A SUPERVISED LEARNING APPROACH TO PREDICTING NBA GAMES

WEDNESDAY, MAY 12, 2021

BENNETT COHEN
CHRISTOPHER LANDGREBE

Department of Industrial Engineering & Operations Research
University of California, Berkeley

Contents

Introduction

NBA betting is a big business, accounting for upwards of $500 million in legal bets placed on every month games throughout its 82 game season. With this in mind, we wanted to explore how we could maximize our betting profits by making more accurate predictions on the outcomes of games through machine learning. We searched for the most thorough and reliable data available, eventually choosing to use the NBA's own highly trusted statistical database website https://www.nba.com/stats/, along with a sports betting research website called from https://www.sportsbookreviewsonline.com). As is discussed in later sections, the raw data lends no feasible value to our task because game data only is available after a game outcome is known, so we spent considerable time cleaning and processing the data to curate statistical features. Using these features, we began implementing machine learning techniques (Logistic Regression, Random Forest, Gradient Boosting, and Neural Networks) to compare against a baseline model of predicting the home team to win in recent NBA seasons. We first implemented a modified 5-Fold Cross-Validation approach on all of our models, followed by an exploration into using a hold-out validation set and blending approach on the predictions from our models to see if we could improve our accuracy (primary performance metric). Overall, we found success with our modeling techniques, and our best performing models performed significantly higher than a baseline model and many pre-existing models. As will be discussed, our largest area for improvement is in feature engineering, which could take this model from exploration into true practical feasibility as a betting engine. The following sections outline our processes of data collection , feature engineering, and modeling, and the appendices that follow provide the codebase and technical explanations of these techniques in greater detail

Data Collection and Feature Engineering

i Data Collection

The data used in this project has been extracted from the NBA's own statistics website (Link: https://www.nba.com/stats/) along with https://www.sportsbookreviewsonline.com. There are ten main data sets, which we later combine in our feature engineering section. There are 10 data sets (some of which created by us for ease), which are described below:

1. GAMES ⟶ the box score of all games between the 2007 and 2018 season, along with the winner of each game.
2. DETAILS ⟶ the box score statistics for each player in each game.
3. PLAYERS ⟶ every NBA player between 2007 and 2018, and what team they played for each season.
4. TEAMS ⟶ statistics about the team itself, including location, arena capacity, founding data, and coach.
5. ODDS ⟶ the winning odds for the home and away team for each game.
6. AWARDS ⟶ a data set we created ourselves for the MVP, DPOY, ROY, and 6MOY in each season of data in the provided sets.
7. FIRST TEAM ⟶ a data set we created ourselves that lists the 5 players on the All-NBA 1st Team in each season of data in the provided sets.
8. SECOND TEAM ⟶ a data set we created ourselves that lists the 5 players on the All-NBA 2nd Team in each season of data in the provided sets.

9. THIRD TEAM $\longrightarrow$ a data set we created ourselves that lists the 5 players on the All-NBA 3rd Team in each season of data in the provided sets.
10. ALTITUDE-TIMEZONE $\longrightarrow$ a data set we created ourselves that lists the altitude and timezone of each NBA team's home arena.

For clarity and, we won't show what each of these data sets looks like in their raw form, but instead will show how we use these data sets in conjunction to construct our features to build models off of.

ii Feature Engineering

Our models are based off the following five feature types (F2, F2, F3, F4, and F5), which are the various factors that we believe might influence which team wins a game. We begin with the most common features and slowly add more unique features to create a more robust and potentially accurate model. We don't do feature selection here, and save analyzing feature significance for later sections.

2.1 F1: Cumulative Team Statistics

First, we look at the GAMES data. The primary issue here is that these features don't actually help make predictions because the box score only is known *after* the game is over. If we were to use these as features, our models would have 100% accuracy but would be infeasible. However, what probably does matter to predicting which team will win is the cumulative box score, or the average statistics for a team in every game up until the game we try to predict. This is built off of the assumption that if a hypothetical home team has performed better than the away team across all games earlier in the season, they should be more likely to win this game, and vice versa. For instance, using the average points-per-game (abbreviated to "PPG"), if PPG_1 = 110.6 and PPG_2 = 96.6, Team #1 is likely to score more points than Team #2, and would win the game. Our GAMES data includes the statistics for the home and away team separately, which leads us to make a key assumption: teams tend to perform differently when they play at home than when they play away. This is touched upon in F5 and can be due to various factors (time zone, traveling, altitude, etc). As a result, in a given example above, PPG_1 = 110.6 means that when Team #1 is the home team, they average 110.6 points per game. It does not mean Team #1 average 110.6 points across all games.The box-score statistics we implement are below (the same features are used for the away team):

Statistic	Meaning
WIN_PCT_home	Percentage of Games Won at Home
PTS_home	Avg. Points-per-Game
FG_PCT_home	Avg. Percentage of Field Goals/Baskets Made
FT_PCT_home	Avg. Percentage of Free Throws Made
FG3_PCT_home	Avg. Percentage of 3 Pointers Made
AST_home	Avg. Number of Assists
REB_home	Avg. Number of Rebounds

2.2 F2: Momentum & Recency

Beyond simple season-long cumulative statistics, we know that hot-streaks and cold-streaks are extremely common in sports. A team can get "hot" at the right time, winning many games in a row, so if a team has a bad start to the season, but is very "hot" at the time of a game we are trying to predict, simply using the season statistics wouldn't be reflective of their current level of play (the same is of course true for a cold streak). This is where the idea of momentum and recency comes from; we need to create features to capture teams most recent performances. To do this, we simple create the same exact features as we did in F1, but not include the entire season. We aren't sure exactly what length of time is considered significant to capture a team's momentum, so we create features for the previous five and previous ten games separately. It's possible one of these is shown to be more significant later on, but because this is rather simple to implement, we do both time frames. Again, we use the seven statistics from the table above. This increases our feature count to $2 \times (7 + 7 + 7) = 42$ features. There is potential collinearity between these because the information for the last five games is captured in the last ten statistics, which is also captured in the season-long statistics, but we don't need to address this just yet until we see how the models perform.

2.3 F3: Cumulative Player Statistics

Delving deeper yet again, we also want to consider how each player performs in each game to potentially unveil certain trends and increase our model performance. For F1 and F2, we only looked the the data from the GAMES (and TEAMS in order to turn the TEAM_ID values into team names), but now we consider the more complicated DETAILS data, which consists of the box-score statistics for each player on each team in each game. This data is much larger than the other two data sets. Another assumption we make is that the majority of the most popular and predictive box score statistics are performed by the starting five of a team (2 Guards, 2 Forwards, 1 Center) and that the contributions of the bench players aren't as important. Further, this makes some intuitive sense in that if a "superstar" player such as Kevin Durant scores 60 points in a game, his team is more likely to win and the bench players probably did less in that game because there is limited time to score that many points. This allows us to simplify our data set and only consider five players from each team. Also, we don't actually use the names of the starters, but their position instead, which we simplify further because the data doesn't distinguish between the two types of guards and forwards, leaving us with only three positions to consider. We then use the same functions from earlier to calculate cumulative season-long and last-five team box score stats, but for each type of player.These statistic definitions are shown below:

2.4 F4: Head to Head Statistics

Another factor of NBA games to consider is the notion that certain teams perform differently against different teams regardless of their cumulative statistics due to the different styles of play. For instance, if Team #1 tends to shoot many perimeter shots (long range shots such as three pointers) and if their opponent's (Team #2) defense doesn't defend well against these types of shot attempts, there should be some inherent advantage to Team #1, but this isn't currently reflected in our features. It's very difficult to identify a team's style of play because it can vary based on their opponent, so we get around this by simple creating statistics for the exact combination of home and away teams; in other words, the cumulative statistics in these head-to-head match ups throughout the season.

2.5 F5: X-Factor Features

For our final feature type, we will consider factors that don't translate into quantitative measures as easily. The features detailed below are ones that are less conventional than the features created by the previous 4 types, but they are ones that we believe have the possibility of predictive power as well.

2.5.1 NBA Awards

Each season the National Basketball Association hands out various awards to deserving league players. Four of the most recognized awards are

1. Regular Season Most Valuable Player (MVP)
2. Defensive Player of the Year (DPOY)
3. Rookie of the Year (ROY)
4. Sixth Man of the Year (6MOY)

NBPlayers who win these awards typically exhibit a level of skill, leadership, and overall player quality beyond what statistical metrics may contain. If a player won one of these awards in recent years, we can make the assumption they are providing a similar level of skill and quality for their team now, though this benefit decreases over time so we decide to only look at the last four years (i.e. the "prime" of an NBA player's career is short enough such that they probably are out of it in five years). We use the AWARDS data set we created to calculate the number of players on the starting five of the home/away roster that won one of these awards in the last four years, creating eight new features (four awards for both home and away).

2.5.2 The "LeBron Effect"

Over the course of the last 15+ years, LeBron James has proven that he is a transcendent player, providing a level of skill and leadership beyond any of the awards above (he has often lost awards simply because voters don't want to give it to him even if he is the obvious choice). To capture his value to a team, we create a binary variable for both home and away to reflect if he is on the starting roster at the time of the game.

2.5.3 All-NBA Teams Selections

At the end of each season the NBA chooses the All-NBA First, Second, and Third Teams. Each of these "teams" is made up of five players, where the All-NBA First Team is the Top 5 players in the league according to the NBA (position included), the All-NBA Second Team is the next best 5 players, and so on. We calculate the same counts for these teams as we did above with awards in the last four years though this feature expands the range of values because we are looking at a five player roster instead of one player. We make the same assumptions as earlier.

2.5.4 Home Court Advantage

Of the four major U.S. sports leagues (NBA, NFL, MLB, NHL), home court advantage is most meaningful and significant in the NBA. As we look at our baseline model, this will become clear, but the home team wins in the NBA in our data set around 59% of the time, a number much higher than in other leagues. That being said, home court advantage varies by team based on many factors and in wha ttime period. As such, we quantify a team's home court advantage as the winning percentage of the home team (when they play at home) over the last four years.

2.5.5 Altitude and Timezone Difference for Away Team

Throughout the NBA season, teams trave all over the country (and out) to play at various arenas, in various timezones and at different altitudes. A significant change in timezone (like for an east coast team playing on the west coast or vice versa) could *potentially* handicap the away team. This could be due to adverse effects of traveling or maybe a negative effect on the away team's sleep schedule. Additionally a drastic change (particularly a drastic *increase*) in elevation could also have a significant effect on the away team's ability to perform as well as they typically do. Many Studies have examined elevation and found that physical activities tend to be more difficult at higher elevations, Meaning that the home team may be acclimated to the elevation. We believe that our current data doesn't sufficiently represent the effects of these two factors so this in mind, we designed two features to hopefully accurately represent their effects. These features are the change in elevation and change in timezone for the away team from their own arena. For instance, if the Knicks travel to Los Angeles to play the Lakers, these features would be the change in elevation between the arenas, along with -3 because Los Angeles is three hours behind New York.

2.5.6 Vegas Betting Odds

Our project is motivated by sports betting, which relies on Vegas bookmakers to set the odds off each team winning in order for bettors to make bets. We believe that these bookmakers use large data sets and robust models to set these odds to maximize their profits (as they are businesses), and their "predictions" about the game are summarized by the odds they make. We can't access their data, but we can access the odds, so we add a feature for Vegas betting odds for the home and away team based off of the ODDS data set.

Building Models to Predict Individual NBA Games

Before building models, we need to actually create training and testing data. Because this model will hopefully be used to predict future games (i.e. the time component of the data is significant). Our training data will be data from the 2007 Season to the 2016 Season, and our testing data will be from the 2017 and 2018 Season. This corresponds to 83% training and 17% testing. One thing we want to note here is that we tried the same approach but with randomly split training and testing data, and achieved accuracy values ≈ 2.5-3% higher than time-based splitting for the best models, but it doesn't make sense to simply split randomly if our goal is to predict future games (i.e. this accuracy increase isn't relevant).

i Approach #1: Modified 5-Fold Cross-Validation

1.1 Model #0: Baseline Model

For our baseline model, we introduce a Dummy Classifier to predict the most common label in the training dataset, which is *HOME_TEAM_WINS* = 1. In the training set, the home team wins ≈ 58.8% of games. An alternative would be to pick the team with a higher win percentage but we choose to follow the metric from class. We also create encoded variables from our categorical variables here.

Baseline Model Confusion Matrix and Performance.

	HOME WIN	AWAY WIN
PRED. HOME WIN	0	1081
PRED. AWAY WIN	0	1545

Table 3.1: Confusion Matrix

	Test Set Statistic
Accuracy	0.5883
TPR	1.0000
FPR	1.0000

Table 3.2: Performance Metrics

1.2 Model #1: Logistic Regression

For our first model after the baseline, we implement regularized Logistic Regression using Sci-Kit Learn. Because there are 832 features used in our model (many of which are likely highly correlated, such as Field Goals Made and Points Per Game), it's possible we will be overfitting the noise of these features. We will be doing feature selection with our Random Forest and XGBoost Models later, but beacuse we aren't sure which have the most predictive power yet, we will use regularization to prevent overfitting.

Logistic Regression Confusion Matrix and Performance.

	HOME WIN	AWAY WIN
PRED. HOME WIN	458	623
PRED. AWAY WIN	275	1270

Table 3.3: Confusion Matrix

	Test Set Statistic
Accuracy	0.6580
TPR	0.8220
FPR	0.5763

Table 3.4: Performance Metrics

1.3 Model #2: Random Forest

As we move to tree based models, we first recognize that our data set and computation power limits us both for cross-validation and using a hold-out validation set. On one hand, our data is too large (12,835 x 832) to do cross-validation on a large grid of values in a reasonable time frame. On the other side, however, we aren't sure there is enough data to feel comfortable enough using a hold-out validation set to tune hyperparameters and reduce overfitting. Instead, we will first use reasonable values for hyperparameters to run a Random Forest on the entire data set. Then, we will look at Feature Importances, and select the features with the most predictive power under this model (where predictive power is defined as the mean and standard deviation of accumulation of the impurity decrease within each tree for each feature). The motivation behind this dimensionality reduction is that there are many features we've created that are highly correlated, so our model is overfitting to the noise of our training set. We then perform a grid search on the *max_features* hyperparameter (the number of features to consider at each split), which runs approximately 100x faster because we never consider a high number of features. In an ideal world, we will see our performance actually increase from our "Reasonable Model." Our "reasonable" outputs the following test set performance metrics, along with the a bar plot of the Top 30 Feature Importances are shown below.

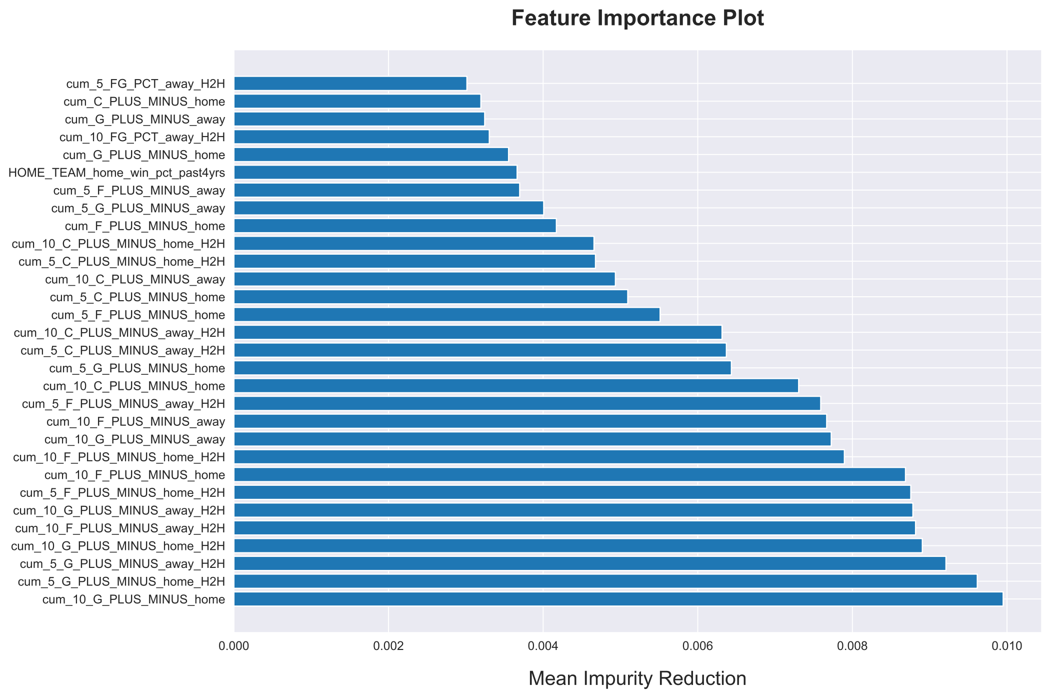
Reasonable Random Forest Confusion Matrix and Performance.

	HOME WIN	AWAY WIN
PRED. HOME WIN	430	651
PRED. AWAY WIN	267	1278

Table 3.5: Confusion Matrix

	Test Set Statistic
Accuracy	0.6504
TPR	0.8272
FPR	0.6022

Table 3.6: Performance Metrics



After using this subset of features, we using GridSearchCV and find out optimal *max_features* = 1. We run our best estimator model on the test set and output the following results. Appendix #3 has the code for cross-validation.

Cross-Validated Random Forest Confusion Matrix and Performance.

	HOME WIN	AWAY WIN
PRED. HOME WIN	481	600
PRED. AWAY WIN	302	1243

Table 3.7: Confusion Matrix

	Test Set Statistic
Accuracy	0.6565
TPR	0.8045
FPR	0.5550

Table 3.8: Performance Metrics

1.4 Model #3: XGBoost with a Sequential 5-Fold Cross-Validation Process

For our boosting model, we use the XGBoost library, and perform a series of grid searches to perform cross-validation to reduce overfitting and improve test set performance. This greedy process and code base for these steps were taken from former Columbia University Masters

Student and Spotify ML Engineer Aarshay Jain (Link: https://www.analyticsvidhya.com). Appendix #3 outlines the sequential cross-validation process, and it drastically reduces our run time by first picking an optimal $n\_estimators$ for learning rate = 0.1, then cross-validating other hyperparameters one by one and picking the optimal at each step, finally decreasing our learning rate to 0.005 at the end. These optimal parameters are shown below. Note that the optimal regularization parameter was found to be very high. This hyperparameter penalizes features which increase the cost and reduce overfitting, indicating that feature selection is probably also useful for this model to reduce is further.

Hyperparameter	Optimal Value
learning_rate	0.005
n_estimators	1142
max_depth	2
min_child_weight	4
gamma	0.0
subsample	0.8
colsample_bytree	0.8
reg_alpha	100

Figure 3.1: Optimal hyperparameters for XGBoost

As with the Random Forest model, we run our best estimator model, and calculate the test set performance statistics. We also use the Feature Importance score to find the Top 30 most importance features in this model.

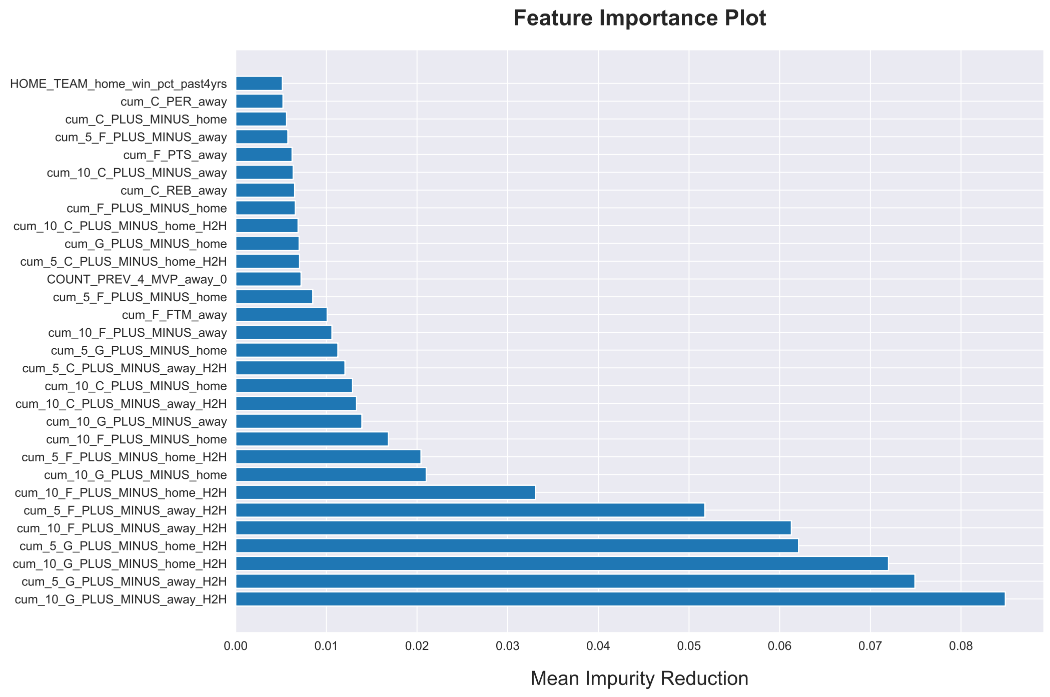
Sequentially Cross-Validation XGBoost Confusion Matrix and Performance

	HOME WIN	AWAY WIN
PRED. HOME WIN	498	583
PRED. AWAY WIN	324	1221

Table 3.9: Confusion Matrix

	Test Set Statistic
Accuracy	0.6546
TPR	0.7903
FPR	0.5393

Table 3.10: Performance Metrics



To see if we dealt with overfitting properly, we re-run our XGBoost model with only the Top 30 features. Below, we see the results compared to using all features. From these tables of outputs, we see that using all the features is slightly better than using very few, likely due to the high regularization hyperparameter value.

	HOME WIN	AWAY WIN
PRED. HOME WIN	491	590
PRED. AWAY WIN	321	1224

Table 3.11: Confusion Matrix

	Test Set Statistic
Accuracy	0.6531
TPR	0.7922
FPR	0.5458

Table 3.12: Performance Metrics

1.5 Model #4: Neural Network (Multilayer Perceptron Classifier)

For our final model, we explore the implementation of a Neural Network using Sci-Kit Learn's MLPClassifier. We perform no cross-validation on the hidden layer sizes or node counts, and create a network with two hidden layers with 25 nodes in each. We attempted to use TensorFlow but couldn't quite tune it properly, so this is more of a toe-dip into the water of deep learning. Regardless, our best outputs are shown below.

Neural Network Confusion Matrix and Performance.

	HOME WIN	AWAY WIN
PRED. HOME WIN	430	651
PRED. AWAY WIN	267	1278

Table 3.13: Confusion Matrix

	Test Set Statistic
Accuracy	0.6504
TPR	0.8272
FPR	0.6022

Table 3.14: Performance Metrics

1.6 Final Model Outputs

We aggregate our results in the table below. We see that

	Client_ID	Start_date	End_date	Num_blog_posts	Num_web
0	0	2015-08-26	2016-11-02	8	
1	1	2011-03-31	2019-08-27	5	
2	2	2010-09-26	2015-06-12	6	
3	3	2012-06-10	2019-01-22	4	
4	4	2011-06-23	2014-12-27	8	

ii Approach #2: Hold-Out Validation Set and Blending

Recall that our limiting factor in building our most robust model (from a modeling standpoint—not in regards to data) is our lack of computational power. That was why we removed potentially unhelpful features before cross-validation. However, we now try the alternative approach, which is to split our data into (1) Training Set, (2) Validation A Set, (3) Validation B Set, and (4) Testing Set. To split this, we first set our Testing Data to be the 2018 Season (8.5%). We then split the remaining data into Train, Validation A, and Validation B randomly such that Validation A and B are each 9.2% of the total data, and Training is 73.2%. Instead of cross-validation, we tune hyperparameters by iterating through a search space and choosing the value that maximizes the accuracy on Validation A. After doing this for all of our models, we use the predicted probabilities for the Validation B Set to train a Logistic Regression model to evaluate on on our test set (effectively creating a blended prediction). The results are shown below. Our main goal of this exploration was to see if a Blended model could outperform our best model from earlier, and it does (albeit by 0.0019)! Interestingly, our Logistic Regression is the second best performing model in terms of accuracy, outperforming our Neural Network.

	Accuracy	TPR	FPR
Baseline Model	0.5883	1.0000	1.0000
Logistic Regression	0.6611	0.8162	0.5606
Random Forest	0.6527	0.8311	0.6022
XGBoost	0.6531	0.7961	0.5513
Neural Network	0.6596	0.8634	0.6318
Blended	0.6664	0.8239	0.5587

Figure 3.2: Final results using a hold-out validation set approach and blending.