Quest for a Better Defensive Metric: Dynamic Catch Probability

Github: https://github.com/[redacted]/SMTDataChallenge2024
Team 57 - Undergraduate Division

Abstract:

In this paper, I set out to improve the catch probability statistic. Instead of assigning an overall catch probability to a play, I assign a catch probability every quarter second. This process allows for analysis of the change in catch probability over the course of a single play, similar to how win probability changes over the course of a game. I also discuss the shortcomings of current defensive statistics and why this would be a welcome improvement to the statistical landscape. Finally, I discuss potential improvements and unresolved questions.

Table of Contents

1. Introduction

It's easy to tell where a player gets their offensive value based only on their stats. A player that accumulates walks and singles will have a very different statistical profile than a player that sacrifices strikeouts for home runs. This sort of statistical differentiation is, I believe, only just beginning for defensive analysis. Outs Above Average (OAA) tells you overall how valuable a player was on a particular play, but doesn't tell you *where* they get their value from. Having a high OAA tells you nothing about a particular play or their player profile. For outfielders: do they get good jumps, or do they get average jumps and make up for it with elite speed? For infielders: do they have a large range? Do they dive well? Field short hops well? Do they have outlier arm strength? Of course stats do exist to answer each of these questions individually, but it still stands that knowing the Outs Above Average for a play doesn't tell you how that number was achieved on that particular play.

Per the official MLB Statcast page, statistics like Outs Above Average (Reference #1) and Catch Probability (Reference #2) are calculated differently for infielders and outfielders (sometimes not at all for infielders). *The analysis in this paper will be limited to outfielders* due to the relative simplicity as well as the fact that the method I will describe makes more sense for catches than for force outs.

So what's wrong with the existing outfield defensive metrics? Like I said, OAA is good, but not as *interpretable* as you would sometimes like on a play-by-play basis. Conventional wisdom states that taking a good route to the ball is important, so maybe this is a solid, interpretable way to quantify how good a particular play is. *Route Efficiency* is a metric that asks a very simple question: compared to how much distance you had to cover, how much distance did you actually cover? The idea is that if you covered much more distance than you had to, you probably took a route that wasn't ideal. Theoretically, this statistic could tell you who in the game is taking the best routes in the outfield. The formula is as follows:

$$\text{Route Efficiency} = \frac{\text{Straight line distance to ball}}{\text{Actual distance covered}}$$

Unfortunately, this statistic simply isn't useful. While good in theory, it turns out that it's easier to get a higher efficiency when the route is simply longer (Appendix B.1). Additionally, I found that the distribution looks the exact same as a completely random distribution, indicating that there's probably no skill involved in having a higher average efficiency (Appendix B.2). Not to

mention the fact that sometimes you actually want to take a curved route to a ball, for example to get momentum into a throw. Statcast seems to agree with this assessment, as mentions of Route Efficiency online are all from the earlier years of Statcast; they seem to have phased it out.

Another existing outfield defensive statistic is *Jump.* Jump is defined by Statcast as being the number of feet covered in the correct direction in the first three seconds of a play (Reference #3, Appendix C.1). Unlike Route Efficiency, I would consider Jump to be a useful statistic. It has a high correlation with Outs Above Average, and the distribution of Jump scores amongst the players in the given data indicates that some people truly are better at getting good jumps (Appendix B.3).

So that's it, right? What is there to improve about Jump? Well for one, only looking at the first three seconds is a bit arbitrary. I couldn't find any information on the Statcast website about why three seconds was chosen. Of course the other issue is that Jump only looks at the beginning of the route. A bad jump could plausibly be made up for by elite tracking and speed.

2. Data

The data for this project was graciously supplied by SportsMEDIA Technology and contains game information on hundreds of minor league baseball games. For each game, we have timestamps for each "event," which includes balls being thrown, balls being hit, balls bouncing, etc. We also have ball and player position coordinates 20 times per second. However, like with all real data, there are some issues with it. Firstly, there's a small amount of missing data. Not every play from every game is accounted for, although that isn't much of an issue for this specific paper; it means we have fewer plays to work with but our analysis doesn't depend on the larger context of the game that would potentially be missing. Additionally, the player IDs are occasionally missing. This means that, while we have the play data, we don't know which player was fielding the ball. We can still analyze the plays, but if we want to compile things on a per-player basis we will be missing some data. There are other additional miscellaneous errors, but nothing that significantly impacts the analysis.

3. Methodology

The core of this paper is a statistic that I'm calling *Dynamic Catch Probability*. Heavily inspired by Win Probability, the goal is to show the probability of catching a fly ball at different times throughout the play and see how it changes. The idea is that, throughout the play, the catch probability should change. For example, imagine a play with a baseline Catch Probability of 50%. After getting a bad jump, this could drop to 20%. However, maybe with elite speed and a direct route the player could make up for this bad jump and up the probability to 70%, eventually making the catch.

To understand this statistic, it is important to first understand how Catch Probability works. Per the Statcast website (Reference #2), four variables are taken into consideration: distance the fielder is required to cover, amount of time the fielder has to cover the distance, direction traveled, and proximity to the wall.

Variable	Calculation method
Distance required	Euclidean distance between player's coordinates at the time the ball was hit and the coordinates of the ball when it is either caught* or bounces. * In the event of a catch, instead of the ball's coordinates we take the player's coordinates. This does create an inconsistency for balls that were caught vs. ones that weren't, but I consider it negligible. Ideally we would take a projection of where the ball would have hit the ground had it not been caught, but given my limited physics knowledge I don't have confidence in developing an accurate model with unknown atmospheric conditions.
Hang time	Amount of time between the bat contacting the ball and the ball either hitting the ground or being caught.
Direction traveled	First find the angle at which the player would have to run to where the ball bounces or is caught and then convert to either forward, back, left, or right. The 360 degrees around the player is split into four equal slices of 90 degrees each (Appendix C.2).
Proximity to wall	Not considered for this analysis.

Table 1: Calculation Method for Each Variable Used by Catch Probability

While there's no specific formula on the website, I suspect these variables are put into a very basic machine learning model since that's exactly what I did and my model correctly predicted if a ball was caught over 95% of the time (Appendix B.4). Using the *sklearn* Python library, I set up a *Multi-layer Perceptron Classifier* (Figure 1). The specifics of this will be left to the Appendix (Appendix C.3), but broadly speaking this is a neural network model that takes in the distance, time, and direction variables and spits out a prediction: catch or no catch. However, crucially, the model can be told to output a probability instead of a binary prediction. *It's this probability that I use as my catch probability.* Statcast rounds their Catch Probability to the nearest 5 percent, and I will as well.

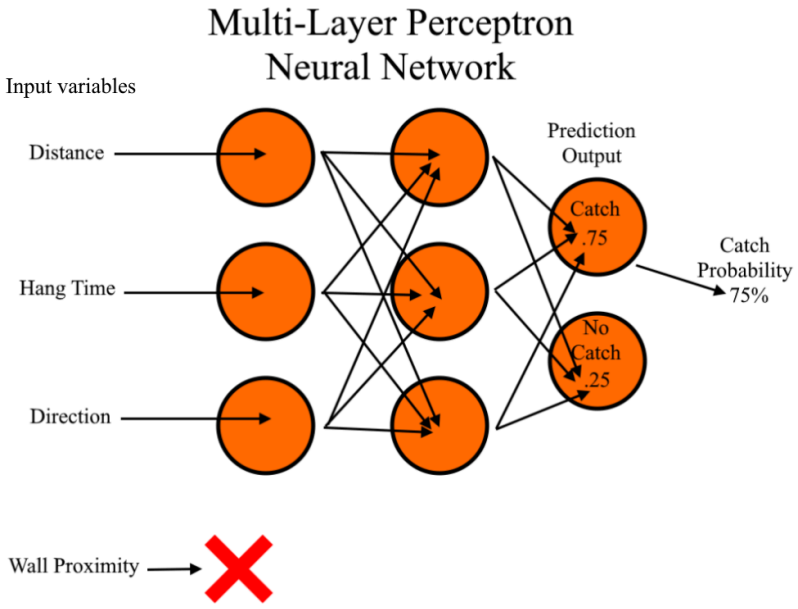

Figure 1: Example diagram of a basic neural network. The model takes in input variables and outputs a catch probability percentage based. Wall proximity is not inputted into this neural network.

Ninety-five percent accuracy sounds quite impressive, but given the distance and hang time it's actually very easy to guess whether or not the ball was caught with high accuracy. As seen in Figure 2, most combinations of hang time and distance will either be caught basically every time or never, with the not-automatic-but-still-doable plays taking up a relatively small amount of the total possibilities. I should note that this includes every ball that was first acquired by an outfielder, meaning this does include ground balls to the outfield as well.

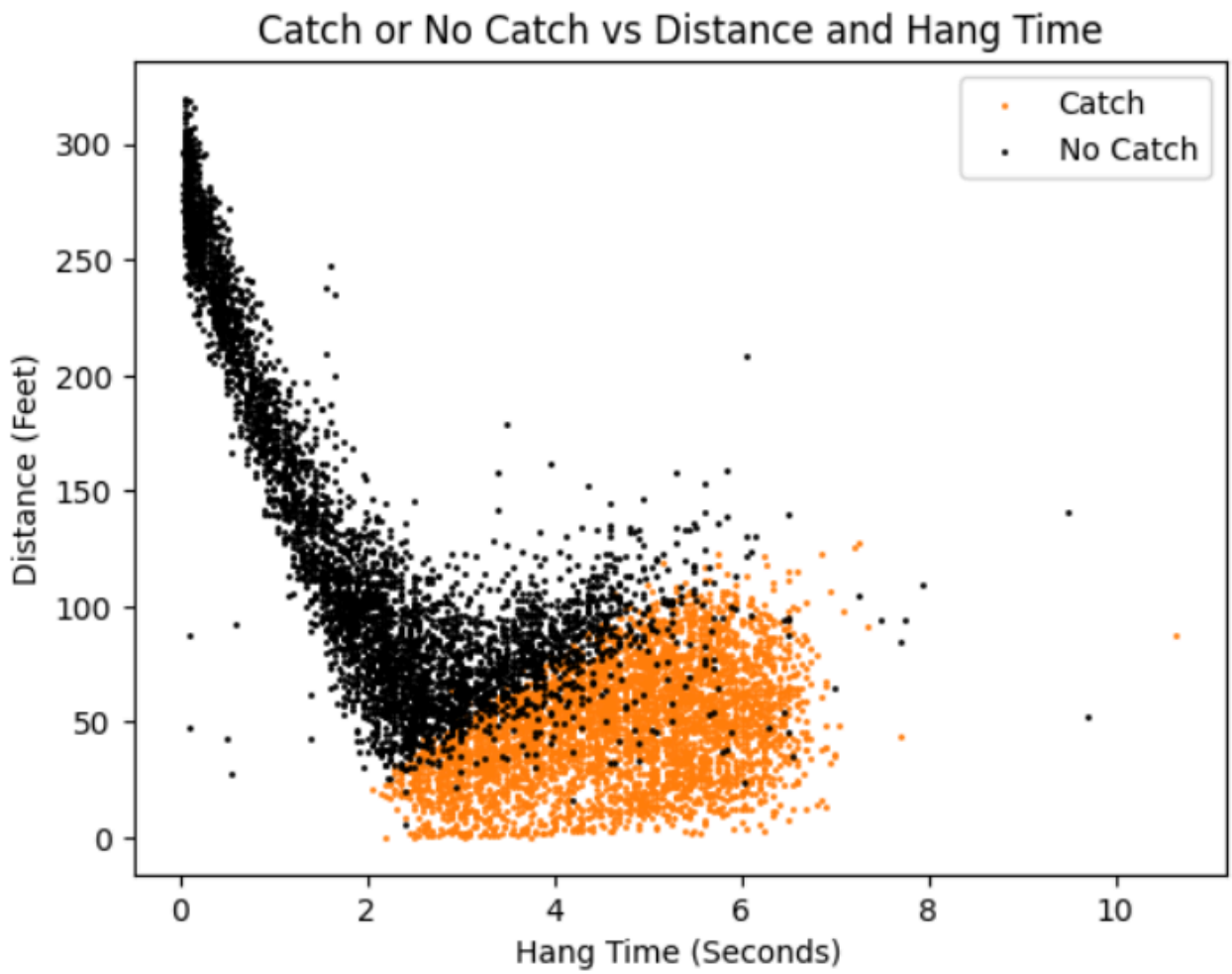

Figure 2: Every play that included an outfielder fielding a ball with orange representing caught balls and black representing balls that weren't caught. There is a clear dividing line between the colors, indicating that how likely a ball is to be caught is almost entirely decided by hang time and distance.

So how do we turn this into something dynamic, something akin to Win Probability that periodically updates? As it turns out, very simply. We only need to make minor changes to the three variables used for our Catch Probability model, as well as add one additional variable. For my model, every quarter second the following variables were recorded (Figure 3).

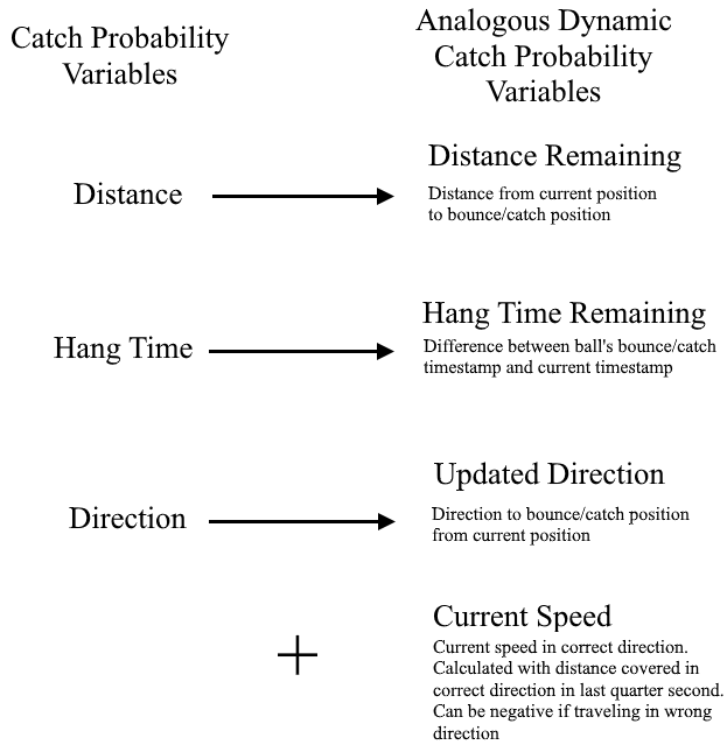

Figure 3: Transformations we make to the Catch Probability variables to make them useful for Dynamic Catch Probability. Current speed is an addition. Bounce/catch position means the coordinates of the ball at the moment it either is caught or hits the ground.

I used the same type of neural network for Dynamic Catch Probability as I did for my own Catch Probability earlier. This means we have a catch probability percentage multiple times for each play, resulting in Dynamic Catch Probability.

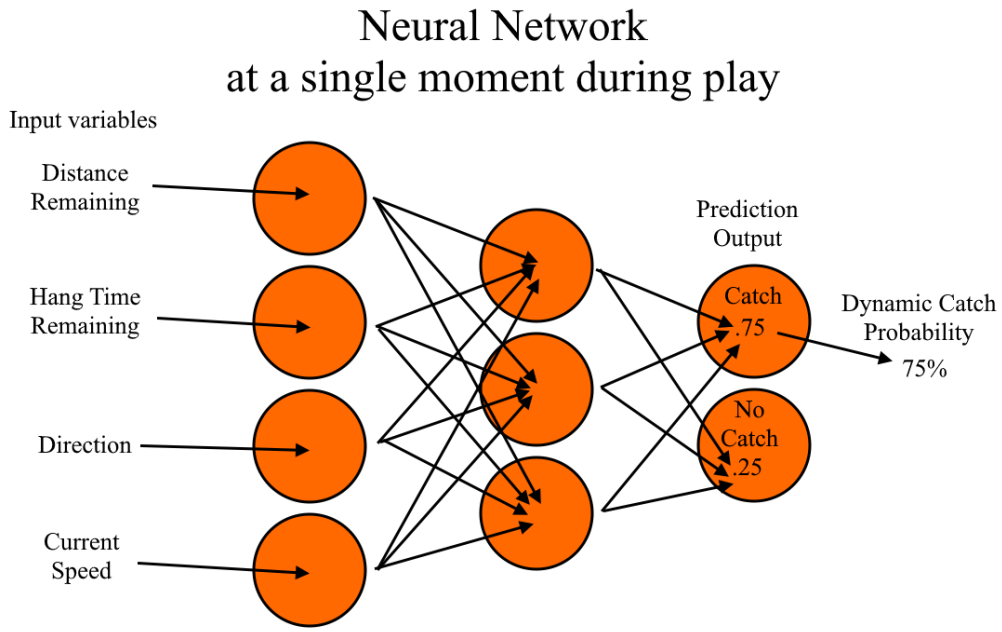

Figure 4: Example diagram of our updated neural network. It works in the exact same way except with four input variables instead of three.

These were the steps that I used to go from raw data to Dynamic Catch Probability:

1. Isolate only the relevant outfield plays from the raw data (Appendix C.4).
2. Find the exact timestamp of the catch or bounce and find ball coordinates at that time.
3. Find player coordinates for all the times between contact and the bounce/catch.
4. For every fifth timestamp, calculate the four variables.
5. Create and train the machine learning model on all of these timestamps' data (Appendix C.5).
6. Use the machine learning model to generate predictions in the form of probabilities.

4. Results

Figure 5 shows the Dynamic Catch Probability of a real play.

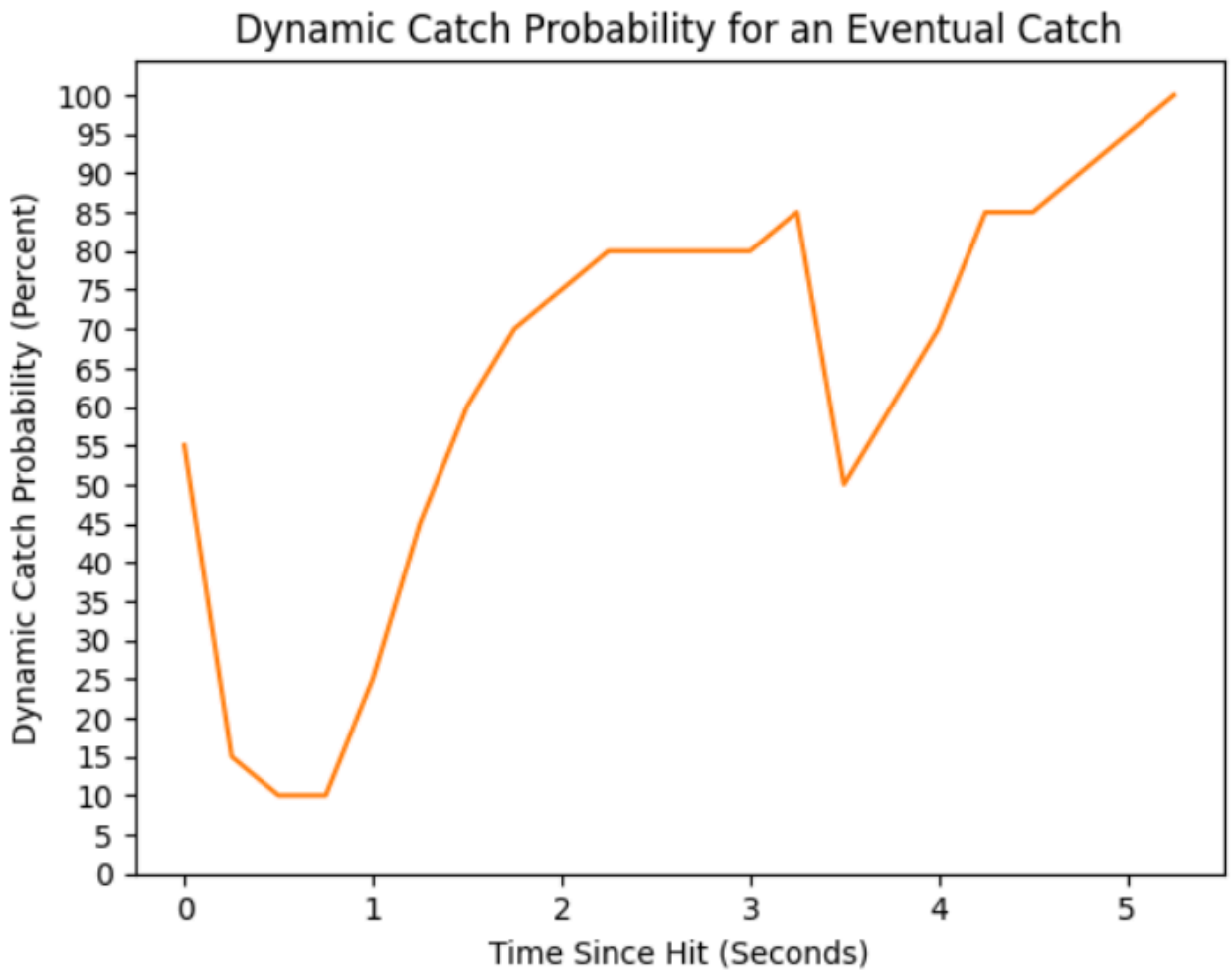

Figure 5: Dynamic Catch Probability for one play. The play starts at a 55% chance of being caught. A bad jump turns that into a 10% chance, but the fielder makes up for it, eventually catching the ball. As you would expect, the Dynamic Catch Probability gets more accurate later in the play.

From left to right, we see the changes in catch probability throughout the play. At the start, there was a 55% chance of the catch being made. However, after a quarter second, this went all the way down to 15%. This indicates that the player got a poor jump. However, they must have made up for it, since the catch probability goes up dramatically, eventually resulting in a catch. Does this assessment agree with the eye test? Here's an animation of this play:

Figure 6: GIF of the example play that resulted in a catch. The right fielder takes a very slight but noticeable step in as the ball is being hit, which the model interprets as a negative velocity (since it's in the wrong direction). This caused the DCP to go down. Seemingly good speed and ball tracking caused it to go back up, resulting in a catch.

The eye test seems to confirm what Dynamic Catch Probability says. If you focus on the right fielder as the ball is hit, you can see a very slight step forwards, likely due to the natural step inwards that defenders generally take as the ball is being thrown. However, they take a fast, direct route to the ball, making a great play.

Here's a similar example:

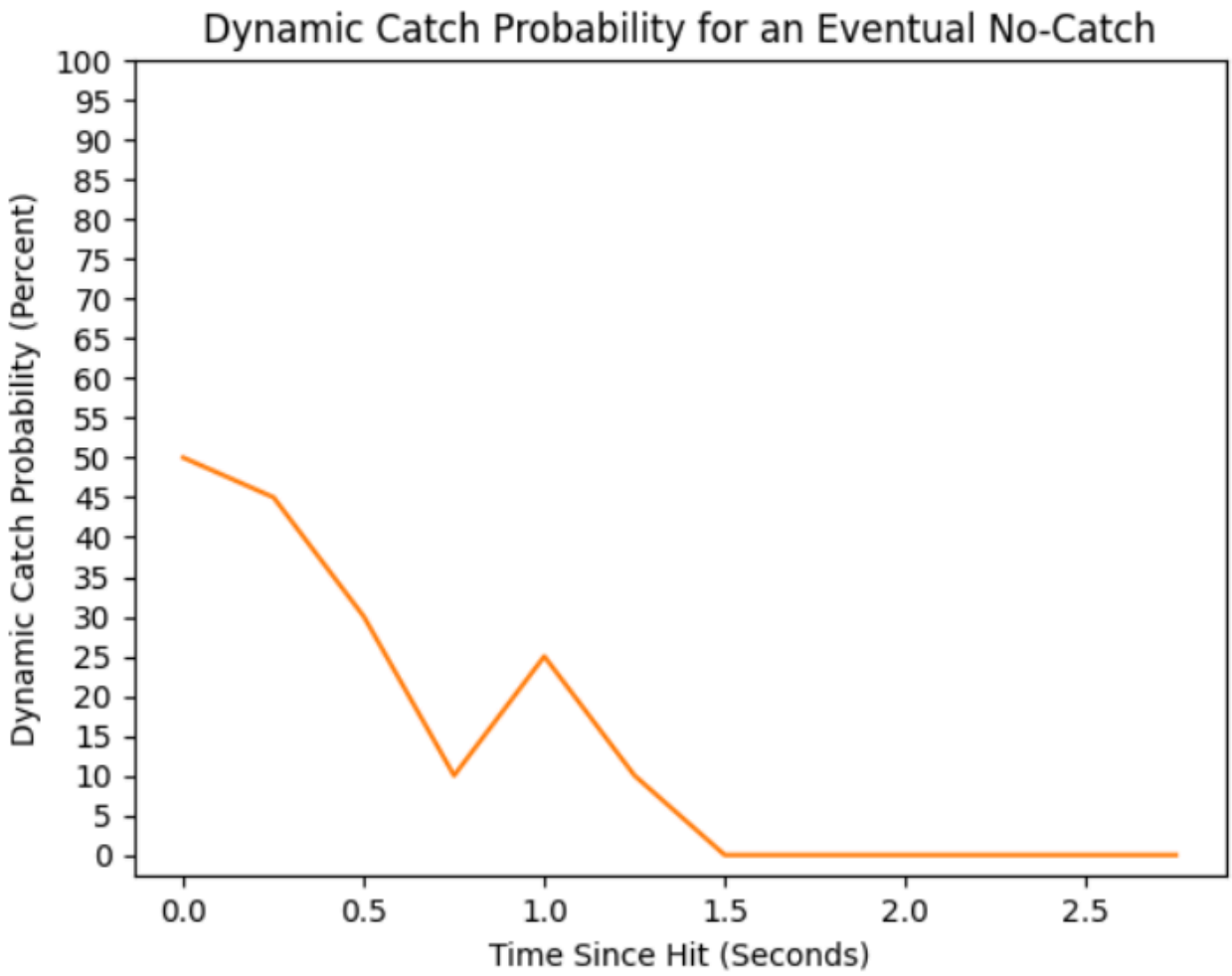

Figure 7: Dynamic Catch Probability for another play. This play starts at a 50% chance of being caught. The jump is fine, but the DCP steadily decreases and the result is a no-catch.

This play started with a 50% chance of being caught, so it was always going to be hard, but this chance declined and resulted in a no-catch. The player's jump wasn't as disastrous as the previous play, but they didn't have the speed or acceleration to capitalize on a relatively neutral jump. Again, does this match the eye test? Here's the play:

Figure 8: GIF of the play that results in a no-catch. This matches exactly what the Dynamic Catch Probability says. The right fielder had a chance at catching the ball, although it would have been a difficult play. However, partially through the play they seem to give up and resign themselves to fielding the ball on a bounce from behind. Animation code from Eddie Dew.

This play seems to match exactly with what the Dynamic Catch Probability says. Based on the small hang time and the distance, the play would have been difficult, thus resulting in a 50% baseline catch probability. The fielder gets a good enough jump to still be competitive, but they seem to give up and change their goal from catching the ball to fielding it from behind.

5. Discussion

Figure 9 shows all of the plays' Dynamic Catch Probability graphs overlaid on each other:

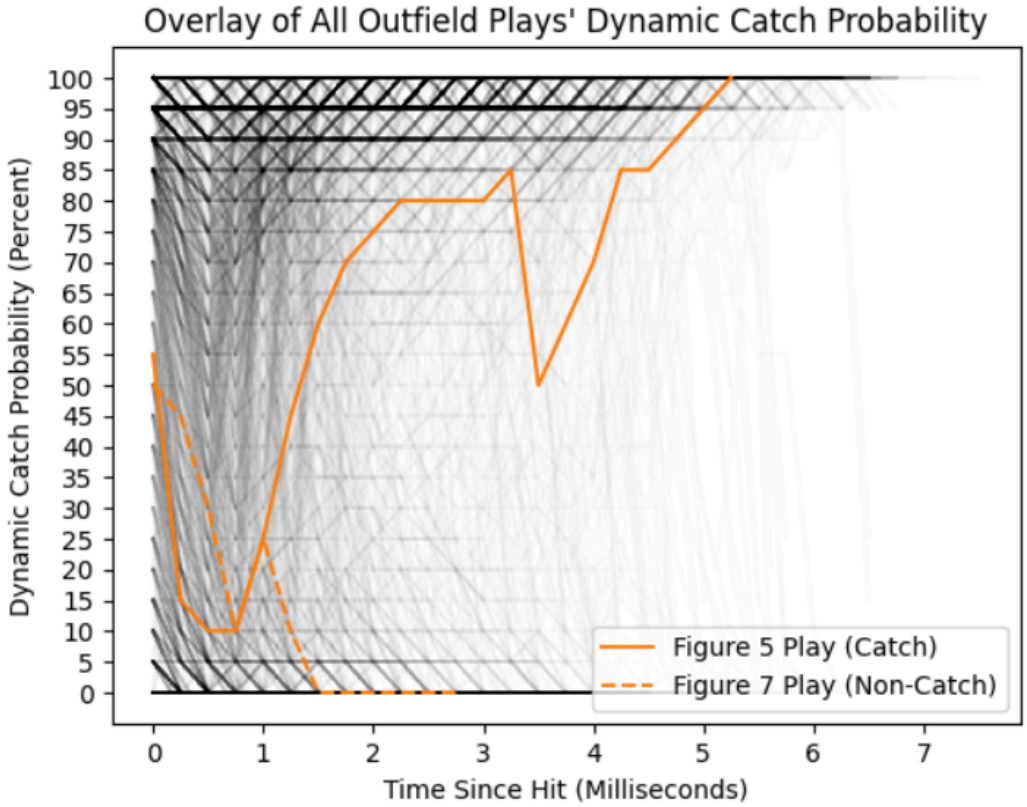

Figure 9: An overlay of all the Dynamic Catch Probabilities for the given data. Darker lines means more overlap. The two previous examples are shown in orange. The dark parts to the left of the two second mark indicate a lot of change in the first two seconds. By four seconds into the play, most plays are solidly above 90% or below 10%. Although there will naturally be less overlap since there are fewer plays that are in the air for four seconds, the pattern still holds (Appendix B.5).

On this graph, the darker lines represent more overlap. As can be seen by the very light lines in the middle right, most plays are either very likely to get caught or very likely to not get caught by about two seconds into the play. This would suggest that the *first two seconds are actually where most of the change in probability takes place*. This is significant because Win Probability works in the *exact opposite way*; bigger changes happen in the later innings. This would suggest that getting a good jump is the most important part of being an outfielder. Looking at MLB stats corroborates this idea; Jump has a noticeably higher correlation with Outs Above Average than Sprint Speed does, for example.

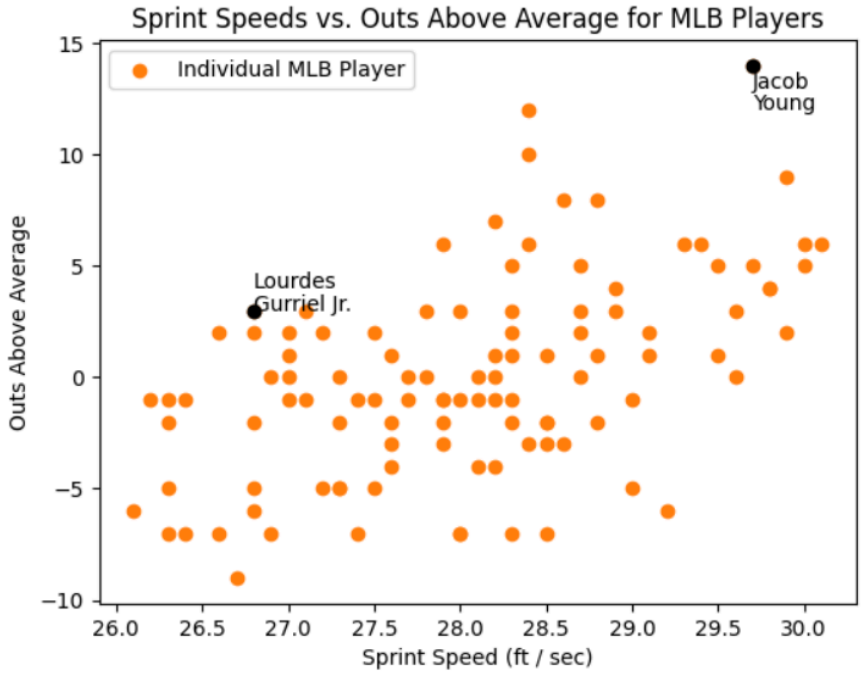

Figure 10: Scatter plot of every qualifying MLB player with Sprint Speed and Outs Above Average. Jacob Young is elite in both statistics. Lourdes Gurriel Jr. has a solid OAA despite having below average Sprint Speed. Data shows a positive correlation, although Gurriel's high Outs Above Average is unexplained. Data gathered on August 4th, 2024.

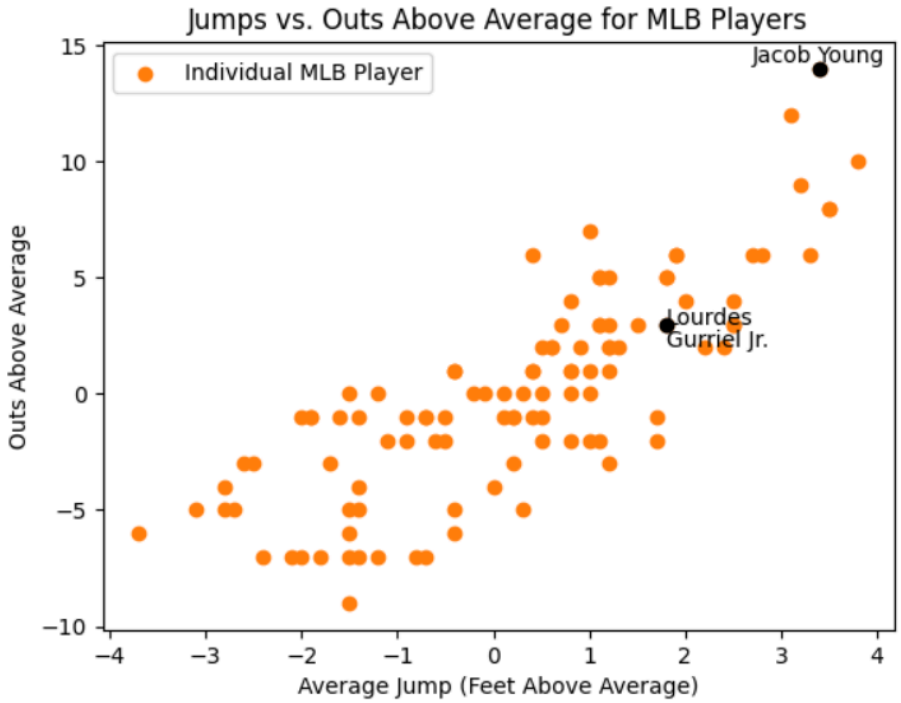

Figure 11: Another scatter plot, but with Average Jump instead of Sprint Speed. A zero on the x-axis indicates league average. With this information, Gurriel's high Outs Above Average makes more sense. Data shows a strong positive correlation. Data gathered on August 4th, 2024.

It should be noted that there is no discernible difference between jumps for players in different levels of the farm system. This suggests that *improving jumps is either not a developmental focus for this organization or it isn't a deciding factor in player promotion.*

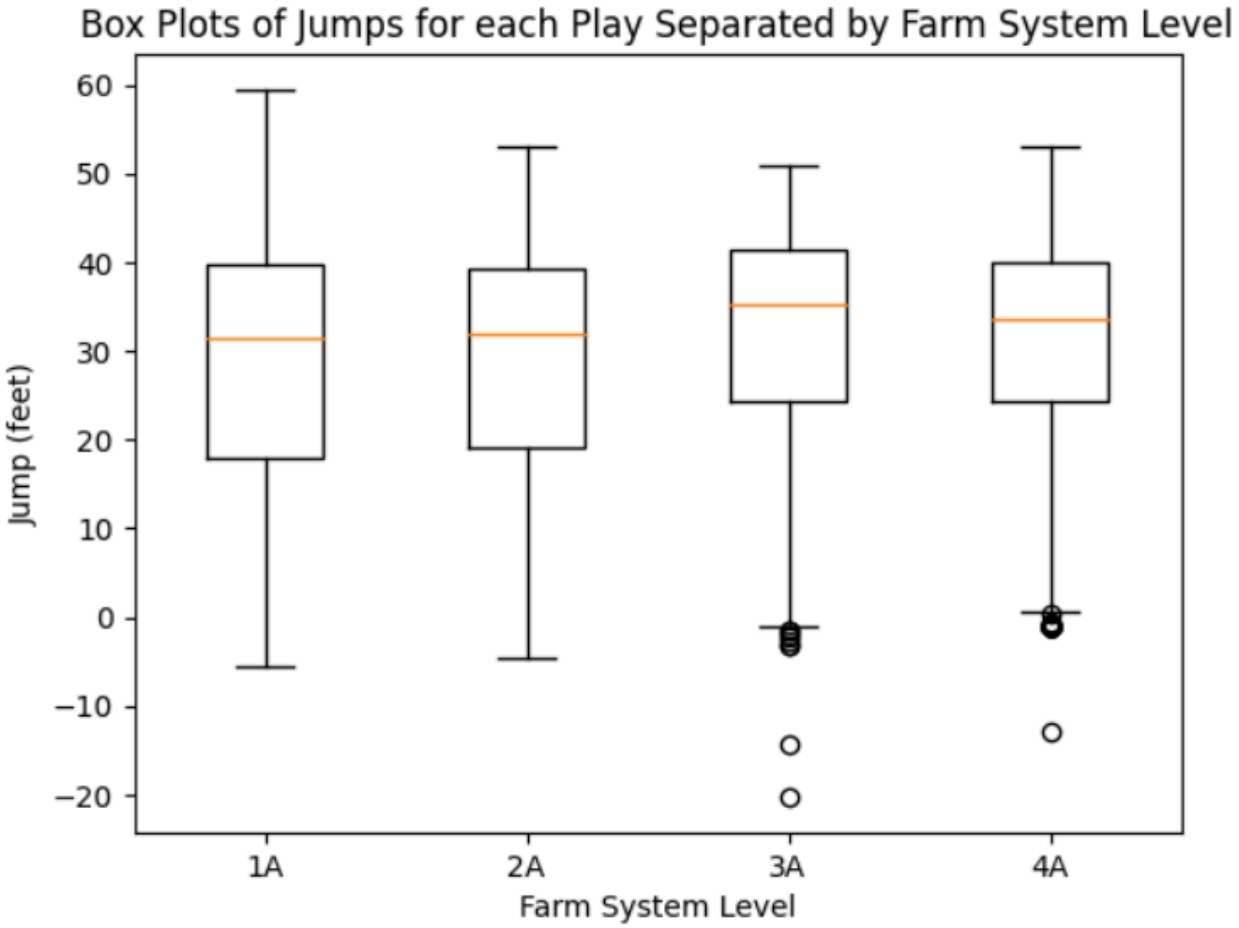

Figure 12: Box plots representing the Jump for each play. Separated by farm system level (4A is the highest). The orange line represents the median. Only "hard" plays were considered; plays with a Catch Probability between 5% and 95%. Plays in higher levels don't have a notably better average jump.

6. Conclusion

To summarize this project's key findings:

1. Route Efficiency is *not useful* for analyzing a player's abilities.
2. Jump *is useful* for analyzing a player's abilities.
3. Statcast's Catch Probability can be closely replicated with a *simple neural network.*
4. ***Dynamic Catch Probability* can be created through a slight modification of Catch Probability and is more interpretable than Catch Probability for individual plays.**
5. The *beginning of a play is the most important* for turning a tough play into a catch. This is *corroborated with MLB data* on Jump, Sprint Speed, and Outs Above Average.
6. Jump is a potential *blind spot* in this organization's minor league outfield development.

7. Acknowledgements

Thank you to SportsMEDIA Technology for organizing this competition. I would like to thank Dr. Meredith Wills for her support throughout the duration of this project. I would also like to thank Kenny Bores for finding me on LinkedIn and telling me about this competition. Without that message I would not have known about it. Thanks to Eddie Dew as well, whose code to turn the plays into animated GIFs helped tremendously. Additional thanks to all of my professors for teaching me data science concepts, as well as any cafe/library that I got work done at in the last two months.

8. References

Reference #1 (Statcast Outs Above Average): https://www.mlb.com/glossary/statcast/outs-above-average
Reference #2 (Statcast Catch Probability): https://www.mlb.com/glossary/statcast/catch-probability
Reference #3 (Statcast Jump): https://www.mlb.com/glossary/statcast/jump

9. Appendix

Appendix A: Suggested Future Work

There are a number of ways Dynamic Catch Probability could be improved. The most obvious one would be to include the proximity to the wall. Additionally, the interval between catch probabilities could be smaller. With this data I could have done 20 times a second, but I had worries about the reliability of the coordinates on this short of a timespan. Still though, more testing could be done on the optimal interval to consider. I would guess that at a certain point getting finer details wouldn't actually give any more information, but I don't know at what point that would be.

Another possible improvement is in the calculation for the current speed that gets put into the DCP model. I was only tracking player positions for the duration of the play, so when I realized I needed a speed at the exact moment the ball hits the bat, I just set it to be zero. In reality, players move slightly before the ball is thrown, and that is not reflected in the current model. I could see a change to this actually having a large effect, since the model could potentially learn that a small step in the wrong direction at the very beginning of the play isn't a death sentence. My hunch is that currently the model is too sensitive to negative velocities. In theory a tweak to this could invalidate my findings about the beginning of the play being the most important, since that's when negative velocities are the most likely to occur. However, given the incredibly strong correlation between Jump and OAA in the MLB data, I don't think this finding would be completely invalidated even with a small tweak.

The ball position methodology could also be improved. For plays where the ball was caught, I simply used the player's coordinates as the ball's coordinates. This is probably accurate to where the ball would have hit the ground to within 5 feet or so, but a sufficiently accurate physics-based model could use information on the ball's spin and velocity, as well as atmospheric conditions, to potentially give more accurate theoretical landing coordinates for the ball. This would be a minor improvement since one of the variables is the distance that the player is from where the ball eventually lands. Another simpler model would be to simply take a projection of the ball's location at the time of the catch onto the ground. This would be akin to assuming the ball falls straight down, which would be somewhat accurate depending on the angle of the ball.

Having a projected landing spot, as well as a projected hang time, could allow Dynamic Catch Probability to be calculated in real time. Currently, the play has to have already happened so that the ball's landing position and time is known, and then it can be calculated. However, I don't see

very much utility for this, since having a real time catch probability doesn't tell you anything different analytically, and plays happen so quickly that I don't think it would add much to the viewing experience either.

Another potential improvement would be making this process work for infielders. Some of the methodology would translate directly; you could take the fielder's distance from the ball, the ball's velocity, the direction of the ball, and the fielder's velocity into account. However, this would likely need to be combined with a completely separate model for determining how good the throw was. I don't see this method being useful for analyzing the ball while it's in the air between the infielder and first base, so I envision a Dynamic Fielding Probability that ends when the throw is made and is paired with a throwing quality statistic. But even this process would ignore the complexities of unassisted putouts and double plays for example, just to name a few.

Appendix B: Evidence for Claims

Claim #1: Longer routes lead to naturally higher route efficiencies.

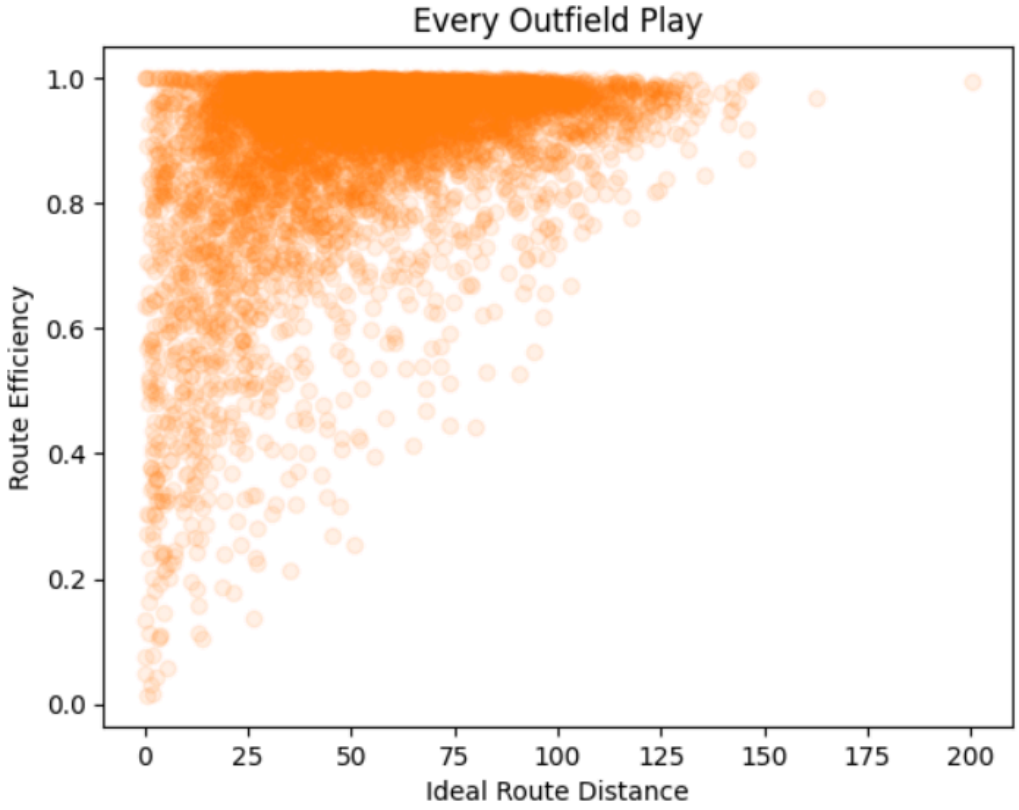

Figure 13: Scatter plot of every outfield play in the data showing the ideal distance and the route efficiency. The pattern is clearly that of a rational function, which makes sense given the formula. Efficiencies clearly trend upwards with longer routes.

Claim #2: The distribution of average route efficiencies amongst players is close to a completely random distribution.

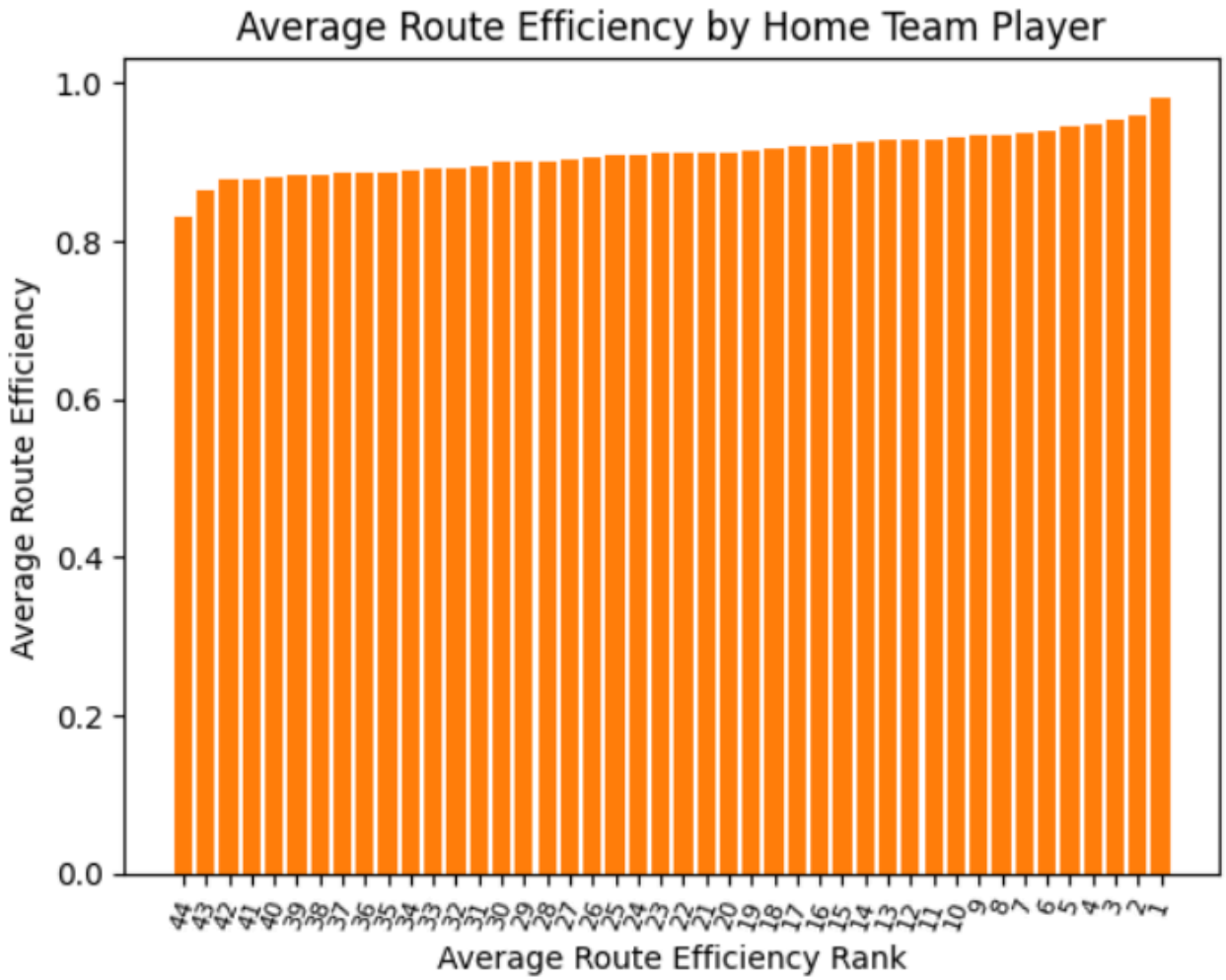

Figure 14: Bar chart showing average Route Efficiencies for each player (44 players total). Only home team players are considered. The x-axis represents the rank for the corresponding player, so the player with the highest average Route Efficiency is on the far right, lowest on the far left.

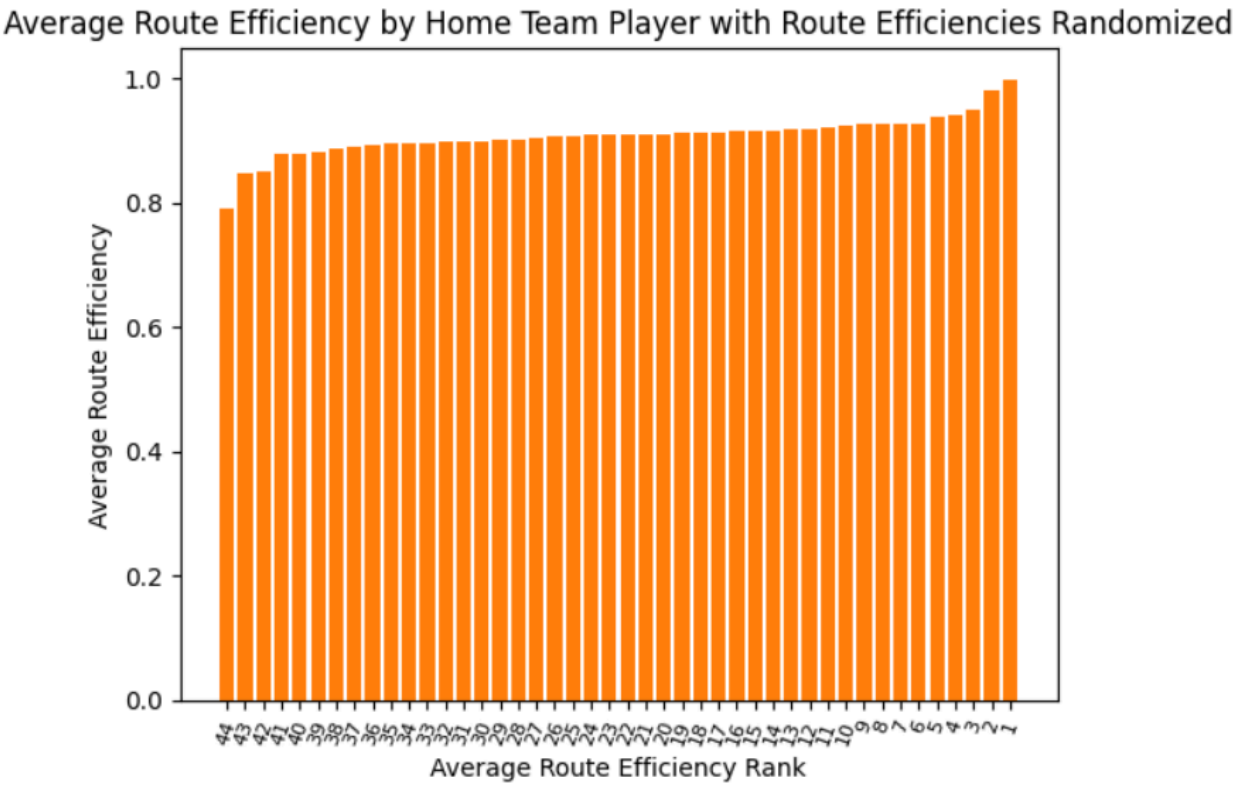

Figure 15: The same chart as above but with randomized Route Efficiencies, which should remove all inherent skill. The distribution looks very similar to the true distribution.

I ran a permutation test on the Route Efficiencies with a test statistic of L1 distance between the true distribution and a uniform distribution of the average Route Efficiency. I had a p-value of 0.354, indicating no statistical significance. This means that the Route Efficiency distribution looks basically random.

Claim #3: The distribution of average jump scores amongst players is sufficiently far from a completely random distribution so as to indicate it being a real skill.

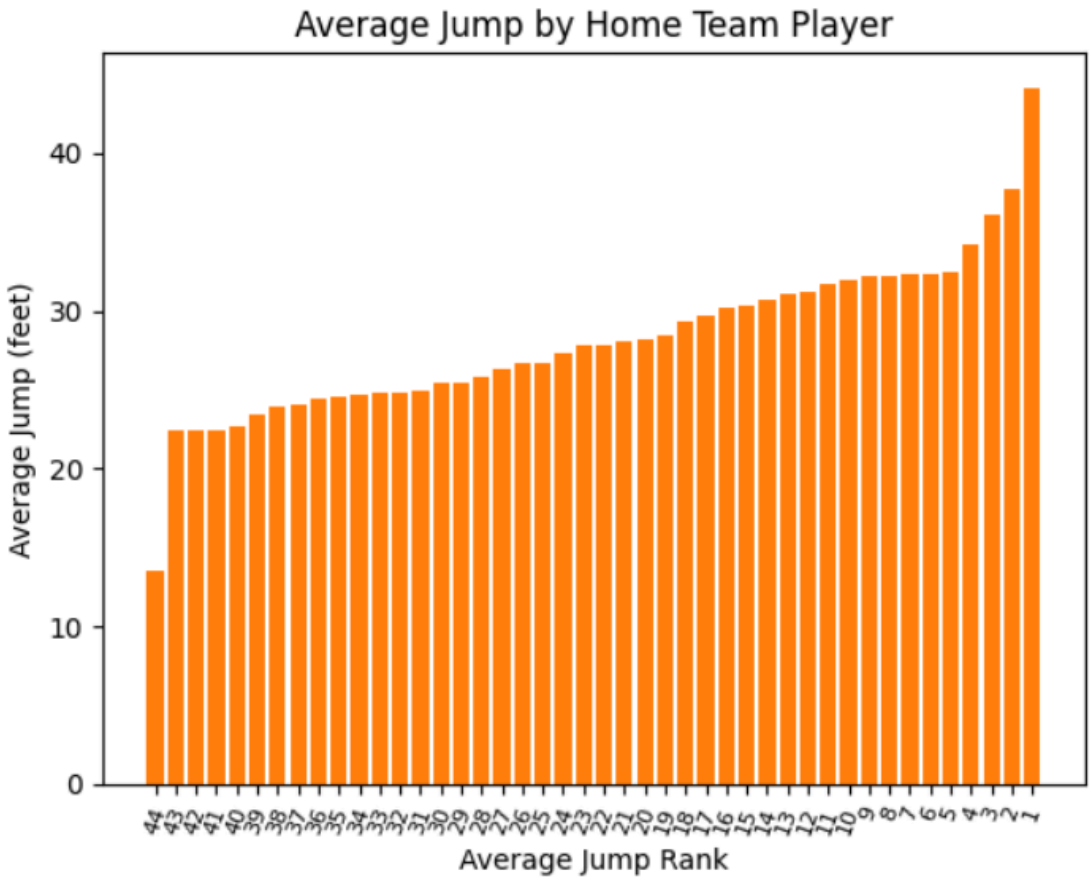

Figure 16: Average jump by player. Only home team players are considered. The x-axis represents the rank of the player that the individual bar represents.

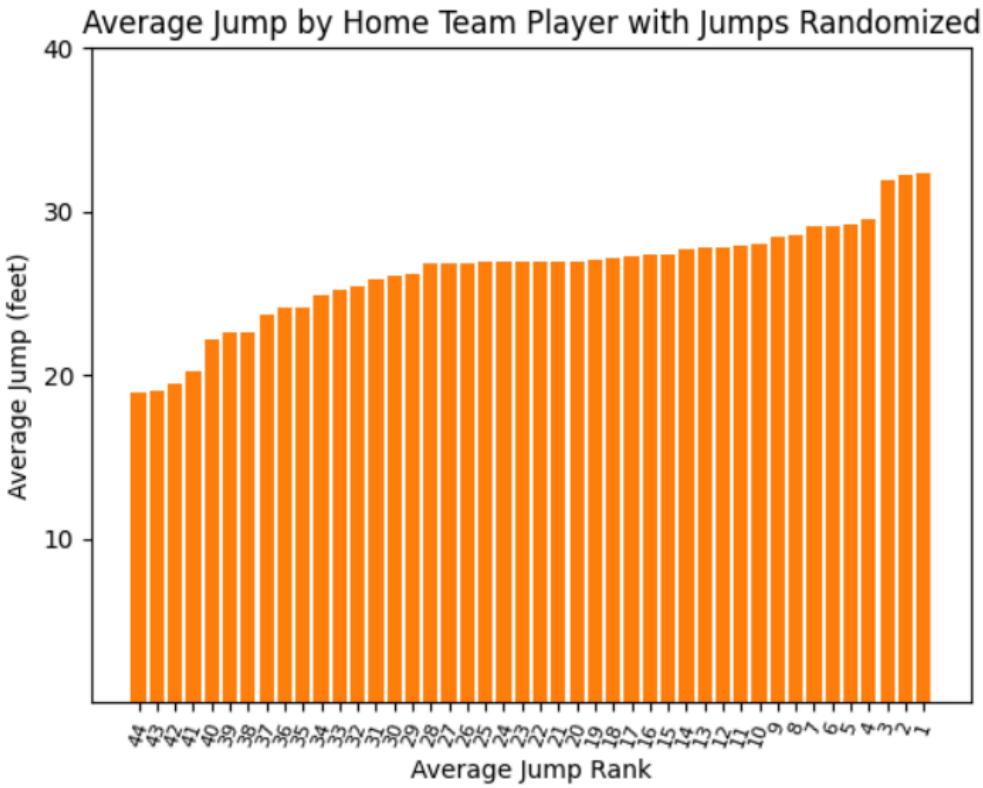

Figure 17: The exact same figure as above but with all of the jumps randomized. This is what a jump distribution would look like if the jumps were random, meaning absolutely zero true skill involved. It looks fairly different from the real distribution.

These two distributions look similar, but they are meaningfully different. I performed a permutation test on the jumps for each play and found a p-value of exactly zero. The test statistic I used was the L1 distance between the true distribution and a uniform distribution where every player's average jump is the overall average jump. I don't think this methodology is flawless, but it gets the point across and is consistent with the MLB data.

Claim #4: My Catch Probability model was correct over 95% of the time.

binary_pred	Catch	No Catch
was_caught		
False	253	3994
True	3975	77

Figure 18: Confusion matrix for my Catch Probability model. The numbers represent the following: 253 balls were predicted to be caught but weren't, 3994 balls were correctly identified

as non-catches, 3975 balls were correctly identified as catches, 77 balls were predicted to not be caught but were.

Simple math. The total number of correct predictions was 3994 + 3975 = 7969. The total number of plays considered was 8299. Simple division shows that 7969/8299 = .96. The exact number varies slightly since the neural network is non-deterministic across different trainings.

Claim #5: Changes in DCP primarily occur at the beginning of the play even when looking only at plays with a specific hang time.

Earlier I made a claim that the beginning of a play is the most important part in terms of changing the probability that a ball is caught. My evidence for this was the higher presence of line overlap on the left middle part of Figure 9. However, it could be argued that this is simply due to there inherently being more lines representing smaller hang times. Figure 19 is the same chart except only plays with a hang time between four and five seconds were considered. We still see the same pattern of the left side having more overlap in the middle percentages, meaning that the probabilities generally converge to either zero or 100 percent as the play goes on. This shows that in fact the beginning is the most important.

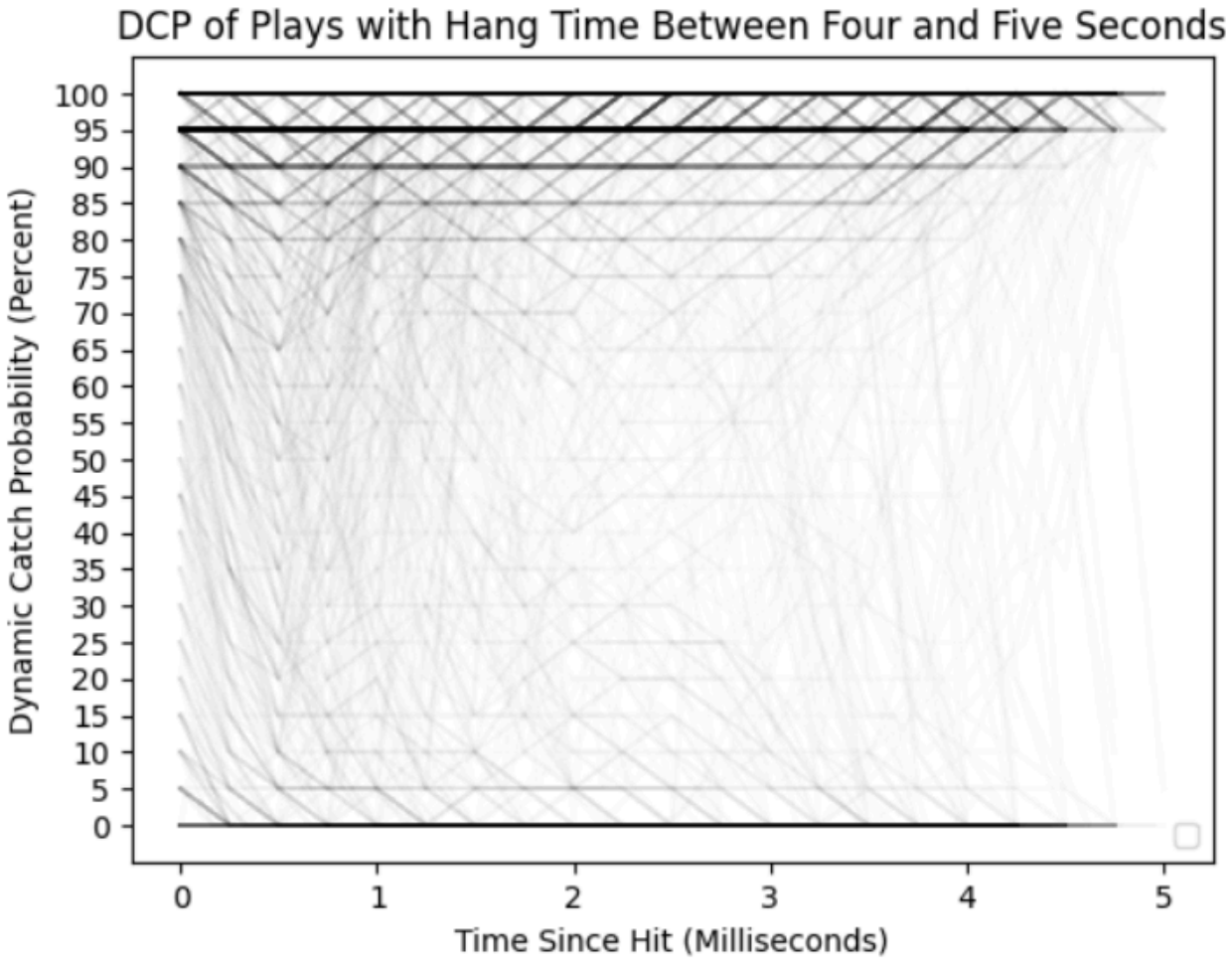

Figure 19: Overlay of the Dynamic Catch Probability graphs for all plays with a hang time between four and five seconds. The pattern of plays generally going to a zero or 100 percent chance of being caught as time goes on persists on this chart, reinforcing my claim about the importance of the jump.

Appendix C: Technical Details

Technical Detail #1: Distance traveled in correct direction

At various times in this paper, I have referred to the distance traveled in the "correct direction." To calculate this quantity, I used the well known formula for finding the length of one vector projected onto another vector.

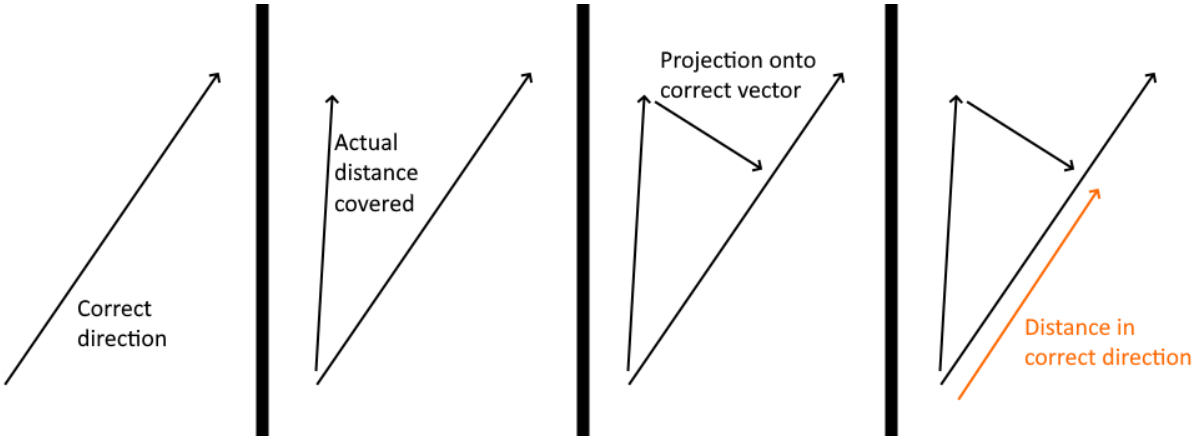

Figure 20: Vector Projection Visualized

If A is the actual vector traveled and B is the correct direction vector, then the distance traveled in the correct direction is simply the dot product of A and B divided by the length of B. After obtaining this number I multiplied by four since I was taking measurements every quarter second and I wanted the units to be feet per second.

Technical Detail #2: Direction traveled calculation

The code to calculate the direction is as follows. It heavily uses the NumPy arctan2 function.

```
def find_direction(start, end):
    direcvec = (end[0] - start[0], end[1] - start[1])
    angle = np.arctan2(start[1], start[0]) - np.arctan2(direcvec[1], direcvec[0])
    if angle < 0:
        angle = angle + 2 * np.pi
    if angle <= np.pi / 4 or angle > 7 * np.pi / 4:
        return "back"
    elif angle <= 3 *np.pi / 4:
        return "left"
    elif angle <= 5 * np.pi / 4:
        return "forward"
    else:
        return "right"
```

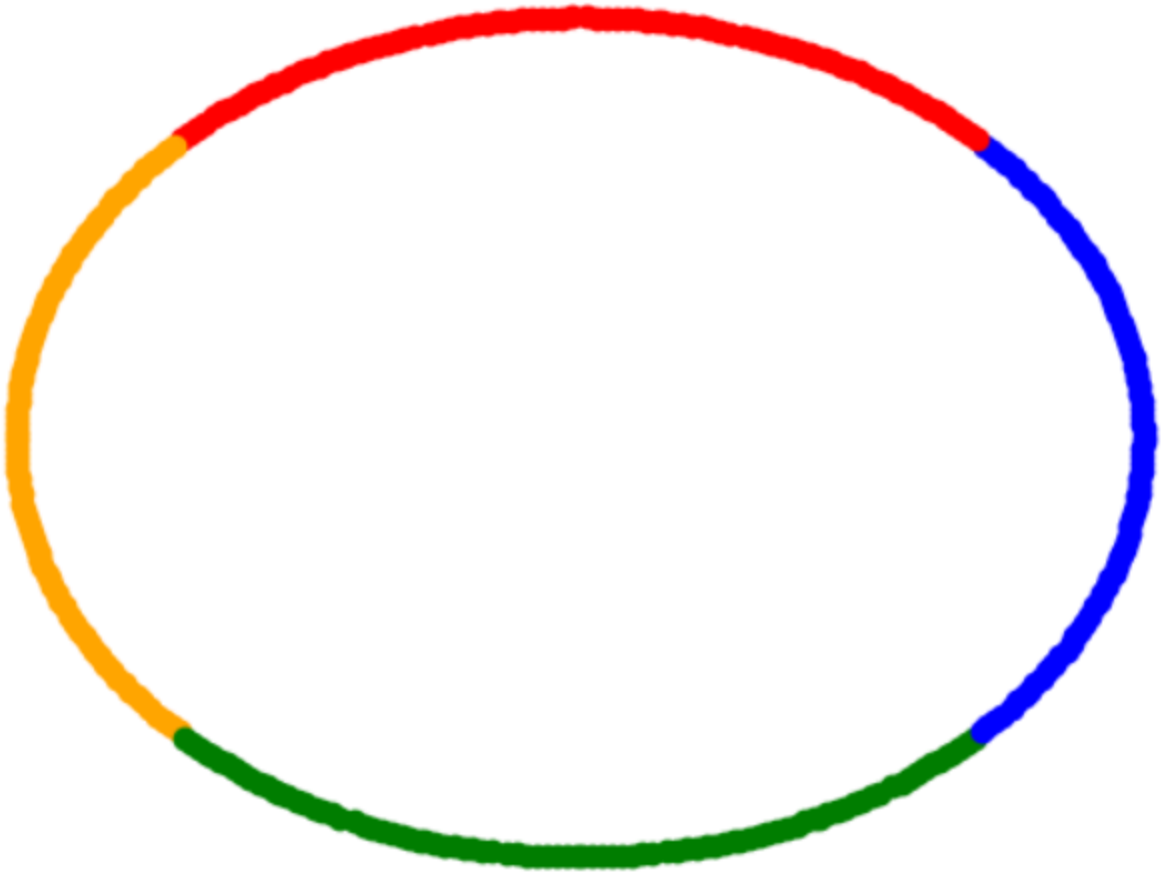

Figure 21: Direction visualization. Red is back, green is forward, blue is left, yellow is right. This is a top down perspective on the outfielder with home plate in the green direction.

Technical Detail #3: Catch Probability neural network

The code for the Catch Probability neural network is as follows. I used a Pipeline from the sklearn Python library. I did a one hot encoding of the direction (dropping the first label) and used a standard scaler.

```
preproc = ColumnTransformer(
    transformers= [
        ("one-hot", OneHotEncoder(drop = "first"), ["direction"])
    ],
    remainder= "passthrough"
)
pl = Pipeline([
    ("preproc", preproc),
    ("scaler", StandardScaler()),
    ("MLP", MLPClassifier())
```

```
])
```

Technical Detail #4: Algorithm for finding relevant plays.

The following conditions are checked in order for each play:

1. The ball was, at some point, acquired by an outfielder
2. The ball was acquired by the outfielder before being acquired by any other player. This eliminates cases of an infielder getting the ball and throwing it to an outfielder.
3. The ball was in fact hit at some point. This eliminates pickoffs and other miscellaneous plays.
4. No infielders interacted with the ball at all before the outfielder acquired the ball. This primarily eliminates cases of the ball deflecting off of an infielder's glove before getting into the outfield. The idea is that if the ball deflected off another player's glove then it isn't the outfielder's fault if they misplayed it so the play is ignored.

This methodology is not perfect. For example, bounces off of the wall were not considered. Errors in the form of straight up drops were also not considered, as I considered them rare enough to not significantly affect the analysis. I'm making the assumption that no player simply drops routine plays enough for it to meaningfully affect their player profile.

Technical Detail #5: Dynamic Catch Probability neural network

The code for the Dynamic Catch Probability neural network is as follows. I used the exact same sklearn Pipeline as for the base Catch Probability. The only difference is that I input an additional variable. The one hot encoding of the direction and the standard scaler remain.

```
preproc = ColumnTransformer(
    transformers = [
        ("one-hot", OneHotEncoder(drop = "first"), ["updated_direction"])
    ],
    remainder= "passthrough"
)
pl = Pipeline([
    ("preproc", preproc),
    ("scaler", StandardScaler()),
    ("MLP", MLPClassifier())
])
```