Beating Vegas in NBA Predictive Modeling

Chapman Beaird, Josh Chambers, Mason Hamilton, Darren Skidmore

Abstract

The National Basketball Association, founded in 1946, is the most prestigious professional basketball league in the United States and the world. The NBA has risen in popularity over the last several decades thanks to transformative players like Michael Jordan, LeBron James, and Stephen Curry, amassing fans globally. As is the case with any major sports league, significant attention is given to advanced stats beyond the box score, which can be used to estimate which team will win any given game and by what margin. Optimizing these predictions can help paint a picture about which teams truly are the best and the worst, and if designed well enough, can even make a profit from betting markets.

1 Introduction

The intent of this project was to attempt to create a number of machine learning models that could predict the actual result of a point spread for NBA games. In all major sports leagues, betting is involved as an activity for people to test their intuition and luck against other sports bettors, follow games closely, and ultimately waste their money on a fruitless endeavor. Nevertheless, Vegas odds (the colloquial term for sports betting figures) are an interesting metric to gauge who might win a game and by how much. In this case, we are looking at classifying the point spread, which is used to estimate how many points a certain team will win or lose by.

Point spreads are written as a positive or negative number which can either be a full or half decimal (e.g., -8.5, 6.0, 12.5) describing *how many points is team X predicted to win by* (note that negative numbers correspond to a winning margin.) After a game ends, the true margin is compared to the predicted point spread and those who predicted the correct "side" will receive a payout, while those who guessed incorrectly essentially lose their money.

Point spreads are maintained by sports bettors, who are often expert statisticians with years of experience in the field. Therefore, this project is a bit unique because it is not viable to make a huge profit against the house; the objective is simply to get slightly above or equal to 50% accuracy. In our research, we found that 52.4% accuracy and above will result in a significant net gain in profit (Byrnes & Farinella, 2016)—thereby *beating Vegas*.

2 Methods

2.1 Data

We used two data sources to obtain our data. The first was Sportsbook Reviews Online which contained point spreads for almost all NBA games between 2007-10-30 to 2023-01-16. Although it is unclear why the data abruptly stopped in the 2023 season, we were still able to get pre-game point spreads for 19,807 games. The data were stored in the form of HTML tables. We used the Python library requests to download the HTML data and parsed the table into a dataframe using pandas. From there, we extracted the team names, dates, and listed point spread, exporting it to a CSV file.

The second data source was the NBA's official website, for which we used the Python library nba_api to quickly retrieve data in JSON format. We retrieved all games between 2004-11-02 to 2024-04-07. The data from this API included essential team attributes like the score, total rebounds, total field goals made, etc. and were henceforth exported to CSV.

From there, the two datasets were combined in one using a simple pandas left-join. A few more features were computed, including the actual point margin and the Elo score for both teams. There were a total of 44 features. If a regression algorithm was used, the float feature `HomeSpreadActual` was used as the target. If a classification algorithm was used, the boolean feature `HomeSpreadCorrectDirection` was used instead.

The final process involved adjusting the features for a given game to reflect the results of *previous games*. If you use the actual statistics of that current game, it defeats the point of predicting the victor. Therefore, we opted to utilize a hyperparameter vector that would weight results from a certain number of previous games and use that as the actual values. The default vector we selected was `[0.4, 0.3, 0.2, 0.1]` where index *n* corresponds to *n+1* games ago.

2.2 Elo Score

Elo is a statistical metric originally developed for chess that aims to quantify players or groups on their ability level. In a nutshell, an average Elo is 1500 with a standard deviation of around 100. All teams' Elo scores were set to a mean of 1500 beginning on 2004-11-02 and computed to 2024-04-07. In the event of the season ending, all teams were slightly regressed to the mean by 25%. This algorithm was adapted from FiveThirtyEight and was used as the primary feature for assessing a team's ability level (Silver & Fischer-Baum, 2015.)

$$E_A = \frac{1}{1+10^{(R_B - R_A/400)}}$$

$$E_B = E_A - 1$$

R_A: Current Rating of Player A

R_B: Current Rating of Player B

E_A: Probability of Player A Winning

E_B: Probability of Player B Winning

$$R'_A = R_A + K(S_A - E_A)$$

Fig. 1 — Formulae for computing Elo

2.3 Selected Models

We used the following models: K-Nearest Neighbors (classifier and regression), Decision Tree classifier, XBoost, and MLP Classifier. Each group member worked on one model. A variation of models was chosen to test a diversity of possible ways to solve this problem and to give more opportunities to reach a very high accuracy. We opted to use a train/test split of 0.8 and 0.2 respectively as those are considered industry-standard figures and changing them rarely results in any significant variation.

3 Initial Results

3.1 K-Nearest Neighbors

The K-Nearest Neighbors regression model was initially run with the default results weighting hyperparameter, a variation of k-values in `[2, 10]`, and all combinations of feature normalization and weighting (either distance or uniform).

We found that the best results occurred as k increased and there was no normalization nor weighting. This is a fascinating find because many might assume that normalization and weighting are automatically better. However, this shows that some values with higher absolute magnitudes (e.g., Elo, points scored) are more important to the model than values with lower absolute magnitudes. The best recorded MAE (mean absolute error) values were around 9.7, showing that the final scoring margin of a game was predicted incorrectly by an average of 9.7 points. This is however a slight improvement over the historical average margin of 11.2 points, demonstrating that some degree of learning is taking place.

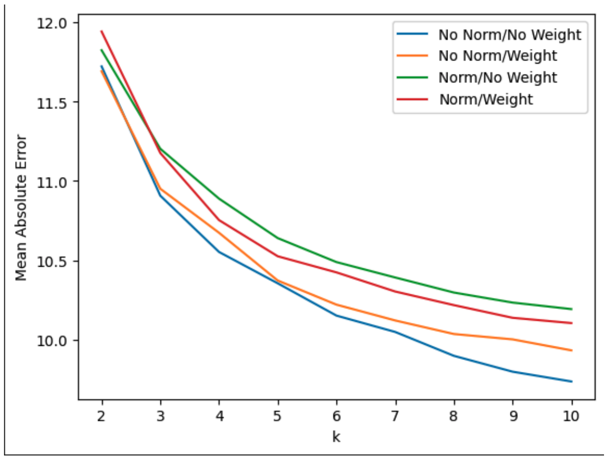

Fig. 2 — Initial MAE (mean absolute error) chart for K-nearest neighbors regression

The K-nearest neighbors classifier was then used to calculate the accuracy of predicting the point spread. Once again, all k-values in the range [2, 10] were tested both with and without normalization.

As was seen in the regression problem, the non-normalized data significantly outperformed the normalized data. The average accuracy for non-normalization was 51.3%. This is not enough to make a profit, but it is better than a random coin flip. The normalized data had an average accuracy of only 50.0%.

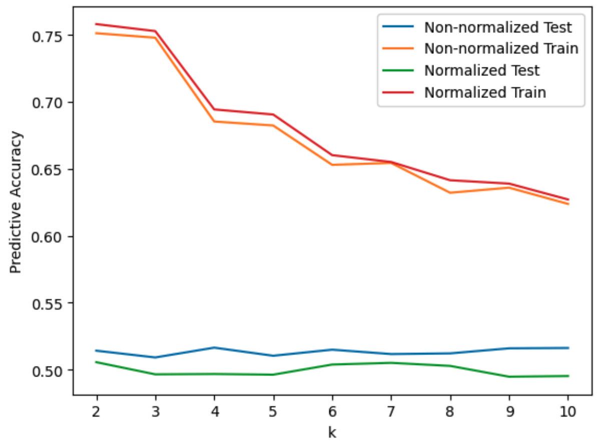

Fig. 3 — Initial accuracy chart for K-nearest neighbors classification

3.2 Decision Tree

As mentioned in 3.1, the use of normalized and or weighting caused the accuracy of our Decision Tree model to decrease. As a result, we used non-normalized and no weighting in our model.

When our Decision Tree model was first generated, we decided to stay within the bounds of the vanilla model to create a good base level in which to compare improvements and hyperparameters to. As a result, on the Decision Tree model with no adjustments, we achieved an accuracy of around 50.8% to 51.2%.

3.3 XGBoost

The initial application of the XGBoost model demonstrated somewhat promising capabilities. Using default settings, we got an accuracy of around 51% (only slightly better than random guessing). This was a solid foundation giving us much hope to what this model could yield.

The XGBoost model was also evaluated based on the confusion matrix. It appears to be the case that the distribution is almost identical. The difference between the false positives and true negatives is only one instance.

The F1-score we received stands at approximately 0.51 for both classes. This confirms that the model has a moderate ability to predict both classes without bias to either.

3.4 MLP Classifier

For initial tests of the MLP Classifier, the vanilla model with all of its default settings was used to get a basis for what the low end accuracy would be before parameters were tested and tweaked to bring up the accuracy. We found that, on average, this model would return an accuracy between 48.6% and 49.7% when it was run on new data. Training and test accuracies on the established data were generally between 50% and 52.4%.

4 Feature/Model Improvements

4.1 K-Nearest Neighbors

In order to try to improve our results, we decided to stick to non-normalized data without weighting and adjust other hyperparameters. We opted to use a GridSearch to find the optimum parameters.

After running a GridSearch with k=10, the optimal parameters were found to be the "auto" algorithm with p=4 and uniform weights. However, the accuracy dipped slightly to 51.1%.

We also attempted to adjust the rolling average vector to different values, including `[1]`, `[0.7, 0.2, 0.1]`, `[0.55, 0.35, 0.1]`, and `[0.16, 0.15, 0.14, 0.13, 0.12, 0.11, 0.1, 0.09]`. However, no changes in accuracy were observed, and the peak accuracy remained 51.6%.

4.2 Decision Tree

As discussed in 4.1, our Decision Tree results were derived from non-normalized data without weighting. We ran GridSearch to find the optimum parameters and the results were mixed.

When we first attempted to make improvements, we adjusted the max_depth to 3. Almost instantly we were able to see an increase from an average accuracy hovering around the 50.4-51% mark to an astounding 53.2% accuracy. In a game of inches, this means a lot.

To further test our model, we ran GridSearch multiple times, each with a slight variation of the previous parameters. Surprisingly, as with K-Nearest Neighbors, GridSearch did not improve accuracy and more often than not, decreased it.

We noticed throughout testing that accuracy was highly dependent on max_depth and came to the conclusion that this was the only parameter that noticeably affected our results.

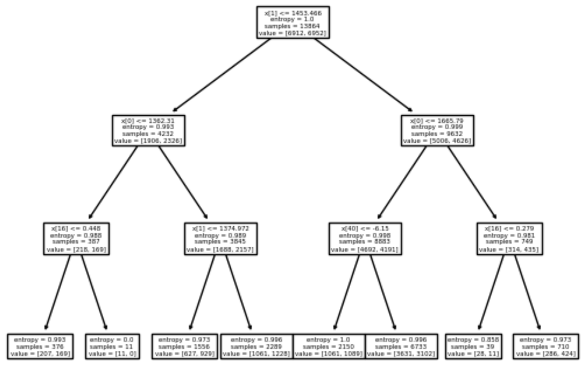

Fig. 4 — Decision Tree with depth 3

4.3 XGBoost

As discussed in previous sections, the results were derived from non-normalized data without weighting. Adjusting the XGBoost model involved fine-tuning the hyperparameters through GridSearch. We ran GridSearch on the XGBoost model with various different parameters to find the optimal parameters for this model.

The grid search was run with various different parameters like max_depth, learning_rate, etc. It fitted 5 folds for each of the 243 candidates which resulted in 1215 total fits. We found it to be the case that the best parameters were colsample_bytree = 0.7, learning_rate=0.01, max_depth=3, n_estimators=300, and subsample=0.8. It is interesting to note that the optimal max_depth is 3 which is the same optimal parameter found in our decision tree model. With these parameters, we found that the best score was about 54.1%. This is a pretty notable improvement from our base model with no improvements done on it which yielded an accuracy of around 51%.

4.4 MLP Classifier

Reiterating what was covered in the other sections, the results were derived from non-normalized data without weighting. When improving the MLP classifier, we focused on three parameters: early stopping, alpha value, and learning rate. Of the three, early stopping had the most impact on the models, with each iteration performed, on average, 2.3% better than when early stopping was not utilized.

While running several iterations of models with different values for alpha and learning rate we were able to determine that using a higher number than the defaults resulted in better results. Anything smaller than the default would usually result in overfitting. Our tests determined that an alpha value of 0.001 and a learning rate of 0.001 resulted in better overall accuracy, with the accuracy for new data sitting between 50% and 51.7%, with some outliers reaching up to 52.7%.

5 Final Results

5.1 K-Nearest Neighbors

Because GridSearch did not result in an improvement, it can be concluded that for KNN classification, most parameters do not significantly alter the accuracy aside from unnormalized data which has an undoubtedly better accuracy than normalized. Because accuracy was generally in the 51.1-51.6% range, the KNN model as designed in this way is unlikely to be effective for making a profit in the markets.

5.2 Decision Tree

Through our testing, we were able to conclude that, like mentioned in 5.1, unnormalized data performed drastically better than normalized and, as referenced in 4.2, max_depth was the only significant parameter in our model. To our surprise, with an accuracy in the range of 53.1% to 53.4%, our model should generate revenue if run on the market.

5.3 XGBoost

Through our testing and manipulating of parameters for the XGBoost model, we can conclude that the model demonstrated significant improvements. The final iteration of the model achieved an accuracy of 54.1% which surpasses the threshold of 52.4% generally required for profitability in sports betting markets.

5.4 MLP Classifier

After testing our model and implementing improvements to it, we were unable to consistently achieve the threshold of 52.4%. While the results were much better than the vanilla models, we aren't confident that it could produce accurate enough results to be an improvement over the current method for spreads.

Other parameters that were tested were the number of hidden layers as well as the momentum, but these were solely through observing iterations and would need to be more thoroughly tested before implementation. While current metrics are not within range, the results could be fine tuned with further research.

6 Discussion and Conclusion

Our models gave us good insight into the underlying data as well as the nature of the field. As noted in Section 5, while XGBoost and Decision Tree produced superior results, all models performed with over 50% accuracy. We found this interesting because it shows that AI Models do work. They produce results that, although may be profitable in our case, beat 50/50 bets. On that note, we also learn that some models perform better than others on specific datasets. XGBoost and Decision Tree did so in our case. This may be the case for a number of reasons, but some of which may be that they are non-linear models, less influenced by outliers, and regulations to help prevent overfitting.

With great power comes … great risk. As much as we want to employ the Decision Tree or XGBoost models and make our millions, it seems a little too good to be true. For this reason, we have not taken it to the market.

7 Future Work

The most glaring absence in our models is using feature reduction methods such as PCA to identify which features are the most impactful on the learning process. Based on the large difference of the performance between un-normalized and normalized data, some features with greater absolute magnitude are clearly among the most important (as they are exemplified in the case of un-normalized data.) Feature reduction may help reduce attributes of lesser importance and lead to accuracy even beyond what we have obtained.

We would also want to consider trying other models such as an MLP regressor and Naive Bayes classifier to gain an even greater scope into the understanding of how our data interacts with various models.

References

Byrnes, T. T., & Farinella, J. A. (2016). The effect of momentum on the NBA point spread market. The Sport Journal, 19, 1-9.

Silver, N., & Fischer-Baum, R. (2015, May 21). *How we calculate NBA Elo ratings*. FiveThirtyEight. https://fivethirtyeight.com/features/how-we-calculate-nba-elo-ratings/

Appendix A Attribute List

Using default rolling average hyperparameters — values do not significantly vary with other settings.

	MAX	MIN
ELO_AWAY	1849.6679	1184.2628
ELO_HOME	1850.2002	1186.7542
MIN_AWAY_RA	291.8	171.6
MIN_HOME_RA	287.9	171.4
PTS_AWAY_RA	145.4	72.5
PTS_HOME_RA	141	71.9
FGM_AWAY_RA	53.2	23
FGM_HOME_RA	52.8	24.3
FGA_AWAY_RA	107.2	61.5
FGA_HOME_RA	105.1	65.8

FG_PCT_AWAY_RA	0.5881	0.3193
FG_PCT_HOME_RA	0.5926	0.3357
FG3M_AWAY_RA	23.6	0.8
FG3M_HOME_RA	23	0.5
FG3A_AWAY_RA	62	4.2
FG3A_HOME_RA	56	3.5
FG3_PCT_AWAY_RA	0.6522	0.0972
FG3_PCT_HOME_RA	0.6393	0.0686
FTM_AWAY_RA	36.1	6.7
FTM_HOME_RA	36.5	6
FTA_AWAY_RA	45	8.7
FTA_HOME_RA	48.8	7.5
FT_PCT_AWAY_RA	0.9619	0.4609
FT_PCT_HOME_RA	0.9802	0.4451
OREB_AWAY_RA	24.2	3.1
OREB_HOME_RA	22.9	3.5
DREB_AWAY_RA	50.1	19.6
DREB_HOME_RA	49.5	18.7
REB_AWAY_RA	59.7	27.3
REB_HOME_RA	59.5	27.1
AST_AWAY_RA	39.5	10.9
AST_HOME_RA	38.8	11
STL_AWAY_RA	15.3	1.7
STL_HOME_RA	15.8	2.1
BLK_AWAY_RA	13.1	0.4
BLK_HOME_RA	12.5	0.5
TOV_AWAY_RA	24.4	5.6
TOV_HOME_RA	24.5	5.7
PF_AWAY_RA	32.5	11.4
PF_HOME_RA	33.2	10.8
PLUS_MINUS_AWAY_RA	34.4	-35.4
PLUS_MINUS_HOME_RA	38.9	-33.9
HomeSpreadActual	58	-73
HomeSpreadCorrectDirection	1	0

Appendix B Source Code

https://github.com/darrenrs/cs270-nbaproj