AI-Based MLB Outfielder Recommendation

FieldVision AI

COMP3106 Fall 24 Final Project

December 3rd, 2024

Group Members:

Mario: 101286566

Paul: 101219909

Fabrice Mikobi: 101196480

2. Statement of Contribution

Mario:
- Project planning: ideas, planning approach to solve problem
- Finding datasets and gathering data, to then analyze predictive utility
- Strategy and algorithm for finding optimal outfielder positions, and applying rule-based systems to best adapt to game situations and coaching

Paul:
- Implementation and optimization of parsing the relevant statistical data for the neural network, and application optimizations.
- Design and implementation of the neural network model and the Outfielder Positioning loss function
- Neural Network Model Training and Testing

Fabrice:
- Application development
- GUI design and implementation
- Merging API functions, application code, and pre-trained model

3. Introduction

3.1. Background and motivation for the project

Drawing inspiration from the film "Moneyball," we recognized baseball's suitability for developing an AI-driven tool to assist coaches. It's a sport with an abundance of metrics and data which can be best used for Artificial Intelligence through Machine learning models. Luckily, placing outfielder's in baseball has a relatively static nature which allows for real-time predictions even with limited model sizes. Our objective then, was to create a tool to assist baseball enthusiasts, particularly MLB coaches, in determining the optimal positions for outfielders based on the current game situation and former player statistics.

Statement of objectives

FieldVisionAI had three main objectives

1) Retrieve the current game state and the relevant statistics for the batter and pitcher who are currently up.
2) Using the current game state in addition to the relevant batter/pitcher statistics, run the features through a complex Neural Network to make a field of predicted ball landing spots.
3) Using this field of predictions, calculate the recommended outfielder positions.

3.2 Related prior work

Creating this program required extensive research into the rules and dynamics of baseball, and understanding and quantifying data to find relevant data points. Thus, we first analyzed the data that the Pybaseball API had available to find data points of relevance.

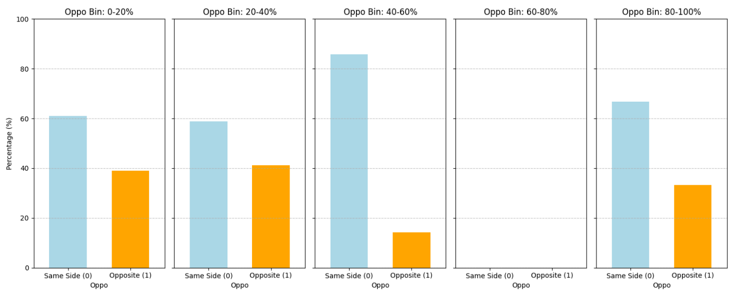
Here are some examples of the graphs and calculations used to find meaningful statistics:

Number of Balls vs Hit/NoHit

Number Balls	Hit%	No Hit%
0	10.92	89.08
1	6.55	93.45
2	7.53	92.47
3	5.84	94.16

From this data, we decided to keep the number of balls as we determined it likely be useful for the Neural Network to make predictions

Batter's Oppo%



Contrary to our initial assumptions, a batter's percentage hitting the baseball to their opposite side (Oppo%) did not show any significant impact on predicting a baseball's landing location, and so we decided against including the statistic to reduce noise in our predictions.

After analyzing dozens of statistics relevant to game events, batter's statistics, and pitcher's statistics, we narrowed down the statistics to the 54 features which we determined to be most relevant to where the baseball would land. The 54 features can be found in the playerStats.py PlayerStats class on our GitHub.

Initially, we envisioned incorporating Computer Vision into the data-gathering process to identify occupied bases. However, upon further analysis of the statistical relevance of bases, we discovered that they had a negligible impact on predicting ball landing locations. Therefore, we deemed it unnecessary to invest effort in developing this feature. On the other hand, we successfully leveraged the information on base occupations to create a feature for outfielder placement based on the current game situation.

4. Methods

4.1 Methods Used from Artificial Intelligence

To make this outfielder prediction system functional, we leveraged the capabilities of Artificial Intelligence to decipher the patterns within the calculated baseball statistics that we believed would be valuable in finding the true landing location of the baseball and therefore the best locations to place outfielders.

To predict the landing position of a baseball, we employed a **Neural Network** model that generates a normalized heatmap of probabilities. The model consists of an input layer that processes 54 chosen features extracted using the getMlbData() method, 3 fully connected hidden dense layers and outputs a probability field matrix for where the baseball may land. The output must then be reshaped to match the grid-like shape of a real baseball field.

To find the recommended outfielder positions we used a simple **Rule Based** system. We set some basic rules to find the most probable landing area and shift relevant outfielders toward these spots. Then, to the coach's choosing, we can tailor the positions to best adapt to the current game's state and the desired strategy.

4.2 Dataset

Our model was trained using historical Major League Baseball (MLB) data from the Pybaseball library, a Python API providing comprehensive statistics ranging from season averages and career records to individual game events. To optimize the model and enhance prediction accuracy, we extracted a dataset of over 310,000 game events spanning the 2020 to 2023 seasons.

As mentioned previously, before gathering any data, we first conducted an extensive analysis of the relevant statistics available from the Pybaseball library and identified a total of 54 features that we determined to be important and accessible.

Working with the Pandas library to gather the data, we initially used a loop-based approach to get the relevant statistics from the API and format each event to fit our model individually. This proved to be computationally expensive, with processing times exceeding two hours for even a few days' worth of events. Since we planned to use a much larger training set of multiple years, optimization was necessary to reduce this time. So to address this, we vectorized all the operations, applying formatting to entire pandas' DataFrames or columns simultaneously and using join operations to merge the event statistics with the pitcher and batter data. This leveraged Python's optimized batch operations, resulting in a significant performance improvement. The complete three-year dataset parsing was significantly faster, taking just 2 minutes from the initial API call, representing a substantial time reduction.

4.3 Model Architecture

Given our dataset, we set out to capture the intricate relationships between the data we determined through testing to be significant and the true location of where the baseball landed. To capture the complex, non-linear relationships between these input features and the ball's landing location, we employed a neural network with three fully connected dense layers, each containing 512 neurons contributing to the prediction. The output layer produces a flattened prediction matrix of size 125*125 which contains the softmax probability matrix for where it predicts the ball will land. By incorporating batch normalization after each dense layer, we were able to keep the network learning efficiently and prevent it from being thrown off by outliers in the data which would be expected given that we are working with real sports data. Additionally, we applied dropout layers to prevent the model from overfitting the training data to ensure it generalizes well to new situations and captures the larger patterns within the data.

Outfielder Positioning Loss

Predicting optimal positions for nine outfielders, while seemingly a classification problem of what position the ball landed, demanded a more sophisticated approach than standard categorical cross-entropy (CCE) loss. The issue was CCE only valued the prediction of the correct class, but since our task involved spatial predictions, a prediction close to the true landing spot was more favourable than a prediction far away from it, so we looked for a loss function which could represent that.

Our initial attempt involved a loss function combining k-means clustering and Gaussian distributions. This aimed to group high-probability predictions while encouraging spread among the outfielders by considering pairwise distances. However, k-means introduced instability due to its inherent randomness, leading to inconsistent gradients and hindering effective learning. Furthermore, the Gaussian convolution proved computationally expensive, increasing training time. Ultimately, the noisy loss landscape generated by this approach made it challenging for the optimization algorithm to converge.

To overcome these limitations, we developed **"Outfielder Positioning Loss (OPL)"**, a novel loss function which offers significant advantages by combining the strengths of CCE with crucial spatial considerations. This function combines multiple components to directly influence outfielder positioning, the three key components are:

- Categorical Cross-Entropy (CCE): This is the foundational component that measures the difference between the predicted probability distribution over the grid cells and the true distribution (the one-hot landing position matrix). By minimizing CCE, the model learns to assign a high probability to where it believes the ball will land.
- Distance Regularization: This component calculates the minimum Euclidean distance of the top nine predicted landing locations and the true location of the ball. By minimizing this distance, the model learns to accurately predict the ball's trajectory for all 9 outfielders, ensuring that high predictions are positioned close to where the ball is likely to land.

- Confidence Regularization: This term encourages the model to have higher confidence in its top nine predictions. It achieves this by penalizing low confidence values among the top predictions, essentially discouraging the model from being uncertain about the ball's potential landing spots. This is done by adding the mean of the squared differences between 1 and each of the top 9 predicted probabilities to the loss.

Our Outfielder Positional Loss (OPL) function presents several key advantages over both categorical cross-entropy and our initial approach. Firstly, OPL incorporates spatial awareness by valuing predictions in proximity to the true landing spot, a feature absent in categorical cross-entropy. Secondly, OPL provides an efficient and stable loss function with computable gradients, facilitating effective backpropagation during model training. Finally, OPL offers tunability through hyperparameters (lambda_dist and lambda_reg) which control the balance between distance and confidence, enabling fine-tuning for specific tasks.

OPL Performance

To evaluate OPL's effectiveness, we used the same model architecture and dataset with both OPL and standard categorical cross-entropy (CCE) loss, observing a significant improvement in predictive accuracy. OPL yielded a 47% increase in top-9 accuracy, measuring the model's ability to correctly predict at least one of the top nine landing spots, and a 34% improvement in standard accuracy, indicating its superior ability to identify the exact landing cell. These gains highlight the benefits of incorporating spatial awareness into the loss function, as OPL guides the model towards more effective outfielder positioning by explicitly considering the proximity of predictions to the true landing location. Importantly, this improvement came without any significant increase in training time, demonstrating that OPL achieves greater accuracy without sacrificing computational efficiency.

4.3 Training and Validation

To expedite model development and allow for rapid iteration, we began by training and evaluating our model on a subset of the data. We used one month of data from May 2023 for training and two days, July 1st and 2nd, 2023 for testing. This allowed for quicker experiments with different model architectures to find what worked best.

Hyperparameter Tuning

We performed hyperparameter tuning using various values to optimize the following hyperparameters: learning rate, neurons per layer, and dropout rate. We evaluated the different hyperparameter combinations based on validation loss and top-9 accuracy on the smaller dataset. The table below shows some of the hyperparameter selections we tested and their corresponding accuracies. After tuning, we arrived at what we feel is the best model we could produce given our dataset and constraints (highlighted in bold).

Learning Rate	Neurons/Layer	Dropout/Layer	Top-1 Accuracy	Top-9 Accuracy
0.001	256	0.2	0.000618812	0.0111386
0.001	**512**	**0.3**	**0.00123762**	**0.00928218**
0.0001	512	0.25	0.000618812	0.0024

Prediction Validaiton

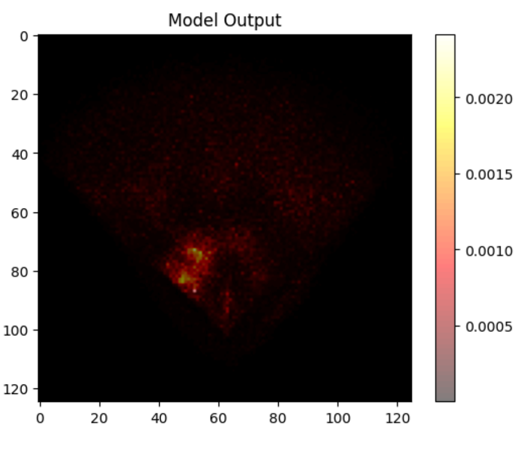
To validate our model's effectiveness in optimizing outfielder positioning, we focused on quantifying the impact of its predictions. We compared the computed outfielder positions, derived from the model's probability matrix, to a standard baseball outfielder formation. This comparison focused on the distance between the closest outfielder and the true ball landing location. We believe this metric accurately reflects successful positioning, as minimizing this distance directly impacts an outfielder's ability to quickly reach the ball and make plays.

5. Results

5.1 Model Prediction Results

Once we finalized our model architecture and hyperparameters, we trained it on our full dataset (2020 April - 2023 March, 310K events) and tested our model on 3 months of data (2023 June-August, 60K events). Since the size of the dataset we trained on increased significantly, we lowered our learning rate by 10% to 0.0001. We then once again measured the model's accuracy by calculating the percentage of instances where one of the top 9 predicted landing locations matched the true landing spot. The final model finished with 0.003 accuracy and 0.021 top-9 accuracy, up 100% from the models with smaller data. Considering this was 31x better than randomly guessing locations and over 10x better than using the average hitting location in the dataset, proving the model successful.

The heatmap figure to the right is an example of a prediction from the model for an event in June. The small pink dot at (50, 90), is the true landing spot. The hottest areas indicate where the model predicts the ball will land.



5.2 Outfielder Prediction Results

5.2.1 Qualitative

To visually interpret the results of our predictions and recommendations, we graphically mapped the landing predictions and recommended outfielder positions, along with the actual (true) landing position of the ball. A large majority of our predictions show accurate predictions, with the ball landing within the hottest hotspot (a pink-yellow-orange zone), and the majority of those that didn't land within the hotspot landed in the same direction as the predicted hotspot. Showing our ability to predict the direction of the hit. Naturally, we did have some predictions that were not close to the actual landing spot, but this can be expected in a sport like baseball.

5.2.2 Quantitative

As mentioned in the validation section, to quantify the efficiency of our predictions we measured how useful our suggested outfielder positionings were in reducing the distance between players and the ball's actual landing spot. We did this by taking a standard outfielder formation and our suggested outfielder formation and calculating the distance between the true ball landing and the closest player. If our suggestion had a negative impact on that distance, and thus gave a larger minimum distance, this counts as our model making a "bad" prediction. If the closest distance using our suggestion was equal to or better, we count this as a "good" prediction. Using 1000 different events, we found that our suggestions were useful 90.5% of the time. This confirmed to us that our NN model effectively predicted potential landing locations, and our algorithm successfully leveraged these predictions to optimize outfielder positioning. These quantitative results demonstrate the model's real-world applicability and its potential to support coaches in real-game scenarios.

5.3 Visualization

We developed a graphical user interface (GUI) to enable coaches to input relevant information and easily use our outfielder prediction system. The application allows users to input key game statistics, such as the inning, count of strikes and balls, and batting and fielding scores, which are then fed to our prediction system. The result is a heatmap of hit locations, and suggested ideal outfielder placements, displayed as blue dots.

Behind the scenes, the application retrieves batter and pitcher statistics for the given season via the Pybaseball API and formats this data for model predictions. The model outputs a probability matrix of hit locations, which is transformed into a heatmap scaled to the dimensions of the baseball field. To enhance visualization, the heatmap features smoothly blended colours, with lighter shades emphasizing areas of higher probability.

Additionally, the application provides coaches with tools to apply their strategy in real-time. Coaches can select between defensive, neutral, or aggressive strategies, to align outfielder positioning with the game's context. For example, a defensive strategy places outfielders farther back to prevent extra-base hits and ensure broad coverage. Ultimately, the application serves as a valuable resource empowering coaches to adapt their strategies dynamically, optimize outfielder positioning, and gain a competitive advantage during the game.

6. Discussion

6.1 Limitations

Neural Nets for intensive predictive tasks are very heavily dependent on the data that are used to train them. Naturally, this was our largest restriction. We relied on (free) external sources, in our case Pybaseball, to collect baseball data which while thorough, is not perfect. For example, many events from each season were unable to be used due to missing data fields, and we were restricted in which seasons we could use, as the data collected changes from time to time and we need to ensure we have the same statistics collected for each event.

In addition, the hitting of a baseball has countless variables affecting it which we are unable to capture. Wind/weather is a great example of one of these external factors that would have a significant impact on landing location but is unfeasible to get for our situation - we would need to get exact wind speed and direction in relation to the batter and the field. A major limitation was the lack of data regarding outfielder location data, which hindered our ability to develop a stronger Neural Network model that directly generated the best outfielder locations. Instead, we utilized the available data to create a system that translated base outfielder predictions based on projected landing locations of the baseball using our computed statistics, thereby achieving a similar outcome.

6.2 Future Work

The success in predicting ball landing location and proactively shifting outfielder formation from these predictions is notable and a sign of the power that data analysis and AI can have on sports, and more precisely baseball. As mentioned, our collection of data was limited so a project with more time, resources and access to data, would yield a more powerful and accurate neural network model.

Maybe the most impactful addition that could be made to this project would be to track the success of different outfielder positionings in real situations. This would allow us to fine-tune our recommendation algorithm over time and ultimately provide coaches with a more complete and useful product.

6.3 Implications of work

As data becomes a larger part of everyday life, business, and even sports, evolving technologies such as AI are destined to continue this trend, and our work is a testament to the power that these technologies can have and will have on sports. Our work can be directly used by MLB coaches to influence their decisions and can influence organizations to find more applications of data analysis and AI. As a team, we proved that by using publicly available and limited data that individually may not have predictive power, we can leverage Artificial Intelligence tools to create real and tangible insights into the game of baseball, and thus, can be extrapolated to sports in general.

7. Conclusion

FieldVision AI successfully demonstrated the potential of AI in enhancing baseball strategy by recommending optimal outfielder positions. Through rigorous data analysis, a carefully designed neural network, and outfielder recommendation algorithms, the project achieves significant success in predicting ball landing spots and positioning players effectively. While further refinements are possible, the results show that our AI-driven approach can outperform standard formations and offer valuable insights to coaches. This project underscores the power of AI in sports analytics and sets the foundation for future advancements in baseball strategy tools.

8. References

- **Pybaseball:** https://github.com/jldbc/pybaseball
- **Tensorflow:** https://www.tensorflow.org/api_docs
- **AI in Sports Applications:** https://appinventiv.com/blog/ai-in-sports/
- **AI in Sports Predictions:** https://ieeexplore.ieee.org/abstract/document/4492661
- **Statistic Definitions:** https://baseballsavant.mlb.com/csv-docs
- **Statistic Definitions:** https://www.mlb.com/glossary
- **Code:** https://github.com/osasuair/FieldVision-AI

Given the novelty of this project, we were unable to find useful and strong prior works - from scholarly articles to online forums that mention the use of statistics for in-game predictions. We gathered as much prior information as we could to help us during our development, and we are happy with the end result.