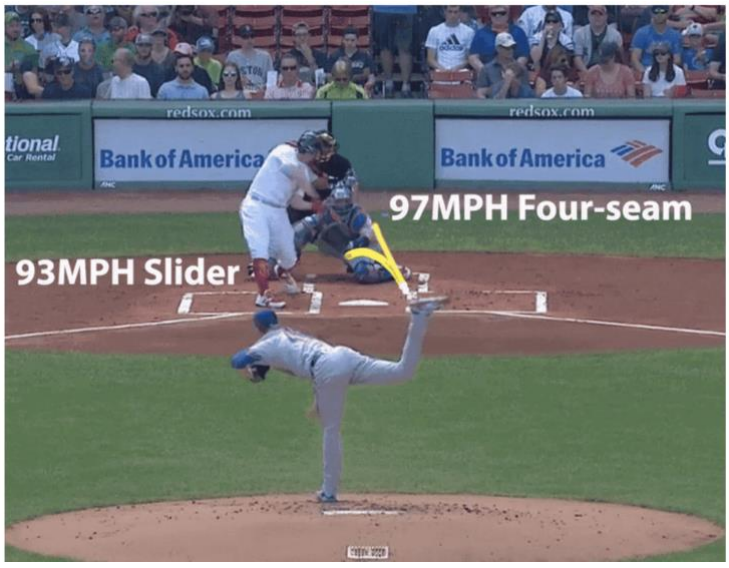
Modeling Pitch Tunneling in Baseball with Deep Learning

COMP642 Final Report
Jacob Lapp (JL372)
jl372@rice.edu

My project combs through an entire season's worth of Major League Baseball data, on the granular level of each individual pitch. Given the extensive amount of data for each pitch (quadratic coefficients of pitch trajectory, coordinates of pitch release and arrival at home plate, pitch spin and type, outcomes of every pitch, etc.), I am deploying a neural network to highlight the most effective tuples of consecutive pitches to try and explore a phenomena known as pitch tunneling. In baseball, effective pitch tunneling is the sequence of throwing consecutive pitches out of the same initial release point and trajectory, as to reduce the amount of time the batter has to identify which pitch is being thrown and if he should swing at it. Using a pitch outcome model developed in a baseball analytics class at Rice[1],

I aim to train a deep neural network on pitch-level data to find which consecutive pairs of pitch features have the highest run value. This will help professional baseball teams make better decisions about which pitchers they have on their team and which pitches they throw during games.

Below is a simple picture illustrating the phenomena of pitch tunneling:



[1] Thank you to Professor Scott Powers for allowing me to participate in the class as well as providing the pitch outcome model itself and the implemented data.

In general, pitch tunneling is still mostly a conjecture amongst coaches and players and does not have extensive research conducted on the subject. Many people have articulated the 'tunnels' and the corresponding pitch shapes that can comprise pitch tunneling.[2] Many of these include overlayed videos of pitches, highlighting the two trajectories, paired with general information about the movement and spin of the pitches. While these are great, intuitive, high-level analyses, we can get much granular and nuanced.

Other more analytical analyses, like done at baseball publication PitcherList, have created a metric to encapture pitch tunneling. PitcherList's metric analyzing pitch tunneling divides the euclidean distance between release points of each pitch by the euclidean distance of the two end coordinates of the two pitches.[3] While this is a sensible way to measure tunneling, there are two main issues:

1. That it measures tunnel value proportionally to distances between two pitches, which essentially focuses pitch sequences to straight fastballs and curving breaking balls. This fails to take into account that offspeed pitches that move less than large curveballs are often better tunneled, since they match the trajectory of the prior pitch longer.

2. This method also only simply measures tunneling in terms of movement. My neural networks train on holistic outcomes corresponding to 'runs' (the score in baseball games) like the expected value, linear weight, run value, etc. of a given pitch.

In total, there are not many analyses conducted on pitch tunneling that utilize machine or deep learning. I will use these techniques to go even further in quantifying this phenomenon.

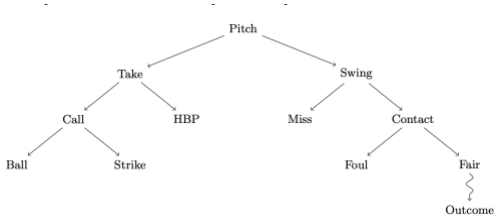
Hypothesis: Using artificial intelligence with a season's worth of pitch-level MLB data can illustrate the features of the most effective two-pitch sequences, aka 'pitch tunneling'. Professional baseball teams can use my model to predict how their current pitchers' pitches will perform, make better decisions about which pitchers are better than others, as well as optimizing which pitches they throw and when.

DATA PREP & MODEL

This analysis is used with entirely publicly-available data. Data can be obtained via the MLB Stats API or other sources such as Baseball Savant. The specific data I use in my project is pitch-level (every single pitch) from the 2023 MLB season. Setting up the data for this project is quite involved and complex, as it involves constructing a 'pitch-outcome model' to

[2] https://alexgravellebusiness.medium.com/a-study-in-pitch-tunneling-bfb7b8363394
[3] https://pitcherlist.com/a-closer-look-at-pitch-tunneling/

systematically assign values to pitches. Given the scope of this project pertains to implementing a deep learning model, I use an implemented pitch-outcome model provided to me by Professor Scott Powers of the Sports Management and Statistics department here at Rice.[4] Below is a tree diagram illustrating the structure of the pitch outcome model, as well as six functions that the model accounts for.



$$f_1(\vec{v},\vec{c}) = \mathbb{P}(\text{Swing} \mid \vec{v},\vec{c})$$
$$f_2(\vec{v},\vec{c}) = \mathbb{P}(\text{HBP} \mid \vec{v},\vec{c},\text{Take})$$
$$f_3(\vec{v},\vec{c}) = \mathbb{P}(\text{Strike} \mid \vec{v},\vec{c},\text{Call})$$
$$f_4(\vec{v},\vec{c}) = \mathbb{P}(\text{Contact} \mid \vec{v},\vec{c},\text{Swing})$$
$$f_5(\vec{v},\vec{c}) = \mathbb{P}(\text{Fair} \mid \vec{v},\vec{c},\text{Contact})$$
$$f_6(\vec{v},\vec{c}) = \mathbb{E}[\text{Outcome} \mid \vec{v},\vec{c},\text{Fair}].$$

A copy of the **R** notebook used to construct the pitch outcome model in SGMT 435 is included in my submitted .zip file (as pitch_outcome_model_R.ipynb) . As the construction of the pitch outcome model is quite laborious, I will only give a high-level explanation of how the pitch-level data was incorporated into Dr. Powers' pitch outcome model, and later my model.

First, to get more complete information about the pitch's trajectory, we fit quadratic equations for every pitch's trajectory, and initiate a column for each coefficient in three dimensions (x, y, and z, totaling 12 columns). Fitting this quadratic model is possible thanks to high-speed cameras located in ballparks that can snap ten pictures of every pitch in the milliseconds it is in the air. Fitting the quadratic then lets us model each pitches' movement (break, spin, etc.) with respect to the forces acting on the baseball (mainly gravity and air resistance).

Then the pitch-outcome model to perform regressions on the above 6 functions that calculate the probability of the events on the tree. This is combined with calculating the run values of each pitch based on the linear weight of the corresponding outcome. Linear weights is a common sabermetric that measures the expected value of each outcome via a recursive sequential Bayesian model that aggregates each count, pitch, and outcome to its expected run value.

L: 0-2	K: 1-2	J: 0-1	I: 2-2	H: 1-1	G: 0-0	F: 1-0	E: 2-1	D: 3-2	C: 2-0	B: 3-1	A: 3-0	Baserunners
												(0 outs)
0.42	0.44	0.47	0.48	0.50	0.51	0.55	0.55	0.59	0.61	0.67	0.74	-- -- --
0.76	0.80	0.84	0.87	0.89	0.90	0.96	0.98	1.03	1.07	1.15	1.25	1B -- --
0.99	1.03	1.10	1.09	1.13	1.15	1.20	1.20	1.18	1.25	1.30	1.38	-- 2B --
1.31	1.34	1.41	1.41	1.47	1.50	1.58	1.57	1.64	1.74	1.83	1.98	1B 2B --
1.23	1.26	1.33	1.34	1.37	1.38	1.44	1.41	1.43	1.48	1.52	1.60	-- -- 3B
1.56	1.60	1.68	1.67	1.74	1.78	1.85	1.82	1.86	1.91	1.98	2.08	1B -- 3B
1.77	1.81	1.91	1.85	1.95	1.98	2.03	1.99	1.97	2.09	2.11	2.19	-- 2B 3B
2.08	2.09	2.21	2.24	2.27	2.32	2.40	2.46	2.59	2.52	2.63	2.89	1B 2B 3B
												(1 outs)
0.21	0.22	0.24	0.25	0.26	0.27	0.30	0.30	0.32	0.35	0.38	0.43	-- -- --
0.41	0.43	0.48	0.49	0.51	0.53	0.57	0.57	0.60	0.64	0.69	0.77	1B -- --
0.55	0.58	0.63	0.63	0.67	0.69	0.73	0.72	0.71	0.79	0.79	0.85	-- 2B --
0.76	0.80	0.86	0.86	0.91	0.93	1.00	1.00	1.06	1.10	1.21	1.33	1B 2B --
0.77	0.80	0.90	0.86	0.93	0.97	1.00	0.99	0.95	1.05	1.06	1.13	-- -- 3B
0.98	1.03	1.13	1.09	1.18	1.21	1.24	1.25	1.24	1.33	1.36	1.44	1B -- 3B
1.10	1.17	1.28	1.22	1.32	1.37	1.42	1.39	1.33	1.48	1.48	1.54	-- 2B 3B
1.27	1.34	1.43	1.46	1.53	1.57	1.68	1.68	1.78	1.82	1.91	2.15	1B 2B 3B
												(2 outs)
0.06	0.07	0.09	0.09	0.10	0.10	0.12	0.12	0.12	0.14	0.16	0.18	-- -- --
0.14	0.16	0.19	0.18	0.21	0.23	0.24	0.24	0.25	0.30	0.32	0.37	1B -- --
0.19	0.22	0.27	0.26	0.31	0.32	0.35	0.34	0.32	0.38	0.39	0.41	-- 2B --
0.26	0.31	0.38	0.38	0.41	0.44	0.49	0.49	0.49	0.57	0.61	0.68	1B 2B --
0.23	0.26	0.32	0.31	0.34	0.36	0.39	0.38	0.37	0.43	0.45	0.49	-- -- 3B
0.31	0.37	0.42	0.44	0.47	0.50	0.55	0.54	0.54	0.63	0.67	0.73	1B -- 3B
0.35	0.39	0.48	0.44	0.53	0.57	0.62	0.57	0.54	0.68	0.69	0.75	-- 2B 3B
0.46	0.54	0.62	0.67	0.74	0.76	0.90	0.89	0.92	1.05	1.18	1.38	1B 2B 3B

In the larger field baseball analytics (sabermetrics), a closely related concept is the RE288 matrix (Run Expectancy for all possible 288 base-out and count scenarios. These matrices are essentially just transition matrices describing the mean change in run value when you move from pitch to pitch in a game. In

[4] SMGT 435: Baseball Analytics, Chapter 5, Scott Powers

the RE288 matrix to the right[5] The columns represent the different counts, the rows correspond to the different runners-on-base situations, and the matrix itself (there are 3) respond to the values for the number of outs.

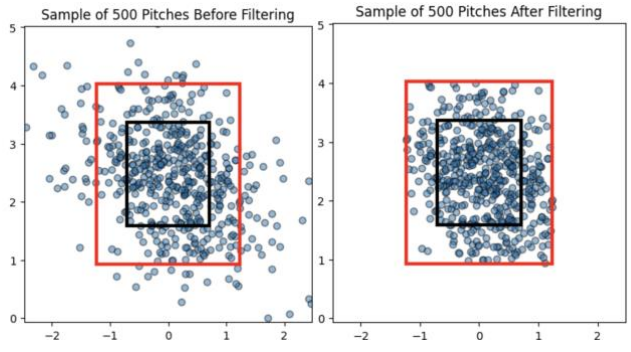
The pitch outcome model works in a very similar manner. Instead of assigning the base-out-count state an expected run value, each *pitch* is assigned a value corresponding to the difference of the count value (measured in runs) before and after the pitch is thrown (if pitch is non-terminal), or the linear weight of the event and the count value (if the pitch is terminal, i.e. leads to change in base-out state). Finally, these 'actual' differences are replaced by the *expected* pitch values, which are the expected linear weight of the pitch conditioned on being fair (see f5 and f6), which is calculated using the quadratic coefficients fitted previously.

Now that the pitch outcome model is fully implemented, the resulting 'data_with_exp_value' is exported. There are just a few small steps before the model can be trained:

I assign 'sequence ids' to each pitch to uniquely identify it so I can group every tuple of consecutive pitches as its own input vector. This process is documented in the 'sequences.ipynb' notebook. Here is a glimpse into the sequence ids:

	sequence1_id	sequence2_id	pitch_value	linear_weight	bat_side	pitch_hand	pitch_type	ax	ay	
629290	<NA>	1	-0.041983	0.474798	R	R	FF	-4.172672	24.453799	-1
396803	1	2	-0.055163	0.474798	R	R	SI	-14.149811	24.987614	-2

Next, we navigate over to the 'pitch_nn.ipynb' notebook, where we import the exported 'sequences.csv' file.



I then filter out sequences where one of the two pitches is outside of an average strike zone expanded by 75%. I do this because my model is training on the difference in pitch value from pitch 1 to pitch 2, so if one of the two pitches is wild (and has very low run value), it can adversely affect the model, as it could reward throwing uncompetitive pitches to maximize change in value. Moreover, to successfully 'tunnel' two consecutive

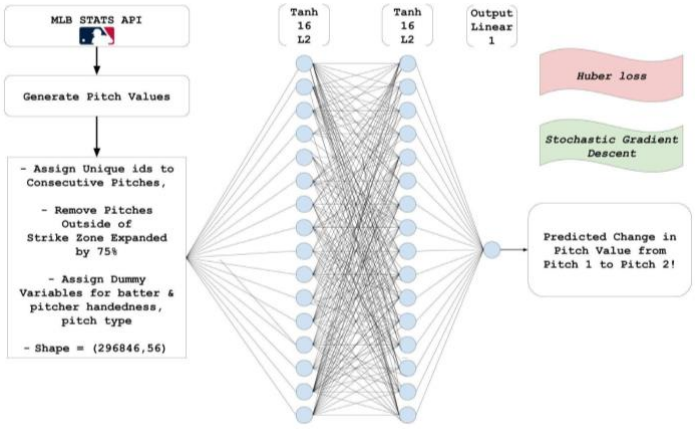
[5] https://www.google.com/imgres?q=re288&imgurl=https%3A%2F%2Fpreview.redd.it%2Fre288-run-expectancy-for-all-possible-scenarios-and-ball-v0-uc55n71v18db1.png%3Fwidth%3D1950%26format%3Dpng%26auto%3Dwebp%26s%3D698810b3dc4fbe0c02c9f96d9bb6903fe17d81a6&imgrefurl=https%3A%2F%2Fwww.reddit.com%2Fr%2Fbaseball%2Fcomments%2F155afsn%2Fre288_run_expectancy_for_all_possible_scenarios%2F&docid=NvkcyAe-v1yT0M&tbnid=0kYhxzGeXj-dIM&vet=12ahUKEwjNjp2ckd6FAxXG_8kDHfZ2AkMQM3oECBYQAA..i&w=1950&h=1247&hcb=2&ved=2ahUKEwjNjp2ckd6FAxXG_8kDHfZ2AkMQM3oECBYQAA

pitches, they by definition need to be relatively close in location (not over the head of the batter, then down around the ankles, for example). Above is a side-by-side of of how the filtering works:

The next part of the data preparation process is to iterate through the rows of the sequence dataframe. If a row's matches the row before, they are concatenated row wise and appended to a new dataframe. The new dataframe is transposed (so now pitch 1 and 2 are columns), indexed by one sequence id. The columns are then condensed into pitch 2 - pitch 1 (for all numeric) and replace the individual pitch features for 1 and 2. Tuples with a pitch identified as knuckleballs, screwballs, forkballs, and ephesus are then removed because they are very very infrequently thrown and could mess with the model.

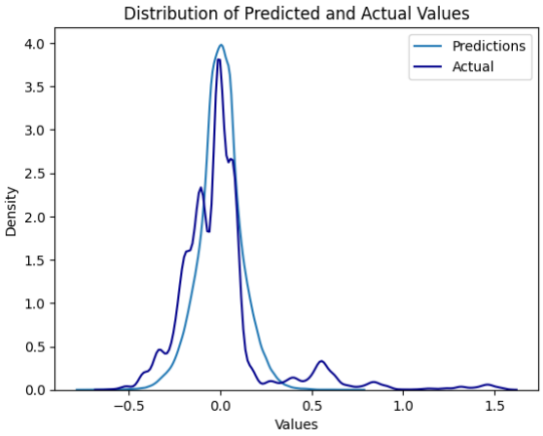
Lastly, the handedness of the batter, the handedness of the pitcher, as well as the pitch type classifications for pitch 1 and pitch 2 are one hot encoded.

Finally, it is time to train the neural network. The target variable is the difference in pitch value between pitch 2 and pitch 1. The data is split into a training (75%) and test (25%) set. The model then trains a neural network on the data with two hidden layers. Both hidden layers are of 16 units, use tanh activation functions, and are regularized with the l2 norm. I chose tanh to prevent a vanishing gradient problem and encourage negative activations. The final layer of the network is a one unit linear activation, since this is a regression task. I use stochastic gradient descent as my gradient optimizer. The loss function is huber, which encourages our target variable (which is loosely normal and centered around 0, no change) to make more varied predictions, instead of always predicting the mean (no difference) by penalizing larger differences linearly, not quadratically (but still quadratically for small errors). Above is the network's architecture

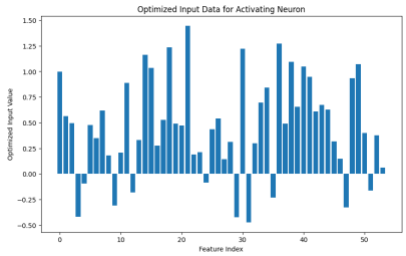


5**. Plots and 1-2 paragraphs summarizing experimental result for each use case**

To the right is a plot of the distributions of the predicted differences in pitch value for the test set. The distributions reflect the predicted change in expected runs based on a pitcher's pitch arsenal. Professional teams can input any two consecutive pitches in a game (or in training), and my model will calculate the expected change in run value based on the data from each pitch. While it matches the distribution of the actual test values pretty well, it tends to cater towards the mean (zero difference). I manually inspected some of the model's predictions and it confirms some of my priors about fastball pairings with cutters and sweepers as being effective sequences.



I also used gradient *ascent* on the input values to randomly generate the optimal feature importance. While this does shed some insight on which input features the network relies on, its randomness changes its results every time it's run. However, first pitch fastballs consistently pop in this feature selector.



Lastly, I employ permutation feature importance to analyze feature significance. This method is also random in nature, and produces slightly different results every time it is run. However, first and second pitch fastballs, first pitch cutters, and first /second pitch sweepers (all pitch types) pop in with this method.

6. 1 paragraph conclusion of why you think you have justified 1 and 3. (Not Required for Mid Report)

All of the above confirm that my model successfully provides MLB teams with the ability to improve their pitchers' pitch selection and identify pitchers that are valuable. I use robust preprocessing methods to accurately value runs in a season's worth of data as well as construct a model to capture a unique and often misunderstood phenomenon in the game with my domain knowledge of the sport as well as technical expertise. This is a viable product that professional teams can use to tailor their gameplans according to their pitchers' strengths and weaknesses, identify value in pitchers on other teams, as well as predict how pitchers will perform in-game with the pitches they throw.

7. (If codes are involved) Submit a zip file containing code, makefile, instructions to execute, and a script that can generate the main plot in the report. **(Not Required for Mid Report)**

- Look at pitch_outcome_r.ipynb to understand pitch outcome model
    - Data is saved in zip as data_with_exp_value.csv
- Load dada_with_exp_value.csv into sequences.ipynb
    - Produces sequences.csv, which is already loaded in zip file
- Load sequences.csv from folder into pitch_nn,ipynb
- Run! (preferably in google colab)