Classifying Batting Success with KNN: A Study of Expectations versus Reality

Cooper Roach

April 2025

Words: 2183

Pages: 11

Abstract

This paper explores the applicability of the K-Nearest Neighbors (KNN) algorithm to baseball analytics, with a specific focus on batting performance. Using data from MLB Baseball Savant, the study aims to classify player performance based on 88 batting statistics, including weighted On-Base Average (wOBA), On-base Plus Slugging (OPS), and Hard Hit Percentage (HH%). The primary objective is to evaluate the effectiveness of KNN as both a binary and multi-class classification tool for determining player success. While KNN is a relatively simple algorithm, its proven effectiveness in other domains motivates this investigation into its potential utility in baseball analytics.

Introduction

Motivation

Baseball is a game of failure. Those who do well are few and far between, so any insight towards improving the chances of success are very beneficial to coaches and players. Using algorithms to classify data and identify patterns, athletes can get the most out of their performance and no longer be reliant on intuition alone. This project seeks to use the algorithm k nearest neighbors (KNN) to classify baseball players based on data collected from Major League Baseball (MLB) via Baseball Savant.[1] Relevant data will be collected from the 2015 through 2024 seasons, which also feature data collected using Statcast. With this in mind, what is KNN? A brief description of KNN can be understood more easily using an example.

[1] https://baseballsavant.mlb.com

Background

Say one has a random data point and all we know is that it is a fruit, along with many of its corresponding characteristics, and our goal is to classify what type of fruit it is between an apple or a banana. To begin this classification process, one needs to determine how many fruits the unidentified data point will be compared against. Setting the k-value, a predetermined and important value, to be three, the algorithm measures the distance between the characteristics of each fruit and returns a value. Since the value k is three, the closest majority, in this case two, of fruits will determine the classification. If the closest fruits were two apples and one banana, the unidentified fruit would be classified as an apple.

While the previous example works for classifying unlabeled fruits, what happens when we want to measure broad information such as success measures, win/loss prediction, player performance comparisons, etc. Multiple categories, or dimensions, of data are generally needed to make this kind of classification and prediction. This paper will cover the classification of players and the potential problems that result from too much and potential mishandling of information.

Methods

MLB uses Statcast to collect a variety of metrics during a game. Batting data consists of hits, swings-and-misses, contact rating, hit location, etc. All of this data is useful, but not all of it will be used in this study. This study is taking a general approach by determining a benchmark of success and then classifying players against that metric. The raw batting data is 1,375 players (rows) by 88 categories (columns). The goal is to use each of these factors effectively to accurately predict success given a batter versus pitcher matchup. Traditionally, KNN uses a distance metric to classify data, but as the data currently stands, many large problems are primed to arise.

Curse of Dimensionality

High dimensionality creates issues due to something known as the curse of dimensionality Altman and Krzywinski, 2018. This is where data becomes difficult to handle due to data points becoming very spread out, diminishing key differences used to classify the data. To further elaborate, KNN uses distance to classify data; if each point becomes equidistant due to high dimensionality, the model will become ineffective. This problem is greatly shown in the Manhattan distance formula,

$$D = \sum_{i=1}^{n} |x_i - y_i| \tag{1}$$

as results become increasingly similar. The primary ways to fix this are to change the way distance is computed or reduce the dimensionality of the data.

Dimensionality Reduction

To avoid dimensionality-based issues, the data can be filtered by the most impactful variables also referred to as principal components. Principal component analysis (PCA) is one of many means to reduce dimensionality. With this analysis, one can determine which variables maintain and maximize variability while also being non-correlated Jollife and

Codima, 2016. The stats that will be focused on for the application are weighted-on-base-average (wOBA), on-base-plus-slugging (OPS), and hard-hit-percent (HH%). OPS and wOBA cover general batting performance while HH% includes the ability to hit the ball above 95 miles per hour, greatly improving the odds of batting success. Brief PCA results are shown in Figure 1, but this is not the full picture.

```
pc1_sorted = loadings['PC1'].sort_values()
print(pc1_sorted[-40:])
```

```
barrel_batted_rate          0.074856
exit_velocity_avg           0.079534
xiso                        0.085246
popups                      0.087449
xwoba                       0.089961
slg_percent                 0.090147
woba                        0.090236
b_total_swinging_strike     0.090818
xslg                        0.090950
on_base_plus_slg            0.091550
```

Figure 1: PCA Results – wOBA and OPS

An alternative approach would be to use a distance formula that can account for high dimensionality. Baseball also has a lot of dependent statistics greatly impacting the output of the algorithm. For this case, Mahalanobis distance would be most appropriate for retaining meaningful outputs with high dimensionality and non-independent stats Yin et al., 2023.

$$D_M(x) = \sqrt{(x-\mu)^T \cdot S^{-1} \cdot (x-\mu)} \tag{2}$$

While this distance metric is featured later in the article, Manhattan distance is primarily used in combination with PCA. Both approaches share a common ground, normalization of data.

Normalization

This study has many features along with different units and scales that have to be accounted for. Normalization allows for these features to be compared without losing their meaning. A generally accepted practice is Min-Max normalization Pandey and Jain, 2017,

$$X_{new} = \frac{X - min(x)}{max(x) - min(x)} \tag{3}$$

While this could work, information loss will be more likely, especially in terms of variability in the data, due to a vast range of possible values for data points. To avoid this, z-score normalization will be used instead.

$$X_{new} = \frac{X - \mu}{\sigma} \tag{4}$$

Z-score allows for data normalized based on the standard deviation from the mean, but can take on negative values as a result Pandey and Jain, 2017.

K Value Fitting

One of the final key aspects to consider is finding the proper k value. This directly influences the final decision of the algorithm, solidifying its importance in this study. The method for determining the k value used in this study was,

$$k = \sqrt{n}, \qquad \text{n = sample size} \tag{5}$$

As seen in the paper by Pandey and Jain, both min-max and z-score have high accuracy with k values approaching 50. With this data set, the k value $= \sqrt{1375} = 37$ rounded, suggesting both would work, but due to the nature of the baseball data, z-score normalization was chosen.

Procedure

All computations and data analysis were performed using Python in Jupyter Lab. The source code is available on Github.[2] Each player was chosen at random with the exception of Jackson Merrill and Jackson Chourio. Merrill and Chourio made their MLB debuts in 2024, making them good candidates to compare rookie results against players who may have played for a longer time. This normally would not have an effect, but the data has players listed over every season they played in the MLB, thus leading to potential self-comparisons. Comparisons between the same player of different seasons are not an issue, but this study avoids comparing a player's stats with the same season as the input.

KNN typically follows a binary classification set, but for this analysis, both quarterly and binary labels are utilized. Using two different sets of classification will help determine the accuracy of the algorithm in the provided situations, the traditional method and an expanded method. Each tested player will be run through both versions of KNN, one with quarterly labels (1: bad, 2: okay, 3: good, and 4: great) and one with binary labels (0: unsuccessful, and 1: successful). Both labeling methods used weights that were determined based on the magnitude of importance found within the PCA analysis. The weights for wOBA, OPS, and HH% were 0.4, 0.4, and 0.2, respectively.

The expected results are noted in the cell and will be compared against the algorithm's predictions.

Results and Analysis

As previously stated, each player was tested in the algorithm under binary and quarterly labels. Per the findings of Pandey and Jain, I also tested each run with the k values, $k = \sqrt{n}$ and $k = 50$, to test the general approach and the paper's findings. First, though, the results of the PCA will be analyzed.

PCA Results

The PCA supported the idea of using wOBA, OPS, and HH% as metrics that influence performance. One can conclude that higher stats, which implies better performance, leads to greater odds of success. Unfortunately, correlation does not mean causation. While the used metrics were impactful, they were not nearly as significant as what are known as counting

[2] https://github.com/CRoach02/Math_498/tree/main

stats, seen in Figures 2 and 3. These are stats that track the number of times something has occurred rather than quality-of-performance. The PCA results suggest that performance is significantly more tied to the number of attempts rather than the quality of each attempt. Although these findings were not pleasant, quality stats are not so insignificant as to not be worth testing.

```
pc1_sorted = loadings['PC1'].sort_values()
print(pc1_sorted[-40:])
```

```
barrel_batted_rate         0.074856
exit_velocity_avg          0.079534
xiso                       0.085246
popups                     0.087449
xwoba                      0.089961
slg_percent                0.090147
woba                       0.090236
b_total_swinging_strike    0.090818
xslg                       0.090950
on_base_plus_slg           0.091550
groundballs                0.108023
single                     0.126183
b_foul_tip                 0.130517
in_zone_swing_miss         0.134526
b_swinging_strike          0.137320
b_called_strike            0.138444
pitch_count_offspeed       0.142598
out_zone_swing             0.143485
```

Figure 2: PCA Results 1

```
out_zone_swing             0.143485
strikeout                  0.147364
linedrives                 0.148196
walk                       0.154411
home_run                   0.157949
barrel                     0.158840
double                     0.158951
batted_ball                0.162860
flyballs                   0.168199
hit                        0.180444
b_foul                     0.182095
pitch_count_breaking       0.182820
pitch_count_fastball       0.187990
ab                         0.188835
in_zone                    0.192059
r_run                      0.192276
in_zone_swing              0.195078
b_ball                     0.195669
b_total_strike             0.197269
b_total_ball               0.197814
pa                         0.198972
pitch_count                0.203916
out_zone                   0.206485
```

Figure 3: PCA Results 2

KNN – Manhattan Distance

Test 1 – Merrill

Jackson Merrill was the first player tested. As seen below, his binary classification was a 1, 'successful,' and his quarterly label was a 4 'great.' In terms of his expected outputs, Merrill should be classified as a 'successful' and 'great' player, but the results in Figure 4 show otherwise.

```
# knn for Jackson Merrill, 2024 using Manhattan quarters
# Expected Success Level: 4
prediction = knn_predict_quad(knn_data, k, 701538, 2024)
print(f'Success level: {prediction}')
```

```
[4, 1, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4,
1, 1]
Success level: 1
```

```
# knn for Jackson Merrill, 2024 using Manhattan quarters
# Expected Success Level: 4
prediction = knn_predict_quad(knn_data, 50, 701538, 2024)
print(f'Success level: {prediction}')
```

```
[4, 1, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4,
1, 1, 4, 4, 1, 1, 4, 1, 4, 4, 4, 1, 1, 1]
Success level: 1
```

```
# knn for Jackson Merrill, 2024 using Manhattan binary
# Expected Success Level: 1
prediction = knn_predict_bin(knn_data, k, 701538, 2024)
print(f'Success level: {prediction}')
```

```
[1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0]
Success level: 0
```

```
# knn for Jackson Merrill, 2024 using Manhattan binary
# Expected Success Level: 1
prediction = knn_predict_bin(knn_data, 50, 701538, 2024)
print(f'Success level: {prediction}')
```

```
[1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0]
Success level: 0
```

Figure 4: KNN Manhattan – Merrill

Thus far, both versions of the Manhattan-based KNN algorithm were incorrect in their predictions. Neither the initial k-value, nor $k = 50$ produced satisfactory results. This may be a product of Merrill being a rookie, but at this time, the cause of the error remains undetermined. Assumptions can be made based on the PCA results, but again, this is a test of average quality per offensive interaction rather than by number of total offensive interactions.

Test 2 – Chourio

Jackson Chourio, like Merrill, is a rookie that debuted in the 2024 season as well. Running his statistics through the algorithm will help with determining potential causes for the lack of accuracy present in Merrill's predictions. Chourio's results, Figure 5, have failed the test as well. So far, the algorithm has shown to be inaccurate in two test cases, although both were fairly similar in terms of background. Selecting a player at random may help glean a better insight into what is happening with the algorithm, as expected outputs are not matching predicted or real outputs.

```
# knn for Jackson Chourio, 2024 using Manhattan quarters
# Expected Success Level: 3
prediction = knn_predict_quad(knn_data, k, 701538, 2024)
print(f'Success level: {prediction}')
```

```
[4, 1, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4,
1, 1]
Success level: 1
```

```
# knn for Jackson Chourio, 2024 using Manhattan quarters
# Expected Success Level: 3
prediction = knn_predict_quad(knn_data, 50, 701538, 2024)
print(f'Success level: {prediction}')
```

```
[4, 1, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4,
1, 1, 4, 4, 1, 1, 4, 1, 4, 4, 4, 1, 1, 1]
Success level: 1
```

```
# knn for Jackson Chourio, 2024 using Manhattan binary
# Expected Success Level: 1
prediction = knn_predict_bin(knn_data, k, 701538, 2024)
print(f'Success level: {prediction}')
```

```
[1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0]
Success level: 0
```

```
# knn for Jackson Chourio, 2024 using Manhattan binary
# Expected Success Level: 1
prediction = knn_predict_bin(knn_data, 50, 701538, 2024)
print(f'Success level: {prediction}')
```

```
[1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0]
Success level: 0
```

Figure 5: KNN Manhattan – Chourio

Test 3 – Grossman

The third and final player was selected at random, which happened to be Eric Grossman. Based on his known data, he should be predicted as a low-performance player, 'unsuccessful' and 'okay.' Grossman's results are found in Figure 6, below. The quarterly labeled classification failed, but it was helpful for another reason. When looking at the binary prediction, the result was correct, but this is likely due to an over-sized range of values. Looking at the nearest values from the quarterly prediction, Grossman should be in the 'okay' range of performance, but it is likely that the binary output is more in line with 'bad' performance.

```
# knn for Robbie Grossman, 2021 using Manhattan quarters
# Expected Success Level: 2
prediction = knn_predict_quad(knn_data, k, 543257, 2021)
print(f'Success level: {prediction}')
```

```
[4, 1, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4,
1, 1]
Success level: 1
```

```
# knn for Robbie Grossman, 2021 using Manhattan quarters
# Expected Success Level: 2
prediction = knn_predict_quad(knn_data, 50, 543257, 2021)
print(f'Success level: {prediction}')
```

```
[4, 1, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4,
1, 1, 4, 4, 1, 1, 4, 1, 4, 4, 4, 1, 1, 1]
Success level: 1
```

```
# knn for Robbie Grossman, 2021 using Manhattan binary
# Expected Success Level: 0
prediction = knn_predict_bin(knn_data, k, 543257, 2021)
print(f'Success level: {prediction}')
```

```
[1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0]
Success level: 0
```

```
# knn for Robbie Grossman, 2021 using Manhattan binary
# Expected Success Level: 0
prediction = knn_predict_bin(knn_data, 50, 543257, 2021)
print(f'Success level: {prediction}')
```

```
[1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0]
Success level: 0
```

Figure 6: KNN Manhattan – Grossman

Manhattan Analysis

Using Manhattan distance with KNN has failed to work with the current data set. Thus far, the only correct prediction has been when the label ranges were at their widest. The normalization method could be causing the problem, but there is not enough evidence to make this conclusion. To further test KNN, Mahalanobis distance can be used instead to handle the variables of interest as they are non-independent.

KNN – Mahalanobis Distance

Figures 7, 8, and 9 display the prediction results of each player using Mahalanobis distance for the algorithm. Analyzing the results shows that the algorithm has failed under the second distance metric. Merrill, who should be labeled as 'great' and 'successful,' was labeled 'bad' and 'unsuccessful,' Figure 7. Chourio, Figure 8, although a generalization, shared the outcome as Merrill. Grossman's predictions, Figure 9, were nearly the opposite of what the expected results were. Both algorithms have failed to perform the tasks required of them.

```
# knn for Jackson Merrill, 2024 using Mahalanobis quarter
# Expected Success Level: 4
prediction = knn_predict2_quad(knn_data, k, 701538, 2024, inverse_covariance_matrix)
print(f'Success level: {prediction}')
```

```
[1, 4, 1, 2, 4, 1, 1, 4, 4, 2, 1, 1, 1, 1, 1, 2, 4, 4, 1, 4, 1, 2, 2, 1, 1, 1, 4, 1, 4, 1, 1, 2, 1, 1,
1, 2]
Success level: 1
```

```
# knn for Jackson Merrill, 2024 using Mahalanobis quarter
# Expected Success Level: 4
prediction = knn_predict2_quad(knn_data, 50, 701538, 2024, inverse_covariance_matrix)
print(f'Success level: {prediction}')
```

```
[1, 4, 1, 2, 4, 1, 1, 4, 4, 2, 1, 1, 1, 1, 1, 2, 4, 4, 1, 4, 1, 2, 2, 1, 1, 1, 1, 4, 1, 4, 1, 1, 2, 1, 1,
1, 2, 4, 2, 4, 1, 2, 1, 3, 4, 4, 1, 1, 1, 1]
Success level: 1
```

```
# knn for Jackson Merrill, 2024 using Mahalanobis binary
# Expected Success Level: 1
prediction = knn_predict2_bin(knn_data, k, 701538, 2024, inverse_covariance_matrix)
print(f'Success level: {prediction}')
```

```
[0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
0, 0]
Success level: 0
```

```
# knn for Jackson Merrill, 2024 using Mahalanobis quarter
# Expected Success Level: 1
prediction = knn_predict2_bin(knn_data, 50, 701538, 2024, inverse_covariance_matrix)
print(f'Success level: {prediction}')
```

```
[0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0]
Success level: 0
```

Figure 7: KNN Manhattan – Grossman

```
# knn for Jackson Chourio, 2024 using Mahalanobis quarters
# Expected Success Level: 3
prediction = knn_predict2_quad(knn_data, k, 701538, 2024, inverse_covariance_matrix)
print(f'Success level: {prediction}')
```

```
[1, 4, 1, 2, 4, 1, 1, 4, 4, 2, 1, 1, 1, 1, 1, 2, 4, 4, 1, 4, 1, 2, 2, 1, 1, 1, 4, 1, 4, 1, 1, 2, 1, 1,
1, 2]
Success level: 1
```

```
# knn for Jackson Chourio, 2024 using Mahalanobis quarters
# Expected Success Level: 3
prediction = knn_predict2_quad(knn_data, 50, 701538, 2024, inverse_covariance_matrix)
print(f'Success level: {prediction}')
```

```
[1, 4, 1, 2, 4, 1, 1, 4, 4, 2, 1, 1, 1, 1, 1, 2, 4, 4, 1, 4, 1, 2, 2, 1, 1, 1, 4, 1, 4, 1, 1, 2, 1, 1,
1, 2, 4, 2, 4, 1, 2, 1, 3, 4, 4, 1, 1, 1]
Success level: 1
```

```
# knn for Jackson Chourio, 2024 using Mahalanobis binary
# Expected Success Level: 1
prediction = knn_predict2_bin(knn_data, k, 701538, 2024, inverse_covariance_matrix)
print(f'Success level: {prediction}')
```

```
[0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
0, 0]
Success level: 0
```

```
# knn for Jackson Chourio, 2024 using Mahalanobis binary
# Expected Success Level: 1
prediction = knn_predict2_bin(knn_data, 50, 701538, 2024, inverse_covariance_matrix)
print(f'Success level: {prediction}')
```

```
[0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0]
Success level: 0
```

Figure 8: KNN Manhattan – Grossman

```
# knn for Robbie Grossman, 2021 using Mahalanobis quarters
# Expected Success Level: 2
prediction = knn_predict2_quad(knn_data, k, 543257, 2021, inverse_covariance_matrix)
print(f'Success level: {prediction}')
```

```
[4, 4, 4, 4, 1, 4, 4, 4, 3, 1, 4, 4, 1, 1, 3, 2, 4, 4, 1, 1, 1, 1, 3, 2, 4, 4, 4, 2, 4, 3, 3, 4, 3, 2, 4,
4, 4]
Success level: 4
```

```
# knn for Robbie Grossman, 2021 using Mahalanobis quarters
# Expected Success Level: 2
prediction = knn_predict2_quad(knn_data, 50, 543257, 2021, inverse_covariance_matrix)
print(f'Success level: {prediction}')
```

```
[4, 4, 4, 4, 1, 4, 4, 4, 3, 1, 4, 4, 1, 1, 3, 2, 4, 4, 1, 1, 1, 1, 3, 2, 4, 4, 4, 2, 4, 3, 3, 4, 3, 2, 4,
4, 4, 4, 4, 4, 3, 4, 4, 3, 4, 1, 2, 1, 1, 4]
Success level: 4
```

```
# knn for Robbie Grossman, 2021 using Mahalanobis binary
# Expected Success Level: 0
prediction = knn_predict2_bin(knn_data, k, 543257, 2021, inverse_covariance_matrix)
print(f'Success level: {prediction}')
```

```
[1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
1, 1]
Success level: 1
```

```
# knn for Robbie Grossman, 2021 using Mahalanobis binary
# Expected Success Level: 0
prediction = knn_predict2_bin(knn_data, 50, 543257, 2021, inverse_covariance_matrix)
print(f'Success level: {prediction}')
```

```
[1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1]
Success level: 1
```

Figure 9: KNN Manhattan – Grossman

Discussion

This project is limited in many ways, including but not limited to data collection, generalization, and data and model accuracy. MLB allows access to pitch by pitch data which would help analyze the game to a more precise degree. My hardware would not support this level of analysis nor would it run in a timely manner if it could. As more types of data are collected, KNN becomes less and less effective due to similarities among data being washed out. One would really have to consider practices for reducing the data's dimensionality while also being able to properly account for statistical influences.

Conclusion

KNN, following this study's implementation, was not a good fit for predicting and classifying player metrics. For a multitude of possible reasons, the application of the model has failed. While failure is not an easy pill to swallow, it does point towards learning better alternative models. As seen in the PCA results, baseball is greatly influenced by a degree of luck and the number of attempts a player gets. Models such as random forest or shallow neural networks (minimal hidden layers) would account for and handle extraneous factors much better than KNN. While the expectation was to potentially predict player success by classification, the reality is that this kind of operation is best performed by other models.

References

Altman, N., & Krzywinski, M. (2018). The curse(s) of dimensionality. *Nature Methods*, *15*, 397–400.

Jollife, I. T., & Codima, J. (2016). Principle component analysis: A review and recent developments. *Philosophical transactions of the Royal Society of London. Series A: Mathematical, physical, and engineering sciences*, *374*, 2015202.

Pandey, A., & Jain, A. (2017). Comparative analysis of knn algorithm using various normalization techniques. *International journal of computer network and information security*, *11*, 36–42.

Yin, L., Lv, L., Dingyi Wang, Y. Q., Chen, H., & Deng, W. (2023). Spectral clustering approach with k-nearest neighbor and weighted mahalanobis distance for data mining. *Electronics*, *12*(15).