"Predicting NBA team performance with machine learning using Python"

Joseph Bu, JeongJun Moon, Lukas Metzner

Summary of research questions and results:

1. How can we use metrics to predict season standings?

To predict the current on-going 20-21 NBA season standings, we trained fundamental variables of a team's offense and defense with the Decision Tree Classifier. Then, we tested this model on the on-going 20-21 NBA season. As we compared the predicted results to the actual current standings, we realized that the outcome was not exactly the same, but it still made notable predictions. For example, when we made the input for our model: 10,000 times, the predictions in the top 5 standings included the two-three teams that were actually in the top 5 (Utah Jazz, Phoenix Suns, Brooklyn Nets). At the same time, the predictions in the lowest five standings included the three teams that were actually in the lowest five (Detroit Pistons, Houston Rockets, Washington Wizards).

2. How can we use recent game team performance to predict the outcome of a game? What are the most effective statistical features for predictors?

We were able to train a Decision Tree Classifier on game outcome (win/loss) labels, and features of average statistics from a team's recent games, which could predict game outcomes with some degree of accuracy. The optimal recent game sample size from which to base the features was 10 games, and the best maximum decision tree depth was 1 node. We found that in these types of predictive models, regardless of parameters, the most important recent performance statistics for outcome prediction included defensive rebounds and turnovers, while the least important included days since the last game and recent game win percentage.

3. How does the difficulty of a team's schedule (number of back-to-back games) affect the winning probability or outcome of a game?

We looked at multiple seasons in the NBA and compared the number of back-to-back games that differed from season to season and also considered the win percentage of back-to-back games vs. non back-to-back games. In order to eliminate the variable of stronger teams and weaker teams, we took the top 5 and bottom 5 teams from a given season and averaged out their win percentages in order to get an average back-to-back and non back-to-back win percentage. We found that although it differs from season to season, there is a significant enough change in win percentage that shows teams generally do worse in back-to-back games than in regular games.

Motivation & Background:

Basketball is a beautiful, dynamic, and unpredictable game, which can never be fundamentally captured or described by numbers. However, quantitative data from basketball games, in the form of various countable statistical parameters, can be used to compare huge numbers of

performances over time to find patterns and trends. This practice, known as “analytics” in the sports world, is now widespread and used by every front office in every major professional sport. It is used to guide decisions from lineup optimization, to contract negotiations, to drafting. It can also be used to predict outcomes of games or seasons.

For this project, we are interested in building predictive models using machine learning with large NBA statistical datasets. The models we generate in this project will allow us to make informed predictions about the season-long performance of teams before a season starts, based on trends from past seasons. Additionally, we will be able to use recent game statistics to predict outcomes of individual games. Although not our primary motivation, this kind of model can be useful in sports betting, which is a growing field in the US. It can be utilized by users to inform their betting decisions by analyzing statistics in a systematic way. It can also be used by bookmakers to set betting odds. Additionally, in the process of building this model we will gain some insights on which team statistical parameters contribute most to predicting team performance. This can contribute to discussions about the quality of teams--for example, is it more important for a team to have good offense or good defense? Are rebounds more important than assists? This can have implications for how fans evaluate teams and players.

Finally, the practice of modeling and predicting outcomes of a complex real-life system through machine learning with specific quantifiable features translates naturally to other endeavors. Building models of things like weather/climate patterns and human bodies, we can use similar computations to predict outcomes that are more socially relevant than basketball. The easily observable and available stats and outcomes of basketball games make it a great training ground for model and feature selection.

Dataset:

- NBA-API: https://pypi.org/project/nba-api/
    - Retrieves various types of data, for example team game logs, or season-long advanced stats, from https://stats.nba.com/ and converts them to usable pandas dataframes.
    - For the creation of the game outcome prediction, we will primarily be using team game logs which contain statistics and information about individual games played by a team across a season, using nba_api.statistics.endpoint.TeamGameLog.

- Basketball Reference https://www.basketball-reference.com/
    - Download .csv format of tabular data, of “Team Per Game Stats” and “Team Ratings”, for a given team for a given season (eg. for 2010-11: https://www.basketball-reference.com/leagues/NBA_2011.html#all_team-stats-per_game; https://www.basketball-reference.com/leagues/NBA_2011_ratings.html)
    - Convert to dataframe using pandas.read_csv method

Challenge Goals:

- **Machine Learning:**
    1. Selecting the appropriate variables of team statistical metrics (points, rebounds, fg% etc.), cleaning/joining the datasets, and selecting features to improve the accuracy of predicting.
    2. Defining the best model in predicting the season standings
    3. Building a decision tree classifier to predict single-game win-loss outcomes

We were able to utilize what we learned about Decision Tree Classifiers and Regressors in the course, as well as feature selection, data splitting, and accuracy evaluation, on two different machine learning tasks. First, we used statistics as features to predict season standings through regression. Then we used running averages on a similar set of features to predict individual game outcomes. We were successful in applying and extending our understanding of Decision Tree machine learning models.

- **Messy Data:**
    1. Extensive historical data on nba statistics is available from its official website, but there is a lot of it, and not necessarily organized in the best way for us to retrieve
    2. We will use the NBA API library and carefully read its documentation to retrieve the data and convert it to pandas dataframes.
    3. We will also reconfigure some game log data to create moving window averages of statistical metrics over a stretch of recent games, to use as features for game prediction.

Data from the nba statistics website, stats.nba.com, is messy, extremely extensive, and not easily accessible. We learned to use methods from the NBA API library to retrieve highly specific data and convert it into easily-usable dataframes. We also used pandas methods to manipulate this data into new and useful data frames with relevant data to our investigation.

- **New Library:**
    1. We will use NBA-API https://pypi.org/project/nba-api/ to retrieve our data
    2. Methods are extensive, so sifting through documentation will be a crucial part of the work
    3. Library contains methods to directly transform tabular data to pandas dataframes

Before even using the library, we needed to learn how to install and use libraries, which was a challenging process because we had only used basic Python, or had coded within a previously-configured CSE 160 virtual environment. NBA API was difficult at first to learn because of our unfamiliarity with API libraries. It was also challenging to learn because it is an uncommon library, with no big tutorials on how to use it, so we needed to look carefully through the documentation. However, before too long we were able to effectively use the methods we needed, and moved on to more familiar territory. It was a useful lesson on APIs in general, and how to learn to use new, obscure libraries.

Methodology:

Season Standing Prediction Model:

As more datasets give more accurate results, we decided to use the last decade of NBA season for this model's training features since we believed the previous decade represents basketball's modern era. The first step is to join two relevant data frames from basketball-reference.com: 'Team Per Game Stats' and 'Season standings.' After the two datasets are joined, the dataset needs to be cleaned so that there are only relevant variables for the analysis: Field Goal Attempt (FG%), Field Goal %(FG%), 3 Point Attempt(3PA), 3 Point %(3P%), 2 Point Attempt(2PA), 2 Point Percentage(2P%), Free Throw Attempt(FTA), Free Throw Percentage(FT%), average Offensive Rebound(ORB), average Defensive rebound(DRB), average Assists, average Steals(STL), average Blocks(BLK), average Turn Over(TOV), average Personal Fouls, average total points per game (PTS). These variables are chosen based on their relevance to fundamental variables in evaluating a team's offense and defense.

After the dataset is cleaned, we will import 'DecisionTreeRegressor' from 'sklearn.tree' and use it as a model to predict the standings: mentioned fundamental variables above will be features, and the Rank variable from 'Team Ratings' will be a label. As these training datasets are from the 19-20 NBA season, instead of splitting the dataset, we will test it on a new dataset from the on-going 20-21 NBA season.

After the rank of the 20-21 season is predicted, we hope to compare with the actual 20-21 standings and see how much they differ from our predicted rank.

Predicting outcomes of individual games based on recent game averages:
We begin by writing a function, taking in a team, season, and window size *w*, to retrieve a particular team's game log data from one season, and turn it into a pandas dataframe with rows as games and columns as statistics from each game. We will create a copy of the game log dataframe, and turn the values in columns for each numbered statistical category for each game (after the *w*th game) into a rolling window average of that statistic for the previous *w* games. We will also change the wins, losses, and win percentage columns to reflect those values just for the past *w* games. Finally, we will convert the game date columns to pandas datetime, and add a column containing the number of days since the last game played, for each game. The function will return this manipulated dataframe, with running averages and win-loss records for a rolling window of *w* previous games, along with number days since the last game, for each game.

We will write another function which will take each of these newly computed features in the manipulated dataframe to be used as features to train a Decision Tree Classifier with the scikit-learn library. For each game, the features will be all of the recent running-average stats. The labels will be the game outcome--'W' for wins, and 'L' for losses. We will split the data into 70% training data and 30% testing data. The resulting classifier will be evaluated based on its training and test accuracy.

To test the model and possible parameters, for each parameter we will test the model's performance on 5 different team-seasons, subjectively selected across a range of team

success, playstyle, and year (after 2001, the year of the last major rule change). When testing each parameter, 10 models will be generated for each team-season, for a total of 50 testing models. We chose to create 50 models to average results when testing each parameter because we found that the variability in this model's accuracy is quite large, and 50 strikes a reasonable balance between sample size and computational intensity.

There will be a function which evaluates the accuracy of the Decision Tree Classifier based on a range of values *w* for window size. We will create the 50 models for each window size (without specifying maximum depth), and take the average training and testing accuracies for each size to account for variance in model fitting. This will tell us the optimal window size to use for this model, based on which window size provides the best testing accuracy (from 1 to 20 games, because anything more than that would cut out too much of the 82-game NBA season). Then we will use the determined optimal window size to test out a good tree depth for the model, which does not underfit or overfit the training data. Again, we will generate the 50 models at each maximum tree depth, between 1 and 10, to compare the average training and testing accuracies at each depth. After optimal window size and tree depth are found, we will use scikit-learn's "feature_importances_" method on the Decision Tree Classifier to assess average importance of each statistical feature through the 50 generated models, using the optimal window size and depth. This should give us a good indication of which team statistical parameters, in recent games, are the best indicators to predict game outcome.

For window size and tree depth, we will use matplotlib to generate line charts comparing the test and training accuracy for the tested range of parameter values, to visualize how model performance changes with these parameters. We will also create a bar chart showing the average importance of each statistical feature across generated models, to visualize the most important features.

Schedule Difficulty Effect Evaluation:

For this research case, our methodology is similar to the methodology described in *Predicting outcomes of individual games based on recent game averages*. However, it is slightly different in that we will take multiple teams into account because different teams may be affected differently by multiple back-to-back games. By analyzing more teams, we can average out the effect and results of having multiple back-to-back games and better analyze the game outcome of a team that has back-to-back games. So to start we can use the function from the methodology above that takes in a team, season, and window size, in order to retrieve a particular team's game log data from one season. Since that function then converts this one season game log data into a pandas dataframe, we can use this function on multiple teams to retrieve data frames of multiple teams. We can then concatenate these data frames from different teams in order to look at the strength of each schedule for each team.

Since the previous function has a variable for pandas datetime and contains a column containing the number of days since the last game played, for each game. We can then write a function to take in this combined data frame and filter out all data that we don't need and also

count the number of back-to-back games per season per team as well as record the result of the game, if their previous game was directly the day before.

Finally we can compare the number of back-to-back games that different teams had in a season as well as the probability of them winning a game if their previous game was the day before.

Results:

How can we use metrics to predict season standings?

By using our methodology, the initial results for this model wildly varied whenever we predicted the standings. Therefore, we augmented our approach by looping the predictions multiple times and returning the average of the prediction standings. This is very similar to getting odds of the head when tossing a coin. Even though we expect 50% for getting heads or tails when flipping a coin, we sometimes end up getting 0 heads or tails out of 10 flips. However, when we flip a coin 100 or 1,000 times, the probability of getting heads or tails ends up near 50%(See figure 1).

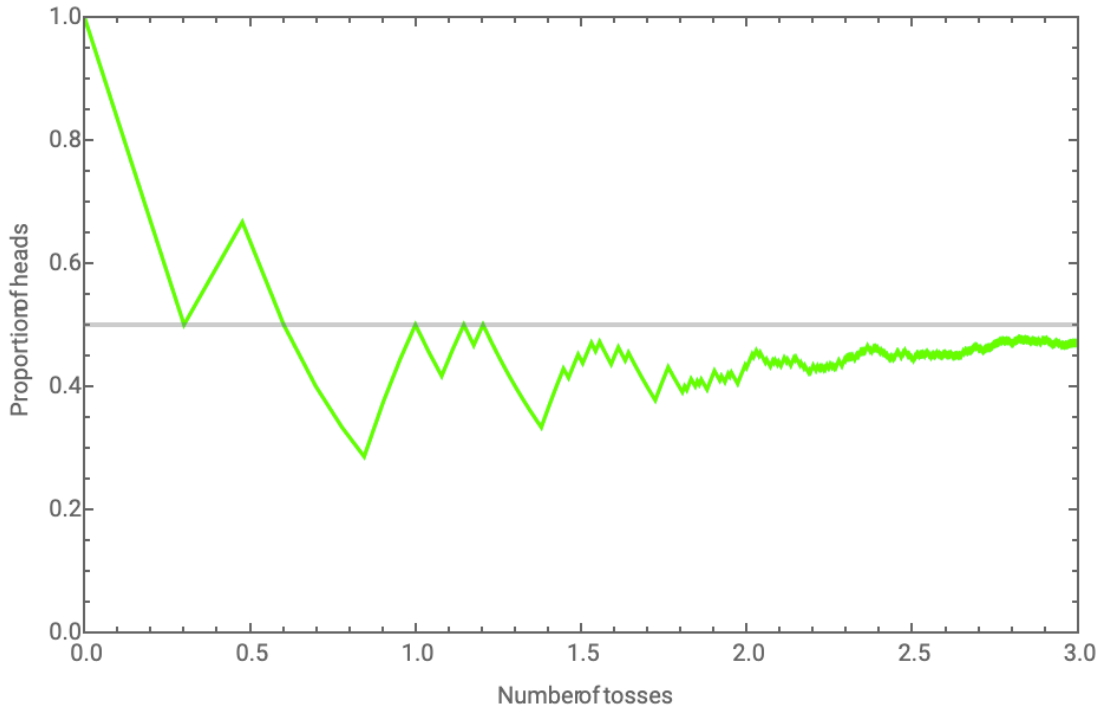


Figure 1: Simulated Coin Tossing Experiments and the Law of Large Numbers (Wolfram Demonstrations Project, Ian McLeod)

After it was revised, we made our input value 10,000 for how many times the prediction is going to be looped, and the result did not vary much like our initial results before.

	Rank	Team	Rank Score	difference
12	1	Dallas Mavericks	1.22	12
18	2	Chicago Bulls	1.43	17
5	3	Phoenix Suns	1.67	3
1	4	Brooklyn Nets	1.91	-2
0	5	Utah Jazz	2.34	-4
15	6	Charlotte Hornets	2.35	10
7	7	Denver Nuggets	3.00	1
21	8	New Orleans Pelicans	3.00	14
6	9	Milwaukee Bucks	3.00	-2
3	10	Los Angeles Lakers	5.57	-6
9	11	Boston Celtics	5.72	-1
24	12	Sacramento Kings	6.44	13
10	13	Golden State Warriors	6.62	-2
19	14	Indiana Pacers	7.15	6
17	15	Atlanta Hawks	8.51	3
20	16	Memphis Grizzlies	9.00	5
4	17	Philadelphia 76ers	9.00	-12
8	18	Portland Trail Blazers	10.59	-9
16	19	Toronto Raptors	10.63	-2
2	20	Los Angeles Clippers	10.78	-17
13	21	Miami Heat	11.39	-7
26	22	Orlando Magic	13.98	5
29	23	Minnesota Timberwolves	15.00	7
14	24	San Antonio Spurs	17.43	-9
11	25	New York Knicks	18.16	-13
25	26	Washington Wizards	18.19	0
22	27	Oklahoma City Thunder	21.61	-4
27	28	Houston Rockets	22.33	0
23	29	Cleveland Cavaliers	24.47	-5
28	30	Detroit Pistons	27.00	-1

Figure 2: Predicted standings by average rank and the difference between the actual standings.

However, we noted that the prediction result had a difference from the actual current standings. Our group might have done a better job improving the accuracy by training more advanced metrics, but we also thought it could be meaningful to see how accurate it can be for a trained model with only fundamental basketball variables.

As the table was not easy to perceive the difference with the actual standings, we visualized the outcome with a bar chart (See figure 3).

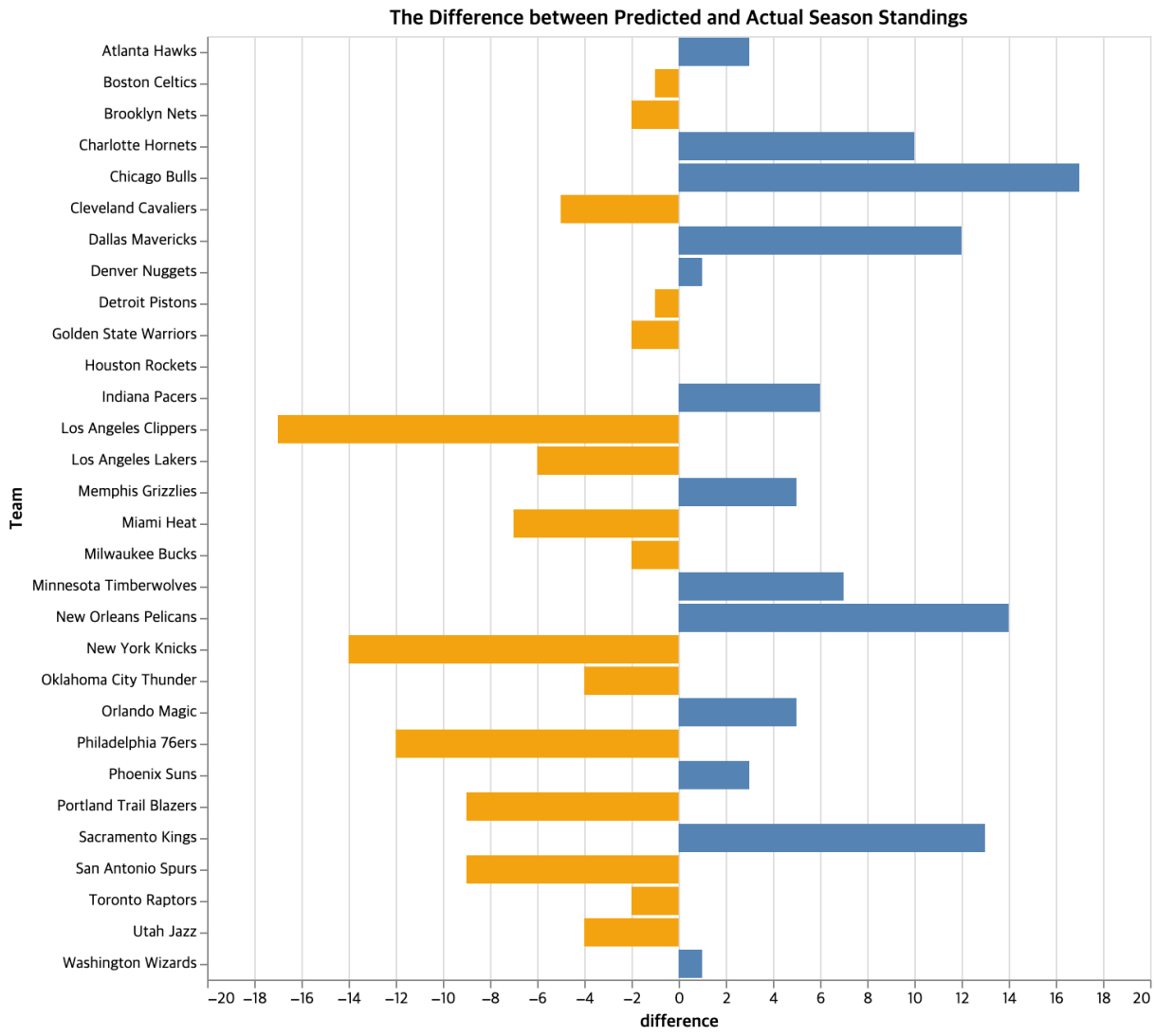


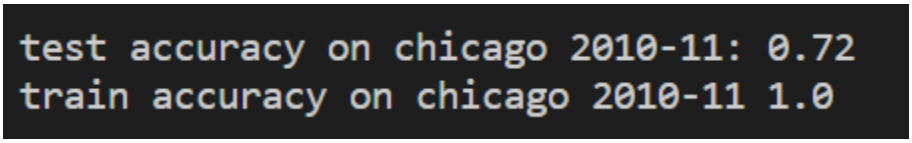
Figure 3: Difference between predicted and the actual 20-21 season standings.

The bars in steel blue color show how much a team jumped from the actual standings. And the bars in orange color show how much a team fell in the prediction. By this bar chart, we are easily able to notice teams that were predicted accurately or inaccurately by the length of the bar.

How can we use recent game team performance to predict the outcome of a game? What are the most effective statistical features for predictors?

In our implementation, we decided to use a rolling average of recent game performance across a range of common team statistics to train a Decision Tree Classifier to predict the outcome of games, given the relationship between that team's performance and outcomes in other games of that season. For this task, we evaluated two primary model parameters, maximum model depth, and the sample size of recent games of the team to use as input features. In addition to

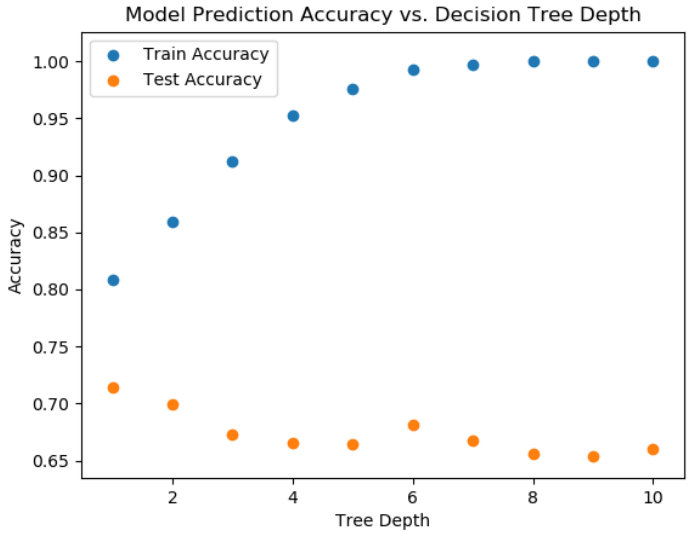
this, we evaluated the most important statistical features when training these trees. For each evaluation, our indications of model performance were training and testing data accuracy in predictions, with emphasis on testing accuracy.



Evaluating a single predictive model for games of the 2010-11 Chicago Bulls

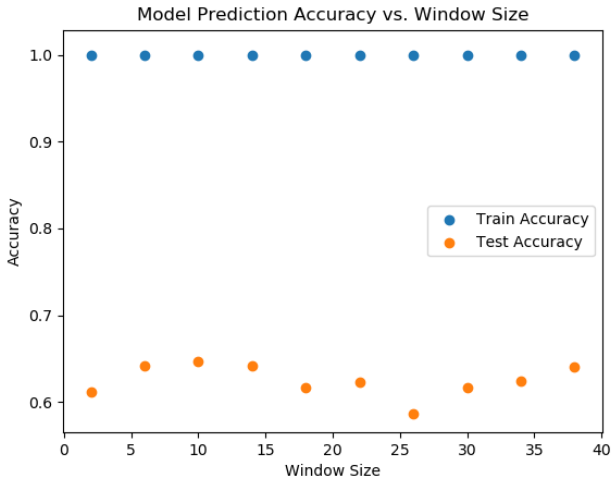
When tested on single season datasets, the testing accuracy of the model varied wildly. In the case of the Chicago Bulls' 2010-11 season, using 10 previous games' averages as inputs and unlimited maximum tree depth, model test accuracy ranged from 0.48 to 0.78, after running the model 20 times, with a mean of 0.69. With the same parameters, the training accuracy for these models was invariably 1.0, possibly indicating overfitting. Interestingly, the model had much higher testing accuracy on datasets of teams which won or lost games at an extremely high rate, such as the 2015-16 Golden State Warriors (73-9), or the 2015-16 Philadelphia 76ers (10-72). In order to visualize the structure of one of these trees, we plotted sample trees. Unfortunately we did not have the proper libraries to effectively visualize the trees, so the nodes themselves were not labeled with comprehensible feature names. However, we were able to assess that when allowed to go to maximum depth, the trees tended to generate 6-9 layers.

After verifying that the model worked and actually generated prediction labels, we assessed the best maximum tree depth for the model to avoid overfitting. When comparing performance of 150 total models on a diverse game log dataset with recent game sample window size 10, we found that test accuracy actually peaked at a maximum tree depth around 1 or 2, slowly decreasing with increased depth, and peaking again at 6. Train accuracy increased steadily by depth until reaching exactly 1 at depth 8. In order to stay true to the original intention of the study, to assess the predictive power of multiple statistical features, we continued to assess parameters in the study while allowing the model to go to unlimited maximum depth.



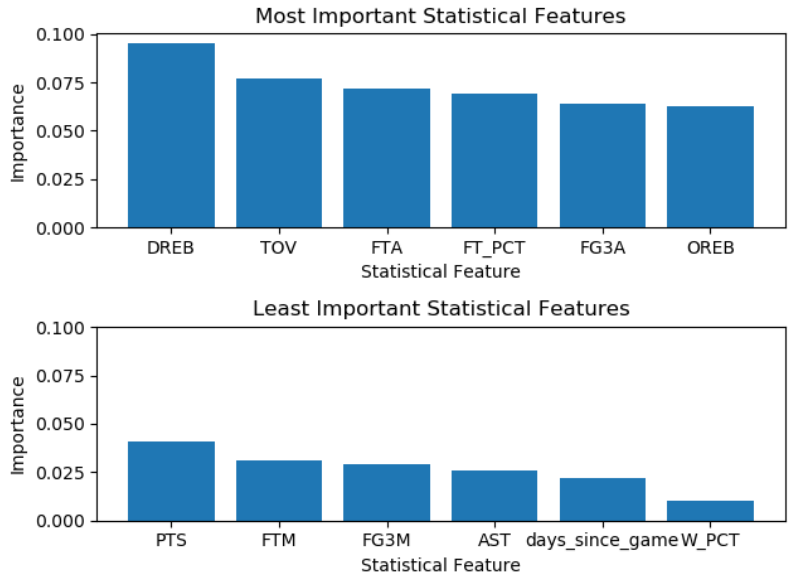
Scatter plot of train and test prediction accuracy of the model across maximum tree depth parameters. Accuracy is shown on y-axis, and tree depth on x-axis.

The next result of our study was based on assessment of optimal recent game sample “window” size. When allowing models to go to unlimited depth, we assessed the performance of 50 models in test and train accuracy across window sizes from 1 to 39 on a diverse game log dataset. We found that test accuracy increased moderately from 1 to 10 games, then decreased slowly on the way to 25 games, then increased again from there to 40 games. Coincidentally, 10 games, our starting window size, was actually the most accurate size that could also preserve most of our dataset. Train accuracy did not stray from 1 throughout the window sizes.



Scatter plot of train and test prediction accuracy of the model across maximum tree depth parameters. Accuracy is shown on y-axis, and tree depth on x-axis.

Finally, we assessed the importance of the features across 1500 models from our diverse dataset, generated with parameters unlimited tree depth and window size 10. We plotted the 6 most important and 6 least important statistical parameters across these models, and found that the most important statistical features for this predictive model were, in descending order, defensive rebounds, turnovers, free throw attempts, free throw percentage, 3-point field goal attempts, and offensive rebounds. The least important were points, free throws made, 3-point field goals made, assists, days since the last game, and win percentage over the past 10 games. These results were somewhat surprising, because things like points and assists are valuable parameters for assessing team performance. However, it is important to keep in mind that the importance of features here only represents the predictive power of recent performance in these categories, relative to a team's own performance, for predicting an upcoming game. Therefore, it doesn't mean that defensive rebounding is more important than points in the NBA overall, but rather that a team which has been rebounding very well recently compared to its performance over the season is more likely to win the game. As an aside, we assessed this same graph with varying window sizes and maximum depths--despite important differences, we always found the same stats at the top and bottom of the list.



Bar chart of importance of statistical parameters across 1500 Decision Tree Classifier models. Relative importance is shown on y-axis, and feature abbreviation is shown on x-axis.

To summarize, we found that the model was decently accurate in predicting game outcomes, especially for particularly good or bad teams. The ideal decision tree depth was either 1, or unlimited, though it did not make a large difference in accuracy. The best window size was either 10 or 39, with 10 allowing for more games to be included across a season. And we found that across parameters, recent average performance in defensive rebounds, turnovers, and free throw attempts were the most important predictive features for a team's upcoming game, while assists, days since the last game, and recent win percentage were least important.

Finally, for our user-friendly predictor, which scrapes data from the nba website to make predictions for upcoming games, I only evaluated it at a surface level. Out of all the games on March 11, only one of them, Hawks vs Raptors, had different outputs for both teams (for other games, the model either predicted wins or losses for both teams). The model predicted that the Hawks win, and Raptors lose, which was surprising because the Raptors are considered a much better team than the Hawks. Lo and behold, the Hawks won the game on a buzzer-beater. So at least from a surface-level assessment, the model is capable of making correct predictions. More importantly, it is easy to use and the code does not need to be updated during the season.

How does the difficulty of a team's schedule (number of back-to-back games) affect the winning probability or outcome of a game?

In our implementation of evaluating a team's schedule difficulty we decided to only focus on the number of back-to-back games. This is because in the NBA, the schedule of a team does not really differ from team to team. More specifically, a season usually consists of 82 games in a season and teams in the NBA will have very similar schedules. A team usually plays every team in the league at least two times. So we decided the difficulty of a schedule only really depends on the number of back to back games a team will have as it differs from season to season and team to team.

We specifically looked at the number of back-to-back games a team had in a season as well as the win percentage of those back-to-back games. We also looked at the win percentage of non back-to-back games in order to compare the two percentages and see if there is any correlation or difference between the two. Since win percentage is also highly dependent on the strength of the team regardless of if the game is back-to-back we decided to write a function to find the top 5 and bottom 5 teams in a given season based on win percentage and compare all back-to-back game stats between them in order to average out the effect of the strength of a team.

The results from our evaluation are surprising, as it depends on the season which we look at. For example, in the 2018-19 season, the last full season, back-to-back games did not really have a strong effect on the winning percentage. The average back-to-back win percentage for that season was 46.9% while the average non back-to-back win percentage was 47.3 % a minimal difference.

	Team	Season	Total_WinPct	Back_to_Back_WinPct	Non_Back_to_Back_WinPct	Num_Back_to_Backs
0	MILWAUKEE	2018-19	0.732	0.538462	0.768116	13
1	TORONTO	2018-19	0.707	0.750000	0.700000	12
2	GOLDEN STATE	2018-19	0.695	0.692308	0.695652	13
3	DENVER	2018-19	0.659	0.923077	0.608696	13
4	PORTLAND	2018-19	0.646	0.416667	0.685714	12
5	NEW YORK	2018-19	0.207	0.133333	0.223881	15
6	CLEVELAND	2018-19	0.232	0.214286	0.235294	14
7	PHOENIX	2018-19	0.232	0.250000	0.228571	12
8	CHICAGO	2018-19	0.268	0.357143	0.250000	14
9	ATLANTA	2018-19	0.354	0.416667	0.342857	12

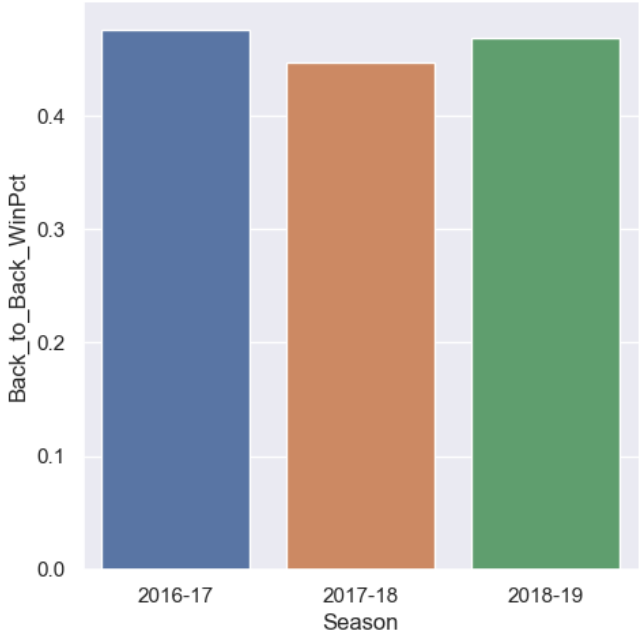
Average Back to Back Win Percentage 0.4691941391941392
Average Non Back to Back Win Percentage 0.47387813399207834
Average Number of Back to Back Games 13.0

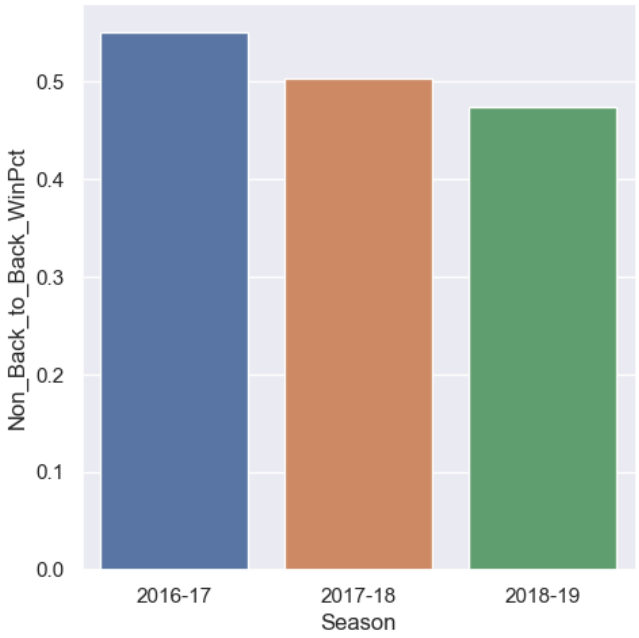
However in the 2016-17 season, back-to-back games definitely had a stronger effect where the back-to-back win percentage was 47.6% and the non back-to-back win percentage was only at 55.1%.

	Team	Season	Total_WinPct	Back_to_Back_WinPct	Non_Back_to_Back_WinPct	Num_Back_to_Backs
0	GOLDEN STATE	2016-17	0.817	0.764706	0.830769	17
1	SAN ANTONIO	2016-17	0.744	0.733333	0.746269	15
2	HOUSTON	2016-17	0.671	0.812500	0.636364	16
3	BOSTON	2016-17	0.646	0.588235	0.661538	17
4	CLEVELAND	2016-17	0.622	0.388889	0.687500	18
5	BROOKLYN	2016-17	0.244	0.071429	0.279412	14
6	PHOENIX	2016-17	0.293	0.357143	0.279412	14
7	LOS ANGELES	2016-17	0.622	0.500000	0.656250	18
8	PHILADELPHIA	2016-17	0.341	0.277778	0.359375	18
9	ORLANDO	2016-17	0.354	0.266667	0.373134	15

Average Back to Back Win Percentage 0.4760679271708684
Average Non Back to Back Win Percentage 0.551002284315772
Average Number of Back to Back Games 16.2

As we can see, the average number of back-to-back games per season also changed every season and it's possible that there is a correlation between the amount of back-to-back games a season and its effect on the win percentage of teams during those back-to-back games. Another interesting effect we can see is that different teams had different results with back-to-back games. Some teams really struggled with winning back-to-back games during the season, while some really did not struggle and tended to do better in back-to-back games. However from what I could see it was pretty random as both of these effects happened to both the top 5 and bottom 5 teams.





As we can see from the two visualizations, the win percentage difference between back-to-back games changes per season and we can see that there definitely is a correlation between win

percentage during back-to-back games and non back-to-back games. While it is only a slight difference, it is still significant enough to acknowledge.
To summarize, the effect of back-to-back games on win percentage is not super strong yet it is still significant enough to draw the conclusion that back-to-back games tend to lower the winning percentage of teams.

Testing:

Season Standing Prediction Model:

Other than functions returning visualizations and prediction models, the functions returning data frames were tested using asserts. Test_load_in_data uses loc to return a specific value of 'PTS', 'TOV', 'PF', 'Team' to match the value in asset_equals. Test_concatenate function finds a specific value from 'PTS', '3P%', FG%', '2PA' and returns the 'FGA' value. Then assert_equals from cse163_utils was used to match the value.

Predicting outcomes of individual games based on recent game averages:

For this section, we mainly were interested in testing the API retrieval game log data frames, as well as the moving-window average statistics for individual games. We made a separate script to test these two critical functions. First, we manually compared individual stats from the data frame generated for the 2019-20 Miami Heat season to stats found on basketball-reference. These were all in order, so we also tested the moving window averages. First, we compared the recent averages for a couple of individual stats (rebounds and assists) for a couple of games, with those averages computed manually by taking averages from 10-game slices prior to that game in the original game log. We also manually computed the average points stats, again manually from basketball-reference, of a 10-game stretch, and asserted equality with the points value of the game immediately after the selection.

Schedule Difficulty Effect Evaluation:

Again for this section, we were also interested in testing the API retrieval game log data frames as well as testing that the correct data frames from year to year were being merged correctly.

Citation

"Simulated Coin Tossing Experiments and the Law of Large Numbers" http://demonstrations.wolfram.com/SimulatedCoinTossingExperimentsAndTheLawOfLargeNumbers/ Wolfram Demonstrations Project Published: March 7 2011

Swar. (n.d.). NBA API - GitHub. Retrieved March 17, 2021, from https://github.com/swar/nba_api