Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Title: NBA Game Predictions

Authors: Conor Heffernan - 16320018

Niall Hunt - 16319138

Owen Burke - 15316452

Module: Machine Learning

Lecturer: Doug Leith

Introduction:

Explain the problem and why it is interesting.

This application aims to address the problem of accurately predicting the outcome of basketball games. It aims to predict whether the home team will win or lose in an NBA (National Basketball Association) game. The application has uses in the popular pastime of NBA Fantasy leagues as well as professional sports betting, in which machine learning models are becoming increasingly prevalent. Basketball, in comparison to other sports, lends itself well to machine learning applications as there is a very high availability of statistics in comparison to leagues such as the NHL or NFL. The outcome of a basketball game is less random compared to other sports where luck has a bigger role in the outcome [1]. This aspect of the sport makes it well suited to an application that will try to predict the outcome as there is a small degree of chance/randomness.

[1] https://www.vox.com/videos/2017/6/5/15740632/luck-skill-sports

State what the input & output are.

The output of the application is a prediction of whether the home team (team that is playing at its own ground) will win the basketball game. The input to the application is a series of features aimed at capturing the behaviour and performance, so far, of the teams playing the game (this includes data such as the number of games each team has won when they were playing both home and away games, the number of games they have lost in the past both at home and on the road, points per game etc).

Dataset & Features:

To gather data, we scraped NBA game statistics across various seasons from basketballreference.com. To scrape the data, we used the requests python library to directly request information about games on various dates. We were able to parse the HTML from the request and search through it to scrape specific pieces of data using xpaths. In some circumstances, the data we wanted was rendered using JavaScript, so it was necessary to proxy our requests through a node server which used puppeteer to render the JavaScript before returning the rendered HTML it to our scraper written in python.

For our training set we collected information about every game of the last 5 NBA seasons (approximately 6375 games ((85 games * 30 teams / 2) * 5 seasons)). There were some statistics such as average points per game, and home win / loss statistics that we could not scrape directly from basketballreference.com, we could compute these as data was read into the scraper game by game. We wrote a Team class to compute statistics about a particular team as the season progressed.

To normalise the data, we used the sklearn pre-processing Standard Scalar function which takes every feature, subtracts the mean of the feature, and divides by the standard deviation of that feature. This is equivalent to the Z score and maps each feature to a value between 0 and 1. This is necessary so that one feature does not dominate the other features in the model

Discuss how you mapped the data to features. Explain the rationale.

Points per game => We were able to aggregate the number of points that a team had scored up to a certain point in a season by adding up their points scored. By dividing this by the number of games that team had played we were able to compute the number of points a team scores in a game on average. As the winner of a basketball game is the team with higher points, intuitively it seemed like the team that scores more points on average is more likely to win.

Current Form => we used the last 3 games that a team has played as a proxy for form. Intuitively we had a notion that games are not independent events. Teams can go on a winning streak which can improve their probability of winning their next game. To account for this, we included 6 features in the model which represent the performance of the home team and away team over their last 3 games, respectively. We used a LIFO queue as we scraped the data game by game, to keep track of each team's results in their last 3 games.

Head-to-head => We scraped the head-to-head data from the previous season. The rationale being that if team A beats team B 4/4 times in the previous seasons, it is more probable that team A will win the next game they play again.

Home and Away win/loss percentage: Teams with a better record of winning both home and away games should beat teams with worse records. When one team does not have a better home and away record, then some weighted combination of these two statistics based on the location of the game should give us a good indication of which team would win.

Feature selection.

After scraping the raw data and parsing it as features. We Initially had 18 features in our model (see feature example below).

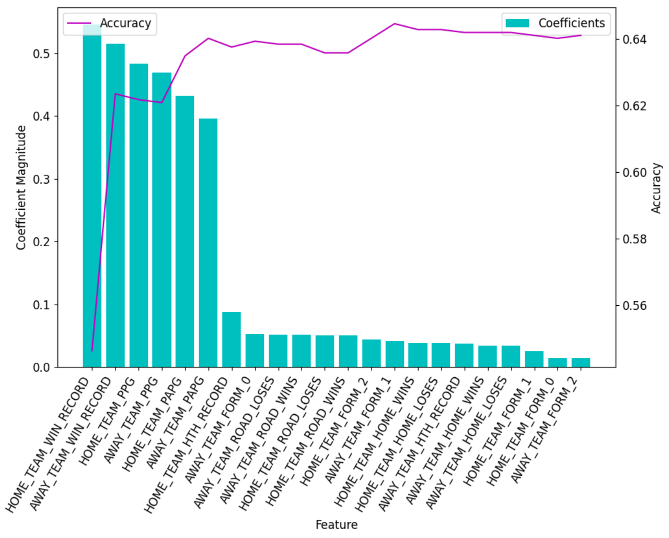
HT_HW	HT_RW	AT_HW	AT_RW	HT_FORM_1	AT_FORM_1	AT_PPG	AT_PAPG	HT_PPG	HT_PAPG
0.125	0.667	0.111	0.111	0	0	90	104	101	102

HT_HTH	AT_HTH	HT_FORM_1	AT_FORM_1	HT_FORM_2	AT_FORM_2	HT_WINS	AT_WINS	RESULT
2	0	0	0	1	0	5	0	1

HT = Home Team. **AT = away Team**. All metrics prefixed by HT will be reciprocated with the same feature prefixed by AT. For the sake of conciseness, I will explain the meanings of the features prefixed by HT. **HT_HW**= win % for the home team when playing at home. **HT_RW** = win % for home team when playing away from their home court. **HT_FORM_X** = result for home team X number of games ago. **HT_PPG** => average number of points per game scored by the home team. **HT_PAPG** => average number of points conceded by the home team. **HT_HTH** => The head-to-head record of the two teams for the previous season i.e., how many times did home team beat away team last year. **HT_WINS** => number of wins achieved by the home team so far in the season. Result => 1 if the home team won

As mentioned previously, we initially selected our features using our intuition. We selected statistics about each game that we thought would have a noticeable impact on the outcome (current win percentage of both teams, points per game of both teams, current form of both teams, etc.). We experimented with our features to investigate the best set of features to use. To do this we used sklearn's recursive feature elimination (RFE) as well as our own feature elimination using the magnitude of the trained model's coefficients. Using sklearn's feature elimination we found that we could get a very similar accuracy with a much smaller set of features. This method eliminates features recursively by training a model with a decreasing set of features and removing the feature with the smallest weight (coefficient in this case because we use logistic regression). It returns the set of features that had the highest accuracy score. Running our model with the RFE selected features we get a mean accuracy of 0.64962999 and a variance of 0.00035448444. Comparing this to our results below it is clear we have the scope to remove features while retaining accuracy.

We trained our logistic regression model and plotted the absolute magnitude of each feature's coefficient to visualise their importance on the outcome. We recursively removed features with the smallest absolute coefficient value to give us decreasing feature set sizes. We plotted the accuracy of each set of features and the features' coefficients below. For each x value in the accuracy plot the feature set used to calculate the accuracy includes the current feature and all previous features (i.e., current x value and all x values to the left of current value). This is the same technique that sklearn's RFE uses however we used the absolute coefficient values to keep features that had a strong negative effect on the outcome.



We experimented using polynomial features however our results showed no increase in accuracy. The use of said features increased training time significantly however and so we refrained from using them.

From our experimentation we decided to use the set of features that corresponded with the greatest peak in accuracy in the graph to the right. This is used when training our final model.

Methods: Describe the learning algorithms that you used. For each algorithm, give a short description

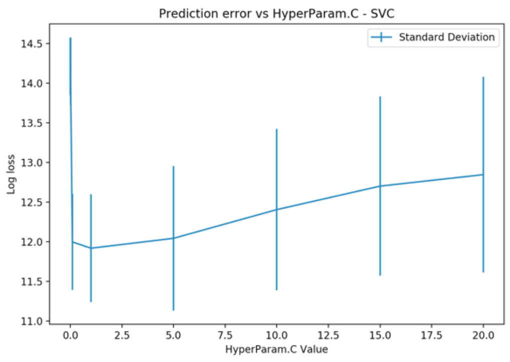
Logistic Regression: logistic regression is a classification model that is used to predict the label of a new set of features given a set of features and labels it has already encountered in training. The model is similar, in a sense, to linear regression in that it uses an equation that is familiar to us to make the predictions, this being $sign(\Theta^T x) = sign(\Theta_0 x_0 + \Theta_1 x_1 + \Theta_2 x_2 + ... + \Theta_n x_n)$ where n is the number of features used to train the model and Θ are our parameter values that must be learned. These values of Θ are found using the following cost function $\log(1 + e^{y\theta^T x})$ / *log(2)*, unlike linear regression which uses the mean squared error. This use of cost function for logistic regression results in a small cost when the result of $\Theta^T x$ is far larger than zero and y=1 as well as when $\Theta^T x$ is much less than zero and y=-1. By minimizing this function, we encourage values of *Θ* that push $\Theta^T x$ far away from the decision boundary (this being the line/plane/hyper-plane where $\Theta^T x = 0$, thus being the boundary between predicting +1 or -1*)*. This cost function is minimised using gradient descent (similarly to linear regression), to provide values of *Θ* that result in the smallest possible return value from the cost function. Gradient descent is an optimization algorithm that uses first order derivatives to find a local minimum of a differentiable function (this being the cost function).

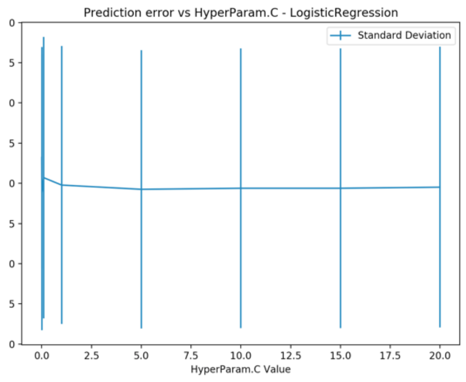
kNN classifier: A kNN classifier makes a prediction by finding a set of points in the training set most like the point for which it is trying to make a prediction (smallest Euclidean distance) and making predictions based on the values of the neighbours. The number of points in the neighbourhood is a hyperparameter, k, of the kNN model. In the case of a classification problem, the model takes a majority vote (for example, if the k (number of neighbours) is 3 and two of the neighbours have a +1 output label, then we predict +1). A kNN model has four aspects that must be specified. As mentioned already, two of these are the distance metric used to get the distance between data points (typically Euclidean distance, but can use others) and the number of neighbours, k. Another aspect is the weights attributed to neighbours. The weights can be assigned uniformly (where each of the k neighbours carry equal weights) or by their distance (Gaussian), where closer neighbours have a greater influence on the prediction. The fourth aspect, as mentioned, is the aggregation method for the neighbourhood (majority vote in classification). With kNNs, increasing k will tend to result in the model under-fitting (as it generalises to a larger neighbourhood) and vice versa, decreasing k will tend to cause over-fitting. Efficiency/computation speed can also be an issue with kNN models, as we need to iterate over all the training data points to find our k nearest neighbours. Therefore, the models are best suited to small datasets.

Kernelized SVM: SVMs (Support Vector Machines) are not dissimilar to Logistic Regression models, however there are a few key differences. SVMs make use of the hinge loss function, given as $max(0, 1 - y\Theta^T x)$. One of the key differences between the two loss functions is that the hinge loss function is not differentiable, and it results in a loss of zero for values of Θ which result in $\Theta^T x >= 1$ when $y = 1$ and $\Theta^T x <= -1$ when $y = -1$. SVMs also require a penalty ($\Theta^T\Theta$ / C) to be added to the cost function to inhibit values of Θ with a large magnitude, as we can otherwise just increase Θ to force $y\Theta^T x > 1$. The hyperparameter, C, to control the penalty must be found through cross-validation. The choice of cost function is the only difference between the two models. A kernelized SVM associates a feature with each piece of training data and makes a prediction based on the features closest to the vector we are trying to make a prediction for, incorporating a similar concept to a KNN classifier as discussed above. Unlike a KNN classifier, the prediction is based on a linear model which weights each feature as a function of the distance from the input vector to the corresponding training vectors ($prediction = sign(\Theta_0 + \Theta_1 y^{(1)} K(x^{(1)}, x) + \Theta_2 y^{(2)} K(x^{(2)}, x) + ... + \Theta_m y^{(m)} K(x^{(m)}, x))$ where m is the number of training points and $K(x^{(i)}, x))$ is a weight applied to each training vector. The kernel, K, is typically a Gaussian function, $K(x^{(l)}, x) = e^{-\gamma d(x^{(i)},x)^2}$, where d is the Euclidean distance. Hyperparameter $\boldsymbol{\gamma}$ controls how quickly $K(x^{(l)}, x)$ decreases as the distance between $x^{(i)}$ and x grows. As we use the training points as features, this model favours smaller datasets, as it is otherwise expensive to use large datasets.

Experiments/Results/Discussion:

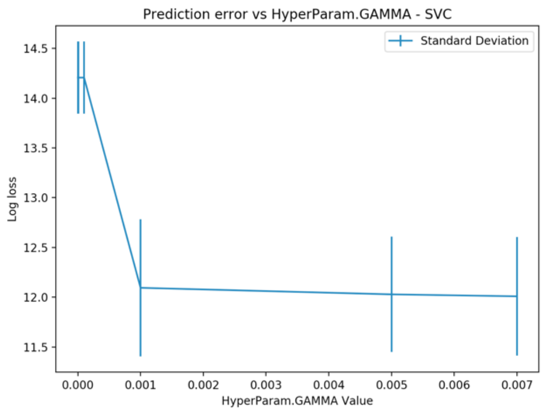
As mentioned above, three models were examined (Logistic Regression, Kernelized SVM, kNN Classifier). In terms of hyperparameters, the penalty value, C, was cross-validated for both the logistic regression model and kernelized SVM (Note: For all hyperparameters, 10-fold cross validation was used to split the training data into *train* and *test*). This cross validation yielded the plots shown above (Note: all cross validation plots use the mean and standard deviation of the log loss, rather than MSE, as we are dealing with classification):

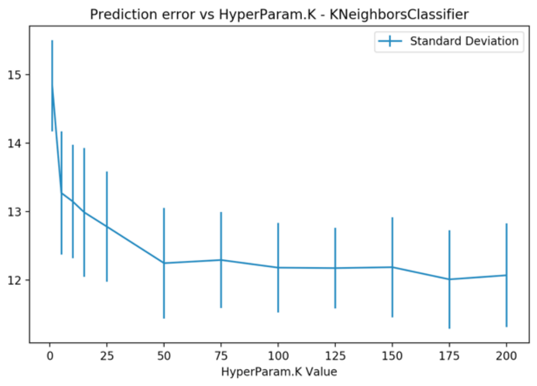




The above plots illustrate that the ideal value for C for both the logistic regression model and SVC are 5 and 1 respectively, as these values result in the lowest mean log loss for the classifiers across the 10 folds, and higher values would result in a greater loss (see SVC plot) or could lead to overfitting as there is no inhibition on the parameter values being of an extremely large magnitude (see Logistic Regression plot).

As well as the hyperparameter C for the kernelized SVM, gamma was also cross validated. As mentioned, gamma controls how quickly the output from the kernel function decreases as the distance between the training feature, $x^{(i)}$, and the input, x, grows. This gamma value can therefore be used to balance the trade-off between over and under-fitting, as a small gamma results in a slower decrease as distance grows (therefore effectively considering a larger neighbourhood and generalising more) while a larger gamma has the opposite effect. The cross validation for gamma for the SVC is shown below. From the above plot, we can see that the ideal value for gamma from our validation is 0.001.





We also performed cross validation on the hyperparameter, K, for the kNN classifier (k being the number of nearest neighbours to consider when making a prediction). Intuitively, with a larger neighbourhood to consider, we can suffer from under-fitting, as the model generalises too much. Conversely, we can suffer from overfitting if k is chosen to be too small. The cross validation for K is shown above. From the above plot, we can see that the ideal value of K to choose from our validation is 125.

In terms of whether we over or under-fit the training data, I do not believe that we suffered massively from either, as we cross validated all our hyper-parameter choices, as well as normalising the input data.

In terms of model parameters and how they are chosen, the logistic regression makes use of gradient descent and kernelized SVM uses sub gradient descent as the hinge-loss is not differentiable. The kNN has no such parameter values to learn.

Regarding model parameters, the following values denote the Θ values from the logistic regression model:

[0.05327453, 0.03906012, -0.03906012, -0.03625056, 0.03625056, 0.06355181, 0.04835476, 0.03740588, 0.04270378, 0.60967686, -0.58291722, 0.51430034, -0.45993147, -0.57177458, 0.49390752] from Θ_1 to Θ_{15}. As mentioned, these values are found through gradient descent. In terms of how many training/tests examples we used, we trained the model on four seasons worth of NBA games and then tested on a new single season

Did the model overfit. Steps taken to overcome overfitting

We did our best to ensure that the model was not overfitting. In this effort we carried out cross validation for all model hyperparameters as discussed above. We chose hyperparameter values that seemed to minimise the log loss without falling into the trap of decreasing the penalty so much that the model overfits the training data. To verify that the model generalised well, we carried out a manual 5-fold validation taking 4 seasons in the interval {2015,2016,2017,2018,2019} as test data and used the data from the remaining season as test data. Doing this and using all models described above, we obtained the following results:

Model / Test	2015	2016	2017	2018	2019	Mean	Variance
Log Regression	0.6752851	0.62996942	0.6559878	0.65980168	0.64448336	0.653105485	0.00023094
KNN	0.6684410	0.63073394	0.6514111	0.64378337	0.617338	0.64234150	0.00030508
SVC	0.6730038	0.63073394	0.6552250	0.65751335	0.61471103	0.64623742	0.00043172

As you can see barring 2016, the logistic regression model outperforms the other models for every year. All models have similar average accuracy (logistic regression has the highest accuracy). The low variance for every model seems to indicate that the models are not overfitting/ are generalising well.

Evaluation / Results

Before you describe your results explain what your primary metrics are: accuracy, precision, AUC, etc.

Our primary metrics when considering the results and performance of our models is the accuracy score (the percentage of predictions that match the known output) as well as considering AUC/ROC curves (illustrates the classification ability of a classifier system as its discrimination threshold is varied. An ideal classifier will have an area under the curve of 1, while a random classifier would have ~0.5 AUC). As our data is balanced, accuracy is an appropriate measure to use. We also look at precision, recall and f1 score. The precision score shows us the ratio of true positives to all guessed positives while recall gives us the ratio of all true positives to all actual positives. These give us a good indication of the performance of our model. The F1 score is a combination of precision and recall it gives us the mean harmonic mean of the two. As with other scores the higher numbers indicate better performance of our classifier.

Confusion matrix and AUC/ROC curves. Performance metrics such as precision, recall, and accuracy.

	Accuracy	Precision	Recall	F1
Logistic Regression	0.653105485	0.66896208	0.7958949	0.72673027
SVC	0.64623743	0.65223737	0.83408203	0.73196238
KNN	0.6423415	0.65256945	0.81881166	0.72620016

The above table uses the same manual 5-fold validation as seen above when testing for generalisation. The values shown here are the averages over the 5 folds for accuracy, precision, recall and f1-score. We can see from these results that logistic regression gives us the best accuracy and precision. However, SVC gives us the best recall and f1-score. We believe the accuracy of logistic regression and the ability to scale with large datasets are the deciding factors in choosing it.

Compare performance against a reasonable baseline

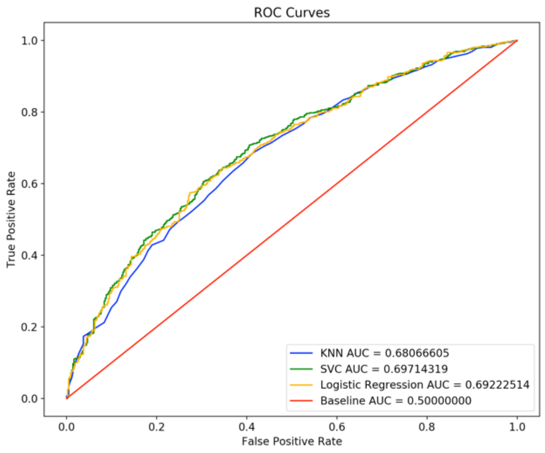
The following are the accuracies of the three models we considered, as well as a reasonable baseline (a most-frequent dummy classifier which simply predicts the most common class from the test data) for the 2019 season

Model	Logistic	SVC	KNN	Baseline
Accuracy	0.653105485	0.64623742	0.64234150	0. 5464098073555

As you can see, all three models outperform the baseline by a statistically significant margin, which is a positive result. In terms of comparing the three models, as mentioned, the logistic regression is fractionally more accurate, although not by any significant amount. However, as the amount of training data could increase, the kernelized SVM and kNN may be too expensive to train as the kNN needs to iterate over all the training points to take its nearest k neighbours and the kernelized model also needs to iterate over all the training points to use them as features.

AUC, ROC & Confusion Matrices

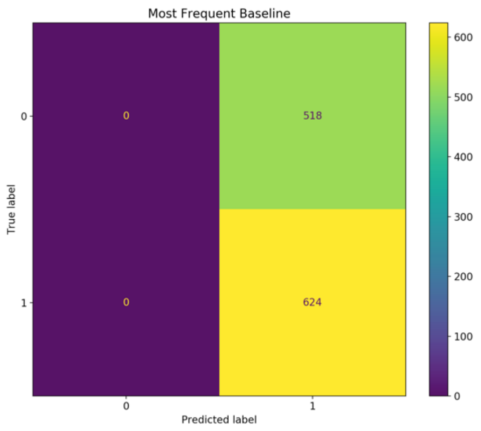
The below plots show the AUC/ROC curves for the three models we are considering and the confusion matrix for logistic regression. The AUC values correlate with the accuracies listed in the table above:

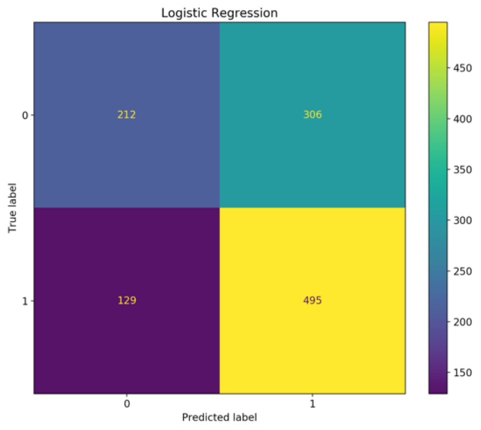


with the logistic regression model outperforming the other models ever so slightly, but not to any significant degree. The below plot is the confusion matrix for the baseline:

Confusion Matrices

From examining the confusion matrices, the logistic regression is a better choice of model as it has far more true negatives, almost half the number of false positives and a similar number of true positives (although it does have more false negatives). While the performance of the models we choose to examine is not extremely high, they are noticeably better than a reasonable baseline which is a positive result and even the sophisticated models from the likes of 'fivethirtyeight' do not achieve a much higher accuracy (https://community.wolfram.com/groups/-/m/t/1730466 *"FiveThirtyEight correctly predicted the winner of 66.42% of games during the 2017-2018 NBA season"*)





Summary

We created an application that uses NBA statistic data to predict the outcome of upcoming games. We get our features by scraping performance statistics for the home and away team of an upcoming game. This is fed into our model to predict if the home team will win or not. We tested this application with three different models: logistic regression, kernelized SVM, and k nearest neighbours. Our results show little difference in the performance of the models, with logistic regression just edging out the others in terms of accuracy. We believe logistic regression is the best choice of model given its ability to scale with increased training data and to converge in a practical timeframe given a large dataset in comparison to kernelized SVM and kNN models. Overall, our application performs significantly better than the baseline and is comparable to many state-of-the-art models. As mentioned above FiveThirtyEight correctly predicted the winner of 66.42% of the games in the 2017-2018 season. Our model correctly predicted the winner of 65.31% of the games in the 2019-2020 season.

Contributions:

Describe what each team member worked on and contributed to the project. It's important to give a decent level of detail e.g., what code was written by whom, which experiments were carried out by whom, who wrote each part of the report. Each team member needs to initial this section (electronic initials are fine).

Owen: Cross validation of hyper-params, confusion matrices, ROC curves, *experiments/results/discussion* in report with Conor, *methods* in report, Code: score_scraper.py, game.py, game_writer.py, main.py, helper_functions.py, CSVGenerator.py, etc (blue = worked collaboratively on code). Vast majority of the code was written over Zoom calls.

Niall: Feature selection and engineering, added ELO ratings as feature (https://fivethirtyeight.com/features/how-we-calculate-nba-elo-ratings/), performance evaluation (precision, recall, f1, accuracy for multiple feature sets). Report: Feature Selection, Performance metrics, Summary. Code: feature_selector.py, feature_processor.py, main.py, CSVGenerator.py (blue = worked collaboratively on code)

Conor: Scraping data (including Node.js proxy), 5-fold validation for model accuracy, feature generation. Discussion in report *Did the model overfit, Models used - Knn Classifier, Mapping Data to Features.* Code score_scraper.py, game.py, main.py, CSVGenerator.py TeamStats.py, Teams.py, PuppeteerServer.js, head_to_head_scraper.py, rankings_scraper.py. (blue = worked collaboratively on code). As Owen mentioned, a lot of the code was written as a group over calls

Include a GitHub link (or similar) to the code written as part of the project:

https://github.com/CHeffernan087/NBA_Machine_Learning_Model