Improving the Scalability and Realism of Game-Theoretic Approaches for Baseball

MS Project Writeup

Spencer Armon

In this research project, I used a game-theoretic model of baseball and enhanced the speed and accuracy. The model suggested Aaron Nola would have a 5.13 ERA vs the Cardinals, which is well higher than his career ERA of 3.70. Additionally, the model had simplified rules such as limiting the game to two bases and limiting at bats to 2 strikes and 3 balls because it took so long to compute a lineup. The three main things I worked on were implementing stochastic runners, speeding up the ERA computation, and enhancing the batter representation.

In the initial model, a single would advance all runners one base and a double would advance all runners two bases. However, runners frequently advance more bases than the hitter, making this simplification inaccurate. Additionally, some runners are far faster than other runners, and treating them all the same is another inaccurate simplification. The solution I used for this was storing empirical data for all batters, calculating the percentage of the time they advance to each base given their starting base and the type of hit. The data was stored in a 2x3x3 tensor where the first dimension indicates single or double, the second dimension indicates whether they started on first, second, or third base, and the third dimension indicates whether they ended up on second base, third base, or home.

Another change I made to the model was adjusting how ERA was calculated. Initially, ERA was calculated by finding the transition distribution for each possible state in the game, determining the strategy for each player, then calculating the value for that state, which is the expected number of runs from that state until the end of the game. This was incredibly

time-consuming because this must be calculated for all 9 innings, with each inning having thousands of states. I believed that a batter/pitcher's strategy stayed the same regardless of the inning of the game since they will always maximize/minimize runs the rest of the inning. In reality, the pitcher/hitter should care slightly about who starts the next inning (which won't exist in the ninth inning), but this is the only change with this new version. If the calculations are cut by a factor of 9 by only focusing on one inning, we can get the expected runs scored for each batter leading off the inning. The only calculation remaining is determining the probability of starting the next inning given who starts the inning. This can be calculated recursively by capping the number of runs in an inning. Then, one can calculate the number of runs scored in the ith inning onward with the jth runner leading off via the equation, $runs_{i,j}+(runs_{i+1}\cdot probs_j)$. Instead of running calculations on 8 additional innings, simply calculate the probability of each runner starting the next inning given the starting runner, then fill in the 9x9 runs table.

The final significant change I made was enhancing the representation of the batter in the system. Using a convolutional neural network, the model takes in a pitcher, a batter, and a pitch type to predict the outcome of a swing (homerun, triple, double, single, out, foul, strike). Initially, the batter was represented by their swing frequency and batting average for each pitching zone and pitch type. However, average indicates the percentage of at-bats in which a player gets a hit, so the model has no sense for how powerful a hitter is. Thus, I added empirical slugging percentage by zone and pitch type for each player so the model can more accurately predict the pitch result. After implementing this change, the cross entropy test loss in the model fell from 1.341 to 1.339. This is a marginal difference, and I believe this is the case because it shouldn't affect non-hit outcomes which is the majority of results and the dataset has high entropy in general so no model will be perfect. Nevertheless, the improvement is still smaller

than I expected.

Overall, I successfully improved the scalability and accuracy of the model. It runs 6 times as fast as before (on a Lenovo ThinkPad), and Aaron Nola's ERA vs the Cardinals fell to 4.4, which is closer to his career average. The model is now ready for future applications such as pinch hitting/running/pitching strategy, finding trade targets, and optimizing the lineup. The repository can be accessed at www.github.com/sparmon7/baseballgametheory.