Machine Learning with PITCHF/x

San Diego Padres Baseball R&D Project

By Christian Hanish

Introduction

For this project, I have been provided with data pertaining to 5,000 pitches across 10 unique pitchers. Each entry contains information regarding the pitch's ID, pitcher's ID, pitcher's throwing arm, pitch velocity, vertical/horizontal break, or spin rate. Among the dataset, 80% of the pitches are identified by classification (i.e. fastball, curveball, slider, changeup). Moreover, these measurements have been collected from two different systems. In particular, just 20% of pitches possess measurements from both Systems A and B, while the remaining 80% of entries have measurements from only System A or only System B. Subsequently, in order to achieve the project's overall objective and accurately predict the 1,000 missing pitch classifications in the dataset, I will first need to cleanse and prep the dataset accordingly. In doing so, I will employ several data pre-processing techniques, highlighted by multiple imputation by chained equations (MICE). Once the data has been appropriately prepped, I will construct a k-nearest neighbors (kNN) machine learning algorithm to perform classification prediction on the data missing pitch types. Throughout this process, I will walk you through my steps for designing, testing, and implementing the kNN model in R. Then, I demonstrate how to test the accuracy of our model and investigate how likely our predictions are to be correct. Finally, I will wrap things up by creating several powerful data visualizations in Tableau to showcase our findings.

Data Cleansing & Preparation

The first step in our data pre-processing phase involves segmenting our data according to system and identifying where there is missing data. As previously noted, 20% of pitches possess measurements from both systems while 80% of pitches possess measurements from only one. Thus, when splitting the data into individual datasets (*System A* and *System B*), each set will be comprised of 3,000 entries in total (1,000 with information from *A and B* and 2,000 with information from *A or B*). Once we have our two sets, we will create a function in R for calculating the percentage of missing data that exists in each row and column. The purpose doing so is to ensure that multiple imputation (where the rule of thumb requires no more than 5% missing data in any row or column) is an apprpriate strategy for handling our missing data. Upon building and applying the function to our datasets, we can confirm that no column in either set contains more than 5% missing data. However, the rows are a different story. According to our function's results, there are 3 rows in System A missing 100% of their PITCHF/x measurements. Moreover, there are 119 rows in System B missing at least 25% of such data, as well. Subsequently, we will need to subset out the bad rows. Specifically, in System A we will retain the 2,997 pitches with no missing data and exclude the 3 pitches missing all their data. Similarly, in System B we will retain the 2,975 pitches with ≤ 25% missing data and exclude pitches with > 25% missing data. Now, both datasets have been effectively prepped and are ready for MICE.

Multiple Imputation By Chained Equations (MICE)

While there are a variety of strategies for handling missing data, many of them are not as effective as they may seem due to their propensity for introducing bias into the data. Susbsequently, my personal preference for addressing missing data involves multiple imputation and the MICE package in R. In simplistic terms, the MICE function will figure out what data is missing and how to replace this for us with plausible values. Upon applying the MICE package to the datasets we have prepped above, we will be presented with two different results. When applied to System A, since none of the pitches in the dataset possess missing data, there will be nothing to MICE and no missing data to correct. In other words, System A is already good-to-go. When applied to System B, however, we'll see MICE create five different imputed datasets. Now that we have successfully applied MICE, we will utilize the complete function within the MICE package to take our replaced data and put everything back together. In setting up the complete function, we need to specify which imputed dataset we want to use. While we could pool the imputed results together, for the purposes of this project and enhancing reproducibility I have simply selected one of the imputed datasets instead. Upon completing MICE, we can use the *cbind* function to put our columns and rows back together where necessary. One final note, in order for MICE to be an appropriate strategy the data of interest must be missing completely at random (MCAR), and this is why we chose to exclude pitches that were missing considerable data. Among the excluded pitches in System A, however, there was a pitch in particular (pitch_ID = 410, pitcher_ID = 480028) that was among the 1,000 entries missing a pitch classification and supposed to be included in our analysis. Not only was this pitch missing all measurements from System A, but it was missing all measurements from System B, as well. Subsequently, we can assume that this entry's data is missing not at random (MNAR) and justify excluding it from the MICE processes. Therefore, moving forward there will be 999 missing pitch types. Now, let's build our machine learning model.

pitch_initial_speed_b	spinrate_b	break_x_b
85.5433	NA	-3.176390
92.7773	2352.280	13.300100
93.2883	2485.160	6.468840
85.1234	NA	-6.692920
92.0530	2322.140	1.922050
84.9271	NA	-6.018440
92.0088	2293.690	15.856500

System B: Before MICE

pitch_initial_speed_b	spinrate_b	break_x_b
85.5433	2478.270	-3.176390
92.7773	2352.280	13.300100
93.2883	2485.160	6.468840
85.1234	2412.260	-6.692920
92.0530	2322.140	1.922050
84.9271	1696.780	-6.018440
92.0088	2293.690	15.856500

System B: After MICE

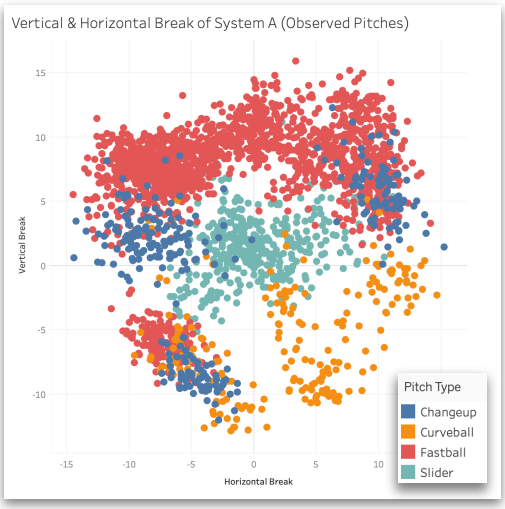
Machine Learning with k-Nearest Neighbors (kNN)

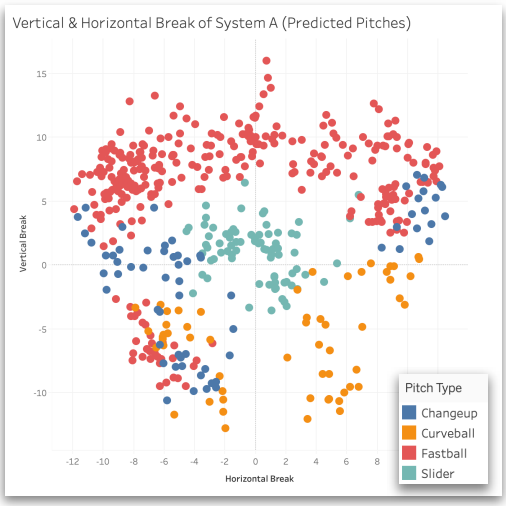
To solve this problem, I will construct a k-nearest neigbhors machine learning algorithm to perform classification prediction on the pitches with missing pitch types. To aid this process, I have taken the liberty of splitting our cleaned data for each system into training and testing sets. Now, we have four datasets: Training A, Testing A, Training B, and Testing B. The pitches within the training sets have corresponding pitch classifications, while the pitches within the testing sets do not. Moving forward, since kNN relies on majority voting rooted in the Euclidean distance, we will need to normalize the values of our features to ensure that the variables with larger values do not have an undue influence on the prediction of our pitch type classifications. In order to normalize the numerical features within our datasets, I have created a normalization function in R. Once defined, I will apply the function to the numerical features within our training and testing sets. Upon successfully normalizing our data, the next steps involve actually constructing and implementing the k-nearest neighbors algorithm.

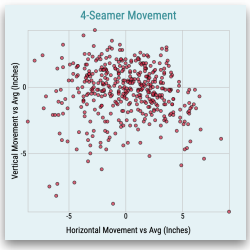
To begin building the model, I will call the *class* package within r. In addition to feeding our training and testing sets into the algorithm, we will need to identify our class and determine the appropriate *k* values (which represents the number of nearest neighbors our model will use to make predictions), as well. In terms of the class, this represents the target feature within our training sets. Thus, in this case the class in both Systems A and B will be the pitch type classification. Moreover, to establish our value for *k*, I will take the square root of the total number of observations within our training sets. This is the rule of thumb, and in this case k will equal 49 for both systems, which is perfect because it is an odd number. Now, we are ready to feed this information into the kNN algorithm. In terms of the arguments for each system: *train* = normalized training set, *test* = normalized testing set, *class* = pitch_type, *k* = 49. Upon running the algorithm it will return the predicted pitch types for the 999 pitches previously missing classifications in our testing sets. Finally, we can use the *cbind* function to combine our pitch classifications with their respective measurements and create our final datasets.

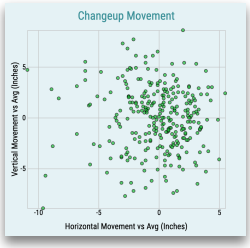
Now that we have made our pitch type classification predictions, we could take things one step further and test the accuracy of such a kNN algorithm against our training dataset for both systems. To do so, we use the same principles to construct a similar kNN algorithm, but instead we would use the training set for everything. Specifically, we could the training set into its own training and testing sets using the 75:25 rule of thumb, calculate a new value for *k*, and use the class of the new training set within our model. Then, we could visualize the results using a confusion matrix to compare our predictions against the actual observations. I have included this code at the end of my R file, as well.

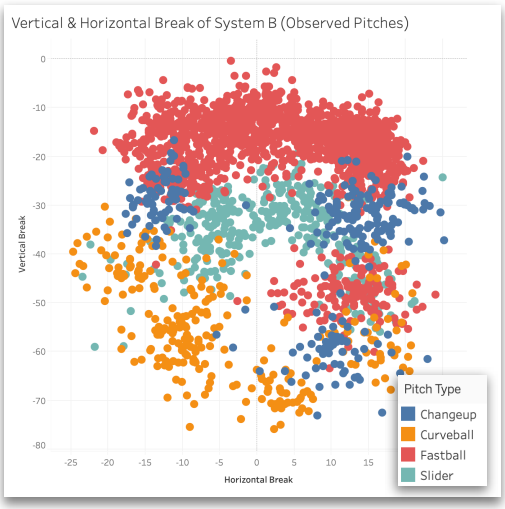
Visualizing Our Findings

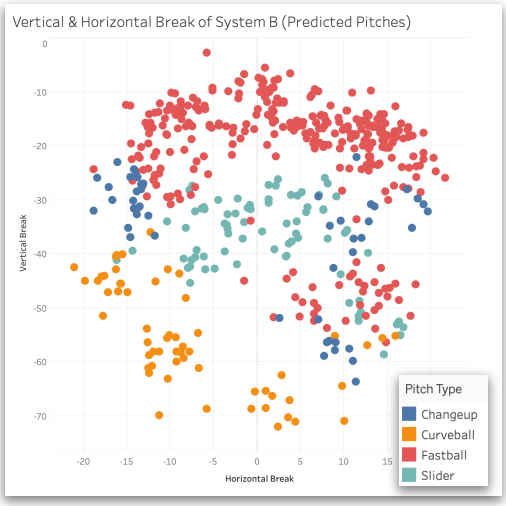


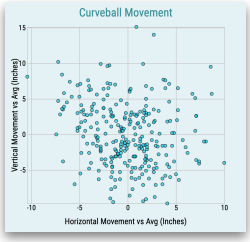


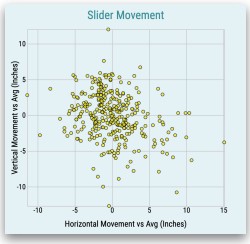












Above are illustrations that I have constructed in Tableau to showcase the trends between pitch type and vertical/horizontal break. From left to right we have the breaks of observed pitch types, predicted pitch types, and Baseball Savant's recorded pitch types from the 2019 MLB season. The purpose of this matrix is to showcase where different pitch types lie on the vertical/horizontal break spectrum, as well as, to give us an idea of how our predictions stack up against pitches in our dataset and real life. As you may notice, the breaks of our predicted pitch types appear to match up quite favorably with trends of observed pitches: fastballs appear to "rise", curveballs appear to "drop", changeups appear to "break horizontally", and sliders appear to have relatively "tight break". Overall, the predictions appear in line with observed trends. Now, if we could only get PITCHF/x and Trackman to agree on their vertical units of measurment.

About the Applicant

Christian Hanish is a recent graduate of Pepperdine University, where he earned his Master of Science (M.S.) in Applied Analytics. While in graduate school, he worked closely with the UCLA & Pepperdine athletics programs in quantitative analysis roles. Recently, he conducted post-graduate research for the Los Angeles Lakers regarding coaching strategy, player development, & prospect evaluation. Moving forward, he hopes to obtain a quantitative analysis position in the MLB.