Predicting Outcomes of NBA Games Through Averaged Player Statistic Models

Abstract

Predicting the outcomes of sports games is an interesting topic, due to the unpredictable nature of these games. A good model must properly leverage historical data and avoid overfitting to upsets in order to accurately predict future games. For this project, we set out to beat the existing prediction systems for National Basketball Association (NBA) games. Scraping the data of 2000 NBA match-ups, we trained on an input of player statistics and an output representing the outcomes of the games. We tested our data over hundreds of runs on an SVM, Naive Bayes, and Neural Net, and ultimately found that the Neural Net best performed on our data, both in accuracy and in consistency. We also discovered that certain features better generalized outcomes than others, and after training an optimized feed-forward neural net on a data set of these features, we were able to successfully reach a maximum of 78.3 percent accuracy on future predictions.

1. Introduction

1.1. Problem and Motivation

The inspiration for this project was in attempting to beat the prediction accuracy of other existing automated sports game predictors. Many of these services are provided for a fee, and in this regard achieving a higher accuracy would mean achieving a more valuable product. Another challenge we set for ourselves was in finding whether we could judge confidence in our predictions based on the closeness to the binary outputs.

After considering the data available to us, we chose to focus on NBA data. The main reason for this is that there are 82 games a year, per team, so we have access to a very large number of data points. Similarly, there are many player statistics to choose from with which we could make the training more accurate. Finally, because the 2015 season is occurring currently, we could test predictions on future games and evaluate those results.

University of Pennsylvania, CIS 419/519 Course Project.

1.2. Numbers to Beat

One such automated system that we aimed to compare our predictions to was TeamRankings (TeamRankings, 2015). Over the previous 1000 games, TeamRankings best accuracy currently stands at 58.7 percent, so we set a goal to beat this number. Further, we wanted to compare our results to those of experts who take into account the less statistical factors of sports, such as streaks and home team advantages. One prediction website, Covers Experts (CoversExperts, 2015), reports that their top three most accurate NBA experts for the previous 30 days achieved individual accuracies of 76, 60, and 59 percent. Together these experts achieved an average 65 percent accuracy.

2. Gathering Data

Our first challenge was to find a reliable source of data for NBA statistics. There were several websites that provided such statistics, but very few provided data available for download. We decided to use the SportRadar API (SportRadar, 2015) because it allowed us to download data in JSON format. Using the API, we wrote a scraper that downloaded seasonal performance averages by team, player performances by game, game outcomes, and end of season rankings. Ultimately, we scraped data for more than 2000 played games, and created a local database from these JSON objects.

3. Techniques & Data Set Manipulation

3.1. Organizing data

The first thing we did in order to better organize our data was to sort all of the games chronologically in folders by team. Using pure JSON formatted data for the neural nets was non-intuitive, so we decided to write wrapper functions that would load the data from our database and convert them into Numpy arrays. This would allow us to get vectors representing the performance of each player in a game, as well as the players average performance over a season. Data is returned from the loader as Numpy arrays, which are then concatenated in another helper function. This allowed us to decide the amount and type of data to use outside of the loading step.

3.2. Deciding what data to use

For the input to the neural net, we decided to create a vector concatenating the performance statistics of five home players and five away players. Each player was represented by a 26 entry performance vector, including values for free throw percentage, fouls, etc. We chose five because at least five players are guaranteed to play in a given game. Additionally, since different teams play different numbers of players, this enabled us to guarantee that the input vector was always of the same length. In order to get the most significant players for the input vectors, we chose each teams' five players with the most play time. The output of the net was a representation of the outcome of the game, which we experimented on.

3.3. Training on Score Differential vs. Binary Outcomes

The two representations of outcomes we chose to experiment on were binary outputs and score differentials.

Testing on the score differentials, we ran into the issue of a very high cost value over iterations, with little decrease, due to the large variability in potential outcomes. We tried to normalize these outputs, but in the case of the feed-forward neural network, non-normalized outputs were still consistently more accurate. On the other hand, the binary outputs had a much lower cost, and was capable of decreasing to a much smaller fraction of its initial cost.

After many tests over the shuffled data sets, we found that training the neural nets on binary outputs averaged an approximate 3 percent increase in accuracy over score differentials. This is likely because the score differential metric includes close games that would have been labeled as absolute 1/0 in the 1/0 system. These close games could make it hard for the classifier to accurately predict outcomes of similar future games.

Overall, because of the higher accuracy and larger decrease in the cost, we decided to go with binary outputs.

3.4. Creating the N parameter

Initially, we trained only on the game day statistics, meaning that the classifier would base a teams outcome directly on how they played that day. We made the ideal assumption that, by looking at previous games, we would be capable of generating a model for the testing data that could represent approximately how they would perform that day. However, we found that game day statistics were training our classifiers to look only for score-based statistics, and so we determined that we would need to adjust what data we provided.

After these realizations, we shifted to both training and testing on the average player statistics for the previous N games, where N was a parameter we optimized. In doing this, we were capable of training the classifier to look for consistent patterns of how a team's recent performances affected the outcomes of their future games.

4. Deciding on Our Classifier

4.1. Neural Network

Our first intuition was that a feed-forward neural network would best perform on the data we had. This is because a neural network is extremely flexible in its hidden layer structure, which would help to reduce the impact of irrelevant features within our instances of many data points. We chose to use the PyBrain (Schaul et al., 2010) library to create a neural network that trains on our input vector and output data. On early experimentation, the neural net achieved accuracies in the mid-60s.

4.2. Naive Bayes

In order to compare our neural network results with other classifiers, we tested on a GaussianNB classifier from Sklearn (Pedregosa et al., 2011). The performance of the NB classifier was on average much worse than that of the neural net. Additionally, the NB would occasionally get accuracies below 50 percent, something that did not often occur in the neural net. This classifier also occasionally guessed all 0s, or all 1s when trained using the score differential, approximately $\frac{1}{3}$ of the time, making this classifier highly unreliable.

4.3. SVM

As another comparison, we chose to test the Gaussian Kernel SVM from Sklearn. On average, it performed at about the same level as the feed-forward neural net, but would often have a wider range of values from the same settings. This high variance made it less reliable than the neural net in the end as it would output entirely different predictions, even when the parameters remained the same.

4.4. Conclusion

Ultimately, we determined that our feed-forward neural net was, in fact, the most consistently accurate on our data set. While the SVM was occasionally as accurate as the Neural Network, it also achieved much lower accuracy at times. The Naive Bayes was not significantly more accurate than a random guess.

Table 1. Classifier accuracies over six runs on shuffled data set

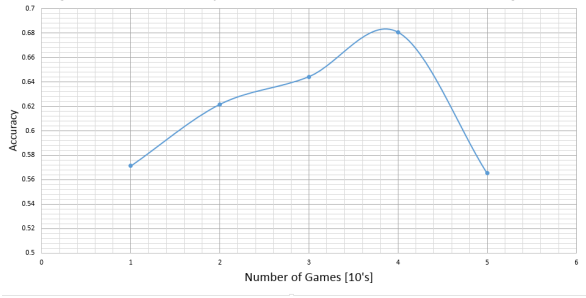
NEURAL NET	NAIVE BAYES	SVM
.73	.54	.66
.60	.54	.66
.78	.53	.70
.67	.51	.67
.68	.58	.60
.72	.56	.65
.69	.55	.66

5. Parameter Improvements

5.1. Optimizing N

Experimenting with the N parameter, representing the number of games we averaged on, we found that there was an optimal N value of 40. Below and above this value, the accuracy decreases. This is because a low N is not capable of finding hot and cold streaks or larger trends in the players' statistics. On the other hand, too high of an N overgeneralizes on too many previous games, by getting approximately the same input vector for every outcome. The figure below demonstrates these facts, with each data point representing the average accuracy over 8 runs for that N value.

Figure 1. Accuracy Vs. Historical Games Leveraged



5.2. Optimizing Features Used

Another issue we found was that there were many statistics fields that appeared irrelevant to the outcome of a game, but still had strong weights and were affecting the predictions. To solve this, we experimented with many different subsets of statistics. Ultimately, we found the best predictors were percentage based. This makes sense because these statistics determine how well a team performs as a whole, rather than how many points they scored, or other statistics which are also reliant on the opposing team. In effect, this avoids overfitting on the scores of previous games, and best generalizes for future games.

Figure 2. Chosen Features

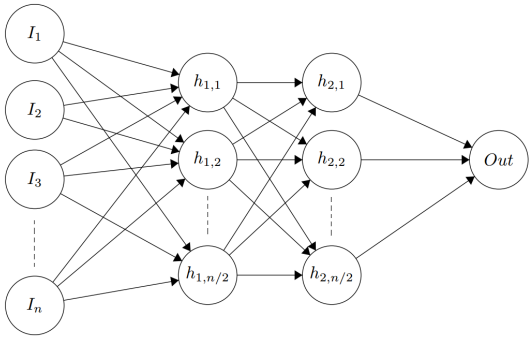
three_points_pct	player_tech_fouls	offensive_rebounds	two_points_att
team_rebounds	field_goals_att	field_goals_made	defensive_rebounds
fast_break_pts	rebounds	personal_fouls	two_points_pct
free_throws_made	steals	blocks	three_points_made
paint_pts	turnovers	field_goals_pct	flagrant_fouls
free_throws_att	coach_tech_fouls,	points_off_turnovers	assists
team_turnovers	assists_turnover_ratio	blocked_att	free_throws_pct
points	three_points_att	two_points_made	second_chance_pts

5.3. Optimizing Neural Net

5.3.1. HIDDEN LAYERS

Along with the number of games we averaged, we tested different shapes of neural nets and including a momentum term. Our early neural nets attempted shapes with multiple hidden layers of two times the input vector and then a second layer one half of the size of the input vector. However, we found that larger neural nets converged faster but they tended to overfit, reducing accuracy. We eventually determined that a neural net with two hidden layers worked best, with each of the middle layers half the size of the first layer. We believe that squeezing the inputs into smaller hidden layers increases accuracy because it prevents the neural net from simply setting the weights corresponding to trained game outcomes.

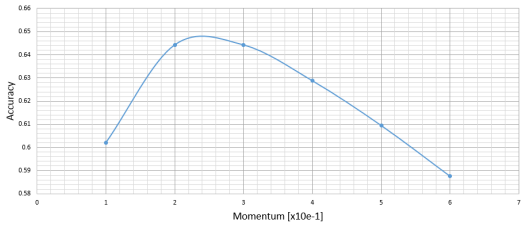
Figure 3. Optimal FFNN Structure



5.3.2. MOMENTUM

Another parameter we tested on was a momentum term. One of the issues that we were having was that the cost function seemed to get stuck in certain areas where it would oscillate between values. Testing different momentum values, we found that with a momentum of between 0.2 and 0.3, the neural net was able to move past these early areas where the cost function would previously get stuck. Too large of a momentum, however, led to more sporadic jumps in cost, often leading to a larger cost in the end, and decrease in accuracy.

Figure 4. Accuracy Vs. Momentum



6. Algorithm

Algorithm 1 Final Algorithm Pseudocode

Get all GameIDs
for each game **do**
  Create instance point by averaging previous N games on relevant features
  Insert instance point into averaged game data matrix
end for
Shuffle the averaged game data
Split into training and testing partitions
Standardize training data
Train neural net until max epoch or convergence met
Standardize test data with training data mean and standard deviation
Predict on each test data instance
Normalize predictions and round to 0 or 1
Run $sklearn.accuracy\_score$ on predictions and true outcomes

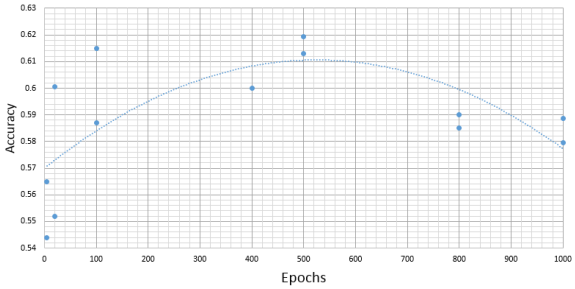
7. Results

7.1. Accuracy Analysis

Overall, we received a maximum accuracy of 78.3 percent, with an average result of 69.17 percent on our neural network runs. This result is largely above the 58.7 percent accuracy of the Team Rankings system. Its consistency over all runs indicates that there is a clear significance to the statistic features we are providing, offering up to a 20 percent increase in insight towards future games. Similarly, we successfully beat the accuracy of the experts by up to 9 percent, demonstrating that using historical data alone is capable of beating the predictions of human experts versed in the field.

One issue we did find from our trials, however, was that the cost would not go below 0.10. We believe that this is due to the unpredictability in upsets and the effect of their outputs on games of similar input statistics. This could potentially be improved upon by acquiring more game instances to reduce the impact of upsets. This was a limitation with our means of acquiring data, as our source did not allow us to acquire data prior to 2013.

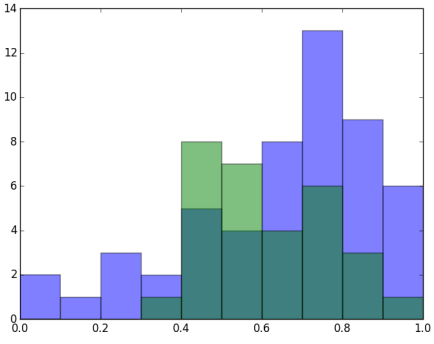
Figure 5. Accuracy Vs. Epochs



Looking at the improvement of accuracy over the number of epochs, we can see that the neural net is indeed properly learning on the data, up to a point. However, we also see that there is a limitation to the accuracy improvement after a certain number of epochs. Once we reach 500 epochs, the neural net begins to overfit, causing the accuracy to decrease past this point. These results again confirm the limitations in the ability to truly fit to basketball data, due to the unpredictability, as overfitting on upsets can lead to incorrect future predictions.

Understanding the possibility of upsets and close games affecting the predictions of future games, we decided to see if the neural network was aware of the unpredictability of such games. With the below figure, we look at how the confidence of the predictions correlates to the corresponding accuracy of the predictions, coloring the incorrect predictions green and correct predictions blue. As we can see, the predictions that near the center tend towards being wrong more often than the predictions that are closer to 0 and 1.

Figure 6. Histogram of Prediction Ranges vs. Accuracy



This information is extremely valuable, as it demonstrates that the neural net is capable of determining the confidence of its predictions. Thus, we can leverage the raw predic-

tion values by labeling our prediction outputs with a corresponding confidence level.

7.2. Predictions of Future Games

In order to test our classifier on real life data, we predicted the outcome of 6 NBA games occurring on a following day. We ran the predictions 20 times and on the game days, compared the results to the actual outcomes. On average, the neural net predicted 4.3 out of the 6 games correctly. This is an accuracy of 71.67 percent and coincides with the accuracies we were seeing on test data.

8. Conclusions

Overall, we found that the results did significantly better than other automated predictions systems that provided their accuracies. Furthermore, our average accuracy was on par with the average of the top three human experts over the past 30 days for Covers Experts. While we tried multiple classifiers, we found that the neural net best performed to our requirements of both consistency and accuracy. Our results demonstrate that the neural net is in fact capable of learning patterns from historical player statistics for future games. While we encountered issues stemming from unpredictability of upsets and close matches, we found that our neural net is also capable of labeling our predictions with a confidence level, making these predictions even more valuable.

9. Future Applications

One future application of our trained neural network could be to predict college games. Because we are strictly using statistics of percentage, and ignoring information that corresponds to specific teams, we have generalized our model to work with any basketball game.

On the other hand, if we did want to be more specific to the NBA, we could further analyze the specifics of a team. Generating statistics such as the teams overall win-loss percentage, as well as its percentage against specific opponents, conferences, or divisions in the league might all provide insight towards how a team will perform against those same categories in the future.

Acknowledgments

None

References

CoversExperts. Top experts sports picks list, 2015. URL https://experts.covers.com/Picks.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Schaul, Tom, Bayer, Justin, Wierstra, Daan, Sun, Yi, Felder, Martin, Sehnke, Frank, Ruckstie, Thomas, and Schmidhuber, Jurgen. Pybrain. *Journal of Machine Learning Research*, 2010.

SportRadar. U.s. sports - live sports and statistics, 2015. URL https://www.sportradar.com/media-services/us-sports-live-scores-statistics/.

TeamRankings. Nba betting picks detailed splits, 2015. URL https://www.teamrankings.com/nba/betting-models/detailed-splits/.