NEW PREDICTION ALGORITHM FOR THE NATIONAL BASKETBALL ASSOCIATION (NBA) REGULAR SEASON GAMES

OYELEYE MOJUBAOLUWA

MSc Data Science Project Report

Department of Computer Science and Information Systems

Birkbeck College, University of London

2020

Table of Contents

Acknowledgements

I would like to acknowledge and thank my supervisor Dr. Alessandro Provetti for his help, support and guidance while executing this project.

Abstract

This project focuses on predicting the outcome of regular season games played in the National Basketball Association (NBA). The objective of this project is to build a data model that answers the question 'Will the home team win the match?'

First, we use the NBA regular season match results to generate ranking vectors. These ranking vectors evaluate a rating value for each team as matches are played. We also generate features that capture fatigue for each team based on the team's schedule. Next, we use different feature extraction and selection techniques to approximate the best combination of features. These features are passed into a logistic regression model. We then train, test and evaluate the model.

We also create a surprise metric that measure how surprising our prediction is and critically evaluate the most surprising predictions to understand the weaknesses of the model better and how these weaknesses can be improved upon.

Supervised by Alessandro Provetti

1. Introduction

1.1. THE NATIONAL BASKETBALL ASSOCIATION

The National Basketball Association (NBA) is a men's professional basketball league based in North America currently with 29 teams in USA and 1 in Canada. The NBA is divided into two conferences, the eastern and western conference, each with 15 teams. Both conferences are divided into three divisions, each with five teams [1].

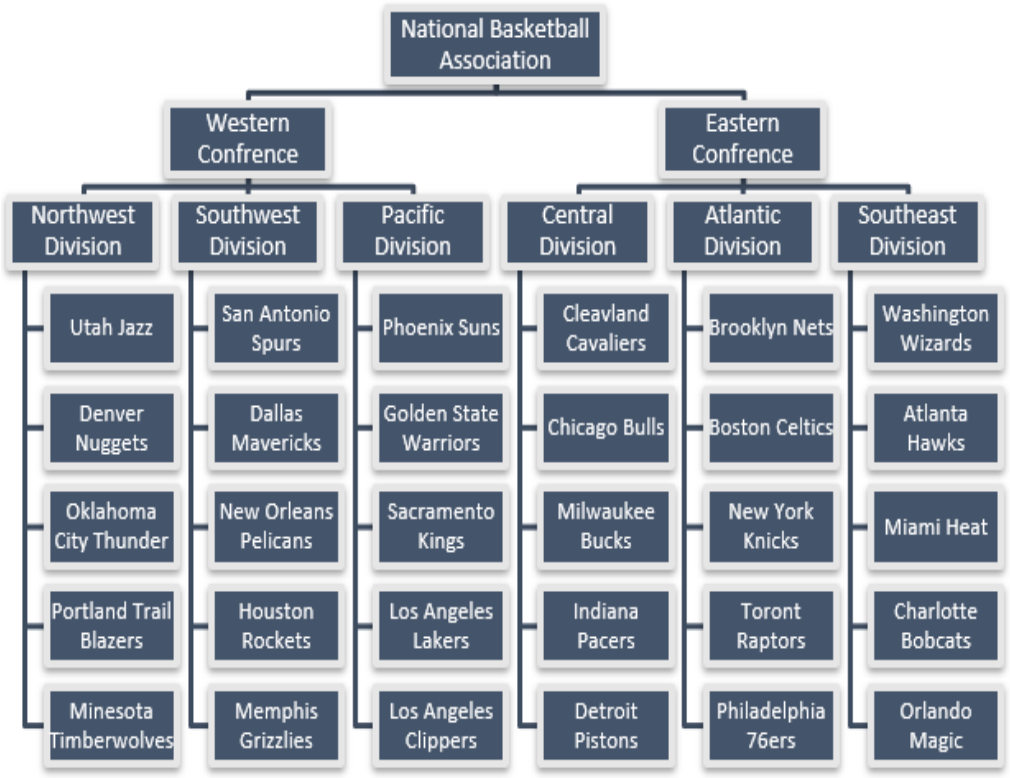

Figure 1:Team breakdown in the NBA

In a full season, each team plays a total of 82 regular season games, 41 home games and 41 away games. A few games each season are played on neutral ground as part of the NBA global games project, nevertheless, this only affects less than 1% of the fixtures. During these global games, one team is still considered the home team and the other the away team even though the game is played on neutral ground. For each team, 4 games are played against each of the other 4 teams in their division (16 games). There are 10 out-of-division but same conference opponents. 4 games are played against 6 of these teams (24 games) and 3 games against the other 4 teams (12 games). 2 games are played against each of the 15 teams in the opposing conference (30 games) [2]. After the regular season, the top 8 teams in each conference move on to a post-season in which each team matchup is a best of 7 series. Two teams, depending on their seeding, play up to 7 consecutive times in predetermined home and away fashion. Games played during the post-season are

often considered different from games played during the regular season [3], hence, only the regular season games were considered in this project.

The data used in this project is from the 2004-2005 season to the 2019-2020 season. During the span, there have been 30 teams in the league. Following the move of the Charlotte Hornets to New Orleans (2002-2003 season), the NBA expanded by adding the Charlotte Bobcats in the 2004-2005 season. This increased the total number of games in a full season to 1230.

The two exceptions are the 2011-2012 season and the 2019-2020 season. The 2011 - 2012 season started late as players and team owners negotiated a new collective bargaining agreement (CBA) [4]. Each team played 66 games, bringing the total number of games played to 990. The 2019-2020 season was suspended due to the coronavirus pandemic [5]. The regular season was put on hold after a total of 971 games. Plans were made to continue the regular season but not in the home and away format. The data from the games played during the continuation was not used in this project.

As at 2019, the average NBA team is worth 1.9 billion dollars, up 13% from 2018 and three times the amount from 2014 [6]. The total NBA revenue for the 2017-2018 season hit 8 billion dollars with only one team losing money [6]. As the NBA popularity and revenue expands, more technology is used in athlete monitoring, and as a result, more data is being collated. This has led to a rise in athlete management systems which try to evaluate players readiness and aptitude to perform by collating, storing and interpreting large volumes of data [7]. Some of the data is collected multiple times on each athlete daily.

The large volumes of data allow for analytical research on millions of data points to facilitate decision making as to how the game should be played. For example, researchers at Ball State University in Muncie, Indiana, analysed over 500,000 free throws and 2 million field goals attempted in NBA matches between 2004 and 2016. The aim of the research was to answer questions concerning the ‘hot-hand’ effect. Does a player become more likely to score with every successful attempt? The researchers found that the effect, which exists but is not strong enough for players or fans to detect, is more noticeable from the foul line. The research measured how the effect grew stronger or weaker with consecutive shots made and how players on offense and defence react to the effect among other things [8]. This is an example of how data can be leveraged in decision making. A team can use findings from similar research to determine who takes a game deciding shot or who to focus on defensively to reduce the likelihood of the opponent team scoring.

The available data also allows for a deeper exploration into how best to determine which team wins a game. If ranking methods, tested on data from 17 consecutive seasons (2004-2005 to 2019-2020), give consistent results with little variation, then the same ranking methods, can be used to predict the outcome of future NBA games.

1.2. RANKING ALGORITHMS

'The problem of ranking in the NBA is elegant and simply – arrange a group of teams (30) in order of relative strength – yet some of its solutions can be complicated and full of paradoxes and conundrums' [9]. An initial approach to solving this problem could be to sum up the individual strengths of each player and coach that has an impact on the team's performance. However, research has shown that team chemistry is a better measure of relative team strength. This is a measure of how well everyone involved (players and coaches) get along together or how well they trust each other. Attempts have been made to quantify team chemistry by a metric called 'prior shared success.' This metric factored in a team's past success in predicting the outcome of games. Across different sports including the NBA, this inclusion was found to improve the accuracy of predicted games by between 2 and 7 per cent [10].

This project follows this logic. Each teams rating is based on how the team performs as one unit. This is measured by looking at a team's prior shared success. Ranking methods from [9] are implemented on each team's past success to determine their relative strengths leading into a game.

1.2.1. MASSEY'S METHOD [9]

The ratings of teams are obtained by solving the system $\bar{\boldsymbol{M}}\boldsymbol{r} = \bar{\boldsymbol{p}}$ to obtain the general Massey rating vector ***r***. To begin, two matrices ***T*** and ***P***, as well as two vectors ***f*** and ***a***, are created.

For ***n*** teams in the league:

T is an $\boldsymbol{n} \times \boldsymbol{n}$ diagonal matrix containing information about the total games played. $\boldsymbol{T_{ii}}$ is equal to the total number of games played by team ***i*** during the season.

P is an $\boldsymbol{n} \times \boldsymbol{n}$ off-diagonal matrix containing pair-wise matchup information. $\boldsymbol{P_{ij}}$ is equal to the number of pair-wise matchups between teams ***i*** and ***j***.

f is an $\boldsymbol{n} \times \boldsymbol{1}$ cumulative points-for vector. $\boldsymbol{f_i}$ is equal to the total points scored by team ***i*** during the season.

a is an $\boldsymbol{n} \times \boldsymbol{1}$ cumulative points-against vector. $\boldsymbol{a_i}$ is equal to the total points scored against team ***i*** during the season.

Next, the matrix ***M*** and vector ***p*** are created as;

$$\boldsymbol{M} = \boldsymbol{T} - \boldsymbol{P}$$

$$\boldsymbol{p} = \boldsymbol{f} - \boldsymbol{a}$$

The rows of $\boldsymbol{M}$ are linearly dependent, so $\boldsymbol{Mr} = \boldsymbol{p}$ does not have a unique solution. Massey's workaround is to replace any row in $\boldsymbol{M}$ with a row of all ones and the corresponding entry of $\boldsymbol{p}$ with a zero. The new system with the row adjustment is denoted

$$\bar{\boldsymbol{M}}\boldsymbol{r} = \bar{\boldsymbol{p}}$$

1.2.2. COLLEY'S METHOD [9]

The ratings of teams are obtained by solving the system $\boldsymbol{Cr} = \boldsymbol{b}$ to obtain the Colley rating vector $\boldsymbol{r}$. $\boldsymbol{C}$ is an $\boldsymbol{n} \times \boldsymbol{n}$ real symmetric positive definite matrix called the Colley matrix defined by;

$$\boldsymbol{C_{ij}} = \begin{cases} \boldsymbol{2 + t_i} & \boldsymbol{i = j}, \\ \boldsymbol{-n_{ij}} & \boldsymbol{i \neq j}. \end{cases}$$

For $\boldsymbol{n}$ teams in the league:

$\boldsymbol{t_i}$ is the total number of games played by team $\boldsymbol{i}$

$\boldsymbol{-n_{ij}}$ is the number of times team $\boldsymbol{i}$ and $\boldsymbol{j}$ faced each other.

The right-hand side vector $\boldsymbol{b}$ is defined by;

$$\boldsymbol{b_i} = \boldsymbol{1} + \frac{\boldsymbol{1}}{\boldsymbol{2}}(\boldsymbol{w_i} - \boldsymbol{l_i})$$

Where:

$\boldsymbol{w_i}$ is the total number of wins accumulated by team $\boldsymbol{i}$

$\boldsymbol{l_i}$ is the total number of losses accumulated by team $\boldsymbol{i}$

1.2.3. THE MARKOV METHOD [9]

This method invokes an old technique from A. A. Markov, and it can be summarized with one word: voting. Every matchup between two teams is an opportunity for the weaker team to vote for the stronger team. In this project, three voting techniques from matchups are explored:

- The losing team votes one point for the team that winning team.
- The losing team votes the point differential for the winning team.
- Both the losing and winning teams vote the points they concede.

This creates a voting matrix, **V** where each entry $\boldsymbol{V_{ij}}$ is the value team, **i**, votes for team $\boldsymbol{j}$. Next, we create a stochastic matrix **S**, by normalising the rows of **V**. This is done by dividing each row by the sum of values in the row. Teams without a loss will result in an all zero row, making **S** substochastic. There are several options for handling undefeated team, however, in this project, a team without a loss will vote 1 for itself.

$$\boldsymbol{S_{ii} = 1 \; \forall \sum_{1}^{j} S_i = 0}$$

To compute the rating $\boldsymbol{r_i}$, for each team, we compute the stationary vector of this stochastic matrix, **S**. The stationary vector, $\boldsymbol{r}$, is the dominant eigenvector of **S** and solves the eigensystem $\boldsymbol{Sr = r.}$

1.2.4. THE OFFENSE-DEFENSE RATING METHOD [9]

The objective is to assign to each team both an offensive rating (denoted by $\boldsymbol{o_i > 0 \; for \; team \; i}$) and a defensive rating (denoted by $\boldsymbol{d_i > 0 \; for \; team \; i}$) that are derived from scores accumulated from past play. Offensive and defensive ratings are intertwined in the sense that each offensive rating is some function of all defensive ratings, and each defensive rating is some function g of all offensive ratings.

To begin with, we create a score matrix A in which $\boldsymbol{A_{ij}}$ is the score that team j generated against team i. (average of the results when the teams have met more than once, and set $\boldsymbol{A_{ij} = 0}$ when the teams have not played each other yet). $\boldsymbol{A_{ij}}$ has a dual interpretation in the sense that:

$$\boldsymbol{A_{ij} = \begin{cases} Number \; of \; points \; that \; j \; scores \; against \; i \\ (an \; offensive \; indicator) \\ and \\ Number \; of \; points \; that \; i \; held \; j \; to \\ (a \; defensive \; indicator) \end{cases}}$$

Since the offensive and defensive ratings are intertwined, it is impossible to directly determine one set without knowing the other. This means that the computing of the offensive-defensive rating can only be accomplished by successive refinement in which one set of ratings is arbitrarily initialized and then successive substitutions are alternately performed until convergence is realized. The defensive ratings vector is initialized with each value as 1.

$$d_0 = \begin{pmatrix} 1 \\ 1 \\ . \\ . \\ . \\ 1 \end{pmatrix}$$

Subsequent offensive and defensive rating vectors are defined by the alternating refinement process

$$\boldsymbol{o_k} = \boldsymbol{A^T d_{k-1}^{\div}},$$

and

$$\boldsymbol{d_k} = \boldsymbol{A o_k^{\div}}, \quad \text{for k = 1, 2, 3, ... 1000}$$

Where $x^{\div}$ is the vector of reciprocals. If

$$x = \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_m \end{pmatrix}$$

Then

$$x^{\div} = \begin{pmatrix} 1/x_1 \\ 1/x_2 \\ . \\ . \\ . \\ 1/x_3 \end{pmatrix}$$

Convergence is measured using the NumPy library in python by checking if the NumPy method, allclose(d_{k-1}, d_k) is False. The maximum number of iterations if there is no convergence is 1000.

Since strong offenses are characterized by big values of $\boldsymbol{o_i}$ while strong defences are characterized by small values of $\boldsymbol{d_i}$, dividing the offensive rating by the corresponding defensive rating makes sense to compute an overall rating. This method allows for the 'large value is better' interpretation of the overall rating $\boldsymbol{r_i}$.

$$r_i = \frac{o_i}{d_i} \text{ for each } \boldsymbol{i}$$

All the ranking algorithms were implemented in python programming language using methods from the Numpy and Pandas libraries for data manipulation. The coding style used is functional programming. The created functions called within the ranking algorithms, as well as the implemented ranking algorithms, are displayed in Appendix 1.

2. Related Work

In literature, there are many proposed methods to predicting the outcome in sports. Most of these proposals try to use win percentages and other team summary statistics as predicting features. A different approach is seen in [11]. Implicit opinions are deduced from match results from 1928-1929 to 2011-2012 seasons in the Spanish Primera Division (Liga) which is the top football league in Spain. These opinions are used in a reputational model based on probabilistic modelling, to compute the strength of each club. The three main concepts are that:

1. The value of an opinion decays with time,
2. The reputation of an opinion source impacts the reliability of the opinion and
3. The certainty of the opinion impacts its weight with respect to other opinions.

Substantial work has also been done on predicting the outcome of regular season NBA games. The problem has been framed in different ways and various techniques have been employed to solve the problem.

In “NBA game prediction based on historical data and Injuries” [12], the goal was to predict the outcome of matches, between the 2011-2012 and the 2015-2016 season, before they happened based on the characteristics of the two teams that are competing. Team-centric metrics, such as average win/loss percentages for both teams participating were collected. Injury records were also tied into each game record. Several machine learning algorithms were used: K-Nearest Neighbours, Linear Regression, Support Vector Machines, J48 Decision Tree, Logistic Model Tree, Naive Bayes, AdaBoost, and a custom majority vote ensemble. Each algorithms performance was measured using 10-fold cross validation. The highest single season accuracy recorded (68.3% using Linear Regression) was over the 2014-2015 season.

In “Prediction of NBA games based on Machine Learning Methods” [13], discovering the most important features in predicting a game’s outcome was one of the objectives. The features used differentiated between the home and away team. Win-Loss percentages of both teams, points differential per game, and head-to-head records both over the whole season and looking only at a preselected number of games. An accuracy rate of 50% was assumed when there was no information available on the teams (the first games of the season or the two teams have equal win percentages). A multi-layer perceptron with two hidden layers (8 and 3 nodes) gave the best average accuracy results (68.44%) when 2006-2007 to 2010-2011 was used as training data and 2011-2012 as testing data.

In “NBA Oracle” [14] , prediction was done using both team and individual player statistics as features. Player statistics (points, rebounds, steals etc.) enable the tracking of team strength depending on the availability of players. Despite player trades and free agency between seasons, the outcomes of games were predicted

based on results from the previous season. The dataset used was for seasons 1992-1993 through 1996-1997. 100-fold cross-validation was used to evaluate performance and the best average accuracy (70.09%) was recorded by a linear regression model.

In “Pre-game Prediction of Outcomes in NBA Matches.” [15], the predictive model was trained and tested on player- and team-based statistics for every game played between the 2002-2003 and 2016-2017 seasons. The dataset includes statistics being recorded at a ‘play-by-play’ event level. The final model has 20 C5.0 decision trees which are polled to get a prediction and give an accuracy of 74.08%. This project also mentions that basketball experts’ on-the-day pre-game prediction accuracy has been estimated to be just below 70% although these experts had the possibility of withholding their predictions in the face of uncertainty.

[14] and [15] used more detailed datasets that included player statistics and [15] also used ‘play-by-play’ event level to predict. As a result, they achieve higher accuracies than [12] and [13].

[12] and [13] use team-centric data which is like the data used in this project. While [13] used only feature vectors generated from previous match results, [12] added in features to account for injuries. In [12], a Naive Bayes model produced the highest average accuracy (66.4%) using data from 2011-2012 to 2015-2016 and cross-validating by holding each seasons data for testing, however, a Linear regression model produced the highest single season accuracy in the 2014-2015 season of 68.3%.

In [13], a multilayer perceptron with 2 hidden layers produced the highest average accuracy (68.44%), however we cannot generalise using the model results since the model was trained using the 2006-2007 to 2011-2012 seasons data and tested on 2012-2013 seasons data. The model may perform well on this seasons prediction but there is no way to know if the model does as well when predicting on other seasons. This result can only be treated as a single seasons accuracy.

Based on the available literature, we aimed to improve on the accuracy results of projects that use similar data. Hence, in this project, we aimed to achieve an average accuracy greater than 67% and a maximum single season accuracy greater than 69%.

3. Design

The algorithm was built in Python using methods and functions from the Pandas, NumPy and scikit-learn libraries. The pandas and numpy libraries were used to manipulate the data to generate features and the scikit-learn library to build machine learning models for classification. Python was selected as the programming language of choice because of the ease and familiarity with dataset manipulations and solving matrix equations with the NumPy and Pandas libraries. Another choice would have been R programming language especially because of the ease with modelling in R.

Figure 2 shows the data flow and interactions between the different parts of the algorithm.

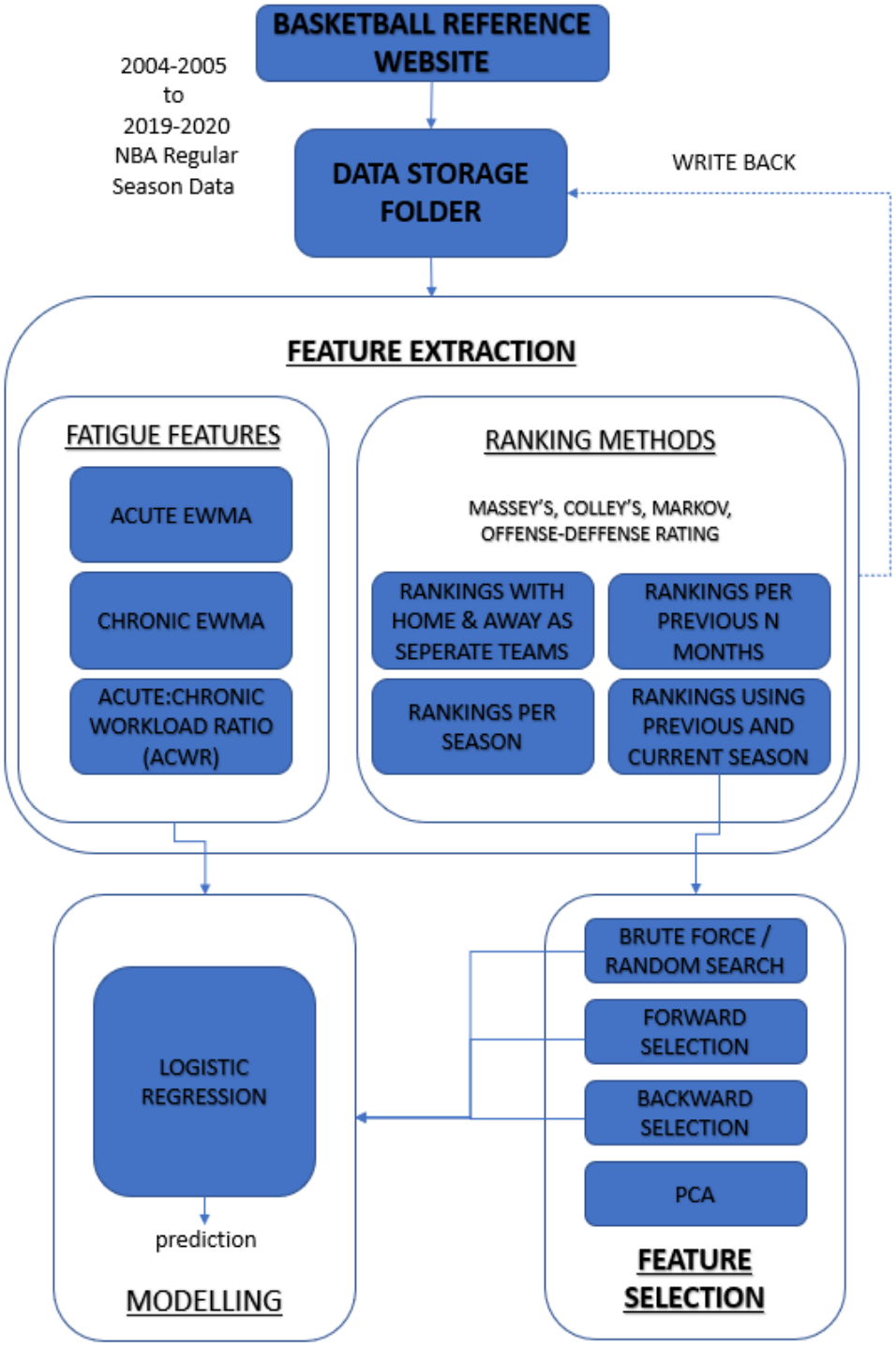

Figure 2: Flow of Data through the design pipeline

CSV files with NBA match results for each season from 2004-2005 to 2019-2020 were downloaded from basketball reference and stored in a directory.

1.3. Feature Generation

Each of the files in the saved directory was used to generate ranking algorithms discussed in Chapter 1. After generating ranking vectors using data from each season separately (called '30 Teams Seasonal' or 'S' in code), the ideas discussed in the project proposal to get a higher accuracy were investigated.

1.3.1. The Cold Start Problem

First, we tried to eliminate the cold start problem. At the start of each season, there was no match history available to rank the competing teams. This resulted in the algorithm guessing for the first few matches each season. To eliminate this problem, all the datasets were joined before ranking so that a new seasons rank is based on all previous seasons result. The results were benchmarked against '30 Teams Seasonal' method and presented in Appendix 2.

From the heat map, more green means '30 Teams Seasonal' method gave results that were better or the same as the method it is compared against. However, observing more closely, the first column, that shows the first season's rankings is completely coloured green. This was expected because there was no previous season for 2004-2005 so the results in the compared datasets were the same. The second column was mostly red. This signified that using one previous season only to tackle the cold start problem could produce better results. This method was tried and the results are shown in Appendix 3. This heat map in Appendix 3 shows that this new ranking (called 'Previous Season' or 'P' in code) gave better results.

1.3.2. Home and away matches treated as separate teams

Next, we investigated splitting a team into two different squads, a home team and an away team, to see if the accuracy could improve. This idea came about during data exploration as some teams performed completely differently during home games as oppose to during away games. An example is the 2019-2020 Philadelphia 76ers which had a 93% winning record playing home games but a 47% winning record playing away games. We tried to improve on the 'Previous Season' ranking method by separating the home teams from the away teams and the results are presented in Appendix 4. This did not improve the accuracy even when used with the '30 Teams Seasonal' method. There is the possibility that this team split provides valuable information in determining the match outcome, for example, it is possible that this split only applies to certain teams, that can be predicted, each season. However, this was not investigated further, and this dataset was discarded moving forward in this project.

1.3.3. Previous 'N' Months

Next, we investigated using the previous N months leading up to a match as the data for ranking where N is an integer and,

$$1 \leq N \leq 6$$

This idea tried to assess whether the 'hot-hand' effect discussed in chapter 1 could be applied on a team level. See Appendix 5,6,7,8,9 and 10 for the results for each value of N. We stopped at the previous 6 months because it covers the span of an entire season. From the heat maps, there was no significant improvement in the accuracy results for any of the values of N.

1.4. Feature exploration

There are 19,180 rows of data used in this project. The dependent variable is a Boolean feature vector that answers the question, 'Did the home team win?' The home team won in 11,345 games and lost in 7835 games which gives a 59/41 target class ratio.

A total of 44 rating feature vectors are generated. 4 ranking methods generate 22 rating feature vectors using only the seasonal data and another 22 using the current and previous seasons data. See Appendix 11 for a list of the 44 vectors. In the code, each feature had a P or S as a one letter suffix signifying whether '30 Teams Seasonal' (S) or the 'Previous Season' (P) data was used in ranking.

In exploring the dataset, we first called the Pandas describe method and transposed the output. The results are presented in Appendix 12. Ratings from the Markov models range between 0 and 0.23 while ratings from Colley's model and Win percentage range between 0 and 1. Ratings from Massey's model have offensive values over 0 and defensive values below 0. The overall ratings vary with a mean of about 0 and a standard deviation close to 5. Offense-Defense ratings have minimum value of 0 but very large maximum values. This is due to the iterative way the rating values are determined. If the values do not converge, they keep increasing exponentially to values as large as 10^{300}. The different ranges of values make scaling necessary before modelling.

In Figure 3, we present the highest accuracy values by each of the ranking methods using '30 Teams Seasonal' (S) data.

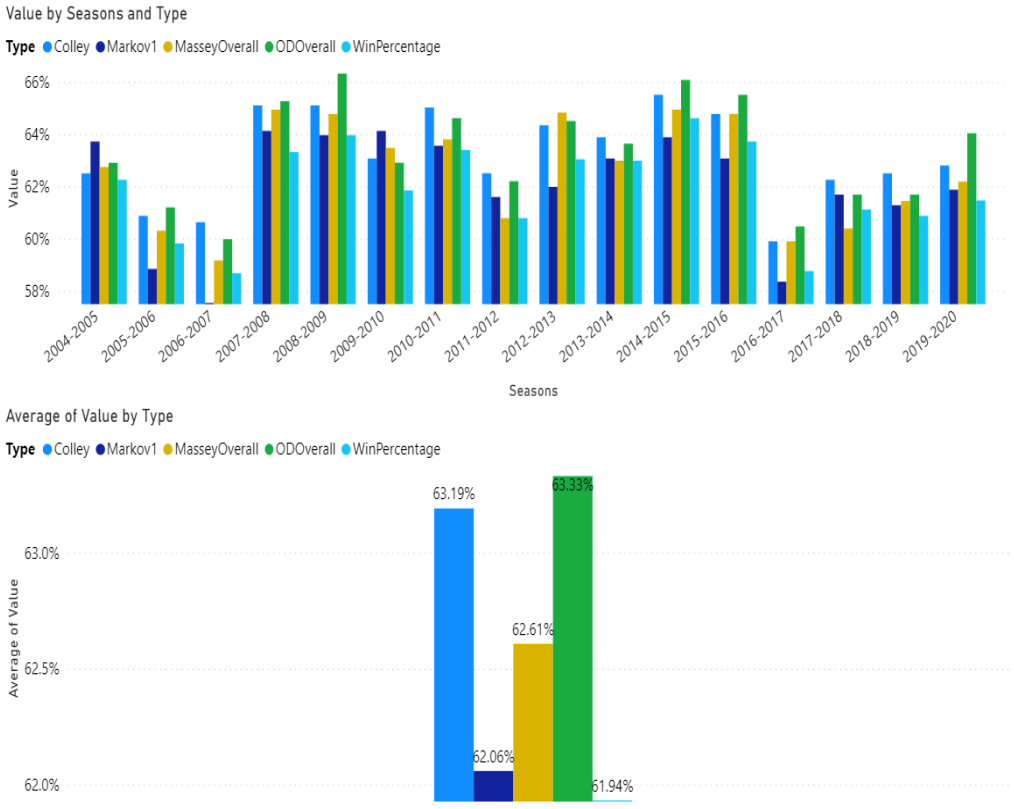

Figure 3:Barchart showing voting accuracies per season and averages using '30 Teams Seasonal' data

The highest average accuracy (63.33%) is from the offense-defense rating method.

In Figure 4, we present the same averages using data from the 'Previous Season' (P) data.

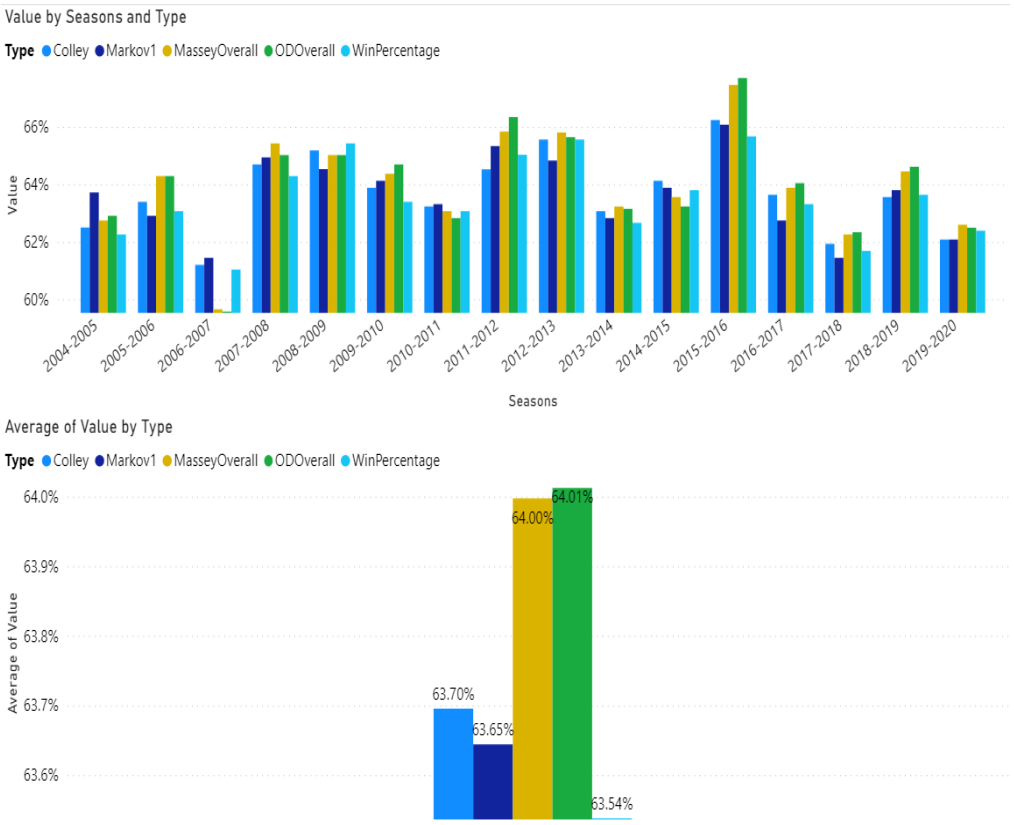

Figure 4:Barchart showing voting accuracies per season and averages using 'Previous Season' data

The highest average accuracy (64.01%) is also from the offense-defense rating method.

Game predictions to calculate accuracy are computed using the difference between the home team and away team rating produced by each rating method.

$$Prediction\ is = \begin{cases} Home\ team, & Home\ team\ rank\ -\ Away\ team\ rank\ >\ 0.00001 \\ Away\ team, & Home\ team\ rank\ -\ Away\ team\ rank\ <\ -0.00001 \end{cases}$$

For similar values for the home and away team, the ranking method does not predict an outcome, and this is assumed to be a wrong prediction when computing the accuracy.

$$Prediction\ is\ wrong\ if f$$

$$abs(Home\ team\ rank\ -\ Away\ team\ rank < 0.00001)$$

In Figure 5 below, we present a heatmap showing the pearson's correlations between the columns. The first thing that stood out was a '#' like shape of low correlation colours running from column numbers 10 to 15 and 32 to 37. The column numbers represent the feature vectors generated by the offense-defense rating method. This could be because the rating method produces very large numbers.

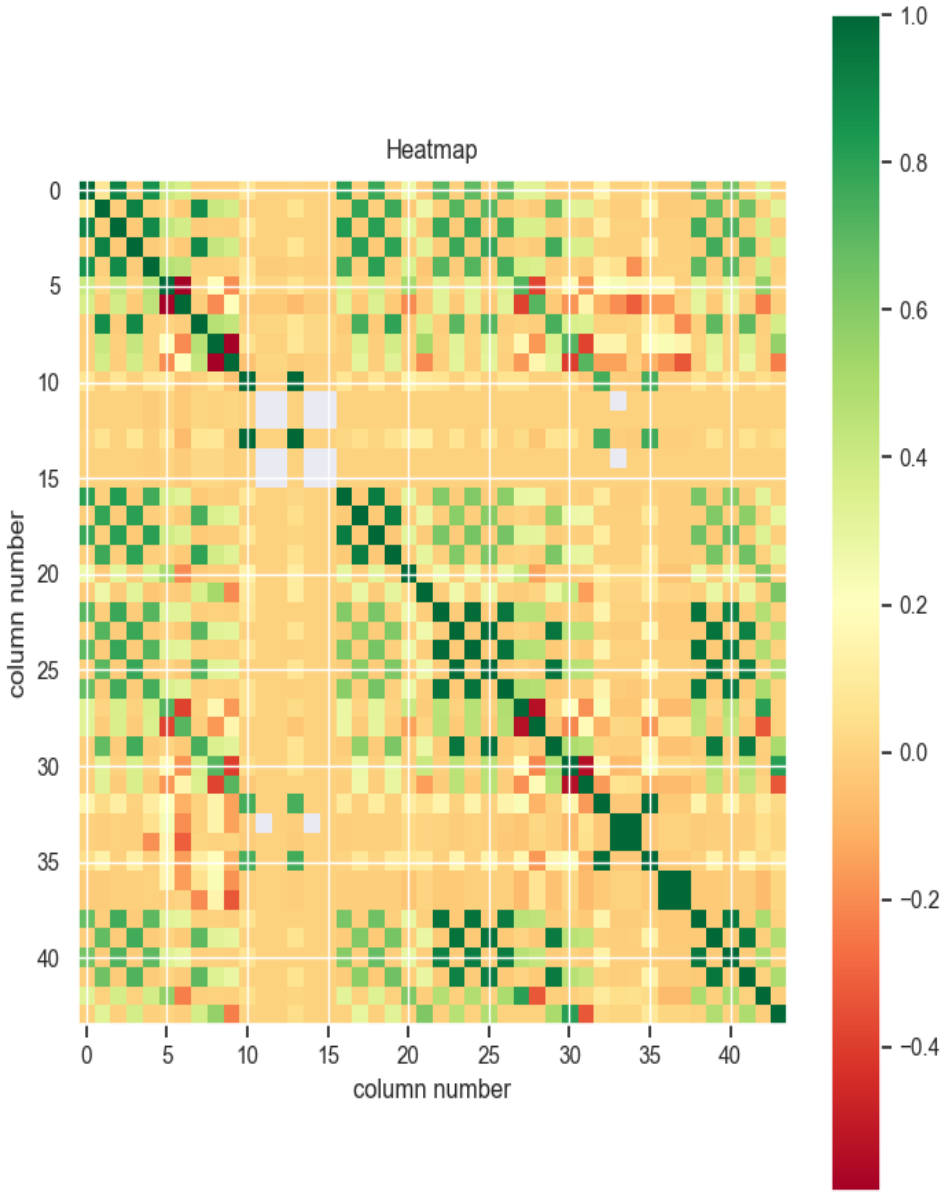

Figure 5:Heatmap showing correlation between feature vectors.

The heatmap also shows that there are not many highly correlated features pairs. This means we can expect to have higher accuracies combining features from different ranking methods to predict in a model.

1.5. Feature Extraction

The next step in the design was to select the features to use in the final model. Again, accuracy was used in evaluating feature performance. Each subset of the extracted features was trained on a logistic regression model using 10-fold cross-validation and the subset with the highest average accuracy was selected as the best feature combination. The standard deviation was also considered to ensure that the selected features will ensure generalization as discussed in Chapter 2. Due to the wide range of rating values by the different ranking methods (Massey's method with negative rating values and Offense-defense method with rating values greater than 1 million), all feature vectors are standardized using Min Max Scaler from the sci-kit learn library.

The feature extraction techniques used were:

1.5.1. Select K Best

This scikit-learn library tool was the only feature selection tool, used in this project, that was a filter method. Filter methods select features without using any machine learning algorithms. Using this method, features were selected based on their scores from a chi-squared statistical test that measures correlation with the dependent variable. Chi-squared can only be applied to non-negative features, so scaling was applied to squash rankings between 0 and 1. See Appendix 13 for the scores for each feature as well as their p-values correct to 4 significant figures.

All possible values of K (1 to 44) were tried and chi-squared was the statistical test. See Appendix 14 for the sorted results. The highest accuracy obtained (66.9%) was with 43 of the 44 features. This showed that this feature selection method did not help to extract the best feature combination.

1.5.2. Sequential Forward Selection

This is the first of the wrapper methods used to select features. It is a greedy search algorithm that provides a suboptimal solution to the best feature combination. During the entire process, the average accuracy value was used to determine which feature combination performed best.

To start the process, we created a list of all the predictors, P. Next, we iteratively tried each of the features, f, as a predictor and the feature that performed the best, b, was removed from P into a separate list F. Next, each of the remaining features was combined one-by-one with b. The feature r, whose combination with b gave the highest accuracy was added to F. this process was repeated iteratively till P was empty and F contained all 44 features.

See Appendix 15 for the results, sorted in decreasing order of accuracy. The highest accuracy obtained (67.14%) was with 22 features.

The function that executes sequential forward selection was written in python programming language using modules from Pandas, NumPy and Sci-kit learn libraries. See Appendix 16 for the code.

1.5.3. Sequential Backward Elimination

This is another greedy search algorithm explored for feature selection. Average accuracy from the cross-validation process was used to determine the best feature combination.

To start the process, we created a list of all the predictors, P. Next, we iteratively removed each of the features from the list, one-by-one. The removed feature that resulted in the highest accuracy was discarded from P. This process was restarted and repeated iteratively, until there was no feature left in P.

See Appendix 17 for the results sorted in decreasing order of accuracy. The highest accuracy obtained (67.15%) was with 27, 29 and 30 features. The feature combination with 27 features was selected as the best since it had the lowest number of features and as a result is the least computationally expensive.

The function that executes sequential backward elinination was written in Python programming language using modules from Pandas, NumPy and Sci-kit learn libraries. See Appendix 18 for the code.

1.5.4. Brute Force and Random Search Approach

With 44 feature vectors, there are approximately 17.6 trillion feature combinations and iterating through ever one of them would be very computationally expensive. For this approach, we combined an exhaustive search algorithm and a random search algorithm.

The algorithm looped through every possible number of features, n (1 feature to 44 features). For values of n less than 5 or greater than 39, there are fewer feature combinations, so an exhaustive search algorithm was used to try every possible feature combination. For feature combinations with n between 6 and 38, a random subset of n features was selected from the 44 features 20,000 times. The randomly selected features were passed into the model and the top 20 accuracy values were stored. See Appendix 19 for the top 20 feature combinations.

The function that executes this feature selection approach was written in Python programming language using modules from Pandas, NumPy, Sci-kit learn and Random libraries. See Appendix 20 for the code.

1.5.5. Principal Component Analysis

Principal component analysis was the only feature extraction technique used in this project that transforms the data. The principal components were derived from all 44 feature vectors and all possible number of principal components were used in the feature selection model. See Appendix 21 for the results sorted in decreasing order of accuracy. The highest accuracy obtained (66.92%) was with 27 principal components.

The highest accuracy value was less than expected. The intuitive thinking is that the different ranking methods will produce feature vectors that approximately vary in the same direction and this is true. Figure 6 shows the cumulative sum of variance explained by the principal components.

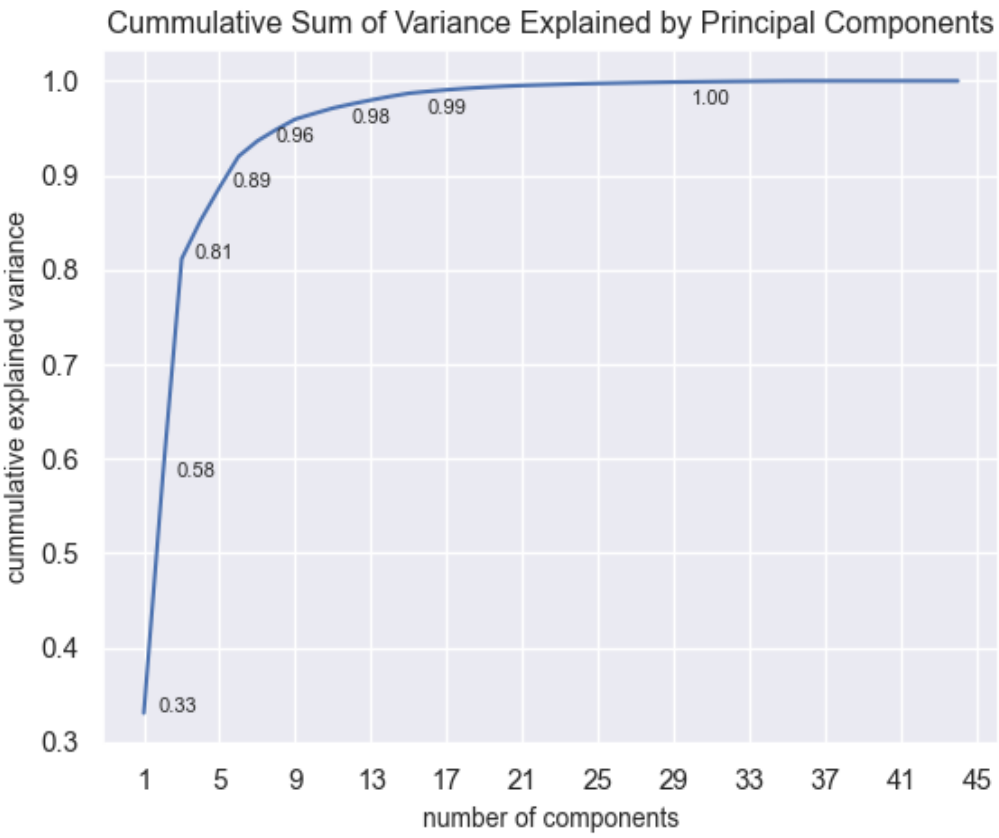

Figure 6: Cummulative Sum of Variance

The top 5 principal components explain 89% of the variance in all 44 feature vectors and the top 17 principal components explain 99% of the variance. However, no combination of principal components led to an accuracy greater than 67%.

The highest accuracy was recorded with 27 principal components (accuracy of 66.92%) but this only explains an additional 0.0384% of variance to 26 principal components (accuracy of 66.89%).

One logical conclusion that can be drawn from this analysis is that some of the feature vectors are bad predictors of the match outcome and adding them in adds noise into the model which reduces the accuracy of the model. This also explains why feature selection techniques that involved eliminating features iteratively produced better accuracy results.

The function that executes the Principal Component Analysis was built in Python programming language using modules from the Pandas, Numpy, Sci-kit learn, Matplotlib and Seaborn libraries. See Appendix 22 for the code.

1.5.6. Conclusion on Feature Extraction

Models with the highest accuracy were chosen from the feature selection methods that were investigated with and that produced accuracies over 67%. These are:

FEATURE EXTRACTION TECNIQUE	NUMBER OF FEATURES IN BEST MODEL	10-FOLD CROSS-VALIDATION ACCURACY	10-FOLD CROSS-VALIDATION STANDARD DEVIATION
SEQUENTIAL FORWARD SELECTION	22	67.12%	1.26%
SEQUENTIAL BACKWARD ELIMINATION	27	67.11%	1.57%
BRUTE FORCE & RANDOM SEARCH APPROACH	20	67.21%	1.40%

The brute force with random search approach produced the highest accuracy with the lowest number of features while the sequential forward selection method produced the lowest standard deviation. However, the goal was to select the best combination of features. These three feature sets are evaluated in the following chapter with better suited validation approaches than 10-fold cross-validation.

In addition to these features, two features that track fatigue were generated, for the home and away teams. All seasons were joined into one dataset and then the acute and chronic versions of the exponential weighted moving averages (EWMA) for workloads were calculated, given by

$$\boldsymbol{EWMA_{today} = LOAD_{today} \times \lambda_a + \left((1 - \lambda_a) \times EWMA_{yesterday}\right)}$$

Where λ_a is a number between 0 and 1 that attributes a decay rate to the load value. This decay rate is calculated as

$$\boldsymbol{\lambda_a = \frac{2}{N + 1}}$$

In this equation, N is the chosen time-decay constant, or in this scenario the time usually given to the acute (7 days) and the chronic (28 days) periods [16].

To calculate the EWMA value for a team, we used the number of minutes the team played each day. An NBA game will have 48 minutes of actual play time. If the scores are tied after the 48 minutes, 5 minutes periods called overtime, are played until there is a winner at the end of an overtime. There is a column from the data source that tells how many overtime periods were played.

The logic behind the load for a team, A, each day is

$$Load = \begin{cases} 0, & Team\ A\ played\ no\ game\ today \\ 48 + (5 * n), & Team\ A\ played\ a\ game\ today \end{cases}$$

$$where\ n = number\ of\ overtime\ periods$$

The fatigue value tends to 0 during the long breaks in between seasons and peaks during the season.

These fatigue features (Acute and Chronic) were added to the model to investigate if either one of them improved the model accuracy. Another concept in measuring load, discussed in [16], is the acute: chronic workload ratio (ACWR), given by

$$ACWR = \frac{Acute\ Workload}{Chronic\ Workload}$$

The ACWR is also passed as a fatigue feature to measure its effect on the final model.

The code that generates fatigue is presented in Appendix 23

1.6. Modelling

All the selected features are passed into a logistic regression model. The logistic model uses the sigmoid function at its core, which can map any real number and map it into a value between 0 and 1, but never exactly those values.

$$f(x) = \frac{1}{(1 + e^{-x})}$$

The logistic regression model works well in a binary classification problem such as this project, in that it is trained to calculate the probability that the target class is 1, Y, and produces a value between 0 and 1. To predict whether the outcome is one or 0, we round up the probability so that

$$Y = \begin{cases} 0, & P(Y=1) < 0.5 \\ 1, & P(Y=1) \geq 0.5 \end{cases}$$

$$where\ P(Y=1)\ is\ the\ probability\ the\ target\ class\ Y\ is\ 1$$

For n features as predictors, we fit the logistic regression model as

$$P(Y=1) = \frac{e^{\beta_0+\beta_1 X_1+\cdots\beta_n X_n}}{1+e^{\beta_0+\beta_1 X_1+\cdots\beta_n X_n}}$$

$$where\ \beta_0, \beta_1, \ldots \beta_n\ are\ weights\ determined\ during\ training\ and$$

$$X_0, X_1, \ldots X_n\ are\ feature\ vectors$$

The logistic regression modelling was done using the sci-kit learn library in Python programming language. The only parameter experimented with is the solver. Since logistic regression is an optimisation problem, different solvers use different methods to find weights combinations that minimize the cost function.

The solvers experimented with are newton-cg, lbfgs, liblinear and saga. Sag was not used since saga is an extension of sag that generally trains faster.

4. Model Evaluation

The goal of the project is to build a model that determines if the home team wins a game. Hence, the metric used in evaluating the final models is accuracy. The data goes through a pipeline built in python using the Pipeline function in the sci-kit learn library, with the following stages:

1. Preprocessing using SimpleImputer to change all nan values to 0. The source of nan values is the inability of the ranking algorithms to produce a rank due to matrix singularity.
2. Preprocessing using MinMaxScaler to squash all values in each feature vector to a feature range between 0 and 1.
3. Feature Selection using predefined functions embedded in FunctionTransformer. The best combination of features from sequential forward selection, sequential backward elimination and the brute force/random search approach are coded into a function that transforms the

dataset before it goes into the model. The fatigue columns are also added in as different features.

4. Modelling using LogisticRegression to predict a 0 or 1. The 4 different models are logistic regression models with different solvers.

In evaluating, data from each season was separated and the model was only trained on data from previous seasons. Two approaches were used:

1.7. 3 Previous Seasons to Predict Current Season

With this validation approach, before predicting on each test season, T, the model was trained on data from the three seasons leading up to T.

$$if\ test\ season = T,$$

$$training\ season = \{T-1, T-2, T-3\}$$

where seasons are consecutive integer numbers ordered by occurrence

The train/test split ratio was approximately 75/25 (not all seasons have the same number of games). Since there were 16 seasons in the dataset, we predicted on 13 seasons, the 2007-2008 to the 2019-2020 season. The validation accuracy values are shown as a table, sorted from the highest to the lowest value in Appendix 24. The highest average across seasons is 67.22% using brute force/random search for feature selection, with the ACWR fatigue feature and liblinear solver.

The highest single season prediction was 2008-2009 with a maximum accuracy of 70.57% using only the backward elimination features with any of the solvers lbfgs, newton-cg or saga.

The top four combinations are presented as a line chart in Figure 7 below using Power BI.

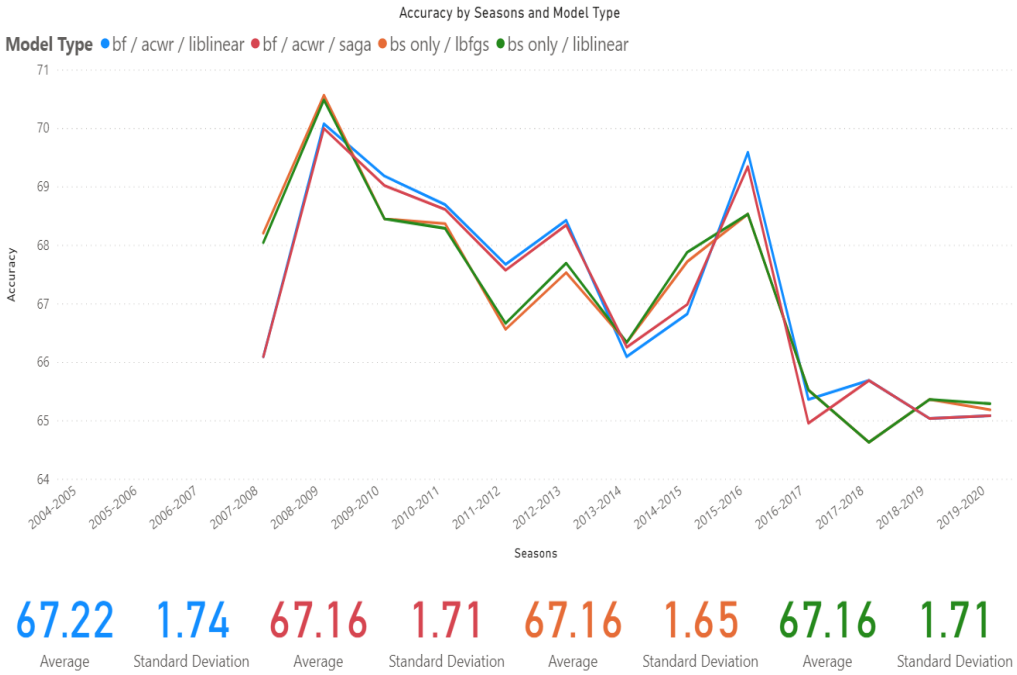

Figure 7: Top 4 models using previous 3 seasons to train

As expected, the models that were built with the same dataset, but different solvers perform similarly in prediction accuracies each season. Predictions over the last 4 seasons (from 2016-2017 to 2019-2020) dipped significantly. It is important to note that the best model included features representing fatigue. Thus, fatigue is a legitimate factor in deciding the outcome of NBA games.

1.8. All Previous Seasons to Predict Current Season

With this validation approach, before predicting on each test season, T, the model was trained on data from all seasons before T.

$$if\ test\ season = T,$$

$$training\ season = \ \ seasons\ where\ season < T$$

where seasons are consecutive integer numbers ordered by occurrence.

The train/test split ratio varies from approximately 50/50 for the 2005-2006 season to approximately 94/6 for the 2019-2020 season.

The validation method helped to investigate whether using all available data to predict a future season would perform better than using a fixed number of prior seasons (3 in this project).

First, predictions were made on every season apart from 2004-2005 as it has no training data. The validation accuracy values are shown as a table, sorted from the highest to the lowest value are presented in Appendix 25. The highest average accuracy is 66.97% which is a significant drop, however, the single season maximum accuracy increased to 70.65% with features from forward selection only and using any of the solvers lbfgs, newton-cg or saga. This showed there was potential to get better results training with all previous seasons.

The 2004-2005 season ranking features were generated differently because there was no previous season data (2003-2004) to create some of its feature vectors. We realised that the noise from this could have potentially distorted the training process, so instead, we used all previous seasons apart from 2004-2005. The validation accuracy values are shown as a table, sorted from the highest to the lowest value are presented in Appendix 26. The highest average accuracy is 67.25% with the backward elimination features only using any of the solvers. This feature vector combination also produced the single season maximum of 70.57% using any of the solvers lbfgs, newton-cg or saga.

The line chart in Figure 8 below was created in Power BI showing the validation results using the 3 different training methods.

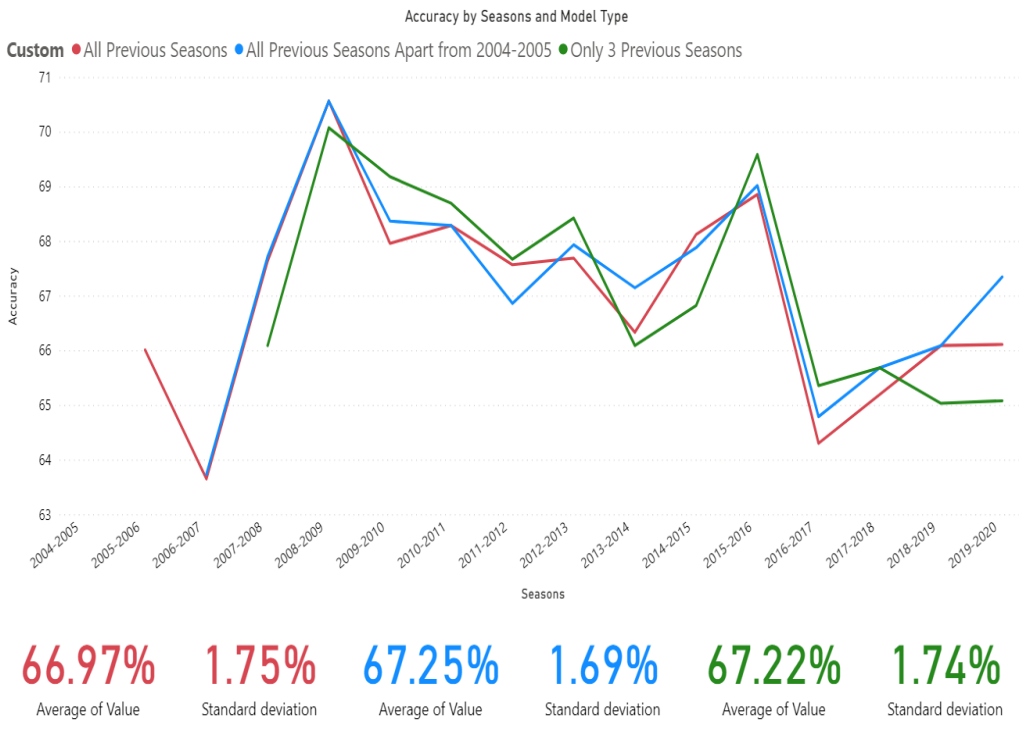

Figure 8: Comparison of the 3 different training methods

The two methods that trained on all seasons data have the lowest recorded accuracy in the 2006-2007 season (63.66% with all previous seasons and 63.74% with all previous seasons except 2004-2005). An explanation could be the lack of enough quality data to train. The model that was trained on all previous seasons apart from

2004-2005 is trained on only the 2005-2006 data to predict the 2006-2007 season. This resulted in a 50/50 train/test ratio and the lower performance can be explained by insufficient training data. Without the prediction on the 2006-2007 season, the average accuracy increases from 67.25% to 67.52%.

In addition, when the models were trained using multiple seasons, the best performing features did not include the fatigue features. The interpretation is that the way fatigue affects match outcomes varies over the year. If a model is trained using a few seasons leading up to a predicted season, fatigue becomes an important feature, however, if a model is trained using a wider range of seasons, fatigue features become noise and diminish the performance of the model.

Apart from fatigue, the only other features used in predicting were generated from the game scores using ranking techniques. Logically, the effect of games scores on the outcome of games will always remain the same in that if a team scores more than the opponent does, the team wins. However, with fatigue, as sports science evolves, the impact of fatigue on game outcome also evolves. NBA teams invest more in Athlete management systems (AMS) to monitor the wellness of their athletes and to reduce the impact fatigue has on players' performance [17]. Also, the NBA has continuously tweaked the schedule to reduce the number of back-to-back games (2 games in two consecutive nights) as well as eliminating sequences of 4 games in 5 nights [18].

5. Surprise Analysis

Predicting a winner when competing teams are of comparable strength is difficult, and this is the case in the NBA. To look deeper at the performance of our model, its limitations as well as discover ways to improve its performance, we did some analysis on the most surprising predictions by our final model in the most recent season (2019-2020).

We selected the 2019-2020 season as it was the most recent season and information about the season was easily accessible.

When writing the project proposal, we hoped to access player availability data from Sports Radar. The presence of certain players in an NBA game has a big impact on winning [19]. An example is Lebron James teams. His teams have a win percentage of 31.9% when he did not play and a win percentage of 66.4% when he played between the 2003-2004 season (his first year) and the 2018-2019 season [20]. However, we did not get a response from the Academic liason at Sports Radar and were unable to get availability data. We proceeded to analyse how much of the surprise predictions from our model can be explained by player availability.

To analyse surprising predictions, we came up with a surprise metric using the probability estimate from the logistic regression models and the score differentials.

The logistic regression model returns a probability estimate for each prediction. This probability estimate informs its prediction. If the probability estimate that a class is 1 is greater than or equal to 0.5 the model predicts the class is 1.

The score differential is calculated as the difference between the home team score and the away team score. This value will be negative if the home team loses and positive if the home team wins. The score differential is then scaled to values between 0 and 1 with the largest win for the home team as 1 and the largest loss for the home team as 0.

The surprise metric is the absolute difference between the scaled points differential and the probability estimate.

$$Suprise = abs(predicted\ probability - scaled\ point\ differential)$$

$$where$$

$$scaled\ point\ differential[i] = \frac{X_i - X_{min}}{X_{max} - X_{min}}$$

$$where$$

$$X(point\ differential\ array) = home\ team\ score - away\ team\ score$$

The surprise metric has a maximum value of 1 if the model is 100% certain the home team wins but the home team loses by the largest margin of the season. Alternatively, if the model is 100% certain the home team loses but the home team wins by the largest margin in the season.

The surprise factor is calculated for all the games played in the 2019-2020 season using the two different training models. We then look at the top 10 surprising results for each of the training models and try to explain the cause of the surprise. See Appendix 27 and Appendix 28 for the top 10 surprising results.

Comparing the two tables, there are 7 of the same top 10 surprise games in the two tables. These games are highlighted. We start the analysis by looking at these 7 games that give surprising predictions independent of how the model was trained.

1) Dallas Mavericks vs Phoenix Suns (104-133)

The best explanation for the surprising Dallas Mavericks loss is that it was a back to back game. This kind of loss is referred to as a scheduled loss by NBA experts [21]. The Dallas Mavericks team played in Oklahoma City the day before, then travelled back home for this game.

2) Los Angeles Clippers vs Memphis Grizzlies (114-140)

The best explanation for the surprising Los Angeles Clippers loss was an unavailable player. Paul George was voted into the top 3 most valuable player the previous season (2019-2020) but was unavailable for this game.

3) Utah Jazz vs Phoenix Suns (111-131)

There is no obvious logical explanation for this surprising loss.

4) Brooklyn Nets vs Memphis Grizzlies (79-118)

This surprising Brooklyn Nets loss can also be explained in that it was a back to back game. The Brooklyn Nets team played the day before in Boston, then travelled back home for this game.

5) Los Angeles Clippers vs Sacramento Kings (103-124)

This loss can be explained by an unavailable player. Kawhi Leonard was the most valuable player in the NBA finals the previous season (2019-2020) but was unavailable for this game.

6) Phoenix Suns vs Houston Rockets (127 -91)

This is another surprising loss that can be explained as it was a back to back game. The Houston Rockets played the day before in Los Angeles before travelling to Phoenix for this game.

A key player was also unavailable for the Houston rockets. Russell Westbrook who won the most valuable player award a few seasons ago (2016-2017) did not play this game.

7) Houston Rockets vs Orlando Magic (106-126)

This is another surprising loss that can be explained as it was a back to back game. The Houston Rockets played the day before in Los Angeles before travelling home for this game.

We also looked at games that were peculiar to one training method. The three games when training with all previous seasons apart from 2004-2005 (See Appendix 28) are:

8) Denver Nuggets vs Golden State Warriors (100-116)

There is no obvious logical explanation for this surprising loss.

9) Atlanta Hawks vs Chicago Bulls (93-113)

This can also be explained as it was a back to back game. The Atlanta Hawks played San Antonio the day before playing this game.

10) Denver Nuggets vs New Orleans Pelicans (100-112)

There is no obvious logical explanation for this surprising loss.

These three surprising outcomes were in the top 20 when training using only the three previous seasons.

The last three surprising games when training with the three previous seasons (See Appendix 27) are:

11) Denver Nuggets vs Cleveland Cavaliers

There is no obvious logical explanation for this surprising loss. This is in the top 20 surprising results when training with all previous seasons apart from 2004-2005.

All the results were surprising irrespective of the training method as they appeared in at least the top 20 surprising result if the other training technique was used. These last two games are different and are only surprising when training with the model with the previous three seasons.

12) Detroit Pistons vs Philadelphia 76ers (111-117)

This game was played close to the beginning of the season and the model makes a surprising prediction. On further investigation, it was discovered that the values from the offense-defense rating method were exceptionally high (over 10^{300}) for the loosing team because it did not converge, and this affected the model's prediction.

13) Golden State Warriors vs Los Angeles Clippers (122-141)

This game was also played to the beginning of the season and the model make a surprising prediction. On further investigation, it was discovered that this prediction was influenced by rating methods that use data from the previous season. In the previous season (2018-2019), Golden State Warriors was a championship contending team winning 57 of their games while the Los Angeles Clippers were an average team winning 48 games. However, during the off season before the start of the 2019-2020 season, players moved to new teams and injured players changed the strengths of the teams dramatically. Golden State Warriors became a below average team while the Clippers became one of the few teams expected to contend for a championship. However, the influence from the previous data affected the model's prediction.

Of the 13 games analysed for their surprise, we could explain the surprise in:

I. 5 games or 38.5% with a back to back scheduled game for the loosing team,
II. 3 games or 23.1% with an unavailable important player on the loosing team,
III. 2 games or 15.4% with the difficulty is solving the cold start problem and
IV. 4 games or 30.8% could not be explained.

We use these probabilities as estimates that can be used for all surprising predictions in the entire dataset.

See Appendix 29 for the code that calculates the surprise metric for each game.

6. Conclusion

At the end of this project, we can conclude that the goals laid out in the proposal were achieved successfully. A data model was built to answer the question, 'Will the NBA home team win?' We set out to answer the question correctly at an average accuracy of 67% and a single season accuracy of 69%. The model was trained and tested using NBA regular season data from the 2004-2005 to the 2019-2020 season.

First, ranking vectors were generated using Massey's, Colley's, Markov and Offense-Defense ranking methods. Fatigue vectors were also generated. Next, we used different feature selection techniques to get the best combinations of features. These features were fed into a logistic regression model for training and testing.

The data was divided by seasons and two training methods were compared and used in validating results. First, the model is trained on 3 consecutive previous seasons to predict on 1 season. The best model had an average accuracy of 67.22% and highest single season accuracy of 70.08%. With the second training method, the model was trained on all the previous seasons apart from the 2004-2005 season because this season had incorrect ranking data. Some of the ranking methods use data from the previous season but since 2004-2005 was the earliest dataset used in this project, it had no previous season. The best model had an average accuracy of 67.25% and highest single season accuracy of 70.49%.

MODEL TRAINING TYPE	HIGHEST AVERAGE ACCURACY	SINGLE SEASON MAXIMUM FROM AVERAGE ACCURACY MODEL	SINGLE SEASON MAXIMUM FROM ANY MODEL
GOAL	**67%**	**69%**	**69%**
3 PREVIOUS SEASONS	**67.22%**	**70.08% (2008-2009)**	**70.57% (2008-2009)**
ALL PREVIOUS SEASONS	**67.25%**	**70.49% (2008-2009)**	**70.57% (2008-2009)**

The Acute Chronic workload ratio (ACWR) was one of the features in the best model when training using 3 previous seasons but there were no fatigue factors in the best model when training with all the previous seasons. From the two training methods, we concluded that the effect that fatigue has on match outcomes has changed over

the years. The major contributors are the NBA scheduling department making deliberate efforts to eliminate this and advances in sports science. As a result, when we train using 3 consecutive seasons, ACWR adds to the accuracy of the results but when we train with all seasons, the fatigue features become noise and reduce the accuracy of the model.

We also analyzed the most surprising predictions to gain more insight on the weaknesses of the predicting model. We used only surprising predictions from the 2019-2020 season as a sample to generalize for all seasons. 30.8% of the most surprising predictions could not be explained by any of the factors we considered. Back to back scheduled games could explain 38.5% of the most surprising predictions, an unavailable player could explain 23.1% and the cold start problem we tried to solve still explained 15.4% of the most surprising outcomes.

The following suggestions should be considered in order to improve the accuracy of the final model as future work.

38.5% of the most surprising outcomes could be attributed to back to back games. We could reduce this percentage by fine tuning the fatigue features. We tried to add a Boolean vector that flagged if a game was the second of a back to back game, but this did not improve the results. However, in calculating the Acute Chronic workload ratio (ACWR) for each team, we used 7 days for the acute period and 28 days for the chronic period. Shorter periods could better capture the effect of a back to back game and capture more information.

To reduce the percentage of surprising outcomes that could be explained by an unavailable player, we would need player availability data. This could be in the form of team lineups for each game. We would also need to capture each player's importance to his team. Player's game statistics could be used. We could then create a measure to estimate what each team misses due to unavailable players.

The last factor that produced surprising predictions was related to the cold start problem. We produced ranking vectors using results from a previous season but between seasons, teams could be completely transformed when players change team. We could account for this with data that captures player trades and club signings. This player data is used in [14] and [15] explains why they report average accuracies above 70%.

Another problem came from the offense-defense ranking method. The cold start problem meant that the iterations did not always lead to team ranks converging. This led to very large values that skewed the model's prediction. In this project, we used 1000 iterations. We could try using fewer iterations or replacing the large ranks with a smaller number.

The accuracy results were higher than accuracy results from existing literature that use similar data and the surprise analysis shows how even higher accuracy would be achieved with more features.

Also, there are other ranking methods that were not considered in this project such as Keener's method and Elo's system [9]. These could be used to create other feature vectors and investigated to see if they improve the model's accuracy.

More work could be done in feature selection also. In this project, we iterated through less that 100,000 out of over 18 trillion feature vector combinations (unique combinations from 50 feature vectors). To try out every feature combination would be very computationally expensive but other feature selection techniques could be employed. One example is stepwise selection which combines forward selection and backward elimination.

Lastly, a more a complex model such as an artificial neural network (ANN) could be used to predict the match outcomes. [13] was the only published work we looked at that used a multilayer perceptron but its best result was achieved using an ANN.

References

[1] Hierarchystructure, "NBA Basketball Team Hierarchy," Hierarchy Structure, 4 December 2017. [Online]. Available: https://www.hierarchystructure.com/nba-team-hierarchy/. [Accessed 21 March 2020].

[2] "How the NBA Schedule is Made," 12th Anniversary, [Online]. Available: https://www.nbastuffer.com/analytics101/how-the-nba-schedule-is-made/. [Accessed 21 March 2020].

[3] "How NBA Playoff Betting Differs From Regular Season," Maddux Sports, 20 March 2013. [Online]. Available: http://www.madduxsports.com/library/nba-playoffs/how-nba-playoff-bets-differs-from-regular-season.html. [Accessed 1 August 2020].

[4] B. Johnson, "Flashback: Key Moments in the 2011 NBA Lockout," 09 April 2020. [Online]. Available: https://basketballforever.com/2020/04/09/flashback-key-moments-in-the-2011-nba-lockout. [Accessed 25 July 2020].

[5] S. Aschburner, "Coronavirus pandemic causes NBA to suspend season after player tests positive," NBA.com, 12 Narch 2020. [Online]. Available: https://www.nba.com/article/2020/03/11/coronavirus-pandemic-causes-nba-suspend-season. [Accessed 25 July 2020].

[6] F. P. Releases, "Forbes Releases 21st Annual NBA Team Valuations," Forbes Press Releases, 6 February 2019. [Online]. Available: https://www.forbes.com/sites/forbespr/2019/02/06/forbes-releases-21st-annual-nba-team-valuations/. [Accessed 25 July 2020].

[7] "Visualizing Data: Athlete Management Systems," BridgeAthletic, 2 November 2018. [Online]. Available: https://blog.bridgeathletic.com/visualizing-data-athlete-management-systems. [Accessed 26 July 2020].

[8] "Success does not always breed success, at least in basketball," Daily chart, 29 May 2020. [Online]. Available: https://www.economist.com/graphic-detail/2020/05/29/success-does-not-always-breed-success-at-least-in-basketball. [Accessed 25 July 2020].

[9] A. N. Langville and C. D. Meyer, Whos #1?: the science of rating and ranking, Princeton, NJ: Princeton University Press, 2012.

[10] N. University, "In team sports, chemistry matters: Sports analytics analysis reveals that past shared success among team members improves odds of future wins," ScienceDaily, 4 December 2018. [Online]. Available: https://www.sciencedaily.com/releases/2018/12/181204095355.htm. [Accessed 28 March 2020].

[11] N. Osman, A. Provetti, V. Riggi and C. Sierra, "MORE: Merged Opinions Reputation Model," in *Multi-agent systems: 12th European Conference, EUMAS 2014*, Prague, 2014.

[12] R. Fang, G. Jean-Baptiste, X. Liu and D. Santiago, "NBA GAME PREDICTION BASED ON HISTORICAL DATA AND INJURIES," [Online]. Available: http://dionny.github.io/NBAPredictions/website/. [Accessed 1 August 2020].

[13] A. Torres, "Prediction of NBA games based on Machine Learning Methods," December 2013. [Online]. Available: http://homepages.cae.wisc.edu/~ece539/fall13/project/AmorimTorres_rpt.pdf. [Accessed 1 August 2020].

[14] M. Beckler, H. Wang and M. Papamichael, "NBA Oracle," 2008. [Online]. Available: https://www.mbeckler.org/coursework/2008-2009/10701_report.pdf. [Accessed 1 August 2020].

[15] "Pre-game Prediction of Outcomes in NBA Matches," All Answers Ltd., November 2018. [Online]. Available: https://ukdiss.com/examples/nba-matches-prediction-of-outcomes.php?vref=1. [Accessed 1 August 2020].

[16] I. Foster, P. J. Byrne, J. A. Moody and P. A. Fitzpatrick, "Monitoring Training Load Using the Acute: Chronic Workload," *ARC Journal of Research in Sports Medicine,* vol. 3, no. 1, pp. 22-28, 2018.

[17] C. Glaeser, "A Buyer's Guide for Athlete Management System Software," Simplifaster, 25 July 2019. [Online]. Available: https://simplifaster.com/articles/buyers-guide-athlete-management-system-software/. [Accessed 12 September 2020].

[18] K. Arnovitz, "By tweaking NBA calendar, league continues the fight against fatigue," ESPN, 5 December 2016. [Online]. Available: https://www.espn.co.uk/nba/story/_/id/18210622/nba-how-adding-days-nba-calendar-help-end-league-fight-fatigue. [Accessed 5 Septembr 2020].

[19] S. K. Deshpande and J. T. Shane, "Estimating an NBA Player’s Impact on His Team’s Chances of Winning," *Journal of Quantitative Analysis in Sports,* vol. 12, no. 2, 2016.

[20] L. Fernández, "Life without LeBron James: what happens when he's out of the lineup?: NBA.com Canada: The official site of the NBA," NBA, 29 December 2018. [Online]. Available: https://ca.nba.com/news/lebron-james-injury-missing-games-los-angeles-lakers-cleveland-cavaliers-miami-heat/1cse7oozda1vv1oploux4nvenz. [Accessed 12 September 2020].

[21] J. Lee, "The anatomy of a scheduled loss," 9 March 2017. [Online]. Available: https://www.goldenstateofmind.com/2017/3/9/14866568/nba-2017-golden-state-warriors-boston-celtics-the-anatomy-of-a-scheduled-loss. [Accessed 12 September 2020].

Appendix

Appendix 1. Ranking Algorithm Functions

```
import numpy as np

import pandas as pd

np.seterr(divide='ignore', invalid='ignore')

def team_order(dataset):

    '''creates a dictionary with teams as items and row number as values to create

    an ordering of teams each season so that matrices and vectors are alligned'''

    teams = sorted(set(dataset['Home Team']).union(set(dataset['Away Team'])))

    return {key: value for value,key in enumerate(teams)}



def points_for(full_data, dataset_of_interest, n):

    '''Creates a vector containing the total number of points scored by each

    team over the time span of the supplied dataset.'''

    team_ordering = team_order(full_data)

    array = np.array([0] * n)

    for i in range(len(dataset_of_interest)):

        #add value of point scored to array

        array[team_ordering[dataset_of_interest['Home Team'][i]]] += dataset_of_interest['Home Score'][i]

        array[team_ordering[dataset_of_interest['Away Team'][i]]] += dataset_of_interest['Away Score'][i]

    return array




def points_against(full_data, dataset_of_interest, n):

    '''Creates a vector containing the total number of points conceded by each

    team over the time span of the supplied dataset'''

    team_ordering = team_order(full_data)

    array = np.array([0] * n)

    for i in range(len(dataset_of_interest)):

        #add value of point conceeded to array
```

```
        array[team_ordering[dataset_of_interest['Home Team'][i]]] += dataset_of_interest['Away Score'][i]
        array[team_ordering[dataset_of_interest['Away Team'][i]]] += dataset_of_interest['Home Score'][i]
    return array


def total_no_played(full_data, dataset_of_interest, n):
    '''Creates a diagonal matrix containing the total number of games played
    by each team over the time span of the supplied dataset'''
    team_ordering = team_order(full_data)
    array = np.array([[0]*n]*n)
    for i in range(len(dataset_of_interest)):
        #add 1 for each game played
        home_index = team_ordering[dataset_of_interest['Home Team'][i]]
        away_index = team_ordering[dataset_of_interest['Away Team'][i]]
        array[home_index, home_index] += 1
        array[away_index, away_index] += 1
    return array


def total_no_won(full_data, dataset_of_interest, n):
    '''Creates a diagonal matrix containing the total number of games won
    by each team over the time span of the supplied dataset'''
    team_ordering = team_order(full_data)
    array = np.array([[0]*n]*n)
    for i in range(len(dataset_of_interest)):
        #add 1 for each game won
        if dataset_of_interest['Home Score'][i] > dataset_of_interest['Away Score'][i]:
            home_index = team_ordering[dataset_of_interest['Home Team'][i]]
            array[home_index, home_index] += 1
        else:
            away_index = team_ordering[dataset_of_interest['Away Team'][i]]
            array[away_index, away_index] += 1
    return array
```

```
def pairwise_matchups(full_data, dataset_of_interest, n):
    '''Creates an off-diagonal matrix containing the number of pairwise
    matchups between teams over the time span of the supplied dataset'''
    team_ordering = team_order(full_data)
    array = np.array([[0]*n]*n)
    for i in range(len(dataset_of_interest)):
        #add 1 for each game between distinct oponents
        home_index = team_ordering[dataset_of_interest['Home Team'][i]]
        away_index = team_ordering[dataset_of_interest['Away Team'][i]]
        array[home_index, away_index] += 1
        array[away_index, home_index] += 1
    return array


def points_given_up(full_data, dataset_of_interest, n, type_):
    '''Creates a matrix containing the total number of points given up to each
    team over the time span of the supplied dataset. The types represent the
    different forms of voting described in the textbook.'''
    team_ordering = team_order(full_data)
    matrix = np.array([[0]*n]*n)
    for i in range(len(dataset_of_interest)):
        #add value of point conceeded to array
        home_index = team_ordering[dataset_of_interest['Home Team'][i]]
        away_index = team_ordering[dataset_of_interest['Away Team'][i]]
        if type_ == 3:
            matrix[home_index, away_index] += dataset_of_interest['Away Score'][i]
            matrix[away_index, home_index] += dataset_of_interest['Home Score'][i]
        elif type_ == 2:
            if dataset_of_interest['Home Score'][i] < dataset_of_interest['Away Score'][i]:
                matrix[home_index, away_index] += dataset_of_interest['Away Score'][i] - dataset_of_interest['Home Score'][
i]
```

```
            else:
                matrix[away_index, home_index] += dataset_of_interest['Home Score'][i] - dataset_of_interest['Away Score'][i]
        elif type_ == 1:
            if dataset_of_interest['Home Score'][i] < dataset_of_interest['Away Score'][i]:
                matrix[home_index, away_index] += 1
            else:
                matrix[away_index, home_index] += 1
    return matrix


def subtract_losses_from_wins(full_data, dataset_of_interest, n):
    '''Creates a vector containing the total number of losses subtracted from
    the total number of wins for each team over the time span of the supplied
    dataset'''
    team_ordering = team_order(full_data)
    array = np.array([0]*n)
    for i in range(len(dataset_of_interest)):
        home_index = team_ordering[dataset_of_interest['Home Team'][i]]
        away_index = team_ordering[dataset_of_interest['Away Team'][i]]
        #checks who won the game
        if dataset_of_interest['Home Score'][i] > dataset_of_interest['Away Score'][i]:
            array[home_index] += 1
            array[away_index] -= 1
        else:
            array[home_index] -= 1
            array[away_index] += 1
    return array



def win_percentage(dataset):
    '''Computes the win percentage of each team. This function works over a
    sorted dataframe by continuously updating the matrices over the span of
```

```
    a supplied dataset.'''

    team_ordering = team_order(dataset)

    n = len(team_ordering)

    home_rating   = []

    away_rating   = []

    unique_dates = dataset.Date.unique()

    for date_index in range(len(unique_dates)):

        data_used_for_ranking = dataset[dataset.Date == unique_dates[date_index-1]]

        data_used_for_ranking.reset_index(inplace = True)

        data_to_be_ranked = dataset[dataset.Date == unique_dates[date_index]]

        data_to_be_ranked.reset_index(inplace = True)

        if date_index == 0:

            total_no_won_array = np.array([[0]*n]*n)

            total_no_played_array = np.array([[0]*n]*n)

        else:

            total_no_won_array   += total_no_won(dataset, data_used_for_ranking, n)

            total_no_played_array   += total_no_played(dataset, data_used_for_ranking, n)

        #r =  np.where(total_no_played_array.diagonal() != 0,

        #              np.divide(total_no_won_array.diagonal(), total_no_played_array.diagonal()), 0)

        #this longer method is necessary to avoid division by 0

        r = []

        for i in range(len(total_no_played_array.diagonal())):

            if total_no_played_array.diagonal()[i] == 0:

                r.append(0)

            else:

                r.append(total_no_won_array.diagonal()[i] / total_no_played_array.diagonal()[i])

        r = np.array(r)

        #append win percentage

        for game_index in range(len(data_to_be_ranked)):

            home_index = team_ordering[data_to_be_ranked['Home Team'][game_index]]

            home_rating.append(r[home_index])
```

```
            away_index = team_ordering[data_to_be_ranked['Away Team'][game_index]]
            away_rating.append(r[away_index])
    dataset['home_win_percentage'] = home_rating
    dataset['away_win_percentage'] = away_rating
    return dataset


def massey_for_a_season(dataset):
    '''Impements the massey ranking method by solving masseys matrix equation
    Mr = p. This function works over a sorted dataframe by continuously
    updating the matrices over the span of a supplied dataset.'''
    team_ordering = team_order(dataset)
    n = len(team_ordering)
    home_overall_rating   = []
    home_offensive_rating = []
    home_defensive_rating = []
    away_overall_rating   = []
    away_offensive_rating = []
    away_defensive_rating = []

    unique_dates = dataset.Date.unique()
    for date_index in range(len(unique_dates)):
        data_used_for_ranking = dataset[dataset.Date == unique_dates[date_index-1]]
        data_used_for_ranking.reset_index(inplace = True)
        data_to_be_ranked = dataset[dataset.Date == unique_dates[date_index]]
        data_to_be_ranked.reset_index(inplace = True)

        if date_index == 0:
            points_for_array = np.array([0]*n)
            points_against_array = np.array([0]*n)
            pairwise_matchups_array = np.array([[0]*n]*n)
            total_no_played_array = np.array([[0]*n]*n)
```

```
else:
    total_no_played_array   += total_no_played(dataset, data_used_for_ranking, n)
    pairwise_matchups_array += pairwise_matchups(dataset, data_used_for_ranking, n)
    points_for_array        += points_for(dataset, data_used_for_ranking, n)
    points_against_array    += points_against(dataset, data_used_for_ranking, n)
#Massy rating calculations start here
T = total_no_played_array
P = pairwise_matchups_array
f = points_for_array
a = points_against_array
p = f - a
M = T - P
#letters are used inline with book to make referencing easy
M[n-1,:] = 1
p[n-1] = 0
try:
    r = np.linalg.solve(M,p)
    d = np.linalg.solve(T+P,np.dot(T,r) - f)
    o = r - d
except np.linalg.LinAlgError:
    d,o,r = ([[None]*n]*3)
#Placing ranks of teams in the dataset
for game_index in range(len(data_to_be_ranked)):
    home_index = team_ordering[data_to_be_ranked['Home Team'][game_index]]
    home_overall_rating.append(r[home_index])
    home_offensive_rating.append(o[home_index])
    home_defensive_rating.append(d[home_index])
    away_index = team_ordering[data_to_be_ranked['Away Team'][game_index]]
    away_overall_rating.append(r[away_index])
    away_offensive_rating.append(o[away_index])
```

```
            away_defensive_rating.append(d[away_index])

    dataset['massey_home_overall_rating']   = home_overall_rating

    dataset['massey_home_offensive_rating'] = home_offensive_rating

    dataset['massey_home_defensive_rating'] = home_defensive_rating

    dataset['massey_away_overall_rating']   = away_overall_rating

    dataset['massey_away_offensive_rating'] = away_offensive_rating

    dataset['massey_away_defensive_rating'] = away_defensive_rating

    return dataset



def colley_for_a_season(dataset):

    '''Impements the colley ranking method by solving colleys matrix equation

    Cr = b. This function works over a sorted dataframe by continuously

    updating the matrices over the span of a supplied dataset.'''

    team_ordering = team_order(dataset)

    n = len(team_ordering)

    home_rating   = []

    away_rating   = []



    unique_dates = dataset.Date.unique()

    for date_index in range(len(unique_dates)):

        data_used_for_ranking = dataset[dataset.Date == unique_dates[date_index-1]]

        data_used_for_ranking.reset_index(inplace = True)

        data_to_be_ranked = dataset[dataset.Date == unique_dates[date_index]]

        data_to_be_ranked.reset_index(inplace = True)

        if date_index == 0:

            subtract_losses_from_wins_array = np.array([0]*n)

            pairwise_matchups_array = np.array([[0]*n]*n)

            total_no_played_array = np.array([[0]*n]*n)

            np.fill_diagonal(total_no_played_array, 2)

        else:

            total_no_played_array   += total_no_played(dataset, data_used_for_ranking, n)
```

```
            pairwise_matchups_array -= pairwise_matchups(dataset, data_used_for_ranking, n)

            subtract_losses_from_wins_array += subtract_losses_from_wins(dataset, data_used_for_ranking, n)

        #Colley rating calculations start here

        T = total_no_played_array

        P = pairwise_matchups_array

        C = T + P

        b = 1 + 0.5 * subtract_losses_from_wins_array

        #letters are used inline with book to make referencing easy

        try:

            r = np.linalg.solve(C,b)

        except np.linalg.LinAlgError:

            r = ([None]*n)

        #Placing ranks of teams in the dataset)

        for game_index in range(len(data_to_be_ranked)):

            home_index = team_ordering[data_to_be_ranked['Home Team'][game_index]]

            home_rating.append(r[home_index])

            away_index = team_ordering[data_to_be_ranked['Away Team'][game_index]]

            away_rating.append(r[away_index])

    dataset['colley_home_rating'] = home_rating

    dataset['colley_away_rating'] = away_rating

    return dataset

def markov_for_a_season(dataset, type_ = 3, beta = 0.6):

    '''Impements the markov ranking method by solving for the stationary

    vector of a voting (winners and losers voting points) stochastic matrix.

    This function works over a sorted dataframe by continuously updating the

    matrices over the span of a supplied dataset.

    types:

    1 -- loser votes only one point for winner

    2 -- loser votes point differential

    3 -- both winner and looser vote points given up

    '''
```

```
    team_ordering = team_order(dataset)

    n = len(team_ordering)

    home_rating = []

    away_rating = []



    unique_dates = dataset.Date.unique()

    for date_index in range(len(unique_dates)):

        data_used_for_ranking = dataset[dataset.Date == unique_dates[date_index-1]]

        data_used_for_ranking.reset_index(inplace = True)

        data_to_be_ranked = dataset[dataset.Date == unique_dates[date_index]]

        data_to_be_ranked.reset_index(inplace = True)



        if date_index == 0:

            voting_matrix = np.array([[0]*n]*n)



        else:

            voting_matrix += points_given_up(dataset, data_used_for_ranking, n, type_)

        #create stocastic matrix

        #line below uses equal voting to other teams by team that has not lost

        #for loop uses vote to self there reorganises the matrix.

        S = np.nan_to_num(voting_matrix/voting_matrix.sum(axis=1, keepdims=True), nan = 1/n)

        for i in range(len(S)):

            if (S[i] == np.array([1/n]*n)).all():

                S[i] = np.array([0]*n)

                S[i][i] = 1

        #calculate the stationary vector

        S = beta * S + (1 - beta) / n * np.array([[1]*n]*n)

        A=np.append(np.transpose(S)-np.identity(n),[[1]*n],axis=0)

        b=np.transpose(np.append(np.array([0]*n),1))

        try:

            r = np.linalg.solve(np.transpose(A).dot(A), np.transpose(A).dot(b))
```

```
        except np.linalg.LinAlgError:
            r = ([None]*n)
        #Placing ranks of teams in the dataset)
        for game_index in range(len(data_to_be_ranked)):
            home_index = team_ordering[data_to_be_ranked['Home Team'][game_index]]
            home_rating.append(r[home_index])
            away_index = team_ordering[data_to_be_ranked['Away Team'][game_index]]
            away_rating.append(r[away_index])
    dataset['markov_home_rating'+str(type_)] = home_rating
    dataset['markov_away_rating'+str(type_)] = away_rating
    return dataset



def od_for_a_season(dataset):
    '''Impements the offense-defence ranking method by solving for the stationary
    vector of a voting (winners and losers voting points) stochastic matrix.
    This function works over a sorted dataframe by continuously updating the
    matrices over the span of a supplied dataset.
    '''
    team_ordering = team_order(dataset)
    n = len(team_ordering)
    home_overall_rating   = []
    home_offensive_rating = []
    home_defensive_rating = []
    away_overall_rating   = []
    away_offensive_rating = []
    away_defensive_rating = []



    unique_dates = dataset.Date.unique()
    for date_index in range(len(unique_dates)):
        data_used_for_ranking = dataset[dataset.Date == unique_dates[date_index-1]]
```

```
        data_used_for_ranking.reset_index(inplace = True)

        data_to_be_ranked = dataset[dataset.Date == unique_dates[date_index]]

        data_to_be_ranked.reset_index(inplace = True)



        if date_index == 0:

            voting_matrix = np.array([[0]*n]*n)



        else:

            voting_matrix += points_given_up(dataset, data_used_for_ranking, n, 3)



        A = voting_matrix

        d = np.array([1.0]*n).reshape(n,1)

        old_d = np.array([0.9]*n).reshape(n,1)

        k = 1

        while k < 1000 and np.allclose(old_d,d) is False:

            old_d = d

            o = np.transpose(A).dot(np.reciprocal(old_d))

            d = A.dot(np.reciprocal(o))

            k += 1

        d,o,r = d,o,o/d

        #Placing ranks of teams in the dataset

        for game_index in range(len(data_to_be_ranked)):

            home_index = team_ordering[data_to_be_ranked['Home Team'][game_index]]

            home_overall_rating.append(r[home_index][0])

            home_offensive_rating.append(o[home_index][0])

            home_defensive_rating.append(d[home_index][0])

            away_index = team_ordering[data_to_be_ranked['Away Team'][game_index]]

            away_overall_rating.append(r[away_index][0])

            away_offensive_rating.append(o[away_index][0])

            away_defensive_rating.append(d[away_index][0])

    dataset['od_home_overall_rating']   = home_overall_rating
```

```
    dataset['od_home_offensive_rating'] = home_offensive_rating

    dataset['od_home_defensive_rating'] = home_defensive_rating

    dataset['od_away_overall_rating']   = away_overall_rating

    dataset['od_away_offensive_rating'] = away_offensive_rating

    dataset['od_away_defensive_rating'] = away_defensive_rating

    return dataset



def rating_for_less(dataset, months, ranking, beta = 0.6):

    '''Impements the different ranking methods over a shorter time span dependent on the

    value of n supplied. n represents how many months (30 days) backward we want to look at

    match results to determine what the team ratings are going to be.

    '''

    unique_dates = dataset.Date.unique()

    if ranking == 'colley' or 'markov' in ranking:

        home_rating   = []

        away_rating   = []

    elif ranking == 'massey' or ranking == 'od':

            home_overall_rating   = []

            home_offensive_rating = []

            home_defensive_rating = []

            away_overall_rating   = []

            away_offensive_rating = []

            away_defensive_rating = []



    for date_index in range(len(unique_dates)):

        end   = pd.to_datetime(unique_dates[date_index])

        start = pd.to_datetime(unique_dates[date_index] - np.timedelta64(30*months,'D'))

        data = dataset.loc[(dataset['Date']>= start)]

        data = data.loc[(data['Date']<= end)]

        if ranking == 'colley':

            colley_for_a_season(data)
```

```
            data = data[data['Date']== unique_dates[date_index]]

            data.reset_index(inplace = True)

            for row in range(len(data)):

                home_rating.append(data.loc[row, 'colley_home_rating'])

                away_rating.append(data.loc[row, 'colley_away_rating'])

        elif  'markov' in ranking:

            type_ = int(ranking[-1])

            markov_for_a_season(data, type_ = type_, beta = beta)

            data = data[data['Date']== unique_dates[date_index]]

            data.reset_index(inplace = True)

            for row in range(len(data)):

                home_rating.append(data.loc[row, 'markov_home_rating'+str(type_)])

                away_rating.append(data.loc[row, 'markov_away_rating'+str(type_)])

        elif ranking == 'massey':

            massey_for_a_season(data)

            data = data[data['Date']== unique_dates[date_index]]

            data.reset_index(inplace = True)

            for row in range(len(data)):

                home_overall_rating.append(data.loc[row, 'massey_home_overall_rating'])

                home_offensive_rating.append(data.loc[row, 'massey_home_offensive_rating'])

                home_defensive_rating.append(data.loc[row, 'massey_home_defensive_rating'])

                away_overall_rating.append(data.loc[row, 'massey_away_overall_rating'])

                away_offensive_rating.append(data.loc[row, 'massey_away_offensive_rating'])

                away_defensive_rating.append(data.loc[row, 'massey_away_defensive_rating'])

        elif ranking == 'od':

            od_for_a_season(data)

            data = data[data['Date']== unique_dates[date_index]]

            data.reset_index(inplace = True)

            for row in range(len(data)):

                home_overall_rating.append(data.loc[row, 'od_home_overall_rating'])

                home_offensive_rating.append(data.loc[row, 'od_home_offensive_rating'])
```

```
                home_defensive_rating.append(data.loc[row, 'od_home_defensive_rating'])
                away_overall_rating.append(data.loc[row, 'od_away_overall_rating'])
                away_offensive_rating.append(data.loc[row, 'od_away_offensive_rating'])
                away_defensive_rating.append(data.loc[row, 'od_away_defensive_rating'])
    if ranking == 'colley':
        dataset['colley_home_%d_month' % months] = home_rating
        dataset['colley_away_%d_month' % months] = away_rating
    elif 'markov' in ranking:
        dataset[ranking + '_home_%d_month' % months] = home_rating
        dataset[ranking + '_away_%d_month' % months] = away_rating
    elif ranking == 'massey':
        dataset['massey_home_overall_%d_month' % months] = home_overall_rating
        dataset['massey_away_overall_%d_month' % months] = away_overall_rating
        dataset['massey_home_offensive_%d_month' % months] = home_offensive_rating
        dataset['massey_away_offensive_%d_month' % months] = away_offensive_rating
        dataset['massey_home_defensive_%d_month' % months] = home_defensive_rating
        dataset['massey_away_defensive_%d_month' % months] = away_defensive_rating
    elif ranking == 'od':
        dataset['od_home_overall_%d_month' % months] = home_overall_rating
        dataset['od_away_overall_%d_month' % months] = away_overall_rating
        dataset['od_home_offensive_%d_month' % months] = home_offensive_rating
        dataset['od_away_offensive_%d_month' % months] = away_offensive_rating
        dataset['od_home_defensive_%d_month' % months] = home_defensive_rating
        dataset['od_away_defensive_%d_month' % months] = away_defensive_rating


    return dataset
def do_seasonal_ranking(data, type_ = 'All'):
    '''takes in a dataset and spits out rankings for every home team and away
    team. This by default doest all types of rankings for a season'''
    year = max(data.Date).year
    percentage = win_percentage(data)
```

```
    #print(percentage)

    colley = colley_for_a_season(data)

    #print(colley)

    massey = massey_for_a_season(data)

    #print(massey)

    od = od_for_a_season(data)

    #print(od)

    markov1 = markov_for_a_season(data,1)

    #print(markov1)

    markov2 = markov_for_a_season(data,2)

    #print(markov2)

    markov3 = markov_for_a_season(data,3)

    #print(markov3[['Away Team','markov_away_rating1', 'markov_away_rating2','markov_away_rating3']])

    #markov3.to_csv('Ranking/' + type_ + ' ' + str(year-1) + '-' + str(year) + '.csv')

    #print(markov3.columns)

    return markov3
def do_timed_ranking(data, type_ = 'All'):

    '''takes in a dataset and spits out rankings for every home team and away

    team. This by default doest all types of rankings for different month lengths

    from 1 month previous to 7 months previous'''

    year = max(data.Date).year

    for i in range(1,8):

        print( 'month', i)

        colley = rating_for_less(data, i, 'colley')

        massey = rating_for_less(data, i, 'massey')

        markov1 = rating_for_less(data, i, 'markov1')

        markov2 = rating_for_less(data, i, 'markov2')

        markov3 = rating_for_less(data, i, 'markov3')

        od = rating_for_less(data, i, 'od')

    #od.to_csv('Rankings/' + type_ + ' ' + str(year-1) + '-' + str(year) + '.csv')

    return od
```

Appendix 2. All Seasons Ranking

30 Teams Seasonal vs One Big Dataset

Type	2004-2005	2005-2006	2006-2007	2007-2008	2008-2009	2009-2010	2010-2011	2011-2012	2012-2013	2013-2014	2014-2015	2015-2016	2016-2017	2017-2018	2018-2019	2019-2020	Averages
Colley																	
30 Teams Seasonal	0.63	0.61	0.61	0.65	0.65	0.63	0.65	0.63	0.64	0.64	0.66	0.65	0.60	0.62	0.63	0.63	**0.63**
Avoid Cold Start	0.63	0.63	0.60	0.58	0.61	0.59	0.59	0.60	0.55	0.52	0.55	0.57	0.57	0.53	0.54	0.57	**0.58**
Markov1																	
30 Teams Seasonal	0.64	0.59	0.58	0.64	0.64	0.64	0.64	0.62	0.62	0.63	0.64	0.63	0.58	0.62	0.61	0.62	**0.62**
Avoid Cold Start	0.64	0.63	0.60	0.58	0.60	0.59	0.59	0.59	0.55	0.52	0.55	0.55	0.56	0.53	0.55	0.57	**0.57**
Markov2																	
30 Teams Seasonal	0.63	0.61	0.57	0.65	0.63	0.64	0.63	0.60	0.63	0.63	0.64	0.64	0.59	0.59	0.60	0.62	**0.62**
Avoid Cold Start	0.63	0.64	0.60	0.59	0.59	0.59	0.58	0.60	0.54	0.54	0.54	0.57	0.58	0.55	0.55	0.57	**0.58**
Markov3																	
30 Teams Seasonal	0.58	0.52	0.51	0.57	0.53	0.55	0.54	0.56	0.56	0.55	0.55	0.55	0.52	0.55	0.54	0.51	**0.54**
Avoid Cold Start	0.58	0.53	0.51	0.54	0.48	0.54	0.53	0.56	0.53	0.52	0.51	0.53	0.54	0.52	0.52	0.54	**0.53**
MasseyDefensive																	
30 Teams Seasonal	0.52	0.54	0.53	0.55	0.61	0.57	0.60	0.57	0.55	0.58	0.58	0.60	0.53	0.53	0.56	0.57	**0.56**
Avoid Cold Start	0.52	0.57	0.53	0.53	0.59	0.53	0.56	0.54	0.51	0.52	0.55	0.53	0.54	0.50	0.51	0.53	**0.53**
MasseyOffensive																	
30 Teams Seasonal	0.57	0.54	0.51	0.59	0.54	0.55	0.55	0.57	0.58	0.58	0.59	0.56	0.55	0.58	0.57	0.55	**0.56**
Avoid Cold Start	0.57	0.52	0.51	0.54	0.49	0.54	0.53	0.56	0.52	0.52	0.51	0.54	0.55	0.53	0.54	0.55	**0.53**
MasseyOverall																	
30 Teams Seasonal	0.62	0.60	0.59	0.65	0.65	0.63	0.64	0.61	0.65	0.63	0.65	0.65	0.60	0.60	0.61	0.62	**0.63**
Avoid Cold Start	0.62	0.64	0.59	0.59	0.60	0.59	0.59	0.59	0.54	0.53	0.55	0.57	0.58	0.55	0.56	0.58	**0.58**
ODDeffensive																	
30 Teams Seasonal	0.50	0.44	0.44	0.44	0.39	0.43	0.41	0.44	0.43	0.43	0.40	0.41	0.44	0.44	0.43	0.38	**0.43**
Avoid Cold Start	0.50	0.43	0.47	0.48	0.42	0.47	0.45	0.47	0.49	0.48	0.45	0.47	0.46	0.49	0.49	0.47	**0.47**
ODOffensive																	
30 Teams Seasonal	0.57	0.52	0.52	0.56	0.52	0.56	0.55	0.57	0.55	0.56	0.55	0.57	0.52	0.55	0.55	0.51	**0.55**
Avoid Cold Start	0.57	0.53	0.51	0.54	0.49	0.54	0.54	0.57	0.52	0.52	0.51	0.54	0.55	0.53	0.53	0.55	**0.53**
ODOverall																	
30 Teams Seasonal	0.63	0.61	0.60	0.65	0.66	0.63	0.65	0.62	0.65	0.64	0.66	0.66	0.60	0.62	0.62	0.64	**0.63**
Avoid Cold Start	0.63	0.64	0.59	0.59	0.61	0.59	0.59	0.59	0.54	0.53	0.55	0.57	0.58	0.55	0.55	0.58	**0.58**
WinPercentage																	
30 Teams Seasonal	0.62	0.60	0.59	0.63	0.64	0.62	0.63	0.61	0.63	0.63	0.65	0.64	0.59	0.61	0.61	0.61	**0.62**
Avoid Cold Start	0.62	0.63	0.60	0.58	0.61	0.59	0.58	0.59	0.55	0.52	0.55	0.57	0.57	0.53	0.54	0.58	**0.57**

* Green if the seasonal ranking value is greater than or equal to the compared value
* Red if the seasonal ranking value is less than the compared value

Appendix 3. Previous Season Alone

30 Teams Seasonal vs Using Previous Season

Type	2004-2005	2005-2006	2006-2007	2007-2008	2008-2009	2009-2010	2010-2011	2011-2012	2012-2013	2013-2014	2014-2015	2015-2016	2016-2017	2017-2018	2018-2019	2019-2020	Averages
Colley																	
30 Teams Seasonal	0.63	0.61	0.61	0.65	0.65	0.63	0.65	0.63	0.64	0.64	0.66	0.65	0.60	0.62	0.63	0.63	**0.63**
Using Previous Season	0.63	0.63	0.61	0.65	0.65	0.64	0.63	0.65	0.66	0.63	0.64	0.66	0.64	0.62	0.64	0.62	**0.64**
Markov1																	
30 Teams Seasonal	0.64	0.59	0.58	0.64	0.64	0.64	0.64	0.62	0.62	0.63	0.64	0.63	0.58	0.62	0.61	0.62	**0.62**
Using Previous Season	0.64	0.63	0.61	0.65	0.65	0.64	0.63	0.65	0.65	0.63	0.64	0.66	0.63	0.61	0.64	0.62	**0.64**
Markov2																	
30 Teams Seasonal	0.63	0.61	0.57	0.65	0.63	0.64	0.63	0.60	0.63	0.63	0.64	0.64	0.59	0.59	0.60	0.62	**0.62**
Using Previous Season	0.63	0.64	0.61	0.64	0.64	0.65	0.63	0.64	0.65	0.63	0.63	0.66	0.62	0.62	0.63	0.61	**0.63**
Markov3																	
30 Teams Seasonal	0.58	0.52	0.51	0.57	0.53	0.55	0.54	0.56	0.56	0.55	0.55	0.55	0.52	0.55	0.54	0.51	**0.54**
Using Previous Season	0.58	0.53	0.51	0.57	0.53	0.55	0.56	0.60	0.59	0.56	0.56	0.59	0.56	0.60	0.56	0.54	**0.56**
MasseyDefensive																	
30 Teams Seasonal	0.52	0.54	0.53	0.55	0.61	0.57	0.60	0.57	0.55	0.58	0.58	0.60	0.53	0.53	0.56	0.57	**0.56**
Using Previous Season	0.53	0.57	0.54	0.55	0.62	0.60	0.60	0.57	0.56	0.59	0.59	0.60	0.57	0.53	0.59	0.60	**0.58**
MasseyOffensive																	
30 Teams Seasonal	0.57	0.54	0.51	0.59	0.54	0.55	0.55	0.57	0.58	0.58	0.59	0.56	0.55	0.58	0.57	0.55	**0.56**
Using Previous Season	0.57	0.52	0.52	0.57	0.53	0.55	0.55	0.60	0.61	0.58	0.59	0.61	0.58	0.61	0.58	0.55	**0.57**
MasseyOverall																	
30 Teams Seasonal	0.62	0.60	0.59	0.65	0.65	0.63	0.64	0.61	0.65	0.63	0.65	0.65	0.60	0.60	0.61	0.62	**0.63**
Using Previous Season	0.63	0.64	0.60	0.65	0.65	0.64	0.63	0.66	0.66	0.63	0.64	0.67	0.64	0.62	0.64	0.63	**0.64**
ODDeffensive																	
30 Teams Seasonal	0.50	0.44	0.44	0.44	0.39	0.43	0.41	0.44	0.43	0.43	0.40	0.41	0.44	0.44	0.43	0.38	**0.43**
Using Previous Season	0.50	0.43	0.46	0.45	0.39	0.42	0.40	0.42	0.47	0.41	0.40	0.39	0.43	0.47	0.41	0.40	**0.43**
ODOffensive																	
30 Teams Seasonal	0.57	0.52	0.52	0.56	0.52	0.56	0.55	0.57	0.55	0.56	0.55	0.57	0.52	0.55	0.55	0.51	**0.55**
Using Previous Season	0.57	0.53	0.52	0.57	0.53	0.55	0.56	0.60	0.59	0.56	0.57	0.60	0.57	0.60	0.57	0.54	**0.56**
ODOverall																	
30 Teams Seasonal	0.63	0.61	0.60	0.65	0.66	0.63	0.65	0.62	0.65	0.64	0.66	0.66	0.60	0.62	0.62	0.64	**0.63**
Using Previous Season	0.63	0.64	0.60	0.65	0.65	0.65	0.63	0.66	0.66	0.63	0.63	0.68	0.64	0.62	0.65	0.63	**0.64**
WinPercentage																	
30 Teams Seasonal	0.62	0.60	0.59	0.63	0.64	0.62	0.63	0.61	0.63	0.63	0.65	0.64	0.59	0.61	0.61	0.61	**0.62**
Using Previous Season	0.62	0.63	0.61	0.64	0.65	0.63	0.63	0.65	0.66	0.63	0.64	0.66	0.63	0.62	0.64	0.62	**0.64**

* Green if the seasonal ranking value is greater than or equal to the compared value
* Red if the seasonal ranking value is less than the compared value

Appendix 4. Home and Away as Separate Teams

30 Teams Seasonal vs Home/Away Different Teams

Type	2004-2005	2005-2006	2006-2007	2007-2008	2008-2009	2009-2010	2010-2011	2011-2012	2012-2013	2013-2014	2014-2015	2015-2016	2016-2017	2017-2018	2018-2019	2019-2020	Averages
Colley																	
Home/Away Seperate	0.66	0.65	0.63	0.62	0.66	0.67	0.65	0.65	0.65	0.63	0.64	0.65	0.64	0.62	0.64	0.63	**0.64**
Previous Season	0.63	0.63	0.61	0.65	0.65	0.64	0.63	0.65	0.66	0.63	0.64	0.66	0.64	0.62	0.64	0.62	**0.64**
Markov1																	
Home/Away Seperate	0.63	0.62	0.60	0.59	0.64	0.63	0.62	0.62	0.62	0.60	0.62	0.63	0.62	0.58	0.61	0.62	**0.62**
Previous Season	0.64	0.63	0.61	0.65	0.65	0.64	0.63	0.65	0.65	0.63	0.64	0.66	0.63	0.61	0.64	0.62	**0.64**
Markov2																	
Home/Away Seperate	0.63	0.63	0.60	0.59	0.64	0.63	0.63	0.62	0.62	0.62	0.62	0.64	0.62	0.60	0.60	0.63	**0.62**
Previous Season	0.63	0.64	0.61	0.64	0.64	0.65	0.63	0.64	0.65	0.63	0.63	0.66	0.62	0.62	0.63	0.61	**0.63**
Markov3																	
Home/Away Seperate	0.54	0.54	0.52	0.53	0.54	0.50	0.52	0.55	0.55	0.53	0.53	0.55	0.54	0.55	0.55	0.52	**0.53**
Previous Season	0.58	0.53	0.51	0.57	0.53	0.55	0.56	0.60	0.59	0.56	0.56	0.59	0.56	0.60	0.56	0.54	**0.56**
MasseyDefensive																	
Home/Away Seperate	0.49	0.50	0.51	0.50	0.51	0.48	0.50	0.50	0.48	0.48	0.49	0.48	0.48	0.50	0.51	0.50	**0.49**
Previous Season	0.53	0.57	0.54	0.55	0.62	0.60	0.60	0.57	0.56	0.59	0.59	0.60	0.57	0.53	0.59	0.60	**0.58**
MasseyOffensive																	
Home/Away Seperate	0.45	0.47	0.46	0.47	0.46	0.48	0.44	0.46	0.45	0.47	0.45	0.47	0.46	0.46	0.44	0.46	**0.46**
Previous Season	0.57	0.52	0.52	0.57	0.53	0.55	0.55	0.60	0.61	0.58	0.59	0.61	0.58	0.61	0.58	0.55	**0.57**
MasseyOverall																	
Home/Away Seperate	0.63	0.64	0.61	0.62	0.65	0.65	0.62	0.64	0.61	0.61	0.61	0.63	0.63	0.61	0.61	0.62	**0.62**
Previous Season	0.63	0.64	0.60	0.65	0.65	0.64	0.63	0.66	0.66	0.63	0.64	0.67	0.64	0.62	0.64	0.63	**0.64**
ODDeffensive																	
Home/Away Seperate	0.47	0.45	0.45	0.45	0.42	0.42	0.41	0.43	0.44	0.44	0.42	0.43	0.43	0.47	0.44	0.43	**0.44**
Previous Season	0.50	0.43	0.46	0.45	0.39	0.42	0.40	0.42	0.47	0.41	0.40	0.39	0.43	0.47	0.41	0.40	**0.43**
ODOffensive																	
Home/Away Seperate	0.53	0.55	0.54	0.54	0.56	0.53	0.53	0.56	0.55	0.55	0.56	0.57	0.56	0.56	0.56	0.52	**0.55**
Previous Season	0.57	0.53	0.52	0.57	0.53	0.55	0.56	0.60	0.59	0.56	0.57	0.60	0.57	0.60	0.57	0.54	**0.56**
ODOverall																	
Home/Away Seperate	0.63	0.64	0.61	0.62	0.67	0.67	0.65	0.65	0.63	0.63	0.63	0.65	0.64	0.62	0.63	0.63	**0.64**
Previous Season	0.63	0.64	0.60	0.65	0.65	0.65	0.63	0.66	0.66	0.63	0.63	0.68	0.64	0.62	0.65	0.63	**0.64**
WinPercentage																	
Home/Away Seperate	0.63	0.63	0.61	0.62	0.66	0.66	0.64	0.64	0.64	0.62	0.63	0.64	0.64	0.61	0.62	0.61	**0.63**
Previous Season	0.62	0.63	0.61	0.64	0.65	0.63	0.63	0.65	0.66	0.63	0.64	0.66	0.63	0.62	0.64	0.62	**0.64**

* Green if the seasonal ranking value is greater than or equal to the compared value
* Red if the seasonal ranking value is less than the compared value

Appendix 5. Previous N months (N = 1)

Previous Season vs Previous 1 Month

Type	2004-2005	2005-2006	2006-2007	2007-2008	2008-2009	2009-2010	2010-2011	2011-2012	2012-2013	2013-2014	2014-2015	2015-2016	2016-2017	2017-2018	2018-2019	2019-2020	Averages
Colley																	
Previous 1Months	0.63	0.60	0.59	0.62	0.64	0.64	0.63	0.61	0.61	0.62	0.64	0.65	0.61	0.60	0.60	0.60	**0.62**
Previous Season	0.63	0.63	0.61	0.65	0.65	0.64	0.63	0.65	0.66	0.63	0.64	0.66	0.64	0.62	0.64	0.62	**0.64**
Markov1																	
Previous 1Months	0.60	0.58	0.56	0.60	0.61	0.61	0.62	0.61	0.59	0.60	0.62	0.62	0.60	0.59	0.59	0.59	**0.60**
Previous Season	0.64	0.63	0.61	0.65	0.65	0.64	0.63	0.65	0.65	0.63	0.64	0.66	0.63	0.61	0.64	0.62	**0.64**
Markov2																	
Previous 1Months	0.60	0.58	0.57	0.60	0.61	0.61	0.63	0.61	0.60	0.61	0.63	0.62	0.60	0.59	0.59	0.59	**0.60**
Previous Season	0.63	0.64	0.61	0.64	0.64	0.65	0.63	0.64	0.65	0.63	0.63	0.66	0.62	0.62	0.63	0.61	**0.63**
Markov3																	
Previous 1Months	0.52	0.50	0.48	0.51	0.51	0.49	0.51	0.53	0.52	0.52	0.54	0.53	0.52	0.52	0.53	0.51	**0.52**
Previous Season	0.58	0.53	0.51	0.57	0.53	0.55	0.56	0.60	0.59	0.56	0.56	0.59	0.56	0.60	0.56	0.54	**0.56**
MasseyDefensive																	
Previous 1Months	0.52	0.54	0.54	0.54	0.57	0.59	0.59	0.58	0.56	0.57	0.55	0.56	0.56	0.54	0.54	0.54	**0.56**
Previous Season	0.53	0.57	0.54	0.55	0.62	0.60	0.60	0.57	0.56	0.59	0.59	0.60	0.57	0.53	0.59	0.60	**0.58**
MasseyOffensive																	
Previous 1Months	0.57	0.53	0.51	0.55	0.55	0.53	0.54	0.55	0.57	0.57	0.58	0.57	0.54	0.54	0.57	0.57	**0.55**
Previous Season	0.57	0.52	0.52	0.57	0.53	0.55	0.55	0.60	0.61	0.58	0.59	0.61	0.58	0.61	0.58	0.55	**0.57**
MasseyOverall																	
Previous 1Months	0.63	0.60	0.58	0.61	0.63	0.63	0.63	0.61	0.61	0.63	0.64	0.64	0.61	0.59	0.61	0.61	**0.62**
Previous Season	0.63	0.64	0.60	0.65	0.65	0.64	0.63	0.66	0.66	0.63	0.64	0.67	0.64	0.62	0.64	0.63	**0.64**
ODDeffensive																	
Previous 1Months	0.46	0.45	0.45	0.44	0.43	0.43	0.43	0.45	0.45	0.45	0.45	0.45	0.44	0.46	0.47	0.44	**0.45**
Previous Season	0.50	0.43	0.46	0.45	0.39	0.42	0.40	0.42	0.47	0.41	0.40	0.39	0.43	0.47	0.41	0.40	**0.43**
ODOffensive																	
Previous 1Months	0.53	0.50	0.49	0.52	0.51	0.49	0.51	0.54	0.52	0.51	0.53	0.54	0.53	0.53	0.54	0.51	**0.52**
Previous Season	0.57	0.53	0.52	0.57	0.53	0.55	0.56	0.60	0.59	0.56	0.57	0.60	0.57	0.60	0.57	0.54	**0.56**
ODOverall																	
Previous 1Months	0.62	0.60	0.59	0.61	0.64	0.64	0.63	0.62	0.62	0.63	0.64	0.65	0.62	0.60	0.61	0.62	**0.62**
Previous Season	0.63	0.64	0.60	0.65	0.65	0.65	0.63	0.66	0.66	0.63	0.63	0.68	0.64	0.62	0.65	0.63	**0.64**

* Green if the seasonal ranking value is greater than or equal to the compared value
* Red if the seasonal ranking value is less than the compared value

Appendix 6. Previous N months (N = 2)

Previous Season vs Previous 2 Month

Type	2004-2005	2005-2006	2006-2007	2007-2008	2008-2009	2009-2010	2010-2011	2011-2012	2012-2013	2013-2014	2014-2015	2015-2016	2016-2017	2017-2018	2018-2019	2019-2020	Averages
Colley																	
Previous 2Months	0.65	0.62	0.59	0.62	0.65	0.65	0.64	0.63	0.64	0.64	0.64	0.64	0.61	0.61	0.62	0.62	**0.63**
Previous Season	0.63	0.63	0.61	0.65	0.65	0.64	0.63	0.65	0.66	0.63	0.64	0.66	0.64	0.62	0.64	0.62	**0.64**
Markov1																	
Previous 2Months	0.62	0.60	0.58	0.60	0.63	0.63	0.63	0.62	0.62	0.63	0.63	0.62	0.60	0.60	0.61	0.61	**0.61**
Previous Season	0.64	0.63	0.61	0.65	0.65	0.64	0.63	0.65	0.65	0.63	0.64	0.66	0.63	0.61	0.64	0.62	**0.64**
Markov2																	
Previous 2Months	0.62	0.61	0.59	0.61	0.64	0.64	0.64	0.62	0.62	0.63	0.63	0.63	0.61	0.59	0.59	0.60	**0.62**
Previous Season	0.63	0.64	0.61	0.64	0.64	0.65	0.63	0.64	0.65	0.63	0.63	0.66	0.62	0.62	0.63	0.61	**0.63**
Markov3																	
Previous 2Months	0.55	0.53	0.52	0.52	0.52	0.52	0.53	0.54	0.54	0.55	0.54	0.53	0.52	0.53	0.54	0.52	**0.53**
Previous Season	0.58	0.53	0.51	0.57	0.53	0.55	0.56	0.60	0.59	0.56	0.56	0.59	0.56	0.60	0.56	0.54	**0.56**
MasseyDefensive																	
Previous 2Months	0.53	0.54	0.55	0.55	0.58	0.59	0.59	0.58	0.56	0.57	0.57	0.58	0.56	0.54	0.54	0.56	**0.56**
Previous Season	0.53	0.57	0.54	0.55	0.62	0.60	0.60	0.57	0.56	0.59	0.59	0.60	0.57	0.53	0.59	0.60	**0.58**
MasseyOffensive																	
Previous 2Months	0.58	0.55	0.53	0.55	0.55	0.54	0.54	0.55	0.57	0.58	0.59	0.58	0.54	0.55	0.57	0.57	**0.56**
Previous Season	0.57	0.52	0.52	0.57	0.53	0.55	0.55	0.60	0.61	0.58	0.59	0.61	0.58	0.61	0.58	0.55	**0.57**
MasseyOverall																	
Previous 2Months	0.64	0.61	0.59	0.62	0.65	0.64	0.63	0.62	0.63	0.64	0.64	0.64	0.61	0.59	0.60	0.62	**0.62**
Previous Season	0.63	0.64	0.60	0.65	0.65	0.64	0.63	0.66	0.66	0.63	0.64	0.67	0.64	0.62	0.64	0.63	**0.64**
ODDeffensive																	
Previous 2Months	0.47	0.45	0.45	0.45	0.42	0.42	0.43	0.43	0.45	0.45	0.43	0.43	0.45	0.46	0.46	0.43	**0.44**
Previous Season	0.50	0.43	0.46	0.45	0.39	0.42	0.40	0.42	0.47	0.41	0.40	0.39	0.43	0.47	0.41	0.40	**0.43**
ODOffensive																	
Previous 2Months	0.54	0.53	0.52	0.52	0.51	0.52	0.53	0.54	0.54	0.54	0.55	0.54	0.52	0.53	0.54	0.53	**0.53**
Previous Season	0.57	0.53	0.52	0.57	0.53	0.55	0.56	0.60	0.59	0.56	0.57	0.60	0.57	0.60	0.57	0.54	**0.56**
ODOverall																	
Previous 2Months	0.64	0.62	0.60	0.62	0.65	0.64	0.64	0.64	0.64	0.65	0.65	0.65	0.62	0.60	0.61	0.62	**0.63**
Previous Season	0.63	0.64	0.60	0.65	0.65	0.65	0.63	0.66	0.66	0.63	0.63	0.68	0.64	0.62	0.65	0.63	**0.64**

* Green if the seasonal ranking value is greater than or equal to the compared value
* Red if the seasonal ranking value is less than the compared value

Appendix 7. Previous N months (N = 3)

Previous Season vs Previous 3 Month

Type	2004-2005	2005-2006	2006-2007	2007-2008	2008-2009	2009-2010	2010-2011	2011-2012	2012-2013	2013-2014	2014-2015	2015-2016	2016-2017	2017-2018	2018-2019	2019-2020	Averages
Colley																	
Previous 3Months	0.63	0.62	0.60	0.62	0.65	0.64	0.64	0.64	0.63	0.64	0.65	0.65	0.62	0.61	0.62	0.63	**0.63**
Previous Season	0.63	0.63	0.61	0.65	0.65	0.64	0.63	0.65	0.66	0.63	0.64	0.66	0.64	0.62	0.64	0.62	**0.64**
Markov1																	
Previous 3Months	0.63	0.61	0.58	0.61	0.64	0.64	0.64	0.63	0.62	0.63	0.63	0.63	0.60	0.60	0.61	0.61	**0.62**
Previous Season	0.64	0.63	0.61	0.65	0.65	0.64	0.63	0.65	0.65	0.63	0.64	0.66	0.63	0.61	0.64	0.62	**0.64**
Markov2																	
Previous 3Months	0.62	0.61	0.59	0.62	0.64	0.64	0.64	0.62	0.61	0.63	0.63	0.63	0.61	0.59	0.60	0.61	**0.62**
Previous Season	0.63	0.64	0.61	0.64	0.64	0.65	0.63	0.64	0.65	0.63	0.63	0.66	0.62	0.62	0.63	0.61	**0.63**
Markov3																	
Previous 3Months	0.57	0.55	0.52	0.53	0.54	0.54	0.54	0.54	0.54	0.54	0.55	0.54	0.52	0.53	0.54	0.53	**0.54**
Previous Season	0.58	0.53	0.51	0.57	0.53	0.55	0.56	0.60	0.59	0.56	0.56	0.59	0.56	0.60	0.56	0.54	**0.56**
MasseyDefensive																	
Previous 3Months	0.53	0.54	0.54	0.54	0.57	0.59	0.59	0.59	0.56	0.57	0.58	0.58	0.56	0.53	0.55	0.57	**0.56**
Previous Season	0.53	0.57	0.54	0.55	0.62	0.60	0.60	0.57	0.56	0.59	0.59	0.60	0.57	0.53	0.59	0.60	**0.58**
MasseyOffensive																	
Previous 3Months	0.57	0.55	0.53	0.55	0.56	0.55	0.55	0.55	0.57	0.58	0.59	0.58	0.55	0.56	0.58	0.57	**0.56**
Previous Season	0.57	0.52	0.52	0.57	0.53	0.55	0.55	0.60	0.61	0.58	0.59	0.61	0.58	0.61	0.58	0.55	**0.57**
MasseyOverall																	
Previous 3Months	0.64	0.62	0.60	0.62	0.65	0.64	0.64	0.62	0.63	0.64	0.64	0.64	0.62	0.60	0.61	0.62	**0.63**
Previous Season	0.63	0.64	0.60	0.65	0.65	0.64	0.63	0.66	0.66	0.63	0.64	0.67	0.64	0.62	0.64	0.63	**0.64**
ODDeffensive																	
Previous 3Months	0.48	0.46	0.45	0.44	0.42	0.42	0.42	0.42	0.44	0.44	0.43	0.42	0.43	0.44	0.43	0.41	**0.44**
Previous Season	0.50	0.43	0.46	0.45	0.39	0.42	0.40	0.42	0.47	0.41	0.40	0.39	0.43	0.47	0.41	0.40	**0.43**
ODOffensive																	
Previous 3Months	0.56	0.55	0.53	0.54	0.53	0.53	0.55	0.56	0.54	0.54	0.55	0.55	0.53	0.53	0.55	0.53	**0.54**
Previous Season	0.57	0.53	0.52	0.57	0.53	0.55	0.56	0.60	0.59	0.56	0.57	0.60	0.57	0.60	0.57	0.54	**0.56**
ODOverall																	
Previous 3Months	0.63	0.62	0.61	0.62	0.65	0.64	0.64	0.63	0.63	0.65	0.65	0.65	0.63	0.60	0.62	0.63	**0.63**
Previous Season	0.63	0.64	0.60	0.65	0.65	0.65	0.63	0.66	0.66	0.63	0.63	0.68	0.64	0.62	0.65	0.63	**0.64**

* Green if the seasonal ranking value is greater than or equal to the compared value
* Red if the seasonal ranking value is less than the compared value

Appendix 8. Previous N months (N = 4)

Previous Season vs Previous 4 Month

Type	2004-2005	2005-2006	2006-2007	2007-2008	2008-2009	2009-2010	2010-2011	2011-2012	2012-2013	2013-2014	2014-2015	2015-2016	2016-2017	2017-2018	2018-2019	2019-2020	Averages
Colley																	
Previous 4Months	0.64	0.62	0.61	0.63	0.65	0.64	0.64	0.64	0.64	0.64	0.64	0.65	0.62	0.61	0.62	0.63	**0.63**
Previous Season	0.63	0.63	0.61	0.65	0.65	0.64	0.63	0.65	0.66	0.63	0.64	0.66	0.64	0.62	0.64	0.62	**0.64**
Markov1																	
Previous 4Months	0.64	0.61	0.58	0.61	0.64	0.64	0.64	0.63	0.62	0.63	0.63	0.63	0.60	0.60	0.61	0.61	**0.62**
Previous Season	0.64	0.63	0.61	0.65	0.65	0.64	0.63	0.65	0.65	0.63	0.64	0.66	0.63	0.61	0.64	0.62	**0.64**
Markov2																	
Previous 4Months	0.63	0.62	0.59	0.61	0.64	0.64	0.63	0.62	0.62	0.63	0.63	0.64	0.61	0.59	0.59	0.61	**0.62**
Previous Season	0.63	0.64	0.61	0.64	0.64	0.65	0.63	0.64	0.65	0.63	0.63	0.66	0.62	0.62	0.63	0.61	**0.63**
Markov3																	
Previous 4Months	0.58	0.55	0.51	0.54	0.55	0.54	0.55	0.55	0.56	0.55	0.55	0.55	0.53	0.52	0.54	0.53	**0.54**
Previous Season	0.58	0.53	0.51	0.57	0.53	0.55	0.56	0.60	0.59	0.56	0.56	0.59	0.56	0.60	0.56	0.54	**0.56**
MasseyDefensive																	
Previous 4Months	0.53	0.54	0.54	0.54	0.58	0.59	0.58	0.59	0.56	0.57	0.58	0.59	0.57	0.54	0.55	0.57	**0.56**
Previous Season	0.53	0.57	0.54	0.55	0.62	0.60	0.60	0.57	0.56	0.59	0.59	0.60	0.57	0.53	0.59	0.60	**0.58**
MasseyOffensive																	
Previous 4Months	0.58	0.56	0.53	0.55	0.57	0.55	0.55	0.56	0.58	0.58	0.59	0.57	0.56	0.56	0.57	0.56	**0.56**
Previous Season	0.57	0.52	0.52	0.57	0.53	0.55	0.55	0.60	0.61	0.58	0.59	0.61	0.58	0.61	0.58	0.55	**0.57**
MasseyOverall																	
Previous 4Months	0.64	0.61	0.60	0.62	0.65	0.64	0.64	0.63	0.63	0.64	0.64	0.65	0.62	0.60	0.61	0.62	**0.63**
Previous Season	0.63	0.64	0.60	0.65	0.65	0.64	0.63	0.66	0.66	0.63	0.64	0.67	0.64	0.62	0.64	0.63	**0.64**
ODDeffensive																	
Previous 4Months	0.49	0.46	0.44	0.44	0.42	0.42	0.42	0.42	0.43	0.43	0.42	0.42	0.43	0.44	0.43	0.41	**0.43**
Previous Season	0.50	0.43	0.46	0.45	0.39	0.42	0.40	0.42	0.47	0.41	0.40	0.39	0.43	0.47	0.41	0.40	**0.43**
ODOffensive																	
Previous 4Months	0.58	0.55	0.52	0.54	0.54	0.54	0.55	0.56	0.56	0.55	0.56	0.56	0.54	0.53	0.55	0.53	**0.55**
Previous Season	0.57	0.53	0.52	0.57	0.53	0.55	0.56	0.60	0.59	0.56	0.57	0.60	0.57	0.60	0.57	0.54	**0.56**
ODOverall																	
Previous 4Months	0.64	0.62	0.60	0.62	0.65	0.65	0.64	0.64	0.64	0.64	0.65	0.66	0.63	0.61	0.62	0.63	**0.63**
Previous Season	0.63	0.64	0.60	0.65	0.65	0.65	0.63	0.66	0.66	0.63	0.63	0.68	0.64	0.62	0.65	0.63	**0.64**

* Green if the seasonal ranking value is greater than or equal to the compared value
* Red if the seasonal ranking value is less than the compared value

Appendix 9. Previous N months (N = 5)

Previous Season vs Previous 5 Month

Type	2004-2005	2005-2006	2006-2007	2007-2008	2008-2009	2009-2010	2010-2011	2011-2012	2012-2013	2013-2014	2014-2015	2015-2016	2016-2017	2017-2018	2018-2019	2019-2020	Averages
Colley																	
Previous 5Months	0.62	0.62	0.61	0.63	0.65	0.64	0.64	0.64	0.64	0.64	0.65	0.65	0.62	0.61	0.62	0.63	**0.63**
Previous Season	0.63	0.63	0.61	0.65	0.65	0.64	0.63	0.65	0.66	0.63	0.64	0.66	0.64	0.62	0.64	0.62	**0.64**
Markov1																	
Previous 5Months	0.63	0.61	0.58	0.61	0.64	0.64	0.64	0.63	0.62	0.63	0.63	0.63	0.61	0.60	0.61	0.61	**0.62**
Previous Season	0.64	0.63	0.61	0.65	0.65	0.64	0.63	0.65	0.65	0.63	0.64	0.66	0.63	0.61	0.64	0.62	**0.64**
Markov2																	
Previous 5Months	0.63	0.62	0.59	0.61	0.64	0.64	0.64	0.62	0.62	0.63	0.64	0.64	0.62	0.59	0.60	0.61	**0.62**
Previous Season	0.63	0.64	0.61	0.64	0.64	0.65	0.63	0.64	0.65	0.63	0.63	0.66	0.62	0.62	0.63	0.61	**0.63**
Markov3																	
Previous 5Months	0.58	0.55	0.51	0.54	0.55	0.54	0.55	0.55	0.56	0.55	0.55	0.55	0.53	0.53	0.54	0.53	**0.54**
Previous Season	0.58	0.53	0.51	0.57	0.53	0.55	0.56	0.60	0.59	0.56	0.56	0.59	0.56	0.60	0.56	0.54	**0.56**
MasseyDefensive																	
Previous 5Months	0.53	0.54	0.54	0.54	0.58	0.59	0.58	0.59	0.56	0.57	0.58	0.59	0.57	0.53	0.55	0.57	**0.56**
Previous Season	0.53	0.57	0.54	0.55	0.62	0.60	0.60	0.57	0.56	0.59	0.59	0.60	0.57	0.53	0.59	0.60	**0.58**
MasseyOffensive																	
Previous 5Months	0.57	0.56	0.53	0.55	0.57	0.55	0.55	0.56	0.57	0.58	0.59	0.58	0.56	0.56	0.57	0.56	**0.56**
Previous Season	0.57	0.52	0.52	0.57	0.53	0.55	0.55	0.60	0.61	0.58	0.59	0.61	0.58	0.61	0.58	0.55	**0.57**
MasseyOverall																	
Previous 5Months	0.63	0.62	0.60	0.62	0.65	0.64	0.64	0.62	0.63	0.64	0.64	0.65	0.62	0.60	0.61	0.62	**0.63**
Previous Season	0.63	0.64	0.60	0.65	0.65	0.64	0.63	0.66	0.66	0.63	0.64	0.67	0.64	0.62	0.64	0.63	**0.64**
ODDeffensive																	
Previous 5Months	0.49	0.47	0.44	0.44	0.41	0.41	0.42	0.42	0.43	0.43	0.41	0.41	0.43	0.44	0.43	0.41	**0.43**
Previous Season	0.50	0.43	0.46	0.45	0.39	0.42	0.40	0.42	0.47	0.41	0.40	0.39	0.43	0.47	0.41	0.40	**0.43**
ODOffensive																	
Previous 5Months	0.57	0.55	0.52	0.54	0.54	0.54	0.55	0.56	0.56	0.55	0.55	0.56	0.55	0.54	0.55	0.53	**0.55**
Previous Season	0.57	0.53	0.52	0.57	0.53	0.55	0.56	0.60	0.59	0.56	0.57	0.60	0.57	0.60	0.57	0.54	**0.56**
ODOverall																	
Previous 5Months	0.63	0.62	0.60	0.63	0.66	0.65	0.64	0.64	0.63	0.64	0.65	0.66	0.63	0.61	0.62	0.63	**0.63**
Previous Season	0.63	0.64	0.60	0.65	0.65	0.65	0.63	0.66	0.66	0.63	0.63	0.68	0.64	0.62	0.65	0.63	**0.64**

* Green if the seasonal ranking value is greater than or equal to the compared value
* Red if the seasonal ranking value is less than the compared value

Appendix 10. Previous N months (N = 6)

Previous Season vs Previous 6 Month

Type	2004-2005	2005-2006	2006-2007	2007-2008	2008-2009	2009-2010	2010-2011	2011-2012	2012-2013	2013-2014	2014-2015	2015-2016	2016-2017	2017-2018	2018-2019	2019-2020	Averages
Colley																	
Previous 6Months	0.63	0.62	0.61	0.63	0.65	0.64	0.64	0.64	0.64	0.64	0.65	0.65	0.62	0.61	0.62	0.63	**0.63**
Previous Season	0.63	0.63	0.61	0.65	0.65	0.64	0.63	0.65	0.66	0.63	0.64	0.66	0.64	0.62	0.64	0.62	**0.64**
Markov1																	
Previous 6Months	0.64	0.61	0.58	0.61	0.64	0.64	0.64	0.63	0.62	0.63	0.63	0.63	0.61	0.60	0.62	0.62	**0.62**
Previous Season	0.64	0.63	0.61	0.65	0.65	0.64	0.63	0.65	0.65	0.63	0.64	0.66	0.63	0.61	0.64	0.62	**0.64**
Markov2																	
Previous 6Months	0.63	0.62	0.59	0.61	0.64	0.64	0.64	0.62	0.62	0.63	0.64	0.64	0.62	0.59	0.60	0.61	**0.62**
Previous Season	0.63	0.64	0.61	0.64	0.64	0.65	0.63	0.64	0.65	0.63	0.63	0.66	0.62	0.62	0.63	0.61	**0.63**
Markov3																	
Previous 6Months	0.58	0.55	0.52	0.54	0.55	0.54	0.55	0.55	0.56	0.55	0.55	0.55	0.53	0.53	0.55	0.53	**0.55**
Previous Season	0.58	0.53	0.51	0.57	0.53	0.55	0.56	0.60	0.59	0.56	0.56	0.59	0.56	0.60	0.56	0.54	**0.56**
MasseyDefensive																	
Previous 6Months	0.53	0.54	0.54	0.54	0.58	0.59	0.58	0.59	0.56	0.57	0.58	0.59	0.57	0.53	0.55	0.57	**0.56**
Previous Season	0.53	0.57	0.54	0.55	0.62	0.60	0.60	0.57	0.56	0.59	0.59	0.60	0.57	0.53	0.59	0.60	**0.58**
MasseyOffensive																	
Previous 6Months	0.57	0.56	0.53	0.55	0.57	0.55	0.55	0.56	0.57	0.58	0.58	0.58	0.56	0.57	0.57	0.56	**0.56**
Previous Season	0.57	0.52	0.52	0.57	0.53	0.55	0.55	0.60	0.61	0.58	0.59	0.61	0.58	0.61	0.58	0.55	**0.57**
MasseyOverall																	
Previous 6Months	0.63	0.62	0.60	0.62	0.65	0.64	0.64	0.62	0.63	0.64	0.64	0.65	0.62	0.60	0.61	0.62	**0.63**
Previous Season	0.63	0.64	0.60	0.65	0.65	0.64	0.63	0.66	0.66	0.63	0.64	0.67	0.64	0.62	0.64	0.63	**0.64**
ODDeffensive																	
Previous 6Months	0.50	0.47	0.44	0.44	0.41	0.41	0.42	0.42	0.43	0.43	0.41	0.41	0.42	0.44	0.44	0.41	**0.43**
Previous Season	0.50	0.43	0.46	0.45	0.39	0.42	0.40	0.42	0.47	0.41	0.40	0.39	0.43	0.47	0.41	0.40	**0.43**
ODOffensive																	
Previous 6Months	0.57	0.55	0.52	0.54	0.54	0.54	0.55	0.56	0.56	0.55	0.55	0.56	0.55	0.54	0.55	0.53	**0.55**
Previous Season	0.57	0.53	0.52	0.57	0.53	0.55	0.56	0.60	0.59	0.56	0.57	0.60	0.57	0.60	0.57	0.54	**0.56**
ODOverall																	
Previous 6Months	0.63	0.62	0.61	0.63	0.66	0.65	0.64	0.64	0.63	0.64	0.65	0.66	0.63	0.61	0.62	0.63	**0.63**
Previous Season	0.63	0.64	0.60	0.65	0.65	0.65	0.63	0.66	0.66	0.63	0.63	0.68	0.64	0.62	0.65	0.63	**0.64**

* Green if the seasonal ranking value is greater than or equal to the compared value
* Red if the seasonal ranking value is less than the compared value

Appendix 11. Ranking Vectors

Each ranking vector is generated using only data leading into each matchup. Feature has a P or S as a subscript signifying whether '30 Teams Seasonal' (S) or the 'Previous Season' (P) data was used in ranking

NO	Name in Code	Description
0	home_win_percentageS	Win percentage of the home team
1	away_win_percentageS	Win percentage of the away team
2	colley_home_ratingS	Colley's method rating of the home team
3	colley_away_ratingS	Colley's method rating of the away team
4	massey_home_overall_ratingS	Massey's method overall rating of the home team
5	massey_home_offensive_ratingS	Massey's method offensive rating of the home team
6	massey_home_defensive_ratingS	Massey's method defensive rating of the home team
7	massey_away_overall_ratingS	Massey's method overall rating of the away team
8	massey_away_offensive_ratingS	Massey's method offensive rating of the away team
9	massey_away_defensive_ratingS	Massey's method defensive rating of the away team
10	od_home_overall_ratingS	The offense-defense rating method overall rating of the home team
11	od_home_offensive_ratingS	The offense-defense rating method offensive rating of the home team
12	od_home_defensive_ratingS	The offense-defense rating method defensive rating of the home team
13	od_away_overall_ratingS	The offense-defense rating method overall rating of the away team
14	od_away_offensive_ratingS	The offense-defense rating method offensive rating of the away team
15	od_away_defensive_ratingS	The offense-defense rating method defensive rating of the away team
16	markov_home_rating1S	The markov method rating for the home team loser voting 1 point
17	markov_away_rating1S	The markov method rating for the away team loser voting 1 point
18	markov_home_rating2S	The markov method rating for the home team loser voting point differential
19	markov_away_rating2S	The markov method rating for the away team loser voting point differential
20	markov_home_rating3S	The markov method rating for the home team loser and winner vote conceded points
21	markov_away_rating3S	The markov method rating for the away team loser and winner vote conceded points
22	home_win_percentageP	Win percentage of the home team
23	away_win_percentageP	Win percentage of the away team
24	colley_home_ratingP	Colley's method rating of the home team
25	colley_away_ratingP	Colley's method rating of the away team
26	massey_home_overall_ratingP	Massey's method overall rating of the home team
27	massey_home_offensive_ratingP	Massey's method offensive rating of the home team
28	massey_home_defensive_ratingP	Massey's method defensive rating of the home team
29	massey_away_overall_ratingP	Massey's method overall rating of the away team
30	massey_away_offensive_ratingP	Massey's method offensive rating of the away team
31	massey_away_defensive_ratingP	Massey's method defensive rating of the away team
32	od_home_overall_ratingP	The offense-defense rating method overall rating of the home team
33	od_home_offensive_ratingP	The offense-defense rating method offensive rating of the home team
34	od_home_defensive_ratingP	The offense-defense rating method defensive rating of the home team
35	od_away_overall_ratingP	The offense-defense rating method overall rating of the away team
36	od_away_offensive_ratingP	The offense-defense rating method offensive rating of the away team
37	od_away_defensive_ratingP	The offense-defense rating method defensive rating of the away team
38	markov_home_rating1P	The markov method rating for the home team loser voting 1 point
39	markov_away_rating1P	The markov method rating for the away team loser voting 1 point
40	markov_home_rating2P	The markov method rating for the home team loser voting point differential
41	markov_away_rating2P	The markov method rating for the away team loser voting point differential
42	markov_home_rating3P	The markov method rating for the home team loser and winner vote conceded points
43	markov_away_rating3P	The markov method rating for the away team loser and winner vote conceded points

Appendix 12. Features Description

NO	COUNT	MEAN	STD	MIN	25%	50%	75%	MAX
0	19180	0.49	0.19	0	0.36	0.5	0.62	1
1	19180	0.5	0.19	0	0.37	0.5	0.62	1
2	19180	0.5	0.15	0.02	0.39	0.5	0.61	0.92
3	19180	0.5	0.15	0.03	0.39	0.5	0.61	0.94
4	18614	-0.04	5.26	-41.22	-3.37	-0.16	3.46	45.23
5	18614	50.37	5.84	-73.2	46.65	50.13	53.71	166.29
6	18614	-50.4	5.76	-175.95	-53.57	-50.51	-47.08	71
7	18614	0.04	5.24	-56.17	-3.26	-0.1	3.52	45.33
8	18614	50.44	5.88	-69.2	46.68	50.21	53.79	167.84
9	18614	-50.4	5.75	-170.95	-53.57	-50.46	-47.06	65
10	18846	4098.26	2346.5	75	2074.03	4053.11	6060.87	9709.79
11	18846	5.98E+298	inf	0	2079.83	4063.28	6057.7	5.87E+302
12	18846	5.69E+296	inf	0	0.95	0.99	1.03	5.36E+300
13	18846	4102.07	2348.28	83	2075.06	4061.5	6064.09	9577.75
14	18846	2.83E+298	inf	0	2095.85	4069.08	6067.96	5.33E+302
15	18846	2.84E+296	inf	0	0.95	0.99	1.03	5.36E+300
16	19180	0.03	0.01	0.01	0.03	0.03	0.04	0.16
17	19180	0.03	0.01	0.01	0.03	0.03	0.04	0.2
18	19180	0.03	0.01	0.01	0.02	0.03	0.04	0.16
19	19180	0.03	0.01	0.01	0.02	0.03	0.04	0.23
20	19180	0.03	0	0.02	0.03	0.03	0.03	0.05
21	19180	0.03	0	0.02	0.03	0.03	0.03	0.05
22	19180	0.5	0.15	0	0.39	0.5	0.61	1
23	19180	0.5	0.15	0	0.39	0.5	0.61	1
24	19180	0.5	0.14	0.11	0.4	0.5	0.6	0.88
25	19180	0.5	0.14	0.1	0.4	0.5	0.6	0.88
26	19151	-0.02	4.26	-45.06	-2.88	-0.02	2.96	36.57
27	19151	50.33	4.43	7	47.18	49.95	53.14	99.94
28	19151	-50.35	4.4	-98	-53.22	-50.57	-47.48	-9.06
29	19151	0.02	4.25	-50.06	-2.84	-0.01	3.01	36.94
30	19151	50.34	4.47	2	47.2	49.97	53.15	92.94
31	19151	-50.32	4.4	-90	-53.2	-50.57	-47.44	-6.06
32	19151	11687.37	3183.16	128.39	9734.42	11869.55	14061.85	18750.25
33	19151	6233814.47	719189561.1	0	9697.37	11849.83	14027.41	97052202438
34	19151	35071.98	3817402.64	0	0.97	1	1.03	499564130.9
35	19151	11691.87	3186.22	123.1	9721.14	11884.41	14070.91	18767.79
36	19151	16961.48	722437.86	0	9701.77	11855.56	14033.03	99981362.13
37	19151	35.04	4643.41	0	0.97	1	1.03	642556.74
38	19180	0.03	0.01	0.01	0.03	0.03	0.04	0.12
39	19180	0.03	0.01	0.01	0.03	0.03	0.04	0.14
40	19180	0.03	0.01	0.01	0.03	0.03	0.04	0.12
41	19180	0.03	0.01	0.01	0.03	0.03	0.04	0.13
42	19180	0.03	0	0.02	0.03	0.03	0.03	0.04
43	19180	0.03	0	0.02	0.03	0.03	0.03	0.05

Appendix 13. Select K Best Scores

COLUMN NUMBER (0 - 43)	SCORE	P-VALUE
24	75.3315	0.0
0	72.7655	0.0
2	70.214	0.0
1	63.8798	0.0
25	60.5866	0.0
3	55.2028	0.0
22	50.2988	0.0
18	44.2324	0.0
23	42.9985	0.0
40	40.016	0.0
41	32.043	0.0
16	25.4764	0.0
19	24.6003	0.0
38	21.8054	0.0
17	16.6388	0.0
39	15.3236	0.0001
4	8.9964	0.0027
26	5.8969	0.0152
7	4.1904	0.0407
29	4.0763	0.0435
12	2.896	0.0888
11	2.7836	0.0952
10	1.9209	0.1658
42	1.574	0.2096
14	1.448	0.2289
15	1.448	0.2289
35	1.3527	0.2448
20	1.3502	0.2452
28	1.3479	0.2456
31	1.2032	0.2727
32	1.0264	0.311
30	0.9798	0.3223
27	0.9165	0.3384
43	0.8969	0.3436
21	0.7279	0.3936
37	0.6588	0.417
13	0.6384	0.4243
6	0.2826	0.595
33	0.2521	0.6156
5	0.246	0.6199
8	0.2433	0.6218
36	0.2058	0.6501
9	0.1804	0.671
34	0.129	0.7194

Appendix 14. Select K Best Cross-Validation Accuracy

COLUMN NUMBER (0 - 43)	NO. OF COLS.	ACCURACY	ST. DEV.
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 36, 37, 38, 39, 40, 41, 42, 43	43	0.669082	0.016496
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43	44	0.668978	0.016577
0, 1, 2, 3, 16, 17, 18, 19, 22, 23, 24, 25, 38, 39, 40, 41	16	0.668717	0.014956
0, 1, 2, 3, 16, 18, 19, 22, 23, 24, 25, 38, 40, 41	14	0.668717	0.014721
0, 1, 2, 3, 16, 18, 19, 22, 23, 24, 25, 40, 41	13	0.668717	0.015017
0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 36, 37, 38, 39, 40, 41, 42, 43	42	0.668561	0.016909
0, 1, 2, 3, 4, 7, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 35, 38, 39, 40, 41, 42	33	0.668509	0.016679
0, 1, 2, 3, 4, 7, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 35, 37, 38, 39, 40, 41, 42, 43	36	0.668405	0.017067
0, 1, 2, 3, 4, 7, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26, 28, 29, 30, 31, 32, 35, 38, 39, 40, 41, 42	32	0.668352	0.017141
0, 1, 2, 3, 4, 7, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 35, 38, 39, 40, 41, 42, 43	34	0.668352	0.016324
0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 37, 38, 39, 40, 41, 42, 43	41	0.668352	0.01755
0, 1, 2, 3, 4, 7, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26, 28, 29, 31, 32, 35, 38, 39, 40, 41, 42	31	0.668352	0.017244
0, 1, 2, 3, 16, 18, 22, 23, 24, 25, 40, 41	12	0.668248	0.015028
0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 37, 38, 39, 40, 41, 42, 43	40	0.668248	0.017175
0, 1, 2, 3, 4, 7, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 35, 37, 38, 39, 40, 41, 42, 43	37	0.668196	0.016905
0, 1, 2, 3, 4, 16, 17, 18, 19, 22, 23, 24, 25, 38, 39, 40, 41	17	0.668144	0.014298
0, 1, 2, 3, 4, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 37, 38, 39, 40, 41, 42, 43	39	0.668144	0.017355
0, 1, 2, 3, 4, 7, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 35, 38, 39, 40, 41, 42, 43	35	0.668144	0.016784
0, 1, 2, 3, 4, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 35, 37, 38, 39, 40, 41, 42, 43	38	0.668092	0.017491
0, 1, 2, 3, 16, 17, 18, 19, 22, 23, 24, 25, 38, 40, 41	15	0.667987	0.014069
0, 1, 2, 3, 4, 7, 16, 17, 18, 19, 22, 23, 24, 25, 26, 29, 38, 39, 40, 41	20	0.667987	0.016029
0, 1, 2, 3, 4, 7, 10, 11, 12, 16, 17, 18, 19, 22, 23, 24, 25, 26, 29, 38, 39, 40, 41	23	0.667831	0.015703
0, 1, 2, 3, 4, 7, 10, 11, 12, 14, 15, 16, 17, 18, 19, 22, 23, 24, 25, 26, 29, 35, 38, 39, 40, 41, 42	27	0.667831	0.016457
0, 1, 2, 3, 4, 7, 10, 11, 12, 14, 15, 16, 17, 18, 19, 22, 23, 24, 25, 26, 29, 38, 39, 40, 41, 42	26	0.667779	0.015922
0, 1, 2, 3, 4, 7, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26, 29, 35, 38, 39, 40, 41, 42	28	0.667779	0.016265
0, 1, 2, 3, 4, 16, 17, 18, 19, 22, 23, 24, 25, 26, 38, 39, 40, 41	18	0.667727	0.014315
0, 1, 2, 3, 4, 7, 11, 12, 16, 17, 18, 19, 22, 23, 24, 25, 26, 29, 38, 39, 40, 41	22	0.667727	0.016428
0, 1, 2, 3, 4, 7, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26, 28, 29, 35, 38, 39, 40, 41, 42	29	0.667727	0.01684
0, 1, 2, 24, 25	5	0.667675	0.014141
0, 1, 2, 3, 24, 25	6	0.667623	0.014176
0, 1, 2, 3, 18, 22, 23, 24, 25	9	0.667623	0.014258
0, 1, 2, 3, 18, 22, 23, 24, 25, 40	10	0.667623	0.014383
0, 1, 2, 3, 18, 22, 23, 24, 25, 40, 41	11	0.667623	0.014775
0, 1, 2, 3, 4, 7, 10, 11, 12, 16, 17, 18, 19, 22, 23, 24, 25, 26, 29, 38, 39, 40, 41, 42	24	0.667623	0.016006
0, 1, 2, 3, 4, 7, 10, 11, 12, 15, 16, 17, 18, 19, 22, 23, 24, 25, 26, 29, 38, 39, 40, 41, 42	25	0.667623	0.016208
0, 1, 2, 3, 4, 7, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26, 28, 29, 31, 35, 38, 39, 40, 41, 42	30	0.667623	0.017182
0, 1, 2, 3, 4, 7, 12, 16, 17, 18, 19, 22, 23, 24, 25, 26, 29, 38, 39, 40, 41	21	0.667518	0.016509
0, 1, 2, 3, 18, 22, 24, 25	8	0.667414	0.01417
0, 1, 2, 3, 4, 7, 16, 17, 18, 19, 22, 23, 24, 25, 26, 38, 39, 40, 41	19	0.667362	0.014926
0, 1, 2, 3, 22, 24, 25	7	0.667205	0.0144
0, 1, 2, 24	4	0.662565	0.014375
0, 2, 24	3	0.638425	0.01194
0, 24	2	0.632169	0.012477
24	1	0.62586	0.015287

Appendix 15. Sequential Forward Selection Accuracies

COLUMN NUMBER (0 - 43)	NO. OF COLS.	ACCURACY	ST. DEV.
0, 2, 4, 8, 9, 11, 12, 14, 15, 16, 17, 19, 21, 22, 25, 26, 30, 33, 34, 36, 38, 41	22	0.67122	0.01257
0, 2, 4, 8, 9, 11, 12, 14, 15, 16, 17, 19, 20, 21, 22, 25, 26, 30, 33, 34, 36, 38, 41	23	0.671168	0.01298
0, 2, 4, 6, 8, 9, 11, 12, 14, 15, 16, 17, 19, 20, 21, 22, 25, 26, 30, 33, 34, 36, 38, 41	24	0.671011	0.013064
0, 2, 4, 6, 8, 9, 11, 12, 14, 15, 16, 17, 19, 20, 21, 22, 25, 26, 30, 32, 33, 34, 36, 38, 41	25	0.670907	0.013349
0, 2, 4, 5, 6, 8, 9, 11, 12, 14, 15, 16, 17, 19, 20, 21, 22, 25, 26, 27, 28, 30, 32, 33, 34, 36, 38, 40, 41, 42	30	0.670907	0.013144
0, 2, 4, 8, 9, 11, 12, 14, 15, 16, 17, 19, 21, 22, 25, 26, 33, 34, 36, 38, 41	21	0.670699	0.013338
0, 2, 4, 6, 8, 9, 11, 12, 14, 15, 16, 17, 19, 20, 21, 22, 25, 26, 27, 28, 30, 32, 33, 34, 36, 38, 40, 41, 42	29	0.670647	0.013014
0, 2, 4, 5, 6, 8, 9, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 30, 32, 33, 34, 36, 38, 40, 41, 42	32	0.670594	0.014028
0, 2, 4, 6, 8, 9, 11, 12, 14, 15, 16, 17, 19, 20, 21, 22, 25, 26, 30, 32, 33, 34, 36, 38, 41, 42]	26	0.67049	0.013411
0, 2, 4, 5, 6, 8, 9, 11, 12, 14, 15, 16, 17, 19, 20, 21, 22, 24, 25, 26, 27, 28, 30, 32, 33, 34, 36, 38, 40, 41, 42	31	0.67049	0.013558
0, 2, 4, 6, 8, 9, 11, 12, 14, 15, 16, 17, 19, 20, 21, 22, 25, 26, 27, 30, 32, 33, 34, 36, 38, 40, 41, 42	28	0.670386	0.013245
2, 4, 8, 11, 12, 14, 15, 16, 17, 19, 21, 22, 25, 26, 33, 34, 36, 38, 41	19	0.670229	0.013373
0, 2, 4, 6, 8, 9, 11, 12, 14, 15, 16, 17, 19, 20, 21, 22, 25, 26, 27, 30, 32, 33, 34, 36, 38, 41, 42	27	0.670229	0.014082
2, 4, 8, 9, 11, 12, 14, 15, 16, 17, 19, 21, 22, 25, 26, 33, 34, 36, 38, 41	20	0.670177	0.013093
2, 8, 11, 12, 14, 15, 16, 17, 19, 21, 22, 25, 26, 33, 34, 36, 38, 41	18	0.670021	0.012979
0, 2, 4, 5, 6, 8, 9, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 30, 31, 32, 33, 34, 36, 38, 40, 41, 42	33	0.669917	0.014953
2, 19, 22, 25, 26, 34, 36	7	0.669812	0.012711
2, 17, 19, 22, 25, 26, 34, 36	8	0.669812	0.012686
2, 11, 17, 19, 22, 25, 26, 34, 36	9	0.669812	0.012718
2, 11, 12, 17, 19, 22, 25, 26, 34, 36	10	0.669812	0.012718
2, 19, 22, 25, 26, 34	6	0.66976	0.012673
2, 11, 12, 17, 19, 22, 25, 26, 33, 34, 36	11	0.66976	0.012759
0, 2, 4, 5, 6, 8, 9, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 30, 31, 32, 33, 34, 36, 38, 40, 41, 42, 43	34	0.66976	0.015619
2, 11, 12, 15, 17, 19, 22, 25, 26, 33, 34, 36	12	0.669708	0.01285
2, 11, 12, 14, 15, 17, 19, 22, 25, 26, 33, 34, 36	13	0.669708	0.01285
0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 40, 41, 42, 43	41	0.669656	0.016884
2, 19, 22, 25, 26	5	0.669604	0.012795
2, 11, 12, 14, 15, 17, 19, 22, 25, 26, 33, 34, 36, 38	14	0.669604	0.012955
2, 11, 12, 14, 15, 17, 19, 21, 22, 25, 26, 33, 34, 36, 38	15	0.669499	0.012536
2, 11, 12, 14, 15, 17, 19, 21, 22, 25, 26, 33, 34, 36, 38, 41	16	0.669395	0.012678
2, 19, 25, 26	4	0.669343	0.012256
2, 11, 12, 14, 15, 16, 17, 19, 21, 22, 25, 26, 33, 34, 36, 38, 41	17	0.669291	0.012763
0, 2, 4, 5, 6, 8, 9, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36, 38, 40, 41, 42, 43	35	0.669291	0.015657
0, 2, 4, 5, 6, 8, 9, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 38, 40, 41, 42, 43	36	0.669082	0.015676
0, 2, 4, 5, 6, 7, 8, 9, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 38, 40, 41, 42, 43	38	0.66903	0.01636
0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43	42	0.668978	0.017282
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43	44	0.668978	0.016501
0, 2, 4, 5, 6, 8, 9, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 38, 40, 41, 42, 43	37	0.668874	0.01521
0, 2, 4, 5, 6, 7, 8, 9, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 40, 41, 42, 43	39	0.668874	0.016636
0, 1, 2, 4, 5, 6, 7, 8, 9, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 40, 41, 42, 43	40	0.668457	0.017407
0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43	43	0.668405	0.017134
2, 25, 26	3	0.667935	0.013168
2, 25	2	0.663504	0.015725
2	1	0.635714	0.01314

Appendix 16. Sequential Forward Selection Code

```
def SequentialForwardSelection(predictors_dataset, Y):
    '''implements the greedy search algorithm sequential forward selection'''
    best = []
    initial_features = predictors_dataset.columns.tolist()
    best_features = []
    count = 0
    for count in range(len(initial_features)):
        print(count+1)
        remaining_features = list(set(initial_features)-set(best_features))
        mean = 0
        for new_column in remaining_features:
            feat = copy.deepcopy(best_features)
            feat.append(new_column)
            estimators_LR = []
            estimators_LR.append(('rescale', MinMaxScaler()))
            estimators_LR.append(('LR_sag', LogisticRegression(max_iter = 10000)))
            pipe = Pipeline(estimators_LR)
            kfold = KFold(n_splits=10)
            scoring = 'accuracy'
            X = predictors_dataset[feat].values
            X = np.nan_to_num(X)
            pipe.fit(X, Y)
            results = cross_val_score(pipe, X, Y, cv=kfold, scoring=scoring)
            if results.mean() > mean:
```

```
                mean = results.mean()
                stdev = results.std()
                new_best = new_column
        best_features.append(new_best)
        column_index = sorted([initial_features.index(i) for i in best_features])
        best.append((column_index, count+1, mean, stdev))
        best = sorted(best, key = lambda x: x[2], reverse = True)
    return best
```

Appendix 17. Sequential Backward Elimination Accuracies

COLUMN NUMBER (0 - 43)	NO. OF COLS.	ACCURACY	ST. DEV.
0, 1, 2, 3, 4, 7, 9, 10, 11, 13, 16, 17, 20, 21, 23, 25, 26, 28, 29, 30, 31, 32, 33, 35, 36, 39, 40, 41, 42, 43	30	0.6710636	0.0156664
0, 1, 2, 3, 4, 7, 9, 10, 11, 13, 16, 17, 20, 21, 23, 25, 26, 28, 29, 30, 31, 32, 35, 36, 39, 40, 41, 42, 43	29	0.6710636	0.0157425
0, 1, 2, 3, 4, 7, 9, 10, 13, 16, 17, 20, 21, 23, 25, 26, 28, 29, 30, 31, 32, 35, 39, 40, 41, 42, 43	27	0.6710636	0.0157166
0, 1, 2, 3, 4, 7, 9, 10, 11, 13, 16, 17, 20, 21, 23, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 36, 39, 40, 41, 42, 43	31	0.6709593	0.0156508
0, 1, 2, 3, 4, 7, 9, 10, 13, 16, 17, 20, 21, 23, 25, 26, 28, 29, 30, 32, 35, 39, 40, 41, 42, 43	26	0.6709593	0.0155305
0, 1, 2, 3, 4, 7, 9, 10, 13, 16, 17, 20, 21, 23, 25, 26, 28, 29, 30, 31, 32, 35, 36, 39, 40, 41, 42, 43	28	0.6709593	0.0156785
0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13, 16, 17, 20, 21, 23, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 36, 39, 40, 41, 42, 43	33	0.6708551	0.0154789
0, 1, 2, 3, 4, 7, 8, 9, 10, 11, 13, 16, 17, 20, 21, 23, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 36, 39, 40, 41, 42, 43	32	0.6708551	0.0155368
0, 1, 2, 3, 4, 7, 9, 10, 13, 16, 17, 23, 25, 26, 28, 29, 30, 32, 35, 39, 40, 41, 42, 43	24	0.6708551	0.0150081
0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13, 16, 17, 19, 20, 21, 23, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 36, 39, 40, 41, 42, 43	34	0.6708029	0.0156503
0, 1, 2, 3, 4, 7, 9, 10, 13, 16, 17, 23, 25, 26, 28, 29, 32, 35, 39, 40, 41, 42, 43	23	0.6707508	0.0151437
0, 1, 2, 3, 4, 7, 9, 10, 13, 16, 17, 20, 23, 25, 26, 28, 29, 30, 32, 35, 39, 40, 41, 42, 43	25	0.6707508	0.0152849
0, 2, 3, 4, 7, 9, 10, 13, 17, 23, 25, 26, 28, 29, 35, 39, 40, 42	18	0.6706986	0.0154739
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17, 18, 19, 20, 21, 23, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 36, 39, 40, 41, 42, 43	38	0.6705944	0.0153409
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 16, 17, 18, 19, 20, 21, 23, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 36, 39, 40, 41, 42, 43	37	0.6705944	0.015227
0, 2, 3, 4, 9, 13, 17, 25, 26, 29, 35, 39, 40	13	0.6705944	0.0146706
0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13, 14, 16, 17, 18, 19, 20, 21, 23, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 36, 39, 40, 41, 42, 43	36	0.6705422	0.0152656
0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13, 14, 16, 17, 19, 20, 21, 23, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 36, 39, 40, 41, 42, 43	35	0.6705422	0.0157615
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 23, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 36, 39, 40, 41, 42, 43	39	0.6705422	0.0154761
0, 1, 2, 3, 4, 7, 9, 10, 13, 16, 17, 23, 25, 26, 28, 29, 35, 39, 40, 41, 42, 43	22	0.6705422	0.0151441
0, 2, 3, 4, 9, 10, 13, 17, 25, 26, 28, 29, 35, 39, 40, 42	16	0.6705422	0.0144176
0, 1, 2, 3, 4, 7, 9, 10, 13, 16, 17, 23, 25, 26, 28, 29, 35, 39, 40, 41, 42	21	0.6704901	0.0148643
0, 1, 2, 3, 4, 7, 9, 10, 13, 17, 23, 25, 26, 28, 29, 35, 39, 40, 42	19	0.6703337	0.0155665
0, 2, 3, 4, 9, 10, 13, 17, 23, 25, 26, 28, 29, 35, 39, 40, 42	17	0.6703337	0.0146748
0, 2, 3, 4, 9, 10, 13, 17, 25, 26, 28, 29, 35, 39, 40	15	0.6703337	0.0154014
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 23, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 39, 40, 41, 42, 43	40	0.6702815	0.0153964
0, 1, 2, 3, 4, 7, 9, 10, 13, 17, 23, 25, 26, 28, 29, 35, 39, 40, 41, 42	20	0.6702294	0.0162252
0, 2, 3, 4, 9, 13, 17, 25, 26, 28, 29, 35, 39, 40	14	0.6702294	0.0147251
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 39, 40, 41, 42, 43	41	0.6701251	0.0153041
0, 2, 3, 4, 9, 13, 17, 25, 29, 35, 39, 40	12	0.6701251	0.014309
0, 2, 3, 4, 13, 17, 25, 29, 35, 39, 40	11	0.670073	0.0147666
2, 3, 4, 13, 17, 29, 35, 40	8	0.6699687	0.0156378
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 38, 39, 40, 41, 42, 43	42	0.6699687	0.0154823
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 38, 39, 40, 41, 42, 43	43	0.6698644	0.0158241
2, 3, 4, 17, 29, 35, 40	7	0.6698644	0.0143047
2, 3, 4, 17, 29, 40	6	0.6696559	0.0146547
2, 3, 4, 13, 17, 25, 29, 35, 39, 40	10	0.6696559	0.0146064
2, 3, 4, 13, 17, 25, 29, 35, 40	9	0.6691867	0.0154053
2, 3, 17, 29, 40	5	0.668926	0.0158158
2, 3, 29, 40	4	0.6682482	0.0160768
2, 29, 40	3	0.6674661	0.0144032
2, 29	2	0.6629301	0.0154018
2	1	0.6357143	0.0131404

Appendix 18. Sequential Backward Elimination Code

```
def SequentialBackwardSelection(predictors_dataset, Y):

    '''implements the greedy search algorithm sequential backward elimination'''

    best = []

    initial_features = predictors_dataset.columns.tolist()

    best_features = predictors_dataset.columns.tolist()

    count = 0

    for count in range(len(initial_features)-1,0,-1):

        print()

        print(count+1)

        remaining_features = copy.deepcopy(best_features)

        mean = 0

        for new_column in remaining_features:

            feat = copy.deepcopy(best_features)

            feat.remove(new_column)

            estimators_LR = []

            estimators_LR.append(('rescale', MinMaxScaler()))

            estimators_LR.append(('LR_sag', LogisticRegression(max_iter = 10000)))

            pipe = Pipeline(estimators_LR)

            kfold = KFold(n_splits=10)

            scoring = 'accuracy'

            X = predictors_dataset[feat].values

            X = np.nan_to_num(X)

            pipe.fit(X, Y)

            results = cross_val_score(pipe, X, Y, cv=kfold, scoring=scoring)
```

```
            if results.mean() > mean:
                mean = results.mean()
                stdev = results.std()
                new_best = feat
        best_features = new_best
        column_index = sorted([initial_features.index(i) for i in best_features])
        best.append((column_index, count, mean, stdev))
        best = sorted(best, key = lambda x: x[2], reverse = True)
        print(best[0][2])
    return best
```

Appendix 19. Brute Force and Random Search Accuracies

COLUMNS	FEATURE COUNT	MEAN	SD
2, 4, 5, 6, 8, 9, 10, 12, 13, 14, 17, 22, 23, 24, 25, 31, 34, 36, 38, 39	20	0.6720542	0.0140211
0, 1, 2, 4, 7, 9, 11, 12, 13, 16, 17, 19, 21, 22, 23, 25, 26, 30, 34, 35, 39	21	0.671585	0.0146996
2, 5, 6, 7, 8, 9, 10, 12, 13, 14, 17, 18, 19, 20, 21, 23, 24, 25, 28, 29, 31, 32, 33, 34, 36, 39, 40, 41, 42, 43	30	0.6715328	0.015419
2, 4, 5, 6, 8, 9, 10, 11, 13, 15, 17, 18, 20, 22, 24, 25, 27, 29, 32, 34, 36, 39, 40, 41	24	0.6714286	0.0136354
2, 9, 10, 11, 13, 14, 24, 25, 28, 31	10	0.6713243	0.014934
0, 2, 4, 5, 8, 9, 11, 12, 13, 17, 19, 22, 24, 25, 28, 31, 32, 33, 35, 40, 41	21	0.6712722	0.0131987
2, 9, 10, 11, 13, 23, 24	7	0.67122	0.0135502
0, 2, 4, 10, 11, 12, 13, 17, 20, 22, 25, 27, 32, 39, 41, 43	16	0.67122	0.0148188
0, 2, 4, 5, 6, 7, 8, 9, 11, 13, 14, 20, 22, 23, 25, 27, 32, 40, 41	19	0.67122	0.014235
0, 2, 4, 5, 6, 7, 8, 10, 12, 13, 17, 19, 23, 24, 25, 28, 29, 37, 42	19	0.67122	0.0144623
1, 2, 5, 8, 9, 10, 11, 13, 15, 16, 18, 20, 21, 22, 24, 29, 30, 32, 36, 39, 41	21	0.6711679	0.0150407
0, 2, 4, 6, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 20, 22, 23, 24, 25, 27, 28, 30, 31, 33, 34, 35, 39, 41	28	0.6711679	0.0139024
2, 10, 12, 13, 18, 21, 23, 24, 26, 29, 33, 39, 40, 41	14	0.6711157	0.0138686
0, 1, 2, 4, 5, 6, 7, 9, 11, 12, 13, 14, 15, 18, 19, 20, 21, 22, 23, 25, 27, 28, 29, 30, 35, 39	26	0.6711157	0.0154837
2, 4, 5, 7, 8, 9, 10, 12, 13, 17, 22, 23, 27, 29, 30, 32, 34, 39	18	0.6711157	0.0157828
2, 4, 6, 7, 9, 11, 14, 15, 17, 19, 23, 24, 25, 31, 38, 41	16	0.6711157	0.0150095
0, 1, 2, 6, 9, 10, 13, 14, 16, 18, 20, 22, 23, 24, 25, 27, 28, 30, 31, 32, 34, 36, 39, 40, 41, 43	26	0.6711157	0.0149441
0, 2, 4, 6, 9, 10, 13, 16, 17, 19, 20, 22, 25, 28, 29, 41, 43	17	0.6711157	0.0137762
0, 2, 4, 5, 6, 7, 9, 10, 11, 13, 15, 17, 19, 22, 23, 24, 25, 26, 27, 28, 30, 31, 32, 34, 41, 43	26	0.6710636	0.0154989
2, 6, 10, 11, 13, 20, 24, 25, 33, 34, 39	11	0.6710115	0.0147117

Appendix 20. Brute Force and Random Search Code

```
def brute_force_lr(predictors_dataset, Y):
    '''implements the exhaustive search for the best feature combination.'''
    best=[('a',0,0,0)]*20
    estimators_LR = []
    estimators_LR.append(('rescale', MinMaxScaler()))
    estimators_LR.append(('LR_sag', LogisticRegression(max_iter = 10000)))
    pipe = Pipeline(estimators_LR)
    kfold = KFold(n_splits=10)
    scoring = 'accuracy'
    start_column = 'a'
    for index in range(6, len(predictors_dataset.columns)):
        print(index)
        if index<6 or index>39:
            for combination in itertools.combinations(predictors_dataset.columns, index):
                columns = [i for i in combination]
                if columns[0] != start_column:
                    start_column = columns[0]
                    print(start_column)
                data = predictors_dataset[columns]
                X = data.values
                X = np.nan_to_num(X)
                results = cross_val_score(pipe, X, Y, cv=kfold, scoring=scoring)
                if results.mean() > best[-1][2]:
                    best[-1] = (combination, index, results.mean(), results.std())
                    best = sorted(best, key = lambda x:x[2], reverse = True)
```

```
                with open('brutefrom5.txt', 'w') as top20:
                    for value in best:
                        top20.write(str(value)+'\n')
    else:
        for number in range(20000):
            if number%100 == 0:
                print(number)
            combination = r.sample(predictors_dataset.columns.tolist(), index)
            data = predictors_dataset[combination]
            X = data.values
            X = np.nan_to_num(X)
            results = cross_val_score(pipe, X, Y, cv=kfold, scoring=scoring)
            if results.mean() > best[-1][2]:
                best[-1] = (combination, index, results.mean(), results.std())
                best = sorted(best, key = lambda x:x[2], reverse = True)
                with open('brutefrom5.txt', 'w') as top20:
                    for value in best:
                        top20.write(str(value)+'\n')
```

Appendix 21. Principal Component Analysis Accuracies

NO. OF PRINCIPAL COMPONENTS	ACCURACY	ST. DEV.
27	0.669239	0.016262
19	0.66903	0.016401
38	0.66903	0.016627
36	0.668978	0.016546
39	0.668978	0.016701
40	0.668978	0.016701
41	0.668978	0.016701
42	0.668978	0.016701
43	0.668978	0.016701
44	0.668978	0.016701
26	0.668926	0.015838
28	0.668926	0.016555
37	0.668926	0.01671
34	0.668717	0.015871
35	0.668717	0.016188
15	0.668613	0.013564
25	0.668561	0.016509
29	0.668509	0.01691
20	0.668457	0.016768
18	0.668248	0.016661
30	0.668248	0.017228
21	0.668196	0.017079
24	0.668196	0.017065
5	0.668144	0.013915
32	0.668144	0.016501
23	0.668144	0.017291
22	0.668144	0.016956
31	0.668092	0.016809
17	0.668092	0.016116
33	0.66804	0.016534
16	0.667987	0.015834
6	0.667675	0.013753
7	0.66731	0.013708
4	0.667101	0.013957
3	0.666684	0.014366
13	0.666423	0.015435
14	0.666163	0.015849
8	0.665902	0.014204
9	0.665746	0.014235
2	0.665589	0.012735
10	0.665537	0.013816
11	0.665224	0.014429
12	0.664807	0.014555
1	0.591554	0.010062

Appendix 22. Principal Component Analysis Code

```
def PCA_(predictors_dataset, Y):
    '''uses the sklearn PCA to iteratively select the best
    feature combination for each number of principal components '''
    best=[]
    for index in range(1, len(predictors_dataset.columns)+1):
        print(index)
        estimators_LR = []
        estimators_LR.append(('rescale', MinMaxScaler()))
        estimators_LR.append(('PCA',  PCA(n_components=index)))
        estimators_LR.append(('LR_sag', LogisticRegression(max_iter = 10000)))
        pipe = Pipeline(estimators_LR)
        kfold = KFold(n_splits=10)
        scoring = 'accuracy'
        X = predictors_dataset.values
        X = np.nan_to_num(X)
        pipe.fit(X,Y)
        results = cross_val_score(pipe, X, Y, cv=kfold, scoring=scoring)
        best.append((index,results.mean(), results.std()))
        #print(index,results.mean(), results.std())
    print(pipe['PCA'].explained_variance_ratio_)
    cumsum = np.cumsum(pipe['PCA'].explained_variance_ratio_)
    print(cumsum)
    plt.xticks(ticks = range(0,46,4), labels = range(1,47,4))
    plt.xlabel('number of components', size = 10)
```

```
plt.ylabel('cummulative explained variance', size = 10)

plt.title('Cummulative Sum of Variance Explained by Principal Components')

plt.plot(np.cumsum(pipe['PCA'].explained_variance_ratio_))

for i, v in enumerate(cumsum):

    if i in [0, 1, 2, 4]:

        plt.text(i+1.75, v, "%.2f" % v, ha="center", size = 8)

    elif i in [8, 12, 16, 30]:

        plt.text(i, v-0.025, "%.2f" % v, ha="center", size = 8)

plt.show()

best = sorted(best, key = lambda x:x[1], reverse = True)

return best
```

Appendix 23. EWMA Code

```
import pandas as pd

def ewma(dataset, days):

    '''returns a dictionary of keys as teams and values as the teams laod over a season
s stretch'''

    dataset['Overtime'] = dataset['Overtime'].fillna('0')

    dataset['Overtime'] = dataset['Overtime'].replace({'OT':'1OT'})

    load_dict = dict() #stores dictionary of team as items and dataframe with loading p
er day as values

    lambda_ = 2 / (days + 1)

    maxdate = max(dataset['Date'])

    mindate = min(dataset['Date'])

    calendar = pd.date_range(start=mindate, end=maxdate)

    teams = sorted(set(dataset['Home Team']).union(set(dataset['Away Team'])))

    for team in teams:

        all_team_data = []

        for date_index in range(len(calendar)):

            row = []

            row.append(calendar[date_index])

            #grab from full dataset only fixtures from that date to check if team playe
d

            dates_data = dataset[dataset.Date == calendar[date_index]]

            dates_data.reset_index(inplace = True)

            #iterate through this mini dataset

            if len(dates_data) == 0:

                loading = 0

            else:
```

```
                for index in range(len(dates_data)):

                    if dates_data.loc[index, 'Home Team'] == team or dates_data.loc[ind
ex, 'Away Team'] == team:

                        #load value is the number of minutes played

                        if 'OT' in dates_data.loc[index, 'Overtime']:

                            #loading is 48 + number of overtime periods times 5

                            loading = 48 + int(dates_data.loc[index, 'Overtime'][0]) *
5

                        else:

                            loading = 48

                        break

                    else:

                        loading = 0

            row.append(loading)

            all_team_data.append(row)

        load_dataset = pd.DataFrame.from_records(data = all_team_data,columns = ['Date'
,'Load'])

        # calculate the ewma

        ewma = []

        ewma_yesterday = 0

        for index in range(len(load_dataset)):

            ewma_today = load_dataset.loc[index, 'Load'] * lambda_ + ((1 - lambda_) * e
wma_yesterday)

            ewma.append(ewma_today)

            ewma_yesterday = ewma_today

        load_dataset['ewma'+ str(days)] = ewma

        load_dict[team] = load_dataset

    return load_dict
```

```
def return_ewma_column(dataset, days):

    '''returns a dataset with additional columns that show the ewma load calculated

    based on a number of days supplied to the function'''

    load_dict = ewma(dataset, days) #calls the other function to get a dictionary

    home_load   = []

    away_load   = []

    for index in range(len(dataset)):

        #ifetch datasets from the dictionary

        home_team_load_dataset = load_dict[dataset.loc[index, 'Home Team']]

        away_team_load_dataset = load_dict[dataset.loc[index, 'Away Team']]

        date      = dataset.loc[index, 'Date']



        try:

            h_load = home_team_load_dataset.loc[home_team_load_dataset['Date'] == date
- pd.DateOffset(1), 'ewma'+str(days)].iloc[0]

            a_load = away_team_load_dataset.loc[away_team_load_dataset['Date'] == date
- pd.DateOffset(1), 'ewma'+str(days)].iloc[0]

            home_load.append(h_load)

            away_load.append(a_load)

        except:

            home_load.append(0)

            away_load.append(0)

    dataset['home_ewma' + str(days)] = home_load

    dataset['away_ewma' + str(days)] = away_load

    return dataset
```

Appendix 24. Best Validation Results Table Using 3 Previous Seasons

Model Type	2004-2005	2005-2006	2006-2007	2007-2008	2008-2009	2009-2010	2010-2011	2011-2012	2012-2013	2013-2014	2014-2015	2015-2016	2016-2017	2017-2018	2018-2019	2019-2020	Average
bf / acwr / liblinear				66.10	70.08	69.19	68.70	67.68	68.43	66.10	66.83	69.59	65.37	65.69	65.04	65.09	67.22
bs only / liblinear				68.05	70.49	68.46	68.29	66.67	67.70	66.34	67.89	68.54	65.53	64.63	65.37	65.29	67.17
bf only / saga				66.99	69.84	69.02	68.46	67.37	68.02	66.18	67.15	69.92	64.96	65.45	65.28	64.47	67.16
bf / acwr / lbfgs				66.10	70.00	69.02	68.62	67.58	68.35	66.26	66.99	69.35	64.96	65.69	65.04	65.09	67.16
bf / acwr / newton_cg				66.10	70.00	69.02	68.62	67.58	68.35	66.26	66.99	69.35	64.96	65.69	65.04	65.09	67.16
bf / acwr / saga				66.10	70.00	69.02	68.62	67.58	68.35	66.26	66.99	69.35	64.96	65.69	65.04	65.09	67.16
bf only / newton_cg				66.99	69.84	69.02	68.46	67.37	68.02	66.18	67.15	69.92	64.88	65.45	65.28	64.47	67.16
bf only / lbfgs				66.91	69.84	69.02	68.46	67.37	67.94	66.18	67.15	69.92	64.96	65.53	65.28	64.47	67.16
bs only / lbfgs				68.21	70.57	68.46	68.37	66.57	67.53	66.34	67.72	68.54	65.53	64.63	65.37	65.19	67.16
bs only / newton_cg				68.21	70.57	68.46	68.37	66.57	67.53	66.34	67.72	68.54	65.53	64.63	65.37	65.19	67.16
bs only / saga				68.21	70.57	68.46	68.37	66.57	67.53	66.34	67.72	68.54	65.53	64.63	65.37	65.19	67.16
bf only / liblinear				66.99	69.84	69.11	68.46	67.27	67.86	65.93	67.32	69.92	64.80	65.61	65.28	64.57	67.15
bs / acwr / liblinear				67.72	70.33	68.54	68.46	66.67	67.45	66.75	67.32	68.05	65.20	65.12	65.20	65.60	67.11
bs / acute / newton_cg				67.64	70.08	67.97	68.21	67.27	68.02	66.75	68.05	68.86	64.96	64.63	64.63	65.29	67.11
bs / acute / saga				67.64	70.08	67.97	68.21	67.27	68.02	66.75	68.05	68.86	64.96	64.63	64.63	65.29	67.11
bs / chronic / liblinear				67.40	70.24	67.48	68.46	66.97	68.10	66.50	67.64	68.46	65.45	64.31	65.53	65.81	67.10
bs / acwr / lbfgs				67.72	70.33	68.54	68.46	66.57	67.78	66.42	67.24	68.05	65.20	65.20	65.20	65.60	67.10
bs / acwr / newton_cg				67.72	70.33	68.54	68.46	66.57	67.78	66.42	67.24	68.05	65.20	65.20	65.20	65.60	67.10
bs / acwr / saga				67.72	70.33	68.54	68.46	66.57	67.78	66.42	67.24	68.05	65.20	65.20	65.20	65.60	67.10
bs / acute / lbfgs				67.64	70.08	67.97	68.21	67.27	68.02	66.67	68.05	68.86	64.96	64.63	64.63	65.29	67.10
bs / acute / liblinear				67.48	70.08	67.97	68.13	67.17	68.10	66.67	67.80	68.86	65.37	64.72	64.63	65.29	67.10
bs / chronic / lbfgs				67.64	70.24	67.64	68.37	66.87	68.02	66.26	67.64	68.46	65.45	64.31	65.53	65.81	67.10
bs / chronic / newton_cg				67.56	70.24	67.64	68.37	66.87	68.02	66.26	67.64	68.46	65.45	64.31	65.53	65.71	67.08
bs / chronic / saga				67.56	70.24	67.64	68.37	66.87	68.02	66.26	67.64	68.46	65.45	64.31	65.53	65.71	67.08
bf / acute / lbfgs				66.75	69.76	68.29	67.89	67.88	68.51	66.26	66.91	69.43	64.96	65.04	64.96	64.68	67.02
fs / acwr / liblinear				67.15	69.59	68.46	67.56	67.68	68.92	66.42	66.75	68.54	64.39	65.53	65.28	64.88	67.01
bf / acute / liblinear				66.67	69.76	68.29	67.89	67.88	68.51	66.26	66.99	69.35	64.88	65.04	64.96	64.68	67.01
bf / acute / newton_cg				66.75	69.76	68.29	67.89	67.88	68.43	66.26	66.91	69.43	64.96	65.04	64.96	64.57	67.01
bf / acute / saga				66.75	69.76	68.29	67.89	67.88	68.43	66.26	66.91	69.43	64.96	65.04	64.96	64.57	67.01
fs / acwr / lbfgs				67.32	69.67	68.21	67.80	67.58	68.76	66.34	66.75	68.29	64.55	65.61	65.04	64.78	66.98
fs / chronic / liblinear				67.64	69.84	68.13	67.89	67.68	68.59	66.18	67.15	68.29	64.63	64.72	65.37	64.47	66.97
fs / acwr / newton_cg				67.32	69.67	68.21	67.80	67.47	68.76	66.34	66.75	68.29	64.55	65.61	65.04	64.68	66.96
fs / acwr / saga				67.32	69.67	68.21	67.80	67.47	68.76	66.34	66.75	68.29	64.55	65.61	65.04	64.68	66.96
fs / chronic / newton_cg				67.64	69.84	67.80	67.97	67.68	68.59	66.18	67.15	68.13	64.63	64.88	65.28	64.47	66.94
fs / chronic / saga				67.64	69.84	67.80	67.97	67.68	68.59	66.18	67.15	68.13	64.63	64.88	65.28	64.47	66.94
fs / chronic / lbfgs				67.64	69.84	67.72	67.97	67.68	68.59	66.18	67.15	68.13	64.63	64.88	65.28	64.47	66.94
fs only / lbfgs				67.72	69.59	68.29	67.80	67.17	68.19	66.26	67.48	67.72	64.80	64.80	65.45	64.78	66.93
fs only / newton_cg				67.72	69.59	68.29	67.80	67.17	68.19	66.26	67.32	67.72	64.80	64.80	65.45	64.78	66.91
fs only / saga				67.72	69.59	68.29	67.80	67.17	68.19	66.26	67.32	67.72	64.80	64.80	65.45	64.78	66.91
fs only / liblinear				67.80	69.43	68.29	67.80	67.27	68.35	66.26	67.32	68.05	64.55	64.63	65.45	64.68	66.91
fs / acute / liblinear				66.91	69.76	67.97	67.32	67.78	69.00	66.34	66.75	68.46	64.31	65.20	65.28	64.47	66.89
bf / chronic / newton_cg				67.07	69.59	67.64	67.72	67.47	68.35	66.18	66.83	68.94	64.63	64.96	65.28	64.78	66.88
bf / chronic / saga				67.07	69.59	67.64	67.72	67.47	68.35	66.18	66.83	68.94	64.63	64.96	65.28	64.78	66.88
bf / chronic / lbfgs				67.07	69.59	67.64	67.72	67.47	68.35	66.10	66.83	68.94	64.63	64.96	65.28	64.78	66.88
bf / chronic / liblinear				66.99	69.59	67.72	67.80	67.47	68.35	66.02	66.83	68.86	64.63	65.04	65.28	64.78	66.88
fs / acute / lbfgs				66.99	69.76	67.56	67.56	67.68	68.92	66.34	66.75	68.46	64.23	65.04	65.04	64.57	66.84
fs / acute / newton_cg				66.99	69.67	67.56	67.56	67.68	68.92	66.34	66.75	68.46	64.23	65.04	64.96	64.57	66.83
fs / acute / saga				66.99	69.67	67.56	67.56	67.68	68.92	66.34	66.75	68.46	64.23	65.04	64.96	64.57	66.83
Average				67.29	69.93	68.23	68.09	67.31	68.26	66.31	67.21	68.70	64.91	65.02	65.18	64.95	67.03

- Feature Selection Methods
  - bf is brute force feature selection method
  - bs is backward elimination feature selection method
  - fs is forward selection feature selection method
- Fatigue features
  - acwr is with acute chronic workload ratio fatigue features
  - only is without load features
  - acute is with acute load fatigue features
  - chronic is with chronic load fatigue features
- Solver Used
  - lbfgs
  - liblinear
  - Newton-cg
  - Saga

Appendix 25. Best Validation Results Using All Previous Seasons

Model Type	2004-2005	2005-2006	2006-2007	2007-2008	2008-2009	2009-2010	2010-2011	2011-2012	2012-2013	2013-2014	2014-2015	2015-2016	2016-2017	2017-2018	2018-2019	2019-2020	Average
bs / acute / newton_cg		66.02	63.66	67.64	70.57	67.97	68.29	67.58	67.70	66.34	68.13	68.86	64.31	65.20	66.10	66.12	66.97
bs / acute / saga		66.02	63.66	67.64	70.57	67.97	68.29	67.58	67.70	66.34	68.13	68.86	64.31	65.20	66.10	66.12	66.97
bs / acute / lbfgs		66.02	63.66	67.64	70.57	67.97	68.21	67.58	67.78	66.34	68.13	68.86	64.31	65.20	66.10	66.01	66.96
bs / acute / liblinear		65.85	63.58	67.48	70.57	67.97	68.29	67.58	67.70	66.34	68.13	68.86	64.31	65.28	66.10	66.12	66.94
bs only / lbfgs		65.37	64.15	68.21	70.33	67.97	68.46	66.77	67.70	67.07	67.56	69.19	63.25	65.85	66.10	65.60	66.90
bs / acwr / saga		65.77	64.07	67.72	70.49	67.64	68.13	67.37	67.70	66.75	67.64	68.86	64.31	65.69	65.77	65.60	66.90
bs only / newton_cg		65.37	64.15	68.21	70.33	67.97	68.46	66.77	67.70	67.07	67.56	69.19	63.25	65.77	66.10	65.60	66.90
bs only / saga		65.37	64.15	68.21	70.33	67.97	68.46	66.77	67.70	67.07	67.56	69.19	63.25	65.77	66.10	65.60	66.90
bs / acwr / newton_cg		65.77	64.07	67.72	70.49	67.64	68.13	67.37	67.70	66.75	67.64	68.86	64.31	65.69	65.77	65.50	66.89
bs / acwr / lbfgs		65.77	64.07	67.72	70.49	67.56	68.13	67.37	67.70	66.75	67.64	68.86	64.31	65.69	65.69	65.60	66.89
bs only / liblinear		65.37	64.15	68.05	70.24	67.97	68.46	66.67	67.78	67.07	67.56	69.19	63.25	65.77	66.10	65.60	66.88
bs / acwr / liblinear		65.61	64.15	67.72	70.49	67.56	68.13	67.37	67.70	66.50	67.64	68.94	64.23	65.69	65.77	65.60	66.87
fs / chronic / liblinear		65.53	64.39	67.64	70.41	67.64	67.24	66.87	68.59	66.42	68.21	68.94	65.04	65.28	65.69	64.78	66.85
fs / chronic / lbfgs		65.61	64.31	67.64	70.41	67.64	67.32	66.77	68.59	66.42	68.21	68.94	65.04	65.28	65.69	64.78	66.84
fs / chronic / newton_cg		65.61	64.31	67.64	70.41	67.64	67.32	66.77	68.59	66.42	68.21	68.94	65.04	65.20	65.61	64.78	66.83
fs / chronic / saga		65.61	64.31	67.64	70.41	67.64	67.32	66.77	68.59	66.42	68.21	68.94	65.04	65.20	65.61	64.78	66.83
fs / acwr / lbfgs		66.02	63.66	67.32	70.16	68.29	67.32	66.97	68.35	66.02	68.46	68.37	64.80	65.69	66.02	64.78	66.81
fs / acwr / newton_cg		66.02	63.66	67.32	70.16	68.29	67.32	66.97	68.35	66.02	68.46	68.37	64.80	65.69	66.02	64.78	66.81
fs / acwr / saga		66.02	63.66	67.32	70.16	68.29	67.32	66.97	68.35	66.02	68.46	68.37	64.80	65.69	66.02	64.78	66.81
fs only / lbfgs		65.37	64.31	67.72	70.65	68.05	67.40	66.46	68.51	66.34	68.13	67.97	64.63	65.53	66.26	64.78	66.81
fs / acwr / liblinear		66.02	63.66	67.15	70.24	68.37	67.32	66.97	68.35	66.10	68.46	68.21	64.80	65.69	66.02	64.68	66.80
fs only / liblinear		65.37	64.31	67.80	70.49	67.97	67.40	66.46	68.51	66.34	68.05	68.13	64.63	65.45	66.26	64.78	66.80
fs only / newton_cg		65.37	64.31	67.72	70.65	67.97	67.40	66.46	68.51	66.34	68.13	67.97	64.63	65.45	66.26	64.78	66.80
fs only / saga		65.37	64.31	67.72	70.65	67.97	67.40	66.46	68.51	66.34	68.13	67.97	64.63	65.45	66.26	64.78	66.80
bf only / newton_cg		65.53	64.88	66.99	69.35	68.05	68.37	66.97	67.45	66.10	67.64	69.19	63.98	65.53	66.18	65.40	66.77
bf only / saga		65.53	64.88	66.99	69.35	68.05	68.37	66.97	67.45	66.10	67.64	69.19	63.98	65.53	66.18	65.40	66.77
bf only / liblinear		65.61	64.63	66.99	69.35	68.05	68.37	66.97	67.53	66.10	67.56	69.19	63.98	65.53	66.10	65.40	66.76
bf only / lbfgs		65.53	64.88	66.91	69.35	68.05	68.37	66.97	67.45	66.10	67.56	69.19	63.98	65.53	66.10	65.40	66.76
bs / chronic / lbfgs		65.28	63.41	67.64	70.57	67.80	67.97	67.07	67.78	66.75	67.89	68.86	64.31	65.45	65.45	65.09	66.75
bs / chronic / newton_cg		65.28	63.41	67.56	70.57	67.80	67.89	67.07	67.78	66.75	67.89	68.86	64.39	65.37	65.53	65.09	66.75
bs / chronic / saga		65.28	63.41	67.56	70.57	67.80	67.89	67.07	67.78	66.75	67.89	68.86	64.39	65.37	65.53	65.09	66.75
bs / chronic / liblinear		65.28	63.50	67.40	70.57	67.80	67.89	67.17	67.78	66.75	67.80	68.78	64.47	65.37	65.53	65.09	66.75
bf / chronic / lbfgs		65.61	64.31	67.07	69.92	67.89	67.64	67.07	68.27	66.34	67.32	69.19	63.82	65.28	65.69	65.60	66.73
bf / chronic / newton_cg		65.61	64.31	67.07	69.92	67.89	67.64	67.07	68.19	66.34	67.32	69.19	63.82	65.28	65.69	65.60	66.73
bf / chronic / saga		65.61	64.31	67.07	69.92	67.89	67.64	67.07	68.19	66.34	67.32	69.19	63.82	65.28	65.69	65.60	66.73
bf / acute / lbfgs		65.45	63.66	66.75	69.51	68.78	67.24	67.07	68.35	66.34	67.40	69.27	63.98	65.93	65.37	65.60	66.71
bf / acute / newton_cg		65.45	63.66	66.75	69.51	68.78	67.24	67.07	68.35	66.26	67.40	69.27	63.98	66.02	65.37	65.60	66.71
bf / acute / saga		65.45	63.66	66.75	69.51	68.78	67.24	67.07	68.35	66.26	67.40	69.27	63.98	66.02	65.37	65.60	66.71
bf / chronic / liblinear		65.45	64.23	66.99	69.92	67.80	67.64	67.07	68.27	66.42	67.32	69.19	63.82	65.28	65.69	65.60	66.71
fs / acute / lbfgs		65.61	63.66	66.99	70.08	67.97	67.48	67.17	68.84	65.61	68.37	68.62	64.23	65.45	65.93	64.68	66.71
bf / acute / liblinear		65.37	63.66	66.67	69.51	68.78	67.24	67.07	68.35	66.34	67.40	69.35	63.98	65.93	65.37	65.60	66.71
fs / acute / newton_cg		65.61	63.66	66.99	70.08	67.97	67.48	67.17	68.84	65.61	68.37	68.62	64.07	65.45	65.93	64.68	66.70
fs / acute / saga		65.61	63.66	66.99	70.08	67.97	67.48	67.17	68.84	65.61	68.37	68.62	64.07	65.45	65.93	64.68	66.70
fs / acute / liblinear		65.69	63.41	66.91	70.08	67.97	67.48	67.17	68.84	65.61	68.37	68.62	64.15	65.45	65.85	64.57	66.68
bf / acwr / liblinear		65.37	63.74	66.10	69.27	67.80	67.48	67.37	68.10	66.34	67.40	69.27	63.98	65.85	65.77	65.60	66.63
bf / acwr / newton_cg		65.28	63.66	66.10	69.27	67.80	67.56	67.37	68.10	66.34	67.40	69.27	63.98	65.93	65.85	65.50	66.63
bf / acwr / saga		65.28	63.66	66.10	69.27	67.80	67.56	67.37	68.10	66.34	67.40	69.27	63.98	65.93	65.85	65.50	66.63
bf / acwr / lbfgs		65.28	63.66	66.10	69.27	67.80	67.48	67.37	68.10	66.34	67.40	69.27	63.98	65.93	65.77	65.60	66.62
Average		65.57	63.97	67.29	70.11	67.98	67.75	67.06	68.11	66.36	67.84	68.88	64.22	65.55	65.86	65.29	66.79

- Feature Selection Methods
  - bf is brute force feature selection method
  - bs is backward elimination feature selection method
  - fs is forward selection feature selection method
- Fatigue features
  - acwr is with acute chronic workload ratio fatigue features
  - only is without load features
  - acute is with acute load fatigue features
  - chronic is with chronic load fatigue features
- Solver Used
  - lbfgs
  - liblinear
  - Newton-cg
  - Saga

Appendix 26. Best Validation Results Using All Previous Seasons Except 2004-2005

Model Type	2004-2005	2005-2006	2006-2007	2007-2008	2008-2009	2009-2010	2010-2011	2011-2012	2012-2013	2013-2014	2014-2015	2015-2016	2016-2017	2017-2018	2018-2019	2019-2020	Average
bs only / liblinear			63.82	67.80	70.49	68.46	68.29	66.67	68.02	67.24	67.72	69.11	64.80	65.69	66.10	67.35	67.25
bs only / newton_cg			63.74	67.72	70.57	68.37	68.29	66.87	67.94	67.15	67.89	69.02	64.80	65.69	66.10	67.35	67.25
bs only / saga			63.74	67.72	70.57	68.37	68.29	66.87	67.94	67.15	67.89	69.02	64.80	65.69	66.10	67.35	67.25
bs only / lbfgs			63.74	67.72	70.57	68.37	68.29	66.87	67.86	67.15	67.89	69.02	64.80	65.69	66.10	67.35	67.25
bf only / lbfgs			64.47	66.99	69.84	68.94	68.78	67.58	68.92	66.26	67.15	69.59	63.50	65.61	65.77	66.43	67.13
bf only / newton_cg			64.47	66.99	69.84	68.94	68.78	67.58	68.84	66.26	67.15	69.59	63.50	65.61	65.77	66.43	67.12
bf only / saga			64.47	66.99	69.84	68.94	68.78	67.58	68.84	66.26	67.15	69.59	63.50	65.61	65.77	66.43	67.12
bf only / liblinear			64.23	66.99	69.84	68.86	68.78	67.58	68.84	66.10	67.15	69.59	63.50	65.61	65.77	66.43	67.09
bs / acwr / liblinear			63.17	67.72	70.33	68.13	68.21	66.87	67.62	66.75	68.05	68.94	64.80	65.61	66.02	66.74	67.07
bs / acwr / newton_cg			63.09	67.64	70.33	68.13	68.21	66.87	67.62	66.59	67.97	68.94	64.80	65.69	65.93	66.63	67.03
bs / acwr / saga			63.09	67.64	70.33	68.13	68.21	66.87	67.62	66.59	67.97	68.94	64.80	65.69	65.93	66.63	67.03
bs / acwr / lbfgs			63.09	67.64	70.33	68.13	68.21	66.87	67.62	66.50	67.97	68.94	64.80	65.69	65.93	66.63	67.03
bs / chronic / newton_cg			63.09	67.89	70.24	68.29	67.80	67.27	67.45	67.40	67.48	68.78	64.80	65.20	65.77	66.84	67.02
bs / chronic / saga			63.09	67.89	70.24	68.29	67.80	67.27	67.45	67.40	67.48	68.78	64.80	65.20	65.77	66.84	67.02
bs / chronic / lbfgs			63.17	67.89	70.24	68.29	67.80	67.17	67.37	67.40	67.48	68.78	64.80	65.20	65.77	66.84	67.02
bs / chronic / liblinear			63.25	67.89	70.24	68.29	67.80	67.17	67.53	67.32	67.32	68.62	64.63	65.28	65.77	67.04	67.01
bf / acute / lbfgs			63.90	67.24	69.76	68.78	67.72	68.38	68.59	66.26	67.07	69.43	64.07	65.28	64.96	66.32	66.98
bf / acute / liblinear			63.98	67.24	69.76	68.86	67.72	68.18	68.59	66.26	67.07	69.43	63.98	65.37	64.96	66.32	66.98
bf / acute / newton_cg			63.90	67.24	69.76	68.78	67.72	68.38	68.51	66.26	67.07	69.43	63.98	65.37	64.96	66.32	66.98
bf / acute / saga			63.90	67.24	69.76	68.78	67.72	68.38	68.51	66.26	67.07	69.43	63.98	65.37	64.96	66.32	66.98
fs only / lbfgs			64.31	67.24	69.59	68.29	68.13	67.58	68.51	66.42	67.72	68.21	64.88	65.37	65.93	65.40	66.97
fs only / newton_cg			64.31	67.24	69.59	68.29	68.13	67.47	68.51	66.42	67.72	68.21	64.88	65.37	65.93	65.40	66.96
fs only / saga			64.31	67.24	69.59	68.29	68.13	67.47	68.51	66.42	67.72	68.21	64.88	65.37	65.93	65.40	66.96
fs only / liblinear			64.47	67.24	69.43	68.37	68.05	67.58	68.43	66.26	67.80	68.05	64.88	65.45	65.85	65.29	66.94
bf / acwr / liblinear			63.74	66.59	70.08	68.46	68.05	67.58	68.67	66.26	67.15	69.35	64.23	65.45	65.77	65.71	66.93
bf / acwr / lbfgs			63.82	66.50	70.00	68.46	67.97	67.47	68.59	66.42	67.15	69.35	64.23	65.37	65.93	65.71	66.93
bs / acute / liblinear			62.76	67.64	70.08	68.70	67.89	66.97	67.70	66.75	67.72	68.46	64.88	65.45	65.61	66.32	66.92
bf / acwr / newton_cg			63.82	66.50	70.00	68.46	67.97	67.47	68.59	66.42	67.15	69.35	64.23	65.45	65.77	65.71	66.92
bf / acwr / saga			63.82	66.50	70.00	68.46	67.97	67.47	68.59	66.42	67.15	69.35	64.23	65.45	65.77	65.71	66.92
fs / acute / liblinear			63.50	68.05	69.76	68.13	67.56	67.98	69.08	66.18	68.29	67.97	64.23	65.28	65.69	65.19	66.92
bs / acute / lbfgs			62.68	67.72	70.08	68.70	67.97	66.67	67.62	66.67	67.89	68.54	64.88	65.61	65.45	66.32	66.91
bs / acute / newton_cg			62.68	67.64	70.08	68.70	67.97	66.67	67.62	66.67	67.89	68.62	64.88	65.53	65.45	66.32	66.91
bs / acute / saga			62.68	67.64	70.08	68.70	67.97	66.67	67.62	66.67	67.89	68.62	64.88	65.53	65.45	66.32	66.91
bf / chronic / liblinear			64.07	67.48	69.59	68.37	68.05	67.47	68.59	66.10	66.83	68.94	64.07	65.37	65.45	66.22	66.90
bf / chronic / lbfgs			64.07	67.48	69.59	68.37	68.05	67.47	68.51	66.26	66.91	68.86	64.07	65.37	65.37	66.22	66.90
bf / chronic / newton_cg			64.07	67.48	69.59	68.37	68.05	67.47	68.51	66.26	66.91	68.86	63.98	65.37	65.37	66.22	66.89
bf / chronic / saga			64.07	67.48	69.59	68.37	68.05	67.47	68.51	66.26	66.91	68.86	63.98	65.37	65.37	66.22	66.89
fs / acute / lbfgs			63.33	67.97	69.76	67.97	67.64	67.88	69.08	66.18	68.21	67.97	64.23	65.45	65.69	65.09	66.89
fs / acute / newton_cg			63.33	67.97	69.67	67.97	67.72	67.88	69.08	66.18	68.13	67.97	64.23	65.53	65.69	65.09	66.89
fs / acute / saga			63.33	67.97	69.67	67.97	67.72	67.88	69.08	66.18	68.13	67.97	64.23	65.53	65.69	65.09	66.89
fs / chronic / newton_cg			63.66	66.83	69.84	67.89	67.64	67.58	69.08	66.02	67.89	68.46	64.55	64.96	65.45	64.57	66.74
fs / acwr / lbfgs			63.66	67.15	69.67	67.72	67.48	67.78	69.08	65.93	68.05	67.40	64.39	65.53	65.85	64.68	66.74
fs / chronic / lbfgs			63.66	66.83	69.84	67.89	67.64	67.58	69.08	66.02	67.89	68.37	64.55	64.96	65.45	64.57	66.74
fs / chronic / saga			63.66	66.83	69.84	67.89	67.64	67.58	69.08	66.02	67.89	68.37	64.55	64.96	65.45	64.57	66.74
fs / acwr / newton_cg			63.66	67.07	69.67	67.72	67.48	67.78	69.08	65.93	68.05	67.48	64.39	65.53	65.77	64.68	66.74
fs / acwr / saga			63.66	67.07	69.67	67.72	67.48	67.78	69.08	65.93	68.05	67.40	64.39	65.53	65.77	64.68	66.73
fs / chronic / liblinear			63.50	66.83	69.84	67.97	67.72	67.47	69.08	65.85	67.72	68.29	64.80	64.80	65.45	64.57	66.71
fs / acwr / liblinear			63.82	67.07	69.59	67.72	67.40	67.78	69.00	66.02	67.97	67.40	64.31	65.37	65.93	64.47	66.70
Average			63.66	67.36	69.93	68.34	67.97	67.44	68.41	66.45	67.60	68.71	64.42	65.43	65.68	65.97	66.95

- Feature Selection Methods
  - bf is brute force feature selection method
  - bs is backward elimination feature selection method
  - fs is forward selection feature selection method
- Fatigue features
  - acwr is with acute chronic workload ratio fatigue features
  - only is without load features
  - acute is with acute load fatigue features
  - chronic is with chronic load fatigue features
- Solver Used
  - lbfgs
  - liblinear
  - Newton-cg
  - Saga

Appendix 27. Top 10 Surprising Predictions Training Using 3 previous Seasons

Date	Home Team	Away Team	Result	Scaled Point Differential	Prediction Probabilities	Surprise
28/01/2020 08:30	**Dallas Mavericks**	**Phoenix Suns**	**104 - 133**	**0.13**	**0.77**	**0.64**
04/01/2020 03:30	**Los Angeles Clippers**	**Memphis Grizzlies**	**114 - 140**	**0.17**	**0.78**	**0.62**
26/10/2019 07:00	**Detroit Pistons**	**Philadelphia 76ers**	**111 - 117**	**0.39**	**1.00**	**0.61**
24/02/2020 09:00	**Utah Jazz**	**Phoenix Suns**	**111 - 131**	**0.23**	**0.82**	**0.58**
04/03/2020 07:30	**Brooklyn Nets**	**Memphis Grizzlies**	**79 - 118**	**0.02**	**0.60**	**0.58**
30/01/2020 10:30	**Los Angeles Clippers**	**Sacramento Kings**	**103 - 124**	**0.22**	**0.78**	**0.56**
07/02/2020 09:00	**Phoenix Suns**	**Houston Rockets**	**127 - 91**	**0.86**	**0.30**	**0.56**
11/01/2020 09:00	**Denver Nuggets**	**Cleveland Cavaliers**	**103 - 111**	**0.37**	**0.89**	**0.52**
08/03/2020 07:00	**Houston Rockets**	**Orlando Magic**	**106 - 126**	**0.23**	**0.74**	**0.50**
24/10/2019 10:30	**Golden State Warriors**	**Los Angeles Clippers**	**122 - 141**	**0.24**	**0.74**	**0.50**

Appendix 28. Top 10 Surprising Predictions Training Using All Previous Seasons except 2004-2005

Date	Home team	Away team	Result	Scaled point differential	Prediction probabilities	Surprise
28/01/2020 08:30	**Dallas Mavericks**	**Phoenix Suns**	**104 - 133**	**0.13**	**0.83**	**0.69**
04/01/2020 03:30	**Los Angeles Clippers**	**Memphis Grizzlies**	**114 - 140**	**0.17**	**0.84**	**0.67**
24/02/2020 09:00	**Utah Jazz**	**Phoenix Suns**	**111 - 131**	**0.23**	**0.83**	**0.60**
30/01/2020 10:30	**Los Angeles Clippers**	**Sacramento Kings**	**103 - 124**	**0.22**	**0.82**	**0.59**
04/03/2020 07:30	**Brooklyn Nets**	**Memphis Grizzlies**	**79 - 118**	**0.02**	**0.60**	**0.58**
03/03/2020 09:00	**Denver Nuggets**	**Golden State Warriors**	**100 - 116**	**0.28**	**0.83**	**0.56**
06/11/2019 07:30	**Atlanta Hawks**	**Chicago Bulls**	**93 - 113**	**0.23**	**0.78**	**0.55**
07/02/2020 09:00	**Phoenix Suns**	**Houston Rockets**	**127 - 91**	**0.86**	**0.31**	**0.54**
08/03/2020 07:00	**Houston Rockets**	**Orlando Magic**	**106 - 126**	**0.23**	**0.76**	**0.52**
25/12/2019 10:30	**Denver Nuggets**	**New Orleans Pelicans**	**100 - 112**	**0.32**	**0.83**	**0.51**

Appendix 29. Code to Calculate the Surprise Metric

```
from Evaluation import backward_selection_only, brute_force_with_ratio
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler, FunctionTransformer
import pandas as pd
import numpy as np
import random as r
r.seed(42)

def probabilities(train, test, type_):
    '''returns the prediction probabilities from predictions training
    using only the previous 3 seasons
    '''
    no_of_columns = len(train.columns)
    X_test  = test.iloc[:, :no_of_columns-2].values
    Y_test  = test.Y.values
    X_train = train.iloc[:, :no_of_columns-2].values
    Y_train = train.Y.values
    #print(X_test.shape, Y_test.shape, X_train.shape, Y_train.shape)
    estimators_LR = []
    estimators_LR.append(('nan_to_num',
                          SimpleImputer(strategy='constant',
                                        fill_value = 0)
                          )
                         )
    estimators_LR.append(('rescale', MinMaxScaler()
                          )
                         )
    if type_ == '3PreviousSeasons':
        estimators_LR.append(('bs only',
                              FunctionTransformer(brute_force_with_ratio)))
        estimators_LR.append(('liblinear', LogisticRegression(max_iter = 10000,
                                                              solver = 'liblinear')))
    elif type_ == 'AllPreviousSeasons':
        estimators_LR.append(('bs only',
                              FunctionTransformer(backward_selection_only)))
        estimators_LR.append(('newtoncg', LogisticRegression(max_iter = 10000,
                                                              solver = 'newton-cg')))

    pipe = Pipeline(estimators_LR)
    #fit
    pipe.fit(X_train, Y_train)
    return pipe.predict_proba(X_test)[:, 1]


def score_differential():
    '''returns the score differential for all games in the 2019-2020 season'''
```

```
    last_season = pd.read_csv(r'Data/1 Season/2019-2020.csv')
    diff =  np.array([int(i.split(' - ')[0]) - int(i.split(' - ')[1]) \
            for i in last_season.Result]).reshape(-1, 1)
    last_season['Scaled Point Differential'] = MinMaxScaler().fit_transform(diff)
    return last_season


def availability_analysis(dataset, top, type_):
    '''performs availability analysis by using the other functions in
    this script to return a dataset with a suprise column sorted to show
    the top (function argument) suprise games'''
    last_season = score_differential()
    test = dataset[dataset.season == '2019-2020']
    if type_ == '3PreviousSeasons':
        train = dataset[dataset.season.isin(['2016-2017', '2017-2018',
                                             '2018-2019',
                                             ])]
        last_season['Prediction Probabilities'] = probabilities(train,
                                                                test, type_)
    elif type_ == 'AllPreviousSeasons':
        train = dataset[dataset.season.isin([  '2005-2006', '2006-2007',
                                               '2007-2008', '2008-2009',
                                               '2009-2010', '2010-2011',
                                               '2011-2012', '2012-2013',
                                               '2013-2014', '2014-2015',
                                               '2015-2016', '2016-2017',
                                               '2017-2018', '2018-2019'
                                               ])]
        last_season['Prediction Probabilities'] = probabilities(train,
                                                                test, type_)
    last_season['Surprise'] = abs(last_season['Scaled Point Differential'] - \
                                  last_season['Prediction Probabilities'])
    last_season = last_season.sort_values('Surprise', ascending = False)
    last_season = last_season.drop('Season', axis = 1)
    last_season = last_season.drop('Overtime', axis = 1)
    print(last_season.head(top))
    topN = last_season.iloc[0:top,:]
    topN.to_csv('Results/TopSurprise'+ type_+'.csv', index = False)
```