Splitter, Sinker, or ... Splinker?

Identifying Paul Skenes' Pitch Using Various Classification Methods

Will Scheib

Contents

1 Problem Statement

Baseball has undergone a recent analytics revolution. In particular, as recently as 2015, Major League Baseball (MLB) launched Statcast, "a state-of-the-art tracking technology that allows for the collection and analysis of a massive amount of baseball data."[1] Statcast built upon the existing pitch tracking hardware that was installed in every MLB ballpark in 2008. Also, a physics paper by Andrew W. Smith[2] from 2020 changed the paradigm when it comes to understanding the movement of a baseball during a pitch.

A few commonly known pitch types are the fastball, curveball, and slider. Also important for this paper are the sinker, a type of fastball that has hard downward and/or arm-side movement (movement toward the pitcher's throwing arm), and the splitter, a type of offspeed pitch (of which the curveball and slider are also examples) that is thrown like a fastball but with a grip that emphasizes minimizing spin in order to cause a late downward, dropping motion.

Due to the availability of nuanced pitch data and the improved understanding of ball movement, pitching in major league baseball has entered a renaissance, where once-unknown and misunderstood factors can be controlled in much more detail to produce desired pitch spin and movement, so that a pitcher's "arsenal," the collection of pitches that he throws, can complement each other well by varying velocity and movement. In many cases, this process results in the creation of new styles of pitch that have never or rarely been seen in MLB. A great case study of this new age pitch development is Paul Skenes, a 21-year-old superstar pitcher for the Pittsburgh Pirates. He has become well known for a pitch many sports media personalities are dubbing a "splinker," a hybrid pitch somewhere between a traditional splitter and sinker.

Since Statcast's launch, many statisticians have written about classifying pitches using various clustering methods and model-based classification methods. For example, an article by Camp et al[3] explores classifying pitch subtypes using K-Means clustering. A second notebook by Chang[4] uses DBSCAN. A third by Mancuso[5] uses Naive Bayes. Many classification techniques have been tried, but few comparison papers exist.

In this paper, I intend to compare five binary classification methods – K Nearest Neighbors, Logistic Regression, Linear Support Vector Machine (SVM), Kernel SVM, and Random Forest – by evaluating which method is most successful at classifying major league sinkers and splitters in general. Finally, I will compare how they classify Paul Skenes' splinker in particular.

2 Data Source

Thankfully, much of Statcast's data is available to the public for free via Baseball Savant, a data search service provided by MLB. There are a handful of independently written Python packages that automate the download process for large queries, and I chose to use pybaseball. I used the appropriate query settings to obtain MLB pitch by pitch data for the years 2023 and 2024, which balances large enough sample size with reasonable computation time. Because over 40,000 sinkers and splitters are thrown in a typical MLB season, sample size is not an issue with the smaller time window, and the runtime of my code is much improved.

The variables I will be considering in my analysis are:

1. p_throws (R or L): Whether the pitcher throws right handed or left handed.
2. release_speed (MPH): The measured velocity of the pitch.
3. effective_speed (MPH): The velocity of the pitch as perceived by the batter (a function of Pitch Velocity and Extension).
4. release_spin_rate (RPM): The number of rotations per minute on the ball.
5. release_pos_x (ft): How far off the center line the pitcher releases the ball.
6. release_extension (ft): How far along the line from the pitcher's mound to home plate the pitcher releases the ball.
7. plate_x (ft): How far from the center of home plate the pitch crosses the front of home plate.
8. plate_z (ft): How high off the ground the pitch crosses the front of home plate.

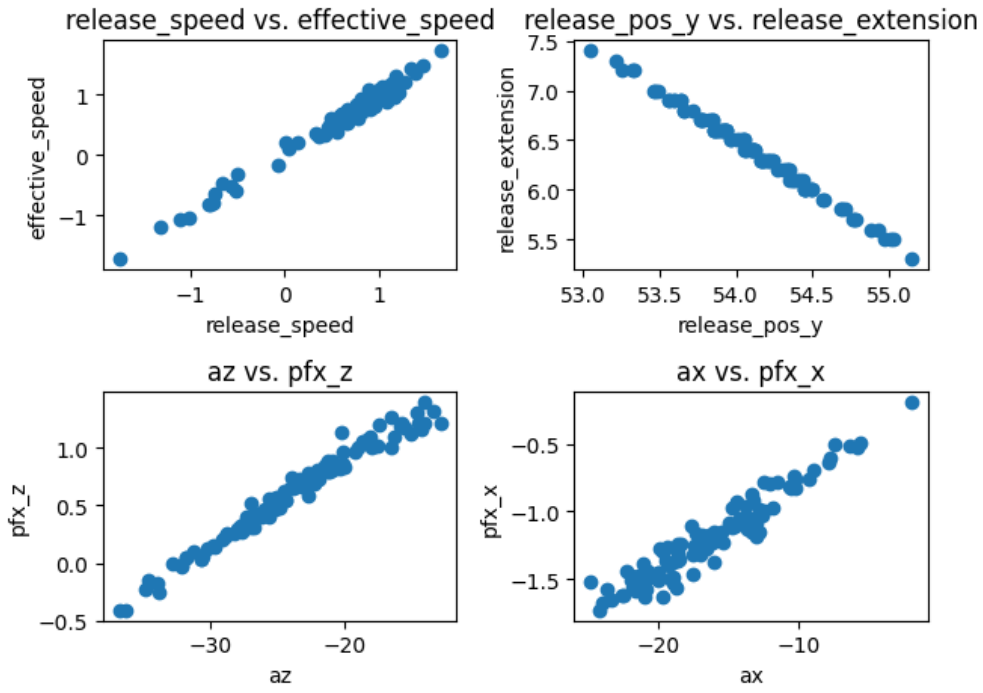
9. pfx_x (ft): Horizontal movement in feet from the catcher's perspective.
10. pfx_z (ft): Vertical movement in feet from the catcher's perspective.
11. vx0 (ft/s): The velocity of the pitch, in feet per second, in the x-dimension, determined at y=50 feet.
12. vy0 (ft/s): The velocity of the pitch, in feet per second, in the y-dimension, determined at y=50 feet.
13. vz0 (ft/s): The velocity of the pitch, in feet per second, in the z-dimension, determined at y=50 feet.
14. ax (ft/s^2): The acceleration of the pitch, in feet per second per second, in the x-dimension, determined at y=50 feet.
15. ay (ft/s^2): The acceleration of the pitch, in feet per second per second, in the y-dimension, determined at y=50 feet.
16. az (ft/s^2): The acceleration of the pitch, in feet per second per second, in the z-dimension, determined at y=50 feet.

I expected to encounter some multicollinearity because these metrics are somewhat vaguely defined on MLB's website, and I believed it to be quite likely that some pairs of variables described the same information.

3 Addressing Multicollinearity

3.1 Visual Inspection

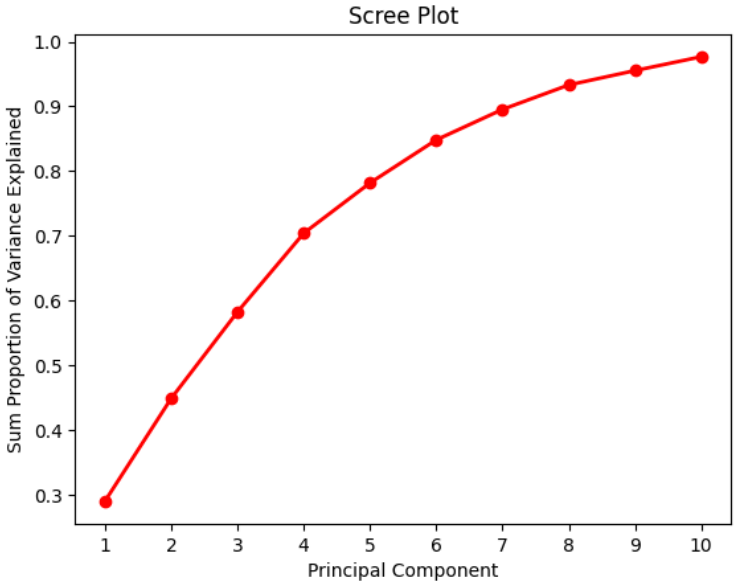
The first step I took to eliminate multicollinearity was to plot a sample of the data on pair plots. After visually inspecting these pair plots, the following pairs of variables could be clearly indentified as describing similar information:



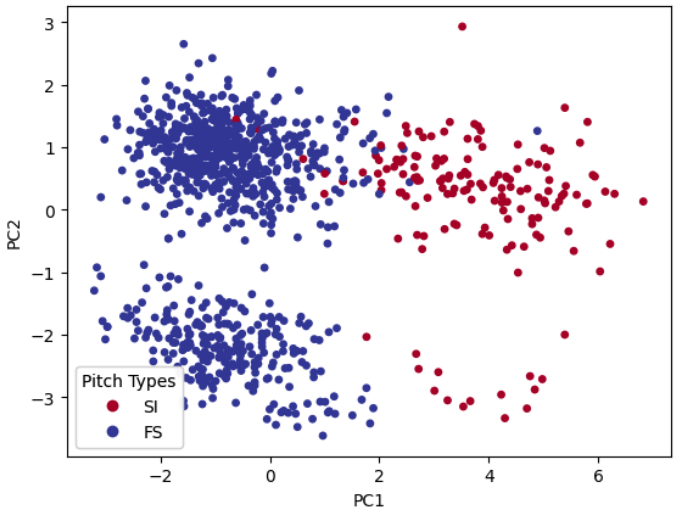
After observing these relationships, I came to the conclusion that I should drop one of each aforementioned pair of variables. I chose to drop release_speed, release_pos_y, pfx_x, and pfx_z.

3.2 Principal Component Analysis

Before fitting the models, as an extra measure of protection against multicollinearity, I performed a Principal Component Analysis (PCA). First, I used a scree plot to look at the sum proportion of explained variance over the principal components. I was looking for an elbow in the plot where the additional explained variance gained by adding another principal component becomes small. In the absence of a clear elbow point, I set a threshold at 90% of the variance having been explained by the principal components.



In this scree plot, there is not an obvious elbow point, so I chose to select eight principal components. 93.3% of the variance in the original twelve predictors is explained by the eight principal components. Here is a visualization of a sample of the data plotted using the first two principal components:



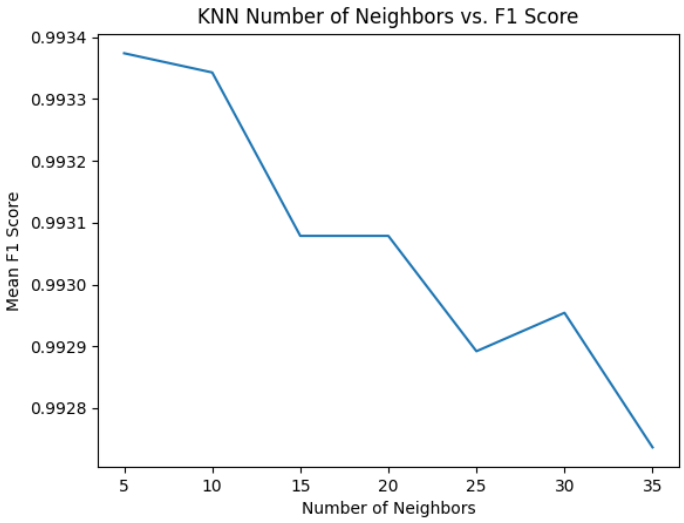
As you can see, even just the first principal component does a respectable job of separating the data, which is a good sign for the models' ability to classify the data correctly.

4 Cross Validation

Cross validation is a technique of optimizing hyperparameters of machine learning models to result in the best possible performance. When evaluating the performance of a model, it is important not to train the model on the same data that it will be tested on. To avoid this problem, I used k-Fold Cross Validation with $k = 5$. As an evaluation metric, I used F1 score because the data set was somewhat unbalanced, including over 54,000 sinkers, but only about 10,000 splitters. Because F1 score is a combination of precision and recall, which both tend to be more effective evaluation metrics than accuracy when working with an imbalanced response variable, it was a good choice for this situation. Each of the cross validation summaries will include one or more plots with various values of a hyperparameter on the x axis and mean F1 score on the y axis. The mean F1 score is the average test F1 score across all 5 folds.

4.1 K Nearest Neighbors

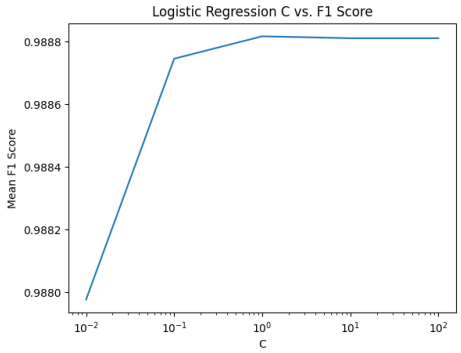
For the K Nearest Neighbors model, the only hyperparameter I needed to optimize was the number of neighbors that would be used to classify a data point. I tested a range from 5 to 35, stepping by 5. The resulting mean F1 scores can be seen in the following plot:

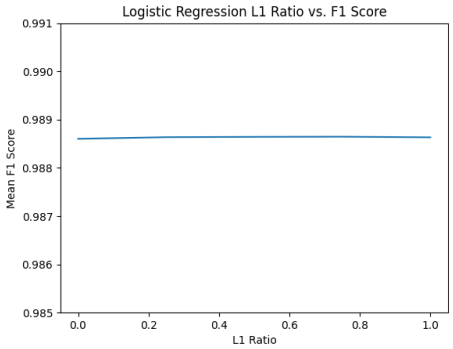


From this plot, I concluded that the best choice for the number of neighbors hyperparameter was 5. It is worth noting, however, that I needed to stretch the y of the plot to see a difference, because all of the number of neighbors values that I tested resulted in very high F1 scores, between 0.985 and 0.99.

4.2 Logistic Regression

For the Logistic Regression model, I needed to optimize two hyperparameters: C and ℓ_1 ratio. C is the inverse of regularization strength. A smaller value of C penalizes complexity more heavily, and a smaller value of C penalizes complexity less heavily. ℓ_1 ratio is the ratio of the penalty term ℓ_1. In an Elastic Net model like the one I used, both the $\ell_1 = ||w||_1$ and $\ell_2 = \frac{1}{2}||w||_2^2$ norm penalty terms are added, and the ℓ_1 ratio determines how much weight is given to each one. Specifically, the regularization term is given by $r(w) = (1-\rho)\frac{1}{2}||w||_2^2 + \rho||w||_1$. I tested C values in powers of 10 from 10^{-2} to 10^2 and ℓ_1 ratios from 0 to 1, stepping by 0.25. The resulting mean F1 scores can be seen in the following plots:

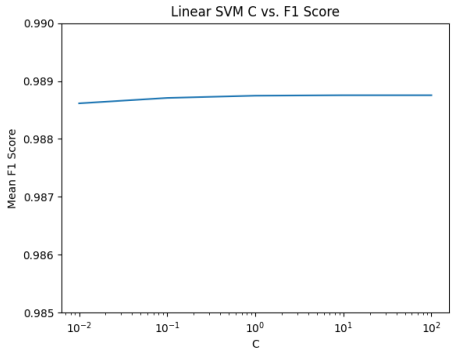


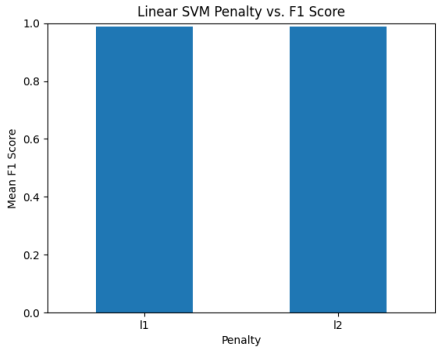


From these results, I concluded that the best choice of C was the elbow point at 10^0 and that my choice of ℓ_1 ratio was fairly inconsequential, but that, very slightly, the best choice of ℓ_1 ratio was 1.0, meaning that the regularization term simplifies to ℓ_1. I chose to use this simplified model. Again, all of the values that I tested resulted in very high and very similar F1 scores.

4.3 Linear SVM

For the Linear SVM model, I needed to optimize two hyperparameters: C and penalty. Again, C is the inverse of regularization strength. Penalty is a binary choice between an ℓ_1 penalty term or an ℓ_2 penalty term, so those were the two I tested. I also tested C values in powers of 10 from 10^{-2} to 10^2. The resulting mean F1 scores can be seen in the following plots:

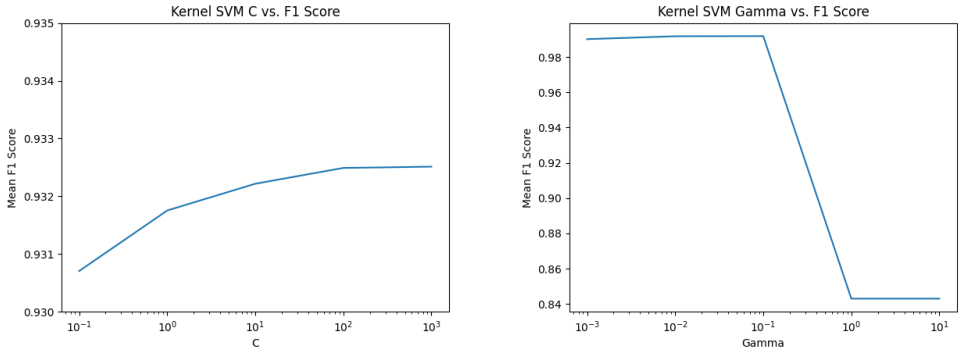




From these results, I concluded that, because the choice of C made little difference in performance, the best choice of C was the default value of 10^0. The choice of penalty also mattered little, so I also chose the default value of ℓ_2. For a third consecutive model, all of the tested values resulted in similarly high F1 scores.

4.4 Kernel SVM

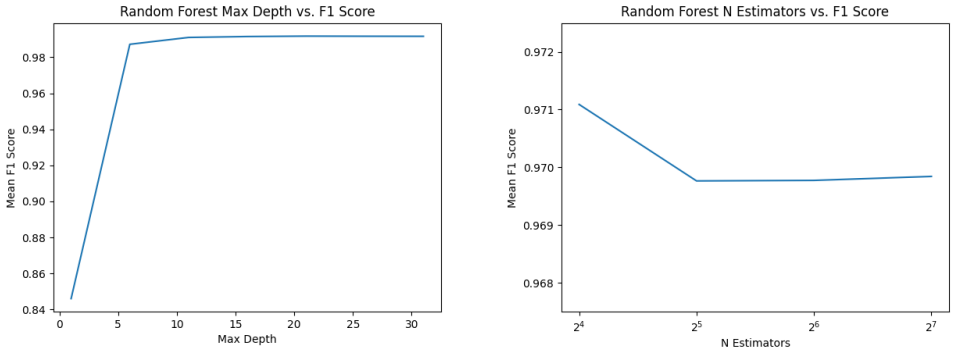
For the Kernel SVM model, I needed to optimize two hyperparameters: C and γ. C is the inverse of regularization strength. γ defines "how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close.'"[6] I tested C values in powers of 10 from 10^{-1} to 10^{3} and γ values in powers of 10 from 10^{-3} to 10^{1}. The resulting mean F1 scores can be seen in the following plots:



From these results, I concluded that the best choice of C was the elbow point at 10^{0} and that the best choice of γ was the elbow point at 10^{-1}.

4.5 Random Forest

For the Random Forest model, I needed to optimize two hyperparameters: maximum depth and number of estimators. Maximum depth is the maximum number of "levels" that each decision tree in the random forest can have. Number of estimators defines the number of trees that make up the forest. More trees will almost always fit the data better, but will also be more prone to overfitting and struggling to make accurate predictions on test data. I tested maximum depth values from 1 to 31, stepping by 5. I tested numbers of estimators in powers of 2 from 2^{4} to 2^{7}. The resulting mean F1 scores can be seen in the following plots:



From these results, I concluded that the best choice of maximum depth was the elbow point at 5. Since all of the values for number of estimators resulted in similar test F1 scores, I decided that the best choice was the simplest value I tested, 2^{4}.

5 Model Testing

After cross validation, the next step in the model life cycle is to split the data into training and testing data, train each model on the training data, and test it on the testing data. I chose to set aside 10% of the data as training data and to use the other 90% as testing data. For model evaluation scores, I included misclassification rate, recall, precision, and F1 score. A lower misclassification rate is better, but for the three other scores, higher is better. Because the data is imbalanced, recall, precision, and F1 score were calculated on a per class basis to help better evaluate the models. The following table shows the test results for each model.

	Misclassification Rate		**Recall**	**Precision**	**F1 Score**
K Nearest Neighbors	0.0062	Sinker	0.9750	0.9848	0.9799
		Splitter	0.9972	0.9954	0.9963
Logistic Regression	0.0107	Sinker	0.9530	0.9774	0.9650
		Splitter	0.9959	0.9914	0.9937
Linear SVM	0.0107	Sinker	0.9510	0.9794	0.9650
		Splitter	0.9963	0.9910	0.9937
Kernel SVM	0.0096	Sinker	0.9530	0.9845	0.9685
		Splitter	0.9972	0.9914	0.9943
Random Forest	0.0135	Sinker	0.9329	0.9790	0.9554
		Splitter	0.9963	0.9878	0.9920

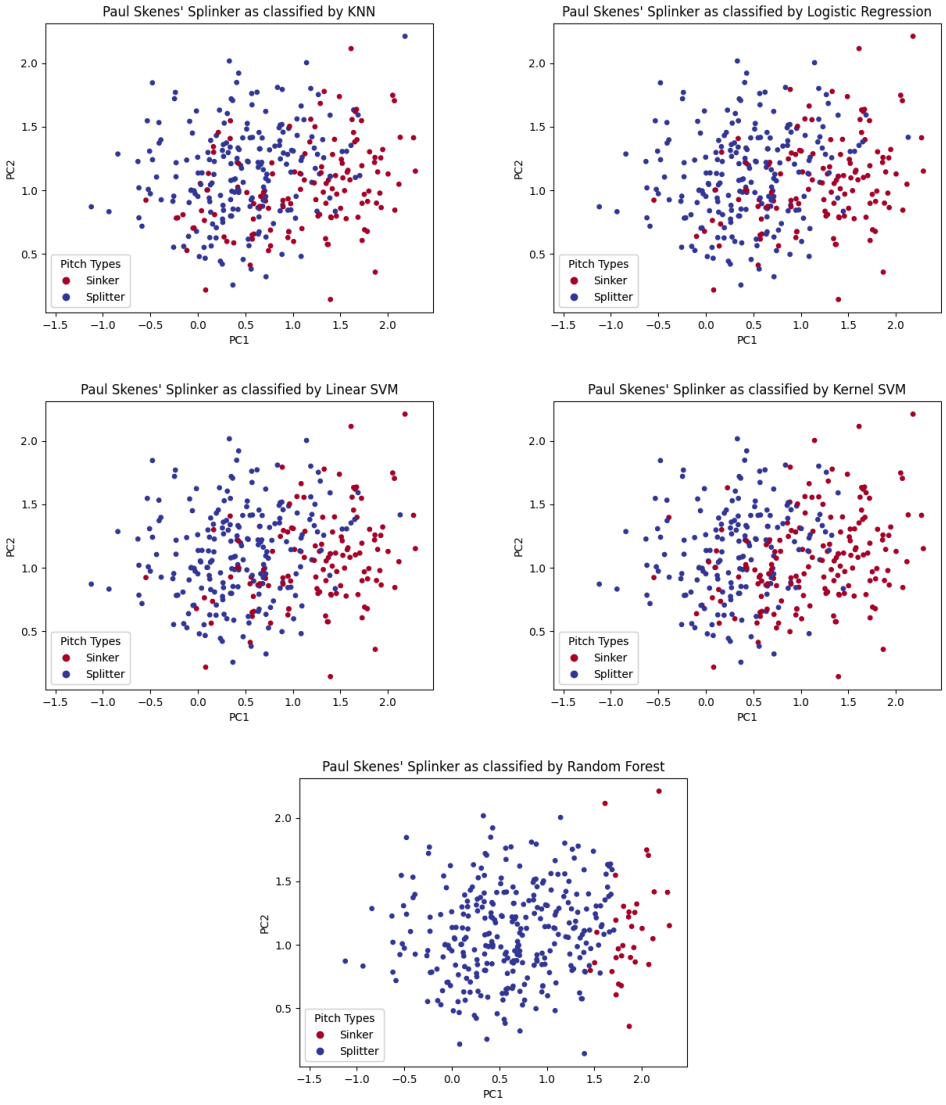
All of the models were very successful at classifying splitters and sinkers. Interestingly, the model with the lowest misclassification rate was K Nearest Neighbors, the simplest of the five models. Also, all five models tended to be better at classifying splitters than sinkers. I believe this is a result of class imbalance, because the models would be more likely to overclassify in the majority class, sinkers, and be more "careful" about classifying into the minority class, splitters. Therefore, one would expect the purity metrics to favor the minority class.

6 So... What is Paul Skenes Throwing?

Ultimately, I wanted to see how these five models classify the splinker. Is it more sinker or more splitter? I trained the models on all of the non-Skenes pitches in the data and used them each to classify Skenes' pitches. I chose to evalute the "sinker-ness" and "splitter-ness" by comparing the classification rates for each pitch type. That is, at what rate did the model classify Skenes' pitch as a sinker and at what rate a splitter? I used this formula for classification rate: $\text{classification\_rate}_{\text{pitch type}} = \frac{\text{\# classified as pitch type}}{\text{total \# splinkers}}$. The resulting classification rates can be found below:

	Sinker	**Splitter**
K Nearest Neighbors	62.09%	37.91%
Logistic Regression	62.09%	37.91%
Linear SVM	63.58%	36.42%
Kernel SVM	50.15%	49.85%
Random Forest	90.15%	9.85%

Here are visualizations of how each model classifies the available splinkers:



The first four models slightly favor classifying the splinker as a sinker and Kernel SVM classifies it almost exclusively as a sinker. Given our observations about class imbalance, it should be expected that, for a less clear classification result, a model will have a slight bias toward the sinker classification. This bias should change our interpretation, particularly of all of the models but Random Forest, which are approximately evenly split in their classification of the splinker. This indecisiveness is definitely notable given the models' general success in distinguishing between sinkers and splitters. After all, none of the models had test misclassification rates over 1.5%.

In conclusion, models that had incredible success distinguishing between the sinkers and splitters thrown by every other pitcher in MLB struggled to choose what to call the Skenes splinker, so sports media is right to call for the creation of an entirely new pitch classification for it. Hence, the newly coined term "splinker" is an appropriate name for this devastating pitch that is carving through MLB batters.

7 References

1. Statcast Glossary. MLB.com. Accessed June 28, 2024. https://www.mlb.com/glossary/statcast
2. Smith AW. Pitched Baseballs and the Seam Shifted Wake. Published online January 1, 2020. doi: https://doi.org/10.26076/e0c9-675d
3. Camp J, Sarris E, Dvorocsik G. Using Clustering to Find Pitch Subtypes and Effective Pairings. Society for American Baseball Research. Published 2020. Accessed June 28, 2024. https://sabr.org/journal/article/using-clustering-to-find-pitch-subtypes-and-effective-pairings/
4. Chang JP. Baseball Pitch Classification through Cluster Analysis. Jason P Chang. Published January 27, 2018. Accessed June 28, 2024. https://jasonpchang.github.io/projects/pitchfx/
5. Mancuso M. Classifying Pitch Types Using Python Modelling. BaseballCloud Blog. Published October 18, 2021. Accessed June 28, 2024. https://baseballcloud.blog/2021/10/18/classifying-pitch-types-using-python-modelling/
6. RBF SVM Parameters — scikit-learn 0.21.3 Documentation. scikit-learn Documentation. Published 2024. Accessed July 28, 2024. https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html