Predicting Justin Verlander's Next Pitch with Machine Learning

Project Category: Athletics & Sensing Devices

Name: Mohamed Owda
SUNet ID: mohamed8
Department of Computer Science
Stanford University
mohamed8@stanford.edu

Name: Griffin Holt
SUNet ID: gholt
Department of Electrical Engineering
Stanford University
gholt@stanford.edu

I. Introduction

Baseball revolves around a pitcher throwing a baseball, an act referred to as "the pitch," and a batter attempting to hit the baseball to then run around the bases and, hopefully, score. To make it more difficult for the batter to hit the baseball, the pitcher will throw different types of pitches, which vary in speed, rotation, vertical movement, and horizontal movement. Being able to guess the upcoming pitch allows batters to know what to expect and adjust accordingly [1]. On the other hand, the less predictable a pitcher is, the higher the chance the pitcher has of catching the batter off guard and "striking" him out.

For our research in particular, we chose to predict the pitches of a single pitcher, Justin Verlander. Verlander has played in Major League Baseball since 2005–which provides us with a fairly large number of data points. He also has a fairly standard repertoire of pitches – consisting of a four-seam fastball ("FF"), a curveball ("CU"), a changeup ("CH"), a slider ("SL"), and a sinker ("SI") – making him a more ideal candidate for a single-pitcher prediction model. This set of 5 pitches will be referred to as valid pitch types (VPT). Given a set of input features (including game situational data, the pitcher's pitching statistics, the batter's hitting statistics, and the results of previous pitches), we will attempt to classify the next pitch Verlander will throw from the set of VPT.

A. Related Work

Ganeshapillai and Guttag [2] first approached the problem of predicting pitches with machine learning, using binary classification for predicting whether a pitch was a fastball or non-fastball. Other groups since then have addressed variations of the same problem. Hoang et al. [3] utilized Support Vector Machines (SVM) and k-Nearest-Neighbor algorithms to approach pitch classification as a multi-class classification problem: they aimed to be more specific in their predictions than simply "fastball" or "non-fastball," and were able to achieve classification accuracy as high as 80%. Sidle and Tran [4] utilized Linear Discriminant Analysis, SVM, and Random Forests to approach the multi-class pitch prediction problem, acheiving classification accuracy results of 66.6%.

As [3], [4] before us, we approached pitch classification as a multi-class classification problem. We utilized some of the same input features, mainly the game situational features and the previous pitch tendency features used by both [3], [4]. However, we used several new input features based on batting statistics that were not used by these groups, including the batter's strike heatmap and pitch-specific slugging averages; these features are further described in Section II. In addition, [2], [3], [4] all had datasets composed of pitches from at least 200 different pitchers–resulting in training set sizes of 100,000 pitches or more. Thus, previous researchers have primarily created general baseball pitch prediction models, applicable to many different pitchers. One of our goals was to see whether or not we could create a pitcher-specific model–trained only on the examples from a single pitcher–which introduced the additional challenge of achieving high accuracy with a limited training set size. Finally, because these papers all saw some success with SVMs, we utilized these models–in addition to Feedforward Neural Networks and Multinomial Logistic Regression–to address our multi-class classification problem.

Lee [5] utilized ensemble models, a method of combining predictions from multiple neural networks, to reach 64.2% accuracy with predicting not only the pitch type but also the pitch location over the base. Furthermore, Yu, Cheng, and Chang [6] used long-short-term memory (LSTM) models, a form of recurrent neural networks, to predict pitch types with an accuracy of 76.7%. While we did not have enough time to try out ensemble or LSTM models, they are certainly areas to explore for future work.

II. Dataset and Features

The data was drawn exclusively from Statcast [7], an automated tool from the MLB to track pitching data by recording information about each pitch thrown. Using the PyBaseball package [8], we gather all pitching data, which starts from the 2008 scene. This final dataset was composed solely of pitches thrown by Justin Verlander, starting from the 2009 season. We have three different groups of features. First, we have "game-situational features" (found in Table I): features that only give information about the game environment that the pitch is being thrown in. Next, we have "pitching features" (found in in Table II): features that are

	Feature	Description
1	Year	The year the pitch was thrown. Zero-indexed on 2010. An integer between 0 and 9, inclusive.
2	Month	The month the pitch was thrown. An integer between 3 and 12, inclusive.
3	Score Difference	The score of the pitching team minus the score of batting team. An integer.
4	Inning	The inning the pitch was thrown. A natural number.
5	Inning Half	0 if pitch is thrown at top of the inning. 1 if pitch is thrown bottom of the inning.
6	Outs	The number of outs in the half of the inning, before the pitch was thrown. An integer between 0 and 2.
7	Balls	The number of balls in the count before the pitch is thrown. An integer between 0 and 3.
8	Strikes	The number of strikes in the count before the pitch is thrown. An integer between 0 and 2.
9	Pitch Number	The pitch number of the plate appearance before the pitch is thrown. A non-negative integer.
10	On 1B	Boolean (0 or 1) on if there's a player on first base.
11	On 2B	Boolean (0 or 1) on if there's a player on second base.
12	On 3B	Boolean (0 or 1) on if there's a player on third base.
13 - 14	OF Alignment	One-hot vector on if the outfield is in a standard or strategic outfielder fielding alignment.
15 - 17	IF Alignment	One-hot vector on if the infield is in a standard, strategic, infield shift fielding alignment.

TABLE I

GAME SITUATIONAL INPUT FEATURES FOR THE BASELINE MODELS

	Feature	Description
18-22	Prev. Pitch Type	A one-hot vector on whether the previous pitch thrown was a FF, CH, CU, SL, SI.
23 - 27	Prev. 5 Pitch Tendencies	For each of {FF, CH, CU, SL, SI}: The fraction of the previous 5 pitches that are of that type. $\in [0, 1]$.
28-32	Prev. 10 Pitch Tendencies	Replicate features 23-27 except this is over the previous 10 pitches (Note that features 30-34 should sum to 1).
33-37	Prev. 20 Pitch Tendencies	Replicate features 23-27 except this is over the previous 20 pitches (Note that features 35-39 should sum to 1).
38-40	Previous Pitch Result	A one-hot vector on whether the previous pitch thrown resulted in a strike, ball, or in-play.
41-45	Total Strike Tendency	For each of {FF, CH, CU, SL, SI}: The fraction of all pitches of that type that resulted in a strike. $\in [0, 1]$.
46-50	Total Pitch Tendency	For each of {FF, CH, CU, SL, SI}: The fraction of total pitches that are of that type. $\in [0, 1]$.
51-55	Prev. 5 Pitch Strike Tendencies	The fraction of the previous 5 CH pitches that are strikes. $\in [0, 1]$.
56-60	Prev. 10 Pitch Strike Tendencies	Replicate features 53-57 except this is over the previous 10 pitches.
61-65	Prev. 20 Pitch Strike Tendencies	Replicate features 53-57 except this is over the previous 20 pitches.

TABLE II

PITCHING FEATURES

	Feature	Description
66	Handedness	0 if batter is left-handed, 1 if batter is right-handed.
67-75	Result of Last At-Bat	One-hot vector of the last at-bat being an out, double, triple, single, strikeout, walk, home run, error or other
76-80	Pitch Strikeout Tendency	The fraction of strikes off each pitch type from {FF, CH, CU, SL, SI} in this batter's career. $\in [0, 1]$.
81	Overall Strikeout Tendency	The fraction of strikes from all pitches in this batter's career. $\in [0, 1]$.
82-86	Pitch-Specific Slugging Average	For each of {FF, CH, CU, SL, SI}: The slugging average from pitches of that pitch type.
87	Overall Slugging Average	The slugging average from all pitches.
88-100	Batting Heat Map	The fraction of pitches that resulted in strikes in 13 different zones. Each feature is $\in [0, 1]$.

TABLE III

BATTING FEATURES

related to different pitching results and pitching tendencies of Justin Verlander based on previous pitches he's thrown. Finally, we have "batter features" (found in Table III): these are various features related to the batter's historical success and tendencies both against a given pitch and against all pitches across all previous at-bats. Both batter features and pitching features are only generated based upon previous pitches, as this is the information the pitcher and batter have at the time. Note that primarily only the game-situational features were provided directly by Statcast: the remaining features required independent extensive preprocessing and data engineering, which we accomplished ourselves.

The classification label for each example is the pitch's type (previously listed). We removed pitches that had faulty data (such as unusual at-bat counts), and only kept pitches that were VPT. In total, we had $n = 35153$ examples. We shuffled the data and created a train-test split of 80-20, stratifying the data so that the labels would be represented proportionally across the training and test set. "FF" was the most prevalent of the pitch classes, accounting for 58.3% of all pitches; "CU" was the second most prevalent, accounting for 16.7% of all pitches, meaning our data is imbalanced.

We wanted to test which features were best for helping our models predict the next pitch. Thus, we created 3 different feature sets against which to train our models: the "Whole" feature set consists of Features 1 - 100 from Tables I, II, and III, the Situational (Sit.) feature set that consists of features only from I, and the "Without Recursive" (WOR) feature set that consists of features 1-17, 41-50, and 66-100 from Tables I, II, and III.

We observe that some features have different scales compared to others (probabilities vs natural numbers) that may result in our model placing higher emphasis on features containing larger values. To counteract this, for each dataset we create a standardized version of the dataset where each feature is forced between 0 and 1, using Scikit-Learn's

MaxMinScaler [9].

Many of the input features are also one-hot vectors due to the discrete nature of the game of baseball. To cut down on this added dimensionality, for each dataset we create a reduced version using Principal Component Analysis. We retained the top 95% of the principal components of each feature set, so as to retain as much of the information as possible while still projecting down into a less-sparse feature space. The Whole feature set was reduced from 100 input features to 35; the WOR feature set was reduced from 53 to 19 input features; and the Sit feature set was reduced from 17 to 12 input features.

To create the heatmap for Features 88-100, we divide the pitching zone into 13 sections [10], the central 9 of which compose the strikezone. For each of these sections for a particular batter, we compute the number of strikes thrown to that zone divided by the number of pitches thrown to that zone.

III. Methods

A. Multi-Layer Perceptron

Our first model type was a Multi-layer Perceptron classifier (MLP) neural network, constructed through Scikit-Learn [9] and Keras [11]. Throughout all of our MLP models, we utilized $\text{ReLU}(x) = \max(0, x)$ as the activation function between layers. The final layer was pushed through the softmax equation, $\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$ to get the output $z = \text{Softmax}(a^{r-1})$ with $z \in R^5$ representing the probability of each class, and a^{r-1} being the final layer. Our objective was to minimize Cross-Entropy Loss

$$CE(y, \hat{y}) = -\sum_{k=1}^{K} y_k \log \hat{y_k},$$

where $\hat{y} \in \mathbb{R}^K$ is the vector of softmax outputs from the model for the training example x, and $y \in \mathbb{R}^K$ is the ground-truth vector for the training example x such that $y = [0, ..., 0, 1, 0, ..., 0]^\top$ contains a single 1 at the position of the correct class (also called a "one-hot" representation) (from CS229 PSET2 Fall 2022 Q4).

We utilized the Adam Optimization Algorithm [12] to minimize this objective.

1) Experiments: k-Fold Cross Validation is a process by which we split the data into k "folds," using $k-1$ of these folds for training and 1 fold for testing. We cycle through each fold being the test set, and generate an accuracy of our model based on the average of the results, thus ensuring our data is both training and validating on each example. We experimented with the neural networks in multiple different ways, conducting hyperparameter searches using 5-fold Cross-Validation, to determine six different hyperparameters:

1) Number and Size of Hidden Layers;
2) Batch Size;
3) Learning Rate;
4) Regularization Parameter;
5) Presence of Dropout Layers;
6) Class Weights.

We experimented with batch sizes of 10, 100, 200, which represents the number of examples passed to the neural network at one time to update the weights. We varied the batch sizes as larger batch sizes are faster to train but tend to be less accurate [13].

As our number of examples were limited, we were concerned about overfitting. Thus, to counteract overfitting, for some models we introduced dropout layers [14] with a dropout rate of 0.1, meaning for each node in each layer there is 0.1 probability this node gets ignored by the following layer. To also counteract overfitting, we also consider regularization parameters of size 10^{-4}, 10^{-3}, 10^{0} and 10^{1}. The regularization parameter acts as a coefficient on the $l2$-norm penalty. The larger the regularization parameter, the greater the loss our model experiences for having the weights θ become too large, meaning θ is pushed to be smaller, making it more difficult for the model to overfit.

To prevent the model from predicting the most common pitch type, we also tested models with class weights. For each class k, we compute a class weight of

$$\frac{n'}{K \cdot c}$$

where n' is the number of examples, K is the number of classes, and c is the number of examples of class k. Thus, classes that occur less will have a higher weight, and will be penalized greater for mistakes in our loss function. We vary the learning rate used by the gradient descent algorithm between $10^{-4}, 10^{-3}$, and 10^{-2}, with larger learning rates converging faster but being prone to overfitting and suboptimal results.

B. Multinomial Logistic Regression

The next model we tested was a Multinomial Logistic Regression classifier. Multinomial Logistic Regression attempts to maximize a weight vector θ for each class according to the cross-entropy loss function

$$J(\theta) = \sum_{i=1}^{n} \sum_{k=1}^{K} 1\{y^i = k\} \log(P(y^i = k | x^i, \theta)$$

where $K = 5$ (our number of classes), n is our number of examples, and the $P(y^i = k|x^i, \theta) = \text{Softmax}(\theta_k^T x^i)$. We then predict whichever class has the greatest likelihood when we multiply the example by θ. The Logistic Regression fitting algorithm used an $l2$-norm penalty applied to θ to ensure the weights do not get too large which would result in overfitting or never converging.

We also experimented with class weighting due to the imbalance of classes in our data.

C. Linear Support Vector Machines

The final model we tested was Support Vector Machines (SVM) [15]. SVM attempts to find the hyperplanes that best separate the data by finding the hyperplane that maximizes the geometric margin (the distance from the hyperplane to an

Model Description	Class Weights	Feat. Set	Standardized	95% PCA	Avg. 5-fold CV Acc.
Baseline: Frequency of Most Common Pitch ("FF")	-	-	-	-	58.27%
Multi-Layer Perceptron Hidden Layer Sizes: [100, 200, 200, 100] Dropout Rate: 0% Regularization Param.: 0.1 Learning Rate: 0.001 Batch Size: 200	None	Whole	No	No	59.59%
Linear SVM	None	Whole	No	No	58.97%
Logistic Regression	None	Whole	Yes	No	58.73%

TABLE IV

TOP CLASSIFICATION ACCURACIES FOR THE THREE MODELS: NEURAL NETWORK, LINEAR SVM, & LOGISTIC REGRESSION

arbitrary point). Our SVM algorithm attempts to minimize the multi-class hinge loss function

$$l(\theta, X, y) = \sum_{i=1}^{n} \sum_{k=1, k \neq y^i}^{K} \max(0, 1 - \theta_{y^i}^T x^i + \theta_k^T x^i)$$

to separate our data into the $K = 5$ classes according to θ. We have an $l2$-norm penalty applied to θ to avoid overfitting.

We also experimented with class weighting due to the imbalance of classes in our data.

IV. RESULTS

Across the 3 different feature sets ("Whole," "Sit," and "WOR"); the 3 different feature selection/reduction techniques ("Normal," "Min-Max Scaling," and "95% PCA"); and the 7 different hyperparameters (see Section III-A.1), we experimented with 81 different MLP models. Similarly, we experimented with 18 different Logistic Regression models and 18 different Linear SVM models.

We assessed the model performance of our 117 different models during training using k-fold Stratified Cross Validation where $k = 5$ (5-fold CV), stratifying so as to maintain the same class distribution from our original dataset. We utilized Classification Accuracy,

$$\frac{\text{\# Classified Correctly}}{\text{\# of Examples in Validation Set}}, \tag{1}$$

to compare model performance and select optimal hyperparameters. Classification accuracy was chosen (as opposed to precision, recall, $F1$-scoring, ROC-AUC or other classification metrics) due to both the multi-class nature of our problem and the fact that classification accuracy was the primary method of performance evaluation in past related work [2], [3], [4], [5], [6], enabling easier comparison.

For each of the three general models, we have displayed the best hyperparameters–including class weighting, feature set, feature selection/reduction–and the resulting highest average 5-fold cross-validation accuracy in Table IV. The Multi-Layer Perception performed the best out of the three models, with a CV accuracy of 59.59%. We then evaluated this model against the test set, where it achieved 58.97% test accuracy. Thus, our best model only performed slightly better than the baseline accuracy of 58.3%: the frequency of the most common pitch ("FF") in Verlander's repertoire.

In Figure 1, we display three confusion matrices for the performance of the best MLP model against the test set. All three confusion matrices display the same data, but are normalized in different ways: 1) across rows; 2) across columns; and 3) across the entire grid. We discuss the significance of these confusion matrices in the next section.

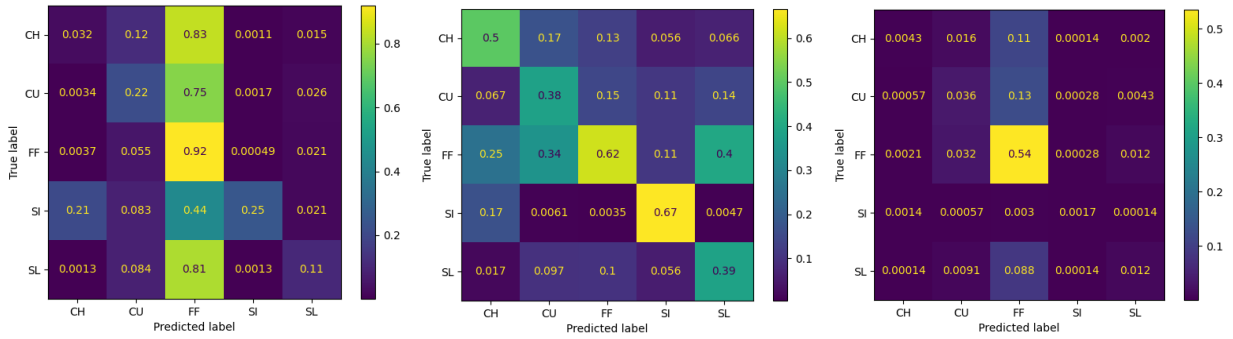
V. DISCUSSION

Throughout our research, we operated under the assumption that more data–specifically, more pitchers–would result in higher training accuracies. This was due to the work of past researchers [3], [4], previously mentioned, that were able to achieve accuracies as high as 70% and 80% on multi-class classification by using data from more than 200 pitchers (100,000's of examples). Part of the challenge of our particular approach was to see if we could achieve similar accuracies on a single pitcher, thus placing a constraint on the size of our dataset.

As can be seen from our results, none of our 117 different models were able to achieve much higher than our baseline naive accuracy of 58.3%. Our experiments, as described in Section III, were a series of sequential attempts to rectify this issue.

As we prepared the dataset for training, we theorized that the sparsity of our dataset–due to the presence of a number of one-hot vectors in the input–contributed to the models' poor performances. After all, a training dataset needs to be representative of all possible combinations of input features in order for a particular model to learn best from that dataset. Sparsity in a dataset can be an obstacle in this goal. Thus, we experimented with feature selection (through the three different feature sets) and feature reduction (through PCA). Although our best models seen in Table IV were all trained on the whole dataset, it is important to note that the reduced feature set models and PCA models did not perform much worse: their average CV accuracies were also in the range of about 56-58% accuracy. Thus, although attempting to address sparsity did not result in better model performances, we learned that it did not significantly hurt model performances.

We also attempted to increase model complexity to see if more complex neural networks (i.e., networks with more layers and more neurons per layer) could achieve higher test accuracy. Although the more complex networks (some as many as seven or eight layers deep) were able to achieve



(a) Normalized by Row / True Label (b) Normalized by Col. / Pred. Label (c) Normalized across Entire Matrix

Fig. 1. Confusion Matrices for the Best MLP Model, evaluated on the Test Set

higher training accuracy, test accuracy never improved. This suggested to us that these more complex networks were simply memorizing the training set without learning any nonlinear feature combinations that might generalize well to the test set. As a result, we stopped trying to increase model complexity and experimented more with other hyperparameters instead.

Because of the imbalances in our data, we also attempted to utilize dropout layers and/or class weightings in our models. However, neither dropout layer nor class weightings improved the models beyond the performances we see in Table IV.

A. A Closer Analysis of the Best MLP Model

Looking at the confusion matrices for our best MLP model in Figure 1, we can gain insight into exactly what our neural networks might have been learning.

First, note that the numbers in Figure IV represent the following quantity: "Given that the true label is X, what is the probability that the model guessed Y?" Across all five of the pitch classes, our model learns to guess "FF" most of the time–this makes sense, as the four-seam fastball was the most prevalent class in the data.

The numbers in Figure IV answer a different question: "Given that the model guessed X, what is the probability that the true label is Y?" These numbers actually suggest that, despite the low classification accuracy, our best model does have some utility. If our model were to be used in a live MLB game where Justin Verlander was pitching and the model guessed that the next pitch to be thrown would be a Sinker ("SI"), the probability that the next pitch would actually be a Sinker would be 67%. Seeing as how Justin Verlander doesn't throw a lot of sinkers (they represented less than 1000 of all his pitches), this is excellent information to have during a game. The predictions for the changeup ("CU") and four-seam fastball ("FF") have similar value.

VI. Conclusions and Future Work

The goal of our paper was, given a set of features related to game, pitcher, and batter information, how well can we predict the next pitch thrown by Justin Verlander. We attempted MLPs, Multinomial Logistic Regression, and SVM models for this classification task, experimenting with various feature sets, and applying standardization and PCA to our features to find what data processing worked best. Additionally, we experimented with a number of hyperparameters meant to address potential overfitting and guessing the most popular pitch that come as a result of our small and imbalanced dataset. In the end, our best MLP model achieved a test accuracy of 58.3%, with specifics regarding CV accuracy in Table IV. While this accuracy is somewhat low, and is on par with just predicting the most common pitch type, further analysis in Section V-A shows that our model learned more than just predicting the most common pitch and thus has value in understanding a pitcher.

A. Future Work

We believe one aspect that made our models struggle was a lack of examples, making the model both prone to overfitting and struggle to predict less relevant pitches. Still, we would like to "figure out" a given pitcher, and the average pitcher will have no more than 40,000 pitches in their career. Thus, one idea would be to cluster similar pitchers to the pitcher you are predicting, and use the pitching history from similar pitches to generate more examples, while finding methods to place more weight on the specific pitcher you are predicting. We could also implement transfer learning [16] by training a neural network model on an increased number of examples ($> 1,000,000$ pitches from many pitchers across multiple years), freezing the first section of layers, and then retraining it specifically on Justin Verlander's data. Thus, our model would benefit from the nonlinear insight gained by the increased number of training examples used to train the general model, but would still be fine-tuned to Justin Verlander's pitching patterns. Inspired by Lee [5] and Yu, Cheng, and Chang [6], we should explore additional neural network techniques such as LSTM models or ensemble models to see if these techniques could improve our accuracy. The LSTM model might be able to fit time-dependent peculiarities in a single pitcher's data for which our standard neural networks could not account. Going forward, it would be useful to develop a better performing ML model as an application for mobile devices so that batting managers could use these results live in-game to direct batters.

VII. Contributions

The processing of data–from a Statcast download to training-ready input matrices–required a significant amount of work. Both Holt and Owda contributed to this process. Holt and Owda split the code-writing portion of this process. Holt then executed the code and monitored its progress to prepare the final dataset.

Owda investigated the Statcast dataset to find how Statcast represented the data and offered analysis on how this combines with our desired experiments. He also researched the PyBaseball library's API and wrote the code to import the data from Statcast. Owda ran the neural network code for seeing the effects of having one of or both of class weights and dropout layers. Owda wrote the code to develop PCA and normalization. Finally, he wrote the Dataset/Features (including the three feature tables) and Methodology sections of this report, as well as much of the Conclusion.

Holt researched possible input features to include in both the baseline dataset and the expanded dataset to be used in the next phase of the project. He investigated the Statcast data for potential faulty data (such as misnamed pitches and incorrect pitch counts) and corner cases. He ran the training for the SVM models and Logistic Regression classifier using SciKit-Learn [9], and neural networks without class weights or dropout layers. Finally, he wrote the Related Works, Results, and Discussion sections, including Table IV and Figure 1, and some of the Conclusion.

References

[1] D. Bernstein, "Astros cheating scandal timeline, from the first sign-stealing allegations to a controversial punishment," 2020.

[2] G. Ganeshapillai and J. Guttag, "Predicting the next pitch," in *Sloan Sports Analytics Conference*, 2012.

[3] P. Hoang, M. Hamilton, H. Tran, J. Murray, C. Stafford, L. Layne, and D. Padget, "Applying machine learning techniques to baseball pitch prediction," 01 2014.

[4] G. Sidle and H. Tran, "Using multi-class classification methods to predict baseball pitch types," *Journal of Sports Analytics*, vol. 4, no. 1, pp. 85–93, 2018.

[5] J. S. Lee, "Prediction of pitch type and location in baseball using ensemble model of deep neural networks," *Journal of Sports Analytics*, no. Preprint, pp. 1–12, 2022.

[6] C.-C. Yu, C.-C. Chang, and H.-Y. Cheng, "Decide the next pitch: A pitch prediction model using attention-based lstm," in *2022 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, 2022, pp. 1–4.

[7] B. Savant, "Statcast," Dec 2022. [Online]. Available: https://baseballsavant.mlb.com

[8] T. Burch. Pybaseball. [Online]. Available: https://pypi.org/

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[10] K. Ruprecht, "Does location of the pitch affect babip?" Jul 2014. [Online]. Available: https://www.beyondtheboxscore.com/2014/7/21/5921769/babip--strikezone-location-mlb

[11] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[13] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," 2016. [Online]. Available: https://arxiv.org/abs/1609.04836

[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[15] M. Hearst, S. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intelligent Systems and their Applications*, vol. 13, no. 4, pp. 18–28, 1998.

[16] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big data*, vol. 3, no. 1, pp. 1–40, 2016.