Machine Learning Implementations in Baseball: An Algorithmic Prediction of the Next Pitch

by

Jacob Morehouse

Bachelor of Arts and Science, UNB, 2016

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

In the Graduate Academic Unit of Mathematics and Statistics

Supervisor(s):	Jeffrey Picka, PhD, Mathematics and Statistics
	Tariq Hasan, PhD, Mathematics and Statistics
Examining Board:	Guohua Yan, PhD, Mathematics and Statistics, Chair
	Usha Kuruganti, PhD, Kinesiology, UNB

This thesis is accepted by the

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

September, 2020

Abstract

Machine Learning (ML) has recently begun gaining traction in the statistical analysis of baseball. Major League Baseball (MLB) has a long history of using statistics to evaluate players, but recent innovations in player tracking have introduced the opportunity for ML to flourish. Statcast is a new tracking system that generates detailed data pertaining to the movement of both the players and the ball using cameras and radar technology. This paper will examine the functionality of predictive models using this data and their applications in baseball. As an example, we will attempt to predict which type of pitch a pitcher will throw next. Random forest and support vector machine algorithms will be created for this learning task.

Dedication

To Mum

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the past, the statistics of Major League Baseball (MLB) were used primarily as a way of recognizing its greatest players. A pitcher that won 20 games in a season was vastly superior to one that only won 10, and a batter that hit for a .250 average was nowhere near the same class as one that hit .300. These facts went unchallenged for decades. Bill James [19] was the first to seriously question this consensus. Modern baseball analytics have done more than chip away at the old misconceptions. They are changing how the game is played.

The flow of information in MLB has worked its way to the field of play, where players and teams are using the current wealth of data to aid in their decision making. Teams are in search of any advantage they can find. For a batter, being able to anticipate what type of pitch the pitcher will throw is invaluable [31]. Before technological advancements made pitch tracking pos-

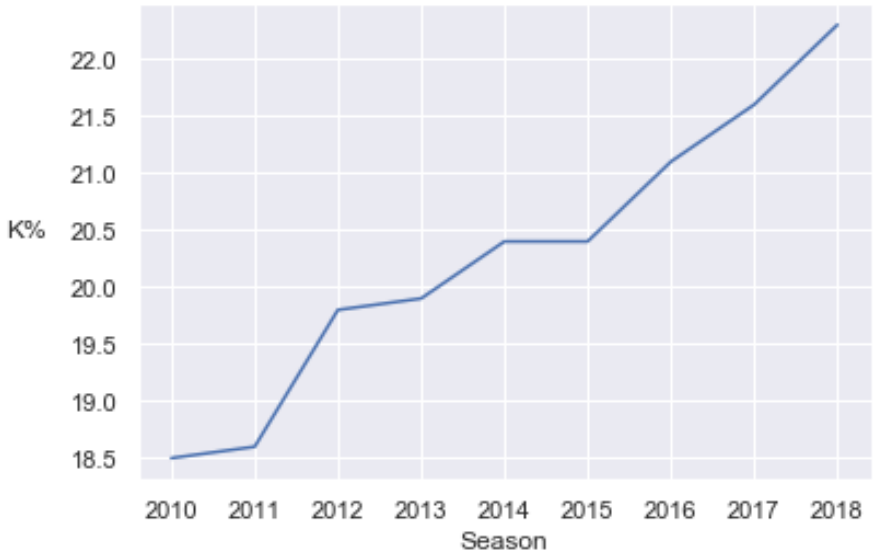
sible, hitters could only give educated guesses at what a pitcher might throw based on anecdotal evidence. Today, hitters have access to detailed breakdowns describing the tendencies of pitchers: what types of pitches they like to throw in a given situation, and where they like to throw them. Likewise, pitchers can see how often hitters swing at different types and locations of pitches. With both sides seeking to leverage the data, the struggle between batter and pitcher has become more complicated than ever.

Hitting a baseball is very difficult. Pitchers have learned what hitters struggle to hit, and have begun striking out batters at an historically high rate (see Figure 1.1). There are likely several contributing factors to this increase, with pitchers optimizing their pitch selection strategies being just one of them. We hope to improve a hitter's odds by aiding in the prediction of the next pitch they will see.

Machine learning (ML) offers a number of algorithms to build predictive classification models. The advantages and issues of specific ML mechanisms relating to baseball will be a key component of this paper. Chapter 2 explores the development of statistical methods in baseball and discusses issues causing complications. The remaining chapters will focus on predicting pitch classes. Chapter 3 presents a literature review of studies involving pitch type prediction. Chapter 4 outlines the data and methodologies of classification for this paper. Two classification methods, random forests and support vector machines, will be considered for pitch type classification using data provided from a technology called Statcast. The results of the analysis are shown in

Chapter 5, where 244 pitchers are used to build models for prediction.

Figure 1.1: **Strikeout Percentage Since 2010**



Strikeout rate, which is the percentage of plate appearances ending with a strikeout, has steadily increased since 2010. This data comes courtesy of fangraphs.com.

Chapter 2

Notes on the Progression of Baseball Analytics

2.1 Historical Review of Sabermetrics

2.1.1 Rise in the Public Sphere

Bill James challenged many of the beliefs held by baseball teams and fans. James published his first *Baseball Abstract* in 1977 and began the task of subverting historical preconceptions. The field of baseball analysis came to be known as sabermetrics, a term created by James himself [23].

The practitioners of sabermetrics have become known as sabermetricians. They have congregated to websites where like-minded fans can share their research and opinions. Much of the most important baseball analysis is avail-

able online on websites such as Fangraphs[1] and Baseball Prospectus.[2] Sabermetrics users have formed their own community where anybody with an understanding of baseball can contribute. They have helped provide historical revisions by finding new ways of evaluating players and discovering optimal game strategies. Although many of the early sabermetricians were hobbyists, they helped to establish new approaches to playing baseball. Tango, Lichtman, and Dolphin[29] provided the most comprehensive strategy guide for MLB games to date, all based on empirical data.

Research has become more focused on forecasting player performance and predictive analytics. The sabermetrics community has become recognized as a legitimate source of knowledge. Their interests have begun spilling in to academic research. This is partially due to the growing barrier in performing new analysis; as the standard for new research has become higher, so has the complexity of the predictive modelling.

MLB teams were slow to adopt advanced analytics. The first well-documented case of a team relying heavily on analytics came in 2003 with the publication of *Moneyball* by Lewis [23]. The shift has been gradual but noticeable. Sabermetrics is now widely practised by professional teams; it also remains popular in public online communities, which have become proving grounds for talented researchers. MLB teams have begun hiring their own sabermetrics users, many of whom started in the public sphere. It is estimated that

[1] www.fangraphs.com

[2] www.baseballprospectus.com

MLB clubs like the Los Angeles Dodgers and Chicago Cubs spend $20 million and $13 million respectively on research and development [30].

2.1.2 An Accumulation of Anecdotes

The implementation of tracking systems have been crucial in the development of baseball analysis. A tracking system is used to monitor movements of the baseball. PITCHf/x is a system used to track pitches thrown in MLB games by captured images from multiple cameras. The technology first appeared in 2006 [1]. The tracking system can measure the velocity, break and position of each pitch thrown. All this data is used to classify the pitch type using a neural network.

Statcast replaced PITCHf/x as the official tracking system used by MLB at the start of the 2017 season. Statcast uses additional cameras to not only track the ball, but also the individual players on the field [1]. The wealth of data that PITCHf/x and Statcast have provided allow for more detailed play-by-play accounts. Prior to the implementation of tracking systems, many of the characteristic skills of a player were assessed anecdotally. For example, a team of people with trained eyes can classify a season's worth of pitches using video [20]. Further back in time historical records of pitcher tendencies become more anecdotal. Neyer and James [21] have compiled an invaluable record of pitcher repertoires; we can see Bob Gibson threw two types of fastballs, a slider, a curveball, and a changeup. How hard was his fastball? How often did he throw a slider with two strikes? There is insufficient data to

answer these questions for a pitcher of Gibson's era. In this regard, the study of pitch sequencing and pitch prediction is a new field of study. MLB teams are gradually putting less weight on the subjective views of their scouts and coaches in favor of quantitative analysts [24].

2.2 Machine Learning Research in Baseball

Research in sabermetrics has an impact on how the game is played; it would be negligent for a team not to incorporate the important results of such studies in to their overall strategy. Lewis showed that a team is capable of using sabermetrics to inform their decision-making. In that case, the Oakland A's exploited the inefficiencies of MLB by targeting undervalued players and introduced strategies that were not popular. They acquired batters that would improve the team's on-base percentage (essentially they made fewer outs) and did not emphasize the defensive quality of their team. The Oakland A's began using analytics heavily before other teams did. Now every team is aware of the necessity to incorporate analytics, and the A's advantage has dissipated.

The integration of machine learning in sabermetrics seems natural given its rise to prominence in other research areas. With game data becoming more widely accessible, sabermetrics users have more flexibility in trying to find answers to their questions. In their overview of machine learning research in baseball, Koseler and Stephan [22] found 32 articles which met their criteria.

Of those papers, they found the earliest paper relating to classification problems (binary or multi-class) was written in 2012. It appears that much of the key machine learning research in baseball is currently coming from academia. The analytics departments of MLB teams are also likely doing key research that is unavailable to the public. In both cases, researchers tend to have academic backgrounds, while also drawing from their knowledge as fans.

Berri and Bradbury [3] note that academics and sabermetrics users have been able to draw from each others' work in order to refine their own research. As sabermetrics users become more acquainted with machine learning techniques, this will likely continue to be the case.

2.2.1 On the Nature of Baseball Statistics

Normally, predictions in sports are dictated by the skills of the players involved. The management of a club is interested in forecasting the performance of their club and the players that constitute it. The skill of the club will largely dictate how many wins it collects, though there are other factors. Injuries and fluctuations in player performance mean that exact predictions on overall team performance are difficult. However, a club's expectations are based on the ability of its players, and over the course of a season those abilities tend to reveal themselves. Predictions are based on data that the club has identified as having some intrinsic relation to the skill of the player. Data measurements pertaining to player ability improve every year. Rather than relying on statistics that focus on outcomes (e.g. batting average), the

new wave of data is far more descriptive of what is happening on the field. Whether a batter is successful in reaching base via hit depends partially on good fortune. In recent years tracking systems have allowed for more intricate measurements of skill. A batter can display high skill by hitting a ball hard yet still make an out. Exit velocity, which measures the speed of the ball off the bat, is a quantitative measure of a batter's ability to hit a ball with force.

Sabermetrics users are better equipped than ever to evaluate the underlying factors that determine outcomes on the field due to a growing base of knowledge. Tango et al. [29] used Markov chains in Chapter 1 to determine a team's run expectancy in an inning based on the arrangement of runners and the number of outs. This combination of runners and outs is known as the base/out state. Their approach helps provide quantitative value for each event on the field. Improved data accessibility has also been crucial to deeper understanding. The PITCHf/x and Statcast tracking systems have enhanced data quality and accessibility. Sabermetrics users may be better at quantitatively measuring the procedures of a baseball game, but they still must identify the effect of random fluctuation in player performance to capture their true skill level.

Sabermetrics users have been aware of the noisiness of baseball statistics for some time. Fluctuations in player performance make it difficult to measure a player's true skill in small sample sizes. Therefore it is beneficial to note the uncertainty of a statistic, or at least the sample size. Baseball statistics

are often presented as measures of skill even when the sample size is not sufficiently large to support that assertion.

Tango et al. [29] describe a Bayesian approach to capturing player skill in Chapter 1 of their work. A new player with no history is assumed to be roughly average until there is data to draw from. When sample sizes are small it is best not to make any strong assumptions about a player's ability. Tango et al. suggest regressing a player's statistic towards the mean of similar players. This is done by adding a fixed number of average results to the player's statistic so that any prediction of performance is not solely reliant on what was found in the small sample. As data points become available the estimates of that player's skill are updated. The more data there is for a player, the less regression to the mean is necessary.

Sabermetrics users have learned to incorporate the inherent randomness in their models rather than ignore it. Consider the discovery that pitchers have limited control over balls put in play. Pitchers have no control over the ability of defensive players in converting balls in play to outs. Pitchers play a part in how hard a ball is struck, but the probability of an out depends on the skill of the fielders once the ball is contacted by the batter. Separating the fielding aspect of defense from the pitching aspect has been a crucial development in evaluating pitchers; whether a ball in play is converted to an out depends partially on the quality of fielders [32]. This has led to sabermetrics users focussing on defense-independent pitching statistics (DIPS) to predict performance because they rely on a pitcher's skill and not the team's

defense. DIPS have less fluctuation over time, so a pitcher that is skilled in striking out batters in the early part of the season will likely carry over that skill later in the year. Other statistics may be dependent on the skill of other defensive players in converting balls in play to outs. Therefore it is important to evaluate a pitcher's performance on aspects within their control.

The most limiting aspect of baseball analysis is undoubtedly related to sample size. This leads to necessary compromises. Studies that focus on individual players often require multiple seasons of data. In the literature review that follows, it will be seen that papers use up to three seasons worth of data to predict pitch types for individual pitchers. That is a small price to pay, though, for having more data for a model to learn from. Restrictions on sample size may also cause systematic bias; injury-prone and lesser-skilled players wind up being dismissed by many studies.

The major issue with predicting pitch types is the trade-off between assumptions in data distribution and sample size. Ideally, multiple seasons of data are used to build a model. This could be sub-optimal because a pitcher's tendencies may change over time. It is unrealistic for every pitcher to maintain the same distribution of data across multiple seasons, thus violating the standard assumption of independent and identically distributed variables. Would a pitcher drastically changing their approach from one season to the next be bad for a predictive model? There may be one advantage in using diverse data across time. If only recent, homogeneous data is used to construct a model it may end up being overfit when used on test data. Incorporating

cross-validation that separates recent data from older data reduces this effect; here it seems the advantages of larger training sizes and increased data diversity mitigate concerns over distribution.

2.3 The Mechanisms of Prediction

Breiman [9] describes the operation of input variables that lead to response variables as a mechanism. In the specific case of pitch type prediction, the mechanism is controlled by the pitcher (and to some degree, possibly the catcher). It is impossible to totally capture the pitcher's thought-process with a model, but they will most likely have some historical tendency to draw on.

Breiman discerns two different approaches to modelling: data modelling and algorithmic modelling. The former typically fits a parametric model, while the latter accepts the mechanism as "complex, mysterious, and, at least, partly unknowable." [9]. Accepting that some aspects of a mechanism are unknowable doesn't preclude discussion of its mechanics. Next, we hope to elucidate the mechanism of certain machine learning algorithms and their relationship to baseball. Complications specific to the pitch type prediction mechanism are also examined.

2.3.1 Bagging, Randomization and their Effects on Data Structures

Bagging (Bootstrap Aggregating) was introduced by Breiman [7] and is known as an ensemble method. Ensemble methods combine M algorithms for the purpose of improving specific aspects of a predictive model. Bagging is particularly effective for reducing variance.

Let a training set $\mathscr{L}$ be made up of data $\{(y_n, \boldsymbol{x_n}), n = 1, ..., N\}$ and let the predictor with inputs $\boldsymbol{x}$ be $\varphi(\boldsymbol{x}, \mathscr{L})$. For the classification task, $\varphi(\boldsymbol{x}, \mathscr{L})$ predicts a class $j \in \{1, ...J\}$. Taking from the distribution of $\mathscr{L}$, bagging seeks to simulate a sequence of training sets $\mathscr{L}_k$ through bootstrapped samples denoted by $\{\mathscr{L}^B\}$. Each bootstrapped sample forms a prediction for the class. The sequence of predictors $\varphi(\boldsymbol{x}, \mathscr{L}^B)$ vote to form the bagged predictor $\varphi_{(B)}(\boldsymbol{x})$. The most common class from voting ends up being the prediction.

Bagging is just one form of randomization. There are other randomization mechanisms to further improve machine learning tasks. In random forests, the randomization comes in the form of feature usage. Not every feature is used for each bagging iteration. Multiple classification trees are grown, each with a different subset of the features used to grow the trees. Randomization in this manner helps to alleviate the correlation between trees that exists with bagging. Although the bagged trees are identically distributed, they are not independent. According to Goldstein et al. [15], if the variance of

each tree is σ^2, the correlation between predictions is ρ, and B is the number of bootstrapped replicates, then the variance of the bootstrapped averages is:

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

As B becomes larger, the variance becomes increasingly reliant on the correlation between trees. As a result bagging becomes less effective as the size of the data grows. By selectively using subsets of features for each iterated tree in the random forest, the correlation is reduced, thereby also decreasing the variance.

It should be noted that models to predict pitch types inherently violate the assumption that data is independent. A pitcher does not decide which pitch to throw independent of what was thrown previously. They factor in the sequence of pitches when making their choice, essentially creating temporal dependency. Randomization can complicate time-dependent learning tasks if the structure of the predictor $\varphi(\boldsymbol{x}, \mathscr{L})$ is perturbed. Suppose a predictor with a time-dependent structure is given by $\varphi(\boldsymbol{x}(t), \boldsymbol{x}(t-1), ..., \mathscr{L})$ at time t [2]. A process such as bagging could affect the temporal relationship of the predictor since the sample is bootstrapped. The study to be performed later in this paper captures the time-dependency of pitch type prediction by including features describing previous pitches (e.g. previous pitch type, previous pitch velocity, etc.).

2.3.2 Game Theory Considerations in Pitching

The interaction between pitcher and batter is an element of baseball that has only recently been studied extensively. Sabermetrics users have shown the capacity to extrapolate player abilities from data. However, they are still learning to describe the relationship between the respective skills of pitchers and batters. What if the nature of the prediction is not strictly related to skill? Again, consider the task of predicting pitch types. The underlying structure of a pitcher's decision-making processes is not usually obvious, in part because it benefits a pitcher not to be. A pitcher does their best to be unpredictable. The challenge becomes identifying the conscious decision-making tendencies of a player rather than their measurable talent.

One of the difficult aspects in predicting pitch types is the inherent game-within-a-game played between pitcher and batter. Game theory would suggest that a pitcher should have a mix of pitches to avoid becoming predictable. Tango et al. [29] note in Chapter 12 the necessity for pitchers to randomly change up their pitch selection strategies. This means generally that they should avoid throwing one pitch exclusively. Regardless of how well a model is able to learn the historical tendencies of a pitcher in a given game situation, there will likely be some unforeseen strategy taken by the pitcher in the future.

The game theory aspect of the batter-pitcher confrontation makes predictive modelling more difficult. A majority of pitchers are undoubtedly wary of becoming predictable in given game situations with given batters. Thus they

end up avoiding their preferred choice for the sake of confusing the batter. This should have the effect of producing a mapping from inputs to outputs that is non-deterministic. Put another way, a pitcher behaving one way in a training set at time t_1 will not necessarily do the same again at time t_2 in the test set, even with the same inputs. Ideally, the training set will capture the diversity of the pitcher's decision-making. If the model becomes overfit it may end up disregarding their willingness to try different pitch types.

Chapter 3

Literature Review

Various types of machine learning methods are available to be used in classification problems, many of which are relatively new forms of data analysis. Two types of machine learning models will be considered here to predict pitch types: random forests and support vector machines. They are popular model types that have been used in other studies regarding pitch type prediction. Our results, which make use of Statcast data, will be compared to these studies that use data from other tracking systems.

3.1 Random Forests

Classification and Regression Tree (CART) models serve as the basis for random forests. While CART models are simple and interpretable, random forests are more difficult to interpret [4]. However, in high dimensional data

sets, random forests improve on CART methods by combining numerous models to find a stabilizing model.

3.1.1 CART

Nodes are where the splits of classification and regression trees are made; CART methods partition data with binary splits at each decision node. Trees require a split at each node into smaller parts until a terminal node, or leaf, is reached. Splitting based on node impurity is a common tactic in this instance. In classification trees, impure nodes have a greater mix of classes than pure nodes. Completely pure nodes are made up of one class entirely. A purer subset of an impure node can be created by splitting it. The Gini diversity index (GDI) described by Breiman [10] splits based on the greatest reduction in node impurity. The probability of an object being in class j at node t is denoted by $p(j|t)$, and the sample variance is $(p(j|t))(1 - p(j|t))$. If there are J total classes, the sum of the variances for all classes yields the formula for the GDI, which is:

$$GDI = 1 - \sum_{j=1}^{J} p^2(j|t)$$

The nodes of the tree are split to maximize the reduction in node impurity given by the GDI formula. After finding the feature that is the best to split on, the splitting criteria that best reduces node impurity is used to build the following child node. The GDI criterion is generally recognized as a superior

splitting criterion to alternatives such as the twoing method, which does not split based on node impurity [10].

Classification and regression trees are grown until they reach some predetermined stopping condition. This causes the resulting tree to be overfit and thus causes high variance. In the case of overfitting, the CART model should then be pruned. Pruning iteratively removes splits of the tree based on the minimization of the cost-complexity function:

$$R_a(T) = R(T) + a|T|$$

where $R(T)$ is the cost of the tree in the training sample, $|T|$ is the number of terminal nodes in the tree, and a is a hyperparameter to be tuned [33]. The cost function $R(T)$ measures the performance of the model on the given training data. Naturally, the classification rate on the trained model will be better if the tree is larger. The complexity term $|T|$ penalizes the model for overfitting. The value of a begins at 0 and gradually increases with each iteration to remove splits which contribute the least to the reduction of $R(T)$. If there are b iterations of tuning, a series of subtrees $T_1 \subseteq T_{b-1} \subseteq T_b$ will be generated.

3.1.2 Random Forest Algorithm

Random forests are an ensemble method of learning first implemented by Breiman [8]. A large number of decision trees are grown for each model, and

all of them are distinct bootstrap samples. Each tree that is grown is left unpruned, which causes the trees to have high variance. A random group of inputs $\boldsymbol{x_m}$ are used to grow tree T_m. If there are p total features, then there will be $F \leq p$ features selected for each candidate split of T_m. The features are selected at random for each split. This injects randomization in to the model that reduces the correlation between trees. Without it, the good predictors would be used repeatedly in each tree. Combined, bagging and random feature grouping help to reduce variance.

Breiman introduces the idea of bagging with replacement in a random forest, a process whereby a random bootstrap sample of the training data is used to grow each tree. As mentioned in Chapter 2, bagging reduces the variance of the model. Bagging has the added advantage of withholding approximately one-third of the objects in each bootstrap sample, which in turn can be used to find the out-of-bag error rate. The out-of-bag error rate is found by using these withheld objects (again, about one-third of the whole training set) in each sample to find the error for each tree that has been grown. This essentially allows each random forest a built-in set to test on.

In the case of classification problems, the result is determined by tallying up the most popular class from the ensemble of trees, of which we say there are M. Each tree votes for a class, and the class with the most votes ends up being the prediction of the random forest algorithm.

Breiman showed that random forests do not become overfit as the number of trees M increases. As a result, M can generally be increased to improve pre-

dictive power. The other parameters of the random forest algorithm require tuning on training data. Good starting points for random forest parameters are discussed in the next section.

3.1.3 Random Forest Parameters

The number of features F used at each split in a random forest depends on the problem at hand. However, in a data set with p total features Breiman [8] suggests using $F = \sqrt{p}$ features for each split in the model. The value of F should be found through tuning. Ideally, random feature selection balances the strength of a full model having all features with the improvement gleaned by lower correlation between trees.

There are several parameters to consider in the random forest model in addition to M and F. The number of points or objects used in each bootstrap sample is represented by A. The value of A is typically set to the number of total objects in the training set n, as demonstrated by Breiman [8]. The minimum node size N controls when construction of each tree is complete. Once all nodes do not contain more than N points, tree fitting is terminated. There is no standard set of procedures for tuning the parameters of a random forest model. There are, however, some suggested values for each of the parameters for classification problems in the machine learning literature. A larger M value decreases the variance of the model and therefore improves the predictive power of the model. The aim in tuning M should be to build the model to a point where prediction accuracy does not fluctuate [4]. In

effect, we are looking for a minimum M to obtain stable predictions while limiting the computational burden.

Díaz-Uriarte and Alvarez de Andrés [13] found that the difference between minimum node sizes N of 1 and 5 was insignificant for a classification problem involving a classification problem based on gene selection and settled on $N = 1$ since using 5 only caused an inconsequential improvement in computing speed. The randomForest package in R and the RandomForestClassifier model in the scikit-learn library in Python both use $N = 1$ as a default for classification problems.

In classification problems, it is recommended to use a smaller F to reduce correlation and increase predictive power, as mentioned previously by Breiman [8]. The choice of $F = \sqrt{p}$, where F is rounded down to the nearest whole number, is often the default value for classification problems. The randomForest package in R and the RandomForestClassifier model builder in the scikit-learn library in Python both currently use this number as a default.

3.2 Support Vector Machines

Support vector machines (denoted by SVM) are a flexible type of learning method with applications as a classifier. SVM models require no assumptions about the distribution of the data they are trained on. The use of SVM models sacrifices simplicity and interpretability for flexibility.

3.2.1 Support Vector Machine Algorithm: C-SVC

The fundamental details of non-linear SVMs were shown by Boser et al. [6]. The algorithm aims to generate one or more hyperplanes that maximizes the margin between two sets of classes. Non-linear SVMs may be linear in their parameters but need not be linear in their dependency on the input components. Cortes and Vapnik [12] also presented key developments in the algorithm by introducing a method to apply the SVM algorithm to data that is linearly non-separable with the use of a soft margin. A penalty term C is introduced to reduce the margins in this case. The following support vector classifier is denoted as C-SVC by Chang and Lin [11].

Let $(\boldsymbol{x_i}, y_i), 1 \leq i \leq N$ be a set of training data. The pairs $(\boldsymbol{x_i}, y_i)$ take the form $y \in \{-1, 1\}$ and $\boldsymbol{x_i} \in \mathcal{R}^d$ with d dimensions and N objects. The value of y indicates which class the object belongs to. A basic support vector machine may only separate two classes. In cases with greater than two classes the separation between each class is handled differently. The methods of separation will be discussed in the sections that follow.

The support vector machine minimizes the function

$$\tau(\boldsymbol{w}, \varepsilon) = \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} + C\sum_{i=1}^{l}\varepsilon_i \tag{3.1}$$

subject to the constraints

$$\varepsilon_i \geq 0, \quad i = 1, ..., l$$

$$y_i(\boldsymbol{w}\cdot \phi(\boldsymbol{x_i}) + b) \geq 1 - \varepsilon_i$$

C is a regularization parameter that is subject to tuning and must satisfy $C \geq 0$. ϕ is a mapping function that projects $\boldsymbol{x_i}$ in to higher-dimensional space in $\boldsymbol{z_i}$. The $\boldsymbol{z_i}$ are the titular support vectors that are close to the separating hyperplane. The values α_i are used to give the optimal vector $\boldsymbol{w}$ satisfying

$$\boldsymbol{w} = \sum_{i=1}^{l} y_i \alpha_i \boldsymbol{z_i}$$

by way of the Kuhn-Tucker theorem [27].

The α_i values are found through quadratic programming by maximizing

$$W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j (\boldsymbol{z_i}\cdot \boldsymbol{z_j})$$

with constraints

$$0 \leq \alpha_i \leq C, \quad i = 1, ..., l$$

$$\sum_{i=1}^{l} \alpha_i y_i = 0$$

The decision function will ultimately become

$$f(\boldsymbol{x}) = \text{sgn}\left(\sum_{i=1}^{N} \alpha_i y_i \cdot (\boldsymbol{z}\cdot \boldsymbol{z_i}) + b \right)$$

3.2.2 Kernels

In certain feature spaces and image mappings, generating an appropriate linear hyperplane can be nearly impossible. Kernel functions allow for computation of a scalar product in the feature space, essentially generating a non-linear classifier from a linear algorithm [25]. The non-linear boundary is mapped to the feature space as a linear decision boundary, effectively introducing a hyperplane. Among the most commonly used kernel functions are the Gaussian radial basis function (RBF), the sigmoidal function, and the polynomial function. The radial basis function creates a margin that encloses a class in the input space, with the other class outside its bounds. The centers, or support vectors, of both the enclosed and outer classes in the input space are those objects which are important in regulating the classification process [27]. Different kernel functions have been shown to share most of the same support vectors in various data sets. For the kernel function $K(\boldsymbol{x}, \boldsymbol{x_i}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{x_i})$, the decision function to use for classification is

$$f(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{i=1}^{N} \alpha_i y_i \cdot K(\boldsymbol{x}, \boldsymbol{x_i}) + b \right)$$

with constraints

$$0 \geq \alpha_i$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

The RBF kernel is a reasonable starting function to consider. This is because

of its relative simplicity compared to the polynomial kernel (there are few parameters to tune) and because of the similarities it has with the sigmoidal kernel under certain parameters [17]. The kernel takes the form

$$K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \exp(-\gamma||\boldsymbol{x_i} - \boldsymbol{x_j}||^2), \gamma > 0.$$

C and γ are the parameters to consider while tuning the RBF kernel. Recall that C is a regularization parameter in Equation 3.1.

3.2.3 One-Versus-All and One-Versus-One Decision Functions

A support vector machine is capable of separating two classes in its basic form. In cases with more than two classes, the hyperplane must be generated using a method that separates points differently. In the multi-class case the SVM is typically run in one of two ways: either a one-versus-all or a one-versus-one method. With k classes, the one-versus-all structure builds k support vector machines. The i-th class (where $i \in \{1, 2 \ldots, k\}$) has its own SVM built, which labels points matching i as positive and labels that do not match i as negative. The SVM is then constructed as it would be in the binary case, where the inputs are mapped to a higher dimensional feature space by an image function and each class is split up by a hyperplane. After building k decision functions, the input point is assigned its class based on the decision function. Note that the decision function here assigns class based

on the largest value rather than the sign of the function.

The one-versus-one method builds SVMs that are trained on data from two different classes. Every class gets an SVM built against all other classes. Once the decision functions are built, the input is passed through each one and the best class in each direct comparison is found. Since direct comparisons between two classes are being done, the binary decision function is once again based on sign. After all the decision functions have voted, the input is assigned to the class with the most votes [18].

3.3 Review of Studies Predicting Pitch Type Classes

Research involving pitch type prediction has previously been focused on two cases. The two types are binary and multi-class models. The binary classification task usually seeks to predict whether a pitcher will throw a fastball or non-fastball, also known as an off-speed pitch. For multi-class prediction, a larger selection of the pitcher's repertoire is considered. The response for pitcher u will take the form $y_i \in \{1, ..., T\}$, where T is the number of pitch types under consideration. Pitchers do not all throw the same pitch types. If all of a pitcher's unique pitch types are being predicted, then T will vary from one pitcher to another.

A naïve model has been commonly used as a baseline of comparison for both binary and multi-class pitch prediction. The naïve model used for the

multi-class case by Sidle and Tran [28] follows a similar method to one used for binary classification by Ganeshapillai and Guttag [14]. Sidle and Tran determined which pitch was most frequent for a given pitcher in their training set. They then found the naive prediction accuracy P_{N_i} for pitcher i on the test set. For example if pitch type j was the most common pitch in the training set, then P_{N_i} would be the proportion of pitches of type j in the test set. From the batter's perspective, this is akin to anticipating a pitcher's preferred type of pitch. It is entirely possible that when facing certain pitchers this is a reasonable strategy, particularly if the pitcher throws one type of pitch predominantly.

3.3.1 Binary Classification Results

In the binary case, machine learning techniques are used to split the classification between fastballs and non-fastballs. Ganeshapillai and Guttag [14] built support vector machine models to predict fastballs and non-fastballs for pitchers that had thrown over 300 pitches in both the 2008 and 2009 MLB seasons. They achieved a correct classification rate of 70% with these models. The support vector machine models used by Ganeshapillai and Guttag significantly improved on naïve models based on a pitcher's prior pitch probability. The average improvement was 18% for each individual pitcher. This particular measurement is reported as a percentage change.

Hamilton et al. [16] trained on 2008 season data and tested on 2009 season data with an overall prediction accuracy of 77.45% and an average improve-

ment of 20.85% over the naïve model for pitchers with over 750 pitches in both seasons. SVM models with linear and Gaussian RBF kernels were used, along with k-nearest neighbours. This study did not fit models with a fixed set of features, but rather used an adaptive set of features that changed for each pitcher. Significance testing was done for each feature to determine if it should be kept in the model. The data set was also partitioned by pitcher and count so that models were built for each pitcher in a given count.

3.3.2 Multi-Class Classification Results

In the multi-class case, Sidle and Tran [28] used linear discriminant analysis, support vector machines, and random forests to make predictions on seven different pitch types for pitchers with over 500 pitches in both the 2014 and 2015 seasons. PITCHf/x data from MLB Advanced Media was used for all seasons included in the study. In total, 287 pitchers met the criterion for inclusion in the analyses. Not all pitchers had the same set of variables in the study since not all pitchers threw all seven pitch types. The average number of variables for a pitcher was 81, and the most was 103. Random forests with 100 classification trees were the most successful, having an accuracy of 66.62% and an average improvement of 12.24% over the naïve guess.

Bock [5] made use of a support vector machine to separately compute predictability for each pitch type. The data was taken from 2011 to 2013. Only pitchers with 1000 pitches in total over the three seasons were included. The study predicted a pitcher's four most common pitch types and used a lim-

ited out-of-sample test set made up of pitches from the 2013 World Series. The model was primarily evaluated 5-fold cross-validation results by taking a weighted average of the prediction accuracy from each slice of data.

3.3.3 Algorithm Review and General Methodology

Previous studies using binary and multi-class pitch prediction were able to improve on naïve models by using machine learning methods. The studies mentioned previously all used different sets of variables for their models, though there was some overlap. Each study used basic situational game information, including: inning, half inning (top or bottom), outs, count, location of baserunners, and score. Sidle and Tran [28] incorporated variables to measure the tendency of a pitcher to throw each of their different pitch types under various game scenarios. The historical percentage of each pitch type thrown by a pitcher was one such measure of tendency. Other measures of tendency were more granular, such as the historical pitch type tendency of a pitcher against the current batter or the pitcher's tendency in more recent pitches. Detailed pitch information measured by PITCHf/x was used to generate variables based on the prior pitch. Their study also included variables with data specific to the tendencies of the batter the pitcher was facing when the pitch was thrown.

The support vector machine model used by Sidle and Tran [28] employed a radial basis kernel function that uses the one-versus-one method of comparison. The random forest classifier they employed comes from MATLAB and

uses the GDI to determine node impurity. This appears to be the only study using random forests as a means of pitch prediction. Each tree generated in the random forest attempted to find the variables that best split the node by maximizing the decrease in impurity at each step. After the appropriate variable to split on is determined at each node, the best splitting criteria for that variable is found.

Support vector machines have been used frequently in studies involving pitch classification. However, those studies have typically been done in binary classification problems. The work done by Sidle and Tran [28] appears to be the only study performing extensive pitch prediction modelling in the multi-class case over a significant period for many pitchers. This study found that SVMs were not as effective as random forests with 100 classification trees in predicting pitch types, with average prediction accuracies of 64.49% and 66.62% for SVMs and random forests respectively (linear discriminants were also used and had an average prediction accuracy of 65.08%). Of the 287 random forest models built for each pitcher in the study, 282 had a higher pitch prediction accuracy than the naïve model for that pitcher. Meanwhile, 251 of the SVM models bettered their naïve model counterparts.

Variable selection was performed by Sidle and Tran [28] as part of the random forest model that was built for multi-class pitch prediction. The overall variable ranking was done by averaging the variable ranking from each random forest built for an individual pitcher. Variables containing information on the game situation were found to be important. Pitch count was found

to be the most important predictor. Unsurprisingly, the number of balls and strikes against a hitter were found to be significant predictors of the upcoming pitch type. The handedness of the batter was also a significant predictor, which is expected since many pitchers are known to throw a different mix of pitches depending on whether the batter is left-handed or right-handed. Variables containing data on the prior pitch were generally found to be useful as well, suggesting that pitchers can fall into patterns with their sequencing of pitches.

Chapter 4

Methods

Statcast data will be used to predict pitch types from the 2018 MLB season using models built for each individual pitcher. All variables contain information only available before the pitch to be predicted is thrown.

4.1 Data Resources

Major League Baseball Advanced Media (MLBAM) has made PITCHf/x and Statcast pitch data available to the public through a website called Baseball Savant, where much of the useful pitch tracking data goes as far back as 2008. It is possible to scrape an entire season's worth of data through the web page. With potentially thousands of pitches to analyze over the course of a season, a pitcher's abilities and tendencies become more apparent. Statcast has had calibration problems since the transition away from PITCHf/x. The

imperfections of the pitch classification algorithm were noticeable at the start of the 2017 season [1]. It has since become generally accepted that the algorithm is accurate, though not flawless.

The data is scraped from the Baseball Savant web page[1] using R. A majority of the code used to scrape the data was written by Petti [26], with modifications made to account for changing field names and new data fields. The code returns data for each pitch thrown on a given day and iterates over a range of dates. All data for the 2017 and 2018 MLB seasons were extracted with this method. Data was then transferred to a MySQL database for storage.

4.2 Sample and Features

In order to be included in this analysis, a pitcher must have 1000 registered pitches during the 2017 and 2018 MLB regular seasons. The data is split in to training and test sets. The training set is made up of pitches thrown before the 2018 MLB All-Star Game, which was played on July 17, a few weeks after the halfway point in the season. The test set consists of all pitches thrown after that date. Due to some occasional data capture issues with Statcast, some pitches are left without a registered pitch type. Those pitches are dropped from each pitcher's sample. This also appears to be the first study of this nature making use of Statcast rather than PITCHf/x.

Since the purpose of this analysis is to perform multi-class prediction, each

[1] baseballsavant.mlb.com

pitcher's response y_i can take on different values. Suppose a pitcher u has T unique pitch types in his arsenal. Then the possible response values for pitcher u are $y_i \in \{1, 2, ..., T\}$ for pitch i. The set of features for the i-th pitch is given by $\boldsymbol{x_i}$. Certain classes of pitches were grouped due to their similarities. Statcast data distinguishes between two-seam fastballs and sinkers, however the two pitch types are essentially the same. Variations of curveballs were grouped together as well so that standard curveballs, knucklecurves, and eephus pitches all belong to the same category. The differences between changeups, splitters, forkballs, and screwballs can also be difficult to delineate, so they were grouped together as off-speed pitches. There are a small number of pitchers in MLB using a knuckleball, and subsequently no pitcher qualifying for this study used one. The label codes for each re-grouped pitch type will be four-seam fastball (1), two-seam fastball (2), cut-fastball (3), curveball (4), slider (5), and changeup (6).

The features used to predict pitch types can be seen in Table 4.1. Tendency features measure the usage in percent of a particular pitch type for each pitcher. For instance, if a pitcher uses a repertoire made up of four different classes (four-seam fastball, two-seam fastball, slider, and changeup as an example), then there will be four 'Pitcher-Batter Tendency' features for that pitcher.

Many of the features in Table 4.1 required additional coding to supplement the Statcast data. Features describing the game state (inning, outs, balls and strikes, etc.) were mostly baked in to the initial Statcast data. The

Table 4.1: **Features Used to Predict Pitch Types**

Feature	**Data Type**	**Description**
Inning	Categorical	
Half Inning	Binary	Top or bottom of inning
Outs	Categorical	
Balls	Categorical	
Strikes	Categorical	
Batter Handedness	Binary	
Runner on First	Binary	Flag for runner on first base
Runner on Second	Binary	Flag for runner on second base
Runner on Third	Binary	Flag for runner on third base
Score Spread	Continuous	Difference in runs between defensive and offensive teams
Pitch Count	Continuous	Number of pitches thrown by pitcher
Batters Faced	Continuous	Number of batters the pitcher has faced in the game
Infield Fielding Alignment	Binary	Alignment can either be standard or strategic
Previous Pitch Type	Categorical	Previous pitch to current batter
Previous Pitch Type 2	Categorical	Pitch sequence of previous two pitches to current batter
Previous Pitch Result	Categorical	Possibilities are ball, strike, or ball put in play
Previous Pitch Location	Categorical	Previous pitch location zone according to Statcast
Previous Pitch Velocity	Continuous	Velocity measured in m.p.h.
Previous Pitch Spin Rate	Continuous	Spin rate measured in r.p.m.
Pitcher-Batter Tendency	Continuous	Tendency to current batter
Previous Five Tendency	Continuous	
Previous Ten Tendency	Continuous	
Previous Twenty Tendency	Continuous	

exception would be score differential, which was trivial to calculate.

Features that described the tendencies of the pitcher and hitter were not present in the scraped Statcast data and necessitated custom code. These features capture how frequently a pitcher throws a particular pitch. 'Pitcher-Batter Tendency' measures how often a pitcher throws each of their respective pitches historically against the current batter. Tendency measures for how often each pitch has been thrown recently in the current game are also included.

4.3 Random Forest and Support Vector Machine Predictive Modelling

Random forest and support vector machine models will be created for each pitcher to predict pitch types. The machine learning library scikit-learn in Python will be used to help fit these models and find optimal parameters. First, the models will be fit and tuned. The tuned model will be used on the test set for each pitcher.

4.3.1 Model Fitting and Hyperparameter Tuning

The random forest model will use cross-validated grid search optimization to find the best mixture of hyperparameters for each pitcher. A grid search tries different combinations of hyperparameter values on the training set to

find the best model. The number of trees grown for each model will be fixed at 100. The hyperparameter values to be used for the grid-search in the RandomForestClassifer() method in scikit-learn are seen in Table 4.2.

The maximum number of features to try in building each tree has a default value of $\sqrt{n\_features}$. The value $log_2(n\_features)$ will also be considered, since preliminary trials suggested it yielded better results in some cases. The combination of hyperparameters with the highest prediction accuracy in the cross-validated grid search is taken as the optimal solution. The pertinent random forest model for each pitcher will be used to evaluate the test set.

The grid search will be randomized. There are four hyperparameters to tune and 256 possible combinations. The randomized grid search will draw from a uniform distribution of set values for each hyperparameter. A total of 10 iterations will be run with 5-fold cross-validation being used for each iteration of the grid search.

Table 4.2: **RandomForestClassifier() Tuning Options**

Hyperparameter	**Setting Options**
$max\_features$	sqrt, log2
max_depth	70, 80, 90, 100, 110, 120, 130, None
$min\_samples\_split$	2, 5, 10, 15
$min\_samples\_leaf$	1, 2, 4, 6

The hyperparameter max_depth is restricts the lengths of the chains of nodes in each decision tree. The values of $min\_samples\_split$ and $min\_samples\_leaf$ place minimum requirements on the number of samples needed to split each

node and the samples needed at each leaf (i.e. terminal node). The classification trees that are grown in the sci-kit learn implementation of the random forest algorithm are not pruned. The hyperparameters *max_depth* and *min_samples_split* are an alternative to prevent overfitting in a fashion similar to pruning.

Support vector machines are sensitive to features with mis-matching data ranges. Features with large numeric ranges dominate features with small numeric ranges [17]. Due to this the data should be scaled. All data will standardized based on calculations in the training set. This means the test set observations will have their features standardized using sample means and sample standard deviations from the training set. The standardized score z_{ij} for the i-th transformed observation on the j-th feature is given by:

$$z_{ij} = \frac{x_{ij} - \overline{x_j}}{s_j}$$

where $\overline{x_j}$ and s_j are the mean and standard deviation of the feature observations.

The support vector machine models will use a radial basis kernel function and the decision function will be one-versus-all. The grid search will tune the C and γ hyperparameters for the SVC() method in scikit-learn. Recall that C is the soft margin regularization parameter and γ, known as *gamma* in sci-kit learn, is the coefficient of the RBF kernel. Ideally both values need to be tuned to find an optimal combination. Of the 20 possible combinations,

10 will be tried for each pitcher. The values for the hyperparameters in the grid-search are given in Table 4.3.

Table 4.3: **SVC() Tuning Options**

Hyperparameter	**Setting Options**
C	0.1, 1, 10, 100, 1000
$gamma$	0.1, 0.01, 0.001, 0.0001

Chapter 5

Results

5.1 Model Results and Comparison

There were 244 pitchers included in this study based on the criteria outlined in the Methods chapter. Three cross-validated models were built for each pitcher to predict pitch types: a naive model, a random forest model, and a support vector machine model. Note that random forests and support vector machines will be referred to as algorithmic models to differentiate them from the simple naive models being used a baseline comparison.

Recall that the data spans the 2017 and 2018 MLB seasons, and the training set consists of data from before the All-Star Game in 2018. The test set is made up of all pitches after the All-Star Game. Each model for a given pitcher will be judged based on its performance with test data, and specifically how well it classifies pitches. The prediction accuracy of the algorithmic model

for pitcher i on the test data is denoted by $P_{A_i} = \frac{C_i}{T_i}$, where C_i is the number of correct predictions and T_i is the number of total predictions. Similarly, the prediction accuracy on the test set for the naive model for pitcher i is given by P_{N_i}. The average prediction accuracy across all K pitchers in a sample is given by $\overline{P_A} = \frac{\sum_{i=1}^{K} P_{A_i}}{K} * 100$. Similarly, the average prediction accuracy is $\overline{P_N} = \frac{\sum_{i=1}^{K} P_{N_i}}{K} * 100$. Imp is the improvement measured as a difference between P_{A_i} and P_{N_i}. To compare the average improvement ($\overline{Imp}$) of the algorithmic models over the naive models, the difference between $\overline{P_A}$ and $\overline{P_N}$ will be taken.

5.1.1 Naive Model

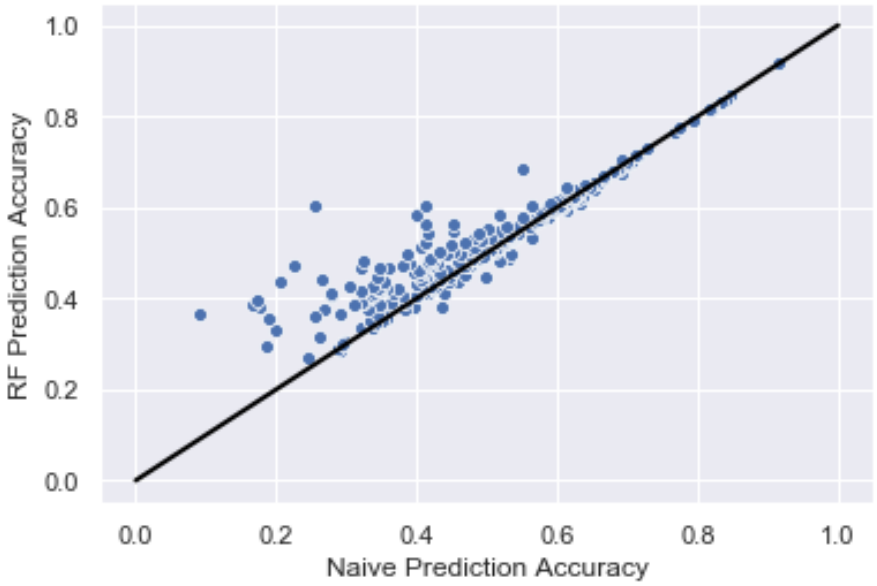
The naive model was fashioned after the one found by Sidle and Tran [28]. First, a pitcher's most used pitch type in the training set was found. Then the percent usage of that pitch was found in the test set. The average naive prediction accuracy across all pitchers was $\overline{P_N} = 47.87\%$, which is well below the value of 54.38 % reported by Sidle and Tran. This is partially due to the fact that the study being conducted here has more starters than relievers. This study is comprised mainly of starters. Relievers typically rely on a smaller repertoire of pitches and will therefore often throw one particular pitch more frequently than others.

5.1.2 Random Forest

Each random forest was made up of 100 classification trees. Tuning was performed to find the best model for each pitcher. The details of the code used to make predictions using these types of models can be found in Appendix A. The average prediction accuracy across all pitchers was $\overline{P_A} = 51.40\%$. Overall, the random forest models performed better than the naive model. The random forest prediction accuracy is plotted against the naive prediction accuracy in Figure 5.1 with a 45° line. The plot shows that the random forest model generally has a higher prediction accuracy than the naive model and is seldom worse. For pitchers with high naive prediction accuracies the random forests do not offer significant improvement. In these cases, the classes are imbalanced because a pitcher relies heavily on one pitch type.

The pitchers with the best results from the random forest models can be seen in Table 5.1. The table is sorted by Imp. Some of the pitchers with large improvements between the random forest and naive models took noticeably different approaches in the test set and the training set. For instance, Wade Miley's naive model predicted the correct pitch just 8.91% of the time. In other words, his most common pitch in the training set, which was a two-seam fastball thrown 27.49% of the time, was only thrown 8.91% of the time in the test set. In the test set Miley adopted the cut fastball as his most common pitch, throwing it 44.81% of the time. The random forest model was able to predict the correct pitch 36.83% of the time despite this change in approach. To a certain degree the improvement is artificial because a

Figure 5.1: **Random Forest Prediction Accuracy vs. Naive Prediction Accuracy**



batter would also be able to recognize that Miley is throwing his two-seam fastball less; a batter would eventually begin expecting cut fastballs rather than naively assuming Miley was following his old patterns. Indeed, many of the pitchers in Table 5.1 have low P_{N_i} values. The improvement in predictions with the random forest can be attributed in part to this.

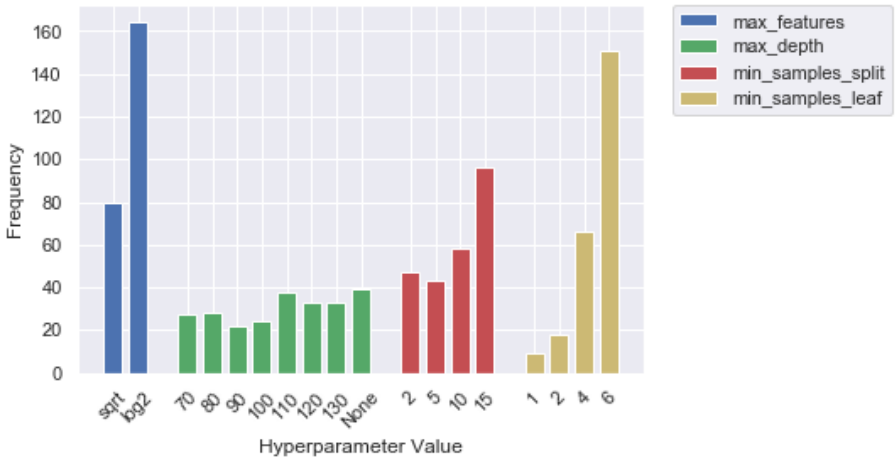
With 244 tuned random forest models (one for each pitcher), it is possible to examine which hyperparameter values occurred most often. The counts of the best hyperparameter values are plotted in Figure 5.2. Limiting the maximum number of features to $log2$ was often found to produce the best tuned model. No one particular fixed depth for each tree stood out. Raising the minimum number of samples above the default value for each split and

Table 5.1: **Random Forest - Pitchers with Greatest Improvement**

Pitcher	Size of Training Set	$P_{N_i}(\%)$	$P_{A_i}(\%)$	Imp
A. J. Cole	1405	25.55	60.20	34.65
Wade Miley	3263	8.91	36.83	27.92
James Shields	4034	22.60	47.31	24.71
Felix Hernandez	3116	20.48	43.52	23.04
Matt Harvey	3185	17.17	39.80	22.63
Wandy Peralta	1539	16.67	38.79	22.12
Jesse Chavez	3191	17.53	38.37	20.84
Kyle Hendricks	3963	41.34	60.52	19.18
Jaime Garcia	3514	40.00	58.55	18.55
Raisel Iglesias	1884	26.32	44.42	18.10

each leaf seems to be preferable for these models.

Figure 5.2: **Random Forest - Best Hyperparameter Results from Cross-Validation**



5.1.3 Support Vector Machine

Support vector machines with radial basis kernel function were fit for each pitcher. The average prediction accuracy was found to be $\overline{P_A} = 50.52\%$. The pitchers with the best results from the support vector machine models can be seen in Table 5.2.

Table 5.2: **Support Vector Machine - Pitchers with Greatest Improvement**

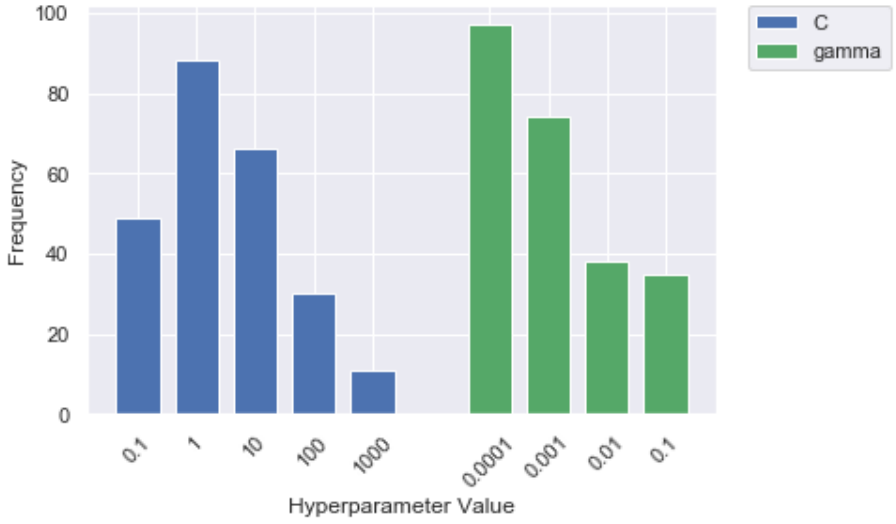
Pitcher	Size of Training Set	$P_{N_i}(\%)$	$P_{A_i}(\%)$	Imp
Felix Hernandez	3116	20.48	43.63	23.15
James Shields	4034	22.60	44.97	22.37
Luis Perdomo	3150	32.31	53.28	20.97
Matt Harvey	3185	17.17	36.49	19.32
Kyle Hendricks	3963	41.34	60.52	19.18
Jaime Garcia	3514	40.00	58.18	18.18
Chris Archer	4800	27.73	42.82	15.09
Jakob Junis	3141	32.02	46.85	14.82
Wandy Peralta	1539	16.67	31.21	14.56
Raisel Iglesias	1884	26.32	40.63	14.31

7 of the 10 pitchers with the greatest improvement using the support vector machines were also among the most improved from random forests. The average improvement for the random forest models was 23.17% for the 10 pitchers , while for the support vector machine models it was 18.20%. For pitchers where the algorithmic models performed best, the random forest models performed better than the support vector machine models in predictive accuracy. The overall difference between the random forest models

and support vector machine models was small. The random forest models performed slightly better by having a $\overline{P_A}$ that was 0.88% higher than the support vector machine models. Similar to the random forest models, many of the pitchers with the highest improvement had low naive model predictive accuracy.

The counts for the best hyperparameter values from the support vector machine classifiers can be seen in Figure 5.3. No C value was obviously preferable to the others, although $C = 1$ was used for 88 out of 244 pitchers. C=1000 was only used 11 times. Smaller *gamma* values were generally better performers, with 171 pitcher models having $gamma = 0.0001$ or $gamma = 0.001$. The other *gamma* values ended up being used with no dominant best choice.

Figure 5.3: **Support Vector Machine - Best Hyperparameter Results from Cross-Validation**



5.1.4 Overall Results and Comparison

The overall results comparing random forest and support vector machine models are in Table 5.3. The total results are broken out by starter and reliever. Starters begin the game as the pitcher for their team, while relievers enter the game after the starter has left. Pitchers were classified as starters or relievers based on the role where they threw the majority of their innings over 2017 and 2018 according to Fangraphs.com. Overall, 57.8 % of the pitchers making it in to this study have been classified as starters. The difference in strategies taken by starters and relievers [29] would seem to make it necessary to split the results in this manner. Relievers tend to vary their pitch selection less than starters.

Table 5.3: **Model Performance Comparison**

Pitcher Type	Pitchers	$\overline{P_N}(\%)$	Measurement	RF	SVC
Starter	141	43.49	$\overline{P_A}(\%)$	47.51	46.76
			$\overline{Imp}$	4.02	3.27
			Count of $P_{A_i} \geq P_{N_i}$	115	119
Reliever	103	53.86	$\overline{P_A}(\%)$	56.72	55.67
			$\overline{Imp}$	2.86	1.81
			Count of $P_{A_i} \geq P_{N_i}$	81	79
All	244	47.87	$\overline{P_A}(\%)$	51.40	50.52
			$\overline{Imp}$	3.53	2.65
			Count of $P_{A_i} \geq P_{N_i}$	196	198

The relievers were more predictable overall with both the random forest and support vector machine models. Their average naive prediction accuracy

was also over 10 % higher. The random forest and support vector machine models for the starters showed a greater average improvement over the naive model than the relievers. The random forest models had a higher $\overline{P_A}$ than the support vector machine models for starters and relievers. On the other hand, SVM models had 198 pitchers out-perform or match their naive models, which was more than the random forests. The 95% confidence intervals for the prediction accuracy using each type of model can be seen in Table 5.4. In the naive case, the 244 values of P_{N_i} are used to build normal confidence intervals for the mean $\overline{P_N}$. The same approach was taken for the random forest and support vector machine models using all P_{A_i} to find the confidence intervals for the mean $\overline{P_A}$.

Table 5.4: **Prediction Accuracy Confidence Intervals**

Model	95% Confidence Interval
Naive	(46.05, 49.69)
Random Forest	(49.98, 52.82)
Support Vector Classifier	(48.93, 52.11)

5.1.5 Permutation Feature Importance

It would be useful to know which features are most are most important in predicting the next pitch. The permutation feature importance (PFI) method first suggested by Breiman [8] provides a way of doing this. A model is fit on training data and evaluated with a baseline model score (prediction accuracy in this case). The training data will remain the same as it was in

previous sections for the purpose of this analysis. Let the baseline prediction accuracy be P. Once a model is fit, data is randomly shuffled in feature column j so that the order of the values is changed. The values in column j do not change, only their order does. The model that was fit before on the training data is used to find prediction accuracy once again, only this time using the data with the permuted feature. If a feature is important in predicting the target label, then the decrease in the prediction accuracy will be larger than if it was unimportant. The difference in the baseline prediction accuracy and prediction accuracy with permuted data is the permutation feature importance score. Additionally, each feature is permuted k times so that the mean of the k prediction accuracies can be used to compare to the baseline model score. For the purposes of this study K will be set to 10. Denote the prediction accuracy with permuted feature j on iteration k as P_{jk}. We can define the permutation feature importance for feature j as

$$PFI_j = P - \frac{1}{K}\sum_{k=1}^{K} P_{jk}$$

The permutation feature importance algorithm is offered in scikit-learn. It will be used to rank feature importances after fitting random forest models for 91 different pitchers with the same randomized grid search discussed before. The pitchers selected to be included are primarily starters, as the threshold for inclusion was 4000 total pitches across 2017 and 2018. Starters have a larger repertoire of pitches, so by focussing on them this examination of

feature importance will place less emphasis on pitchers with overfit predictive models. The overfitting phenomena is mostly associated with relief pitchers (see the next section for more on this).

We will calculate the mean permutation feature importance for each feature across all M=91 pitchers. If pitcher m achieves a permutation feature importance score of PFI_{mj} when j is shuffled, then the mean permutation feature importance of j will be

$$\overline{PFI_j} = \frac{1}{M} \sum_{m=1}^{m} PFI_{jm}$$

.

The results for the total mean feature importance can be seen in Figure 5.4. The categorical features are dummy encoded so that each level is assigned as its own feature. This has the effect of splitting up the overall importance of the feature in to its constituent parts. After encoding them in this manner there are 138 total features. Figure 5.4 ranks all 138 of them.

Figure 5.4 shows that a number of features have small mean permutation feature importances when averaged across 91 pitchers. To display the features which were important, we will zoom in on those ranked in the top 25. The features can be seen in Figure 5.5.

The coding used for the feature names that appear in Figure 5.5 may be difficult to recognize. To translate the coding in Python in to something more understandable, refer to Table 5.5. Recall that these features originally

Figure 5.4: **Mean Permutation Feature Importance**

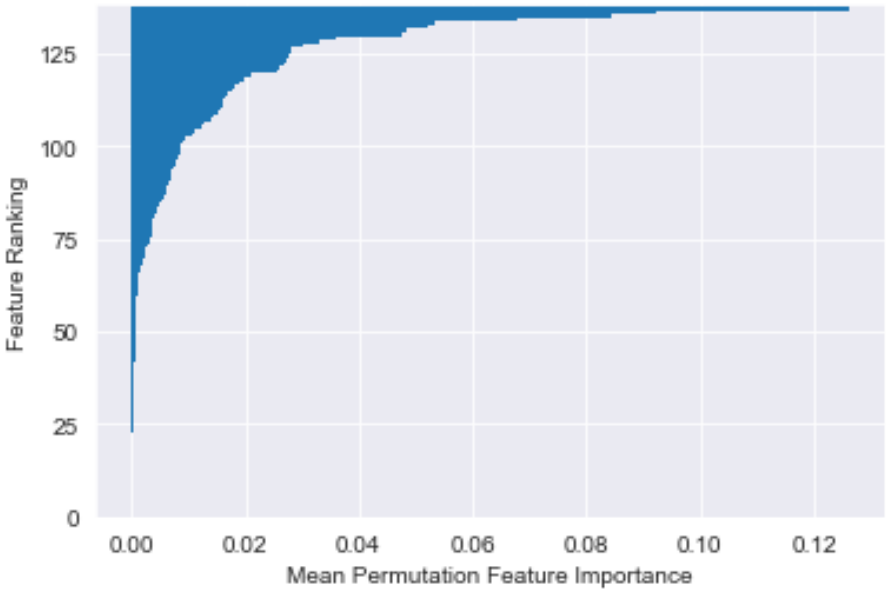
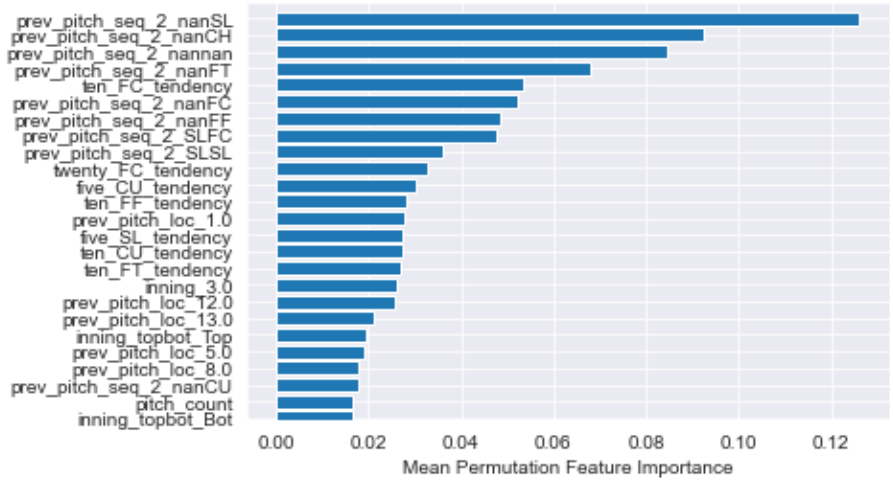


Figure 5.5: **Top 25 Mean Permutation Importance Features**



appeared in Table 4.1. Notice that the coding names for categorical and binary features in Table 4.1 differ from those in Figure 5.5. The coding names in Figure 5.5 have the category appended to the end of the feature where it applies. For instance 'prev_pitch_seq_2_nanSL' is a dummy feature indicating what the previous two pitches in the plate appearance were. In that case, 'nan' means there was no pitch to the current batter two pitches ago. The pitcher has only thrown one pitch in the current plate appearance, which was a slider ('SL').

The features that are most important capture what a pitcher has done recently in the game. This is expected. A pitcher's strategy is known to depend on the sequence of pitches to the current batter. The tendency of pitchers beyond their previous two pitches is also important. In terms of continuous numerical features, the pitchers' tendencies in the previous five, ten, and twenty pitches are predictive as well. Pitch count is another numerical predictor with notable importance because a pitcher may alter their strategy as the game goes along.

Table 5.5: **Description of Features with Highest PFI**

Coding	Data Type	Description
prev_pitch_seq_2	Categorical	Previous two types of pitches in plate appearance
ten_FC_tendency	Numerical	Percentage of cut fastballs in previous ten pitches
twenty_FC_tendency	Numerical	Percentage of cut fastballs in previous twenty pitches
five_CU_tendency	Numerical	Percentage of curveballs in previous five pitches
ten_FF_tendency	Numerical	Percentage of four-seam fastballs in previous ten pitches
prev_pitch_loc	Categorical	Previous pitch location in strike zone
five_SL_tendency	Numerical	Percentage of sliders in previous five pitches
ten_CU_tendency	Numerical	Percentage of curveballs in previous ten pitches
ten_FT_tendency	Numerical	Percentage of two-seam fastballs in previous ten pitches
inning	Categorical	Inning of the game
inning_topbot	Binary	Top or bottom of inning
pitch_count	Numerical	Number of pitches thrown in game

5.2 Discussion of Results

5.2.1 On the Practicality of Pitch Type Prediction Models

The question should be raised: how useful is it to build models predicting pitch classes? The applicability of these predictions rely largely on the pitchers they are being used on. For pitchers that use one pitch predominantly, algorithmic models seem to offer diminishing returns over the naive model. This is particularly true for relievers. The models become overfit and naively assume a pitcher will throw their dominant pitch. This sort of problem is common with data that has class imbalance. Random forest results for some notable relievers that are especially predictable can be seen in Table 5.6. The prediction accuracy of the random forest model P_{A_i} is the same as the naive model's P_{N_i} for each pitcher. In these cases, the random forests naively predicts a pitcher's favoured pitch.

Table 5.6: **Predictable Pitchers**

Pitcher	$P_{N_i}(\%)$	$P_{A_i}(\%)$
Bryan Shaw	91.58	91.58
Jake McGee	84.49	84.49
Chad Green	83.33	83.33
Kenley Jansen	81.53	81.53
Josh Hader	77.41	77.41

Starting pitchers had a greater improvement in their algorithmic model pre-

diction accuracy than relievers did. Starters generally use their mix of pitches more situationally, which an algorithmic model is able to pick up on (i.e. they have pitches that are used in specific game scenarios). Situations where the pitch classes are imbalanced such as we see in Table 5.5 require different treatments if we wish to have better predictions of the minority classes.

The biggest constraint in predictive pitch type modelling is sample size. This study, and others like it, require pitchers to have thrown hundreds of tracked pitches. In reality, a large portion of pitches thrown in MLB come from pitchers with limited data attached to them. Constructing predictive models in those cases may not be productive and further reduces how broadly these sorts of models should be employed.

5.2.2 Future Work

All of the preceding analyses has been done under the assumption that Statcast data is accurate in its measurements. However, there is some chance that sections of the Statcast data are not calibrated correctly, or at least differently from PITCHf/x data. Sidle and Tran [28] used PITCHf/x data for their analysis. The potential calibration issues of Statcast were noted by Arthur [1]. The difference between the results here and those found by Sidle and Tran could in part be due to differences in tracking systems. In addition, only two full seasons of Statcast data were available at the undertaking of this study compared to the three they used.

Statcast takes the measurements of each pitch and uses a neural network

to classify the pitch. The details of the neural network are not made public, therefore it is unclear exactly how pitches are classified. The algorithm originally used by PITCHf/x was tweaked and improved over time. PITCHf/x used to attach a confidence number to the algorithm so it could identify which pitches were difficult to classify. It appears Statcast does not offer the same thing. The work of Sidle and Tran only included pitches that could be classified with 80% confidence. This is the confidence the classification algorithm has in being correct, and not a confidence level. Ultimately, the exact differences between the PITCHf/x and Statcast classifiers remain uncertain. Due to the possible issues with Statcast's classification algorithm, it seems reasonable to explore other methods of grouping pitches. That would involve either re-classifying the data with a new algorithm or adjusting the existing data. The adjustment could come in the form of systematically removing pitches with questionable values associated with them (e.g. pitches with aberrant velocity or movement for their given type).

The grouping of pitch classes should cater to the preferences of the batter. Binary models which predict whether the next pitch will be a fastball or non-fastball have higher prediction accuracy than multi-class models. Although a binary model may not be able to tell a batter which specific pitch is coming, it could help them strategically rule out particular pitches. Future work could also be done to reduce the number of features so a batter can focus on those with strategic significance. Using permutation feature importance, we know which features may be most predictive. Future models could be built

using those important features as a benefit to the batter.

This study does not make any predictions using data gathered after the delivery of the pitch. Since we are primarily interested in building models to simulate real game situations, batter and pitcher tendencies only include data that would be available beforehand to each party. A portion of the predictive work done by Sidle and Tran uses batter and pitcher tendencies with data points posterior to the pitch being predicted, although it is done exploratorily to compare algorithms. This could, in part, explain the differences in results. Sidle and Tran also include a section where they make predictions in 'real-time' with methods similar to ours. When using these methods, their models had an overall lower prediction accuracy. The results in that instance were closer to what we found in this study.

References

[1] R. Arthur, *Baseball's new pitch-tracking system is just a bit outside*, `https://fivethirtyeight.com/features/baseballs-new-pitch-tracking-system-is-just-a-bit-outside/`, April 28, 2017, Accessed: 2019-10-29.

[2] Sebastián Basterrech and Andrea Mesa, *Bagging technique using temporal expansion functions*, Proceedings of the Fifth International Conference on Innovations in Bio-Inspired Computing and Applications IBICA 2014 (Cham) (Pavel Kömer, Ajith Abraham, and Václav Snášel, eds.), Springer International Publishing, 2014, pp. 395–404.

[3] D. J. Berri and J. C. Bradbury, *Working in the land of the metricians*, Journal of Sports Economics **11** (2010), no. 1, 29 – 47.

[4] G. Biau and E. Scornet, *A random forest guided tour*, TEST **25** (2015), 197 – 227.

[5] J. Bock, *Pitch sequence complexity and long-term pitcher performance*, Sports **3** (2015), no. 1, 40–55.

[6] B. E. Boser, I. M. Guyon, and V. N. Vapnik, *A training algorithm for optimal margin classifiers*, Proceedings of the Fifth Annual Workshop on Computational Learning Theory (New York, NY, USA), COLT '92, ACM, 1992, pp. 144–152.

[7] L. Breiman, *Bagging predictors*, Machine Learning **24** (1996), no. 2, 123–140.

[8] L. Breiman, *Random forests*, Machine Learning **45** (2001), no. 1, 5–32.

[9] ———, *Statistical modeling: The two cultures*, Statistical Science **16** (2001), no. 3, 299 – 215.

[10] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and regression trees*, Wadsworth International Group, Belmont, CA, 1984.

[11] C.C. Chang and C.J. Lin, *Libsvm: A library for support vector machines*, ACM Trans. Intell. Syst. Technol. **2** (2011), no. 3, 27:1–27:27.

[12] C. Cortes and V. N. Vapnik, *Support vector networks*, Machine Learning **20** (1995), 273–297.

[13] R. Díaz-Uriarte and S. Alvarez de Andrés, *Gene selection and classification of microarray data using random forest*, BMC Bioinformatics **7** (2006), no. 1, 3.

[14] G. Ganeshapillai and J. V. Guttag, *Predicting the next pitch*, MIT Sloan Sports Analytics Conference, 2012.

[15] Benjamin Goldstein, Eric Polley, and Farren Briggs, *Random forests for genetic association studies*, Statistical Applications in Genetics and Molecular Biology **10** (2011), 32–32.

[16] P. Hoang, M. Hamilton, H. Tran, J. Murray, C. Stafford, L. Layne, and D. Padget, *Applying machine learning techniques to baseball pitch prediction*, International Conference on Pattern Recognition Applications and Methods, 01 2014.

[17] C.-W. Hsu, C.C. Chang, and C.-J. Lin, *A practical guide to support vector classification*, Tech. report, Department of Computer Science, National Taiwan University, 2003.

[18] C.-W. Hsu and C.-J. Lin, *A comparison of methods for multiclass support vector machines*, IEEE Transactions on Neural Networks **13** (2002), no. 2, 415–425.

[19] B. James, *The new Bill James historical baseball abstract (first Free Press trade paperback edition)*, Free Press, New York, NY, 2003.

[20] ———, *The Bill James handbook 2012*, Acta Sports, Chicago, IL, 2011.

[21] B. James and R. Neyer, *The Neyer/James guide to pitchers*, Fireside, New York, NY, 2004.

[22] K. Koseler and M. Stephan, *Machine learning applications in baseball: A systematic literature review*, Applied Artificial Intelligence **31** (2018), no. 9 - 10, 745 – 763.

[23] M. Lewis, *Moneyball*, W.W. Norton & Company Ltd., New York, NY, 2004.

[24] B. Lindbergh and T. Sawchik, *The MVP machine*, first ed., Basic Books, New York, NY, 2019.

[25] K. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf, *An introduction to kernel-based learning algorithms*, IEEE Transactions on Neural Networks **12** (2001), no. 2, 181–201.

[26] B. Petti, *How to build a Statcast database from BaseballSavant*, `https://billpetti.github.io/2018-02-19-build-statcast-database-rstats/`, Accessed: 2020-03-02.

[27] B. Scholkopf, Kah-Kay Sung, Christopher Burges, Federico Girosi, Partha Niyogi, Tomaso Poggio, and Vladimir Vapnik, *Comparing support vector machines with Gaussian kernels to radial basis function classifiers*, Signal Processing, IEEE Transactions on **45** (1997), 2758 – 2765.

[28] G. Sidle and H. Tran, *Using multi-class classification methods to predict baseball pitch types*, Journal of Sports Analytics **4** (2017), 1–9.

[29] T.M. Tango, M.G. Lichtman, and A.E. Dolphin, *The book*, Potomac Books, Inc., Dulles, VA, 2007.

[30] T. Verducci, *Seeing is believing*, Sports Illustrated **130**, no. 6, 44–54.

[31] T. Williams and J. Underwood, *The science of hitting (revised Fireside edition)*, Simon & Schuster, Inc., New York, NY, 1986.

[32] K. Woolner and D. Perry, *Why are pitchers so unpredictable?*, Baseball Between the Numbers (J. Keri, ed.), Basic Books, New York, York, 2007, pp. 48–57.

[33] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg, *Top 10 algorithms in data mining*, Knowledge and Information Systems **14** (2008), no. 1, 1–37.

Appendix A

Building Random Forest Models with a Randomized Grid Search

The coding language used to perform analysis in this study was Python. The predictive models were fit with algorithms from the scikit-learn library. The code that follows in this appendix outlines the implementation of a random forest model with randomized grid search for hyperparameter tuning. The full list of packages and libraries used to perform this analysis can be loaded with the code that follows. This list includes packages not used in the appendix. However, they were used elsewhere over the course of this study. Interested parties are encouraged to research the documentation of these libraries and packages.

```
import mysql.connector
import pandas as pd
import numpy as np
from statistics import mean
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn import tree, preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.model_selection import RandomizedSearchCV
from sklearn.inspection import permutation_importance
```

The random forest model for each pitcher is built using a number of variables. Suppose that *data* is a dataframe of features that can be used to make predictions about what the next pitch will be. The column of y values that we would like to predict is labelled 'adj_pitch_type' and represents the type of pitch thrown in that instance. The 'game_date' column represents the date the game was played and will be dropped after parsing the data appropriately. Games after July 17, 2018 will make up the test set. The All-Star Game was held on that date in 2018.

```
#Sort games by date
data = data.sort_values(by=['game_date'])
```

```
#Split up data in to train and test set
X_train = data.loc[data['game_date'] <= '2018-07-17',\
data.columns !='adj_pitch_type']

X_test = data.loc[data['game_date'] > '2018-07-17',\
data.columns !='adj_pitch_type']

#The 'game_date' column is dropped
X_train.drop(['game_date'],axis=1,inplace=True)
X_test.drop(['game_date'],axis=1,inplace=True)

y_train =\
data.loc[data['game_date'] <= '2018-07-17',['adj_pitch_type']]

y_test =\
data.loc[data['game_date'] > '2018-07-17',['adj_pitch_type']]

#Convert training and test sets to arrays
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
```

There must be a grid of options for the randomized search to work on. The grid used for the random forest model can be seen below, with all values tried for each hyperparameter.

```
#Create grid of options for grid search
n_estimators = [100]
max_features = ['auto', 'log2']
max_depth = [70,80,90,100,110,120,130]
max_depth.append(None)
min_samples_split = [2,5,10,15]
min_samples_leaf = [1,2,4,6]
grid = {'n_estimators': n_estimators,
                'max_features': max_features,
                'max_depth': max_depth,
                'min_samples_split': min_samples_split,
                'min_samples_leaf': min_samples_leaf}
```

Predictions will be made using a random forest model. The model will try 10 combinations from the options listed in the grid and select the best one. The combination of parameters with the highest correct classification rate on the training data will be selected as the best model.

```
#Create random forest
rf = RandomForestClassifier(random_state=42)
```

```
#Random search of parameters,
#using 5 fold cross validation,
#search across 10 different combinations,
#and use all available cores
rf_random = RandomizedSearchCV(estimator = rf,
                               param_distributions = grid,
                               n_iter= 10,
                               cv =5,
                               random_state=24
                               verbose=2,
                               n_jobs = -1)

#Fit the random search model
rf_random.fit(X_train, y_train)

#Extract model with best results
#The best model has the highest correct classification rate
best_grid = rf_random.best_estimator_

#Make predictions using test data
y_best_pred_rf = best_grid.predict(X_test)
```

This method of prediction was repeated for each pitcher. Similar code was

used to make predictions using support vector machine models. The SVM model also used 10 iterations to search for the best hyperparameter combinations.

Vita

Candidate's full name: Jacob Morehouse
University attended: Bachelor of Arts and Science, UNB, 2016
Publications: None
Conference Presentations: None