ABSTRACT

SIDLE, GLENN DANIEL. Using Multi-Class Machine Learning Methods to Predict Major League Baseball Pitches. (Under the direction of Hien Tran.)

As the field of machine learning and its applications grow, there is a need to expand the ability to classify and predict beyond just being able to handle a binary problem. While the ability to predict a yes or no answer is still valuable, in a world of increasing complexity, machine learning methods are now employed in a multi-class problem setting more than ever.

Major League Baseball provides a rich and detailed data set with its PITCHf/x system that tracks every baseball pitch thrown in every stadium in every game. Pitch prediction has been a topic of previous research that has mostly focused on the binary split between fastballs and other pitch types. We extend the binary problem to a multi-class one that involves up to seven unique pitch types. The work done with baseball pitches can be used as a template to expand a wide variety of other binary predictions into accommodating more than a simple two class approach. To accomplish the multi-class prediction, we examine multiple machine learning methods to find the most efficient and accurate algorithm.

In this dissertation, we first explore the results given by each of three different methods: Linear Discriminant Analysis, Support Vector Machines, and bagged random forests of Classification Trees. Our feature set is created from both categorical and continuous data, with features that can be taken from the observation of a game used alongside synthetic features generated from historical data. Using the same methods, we explore adaptive feature selection methods in a novel approach, using Decision Directed Acyclic Graphs to allow for the use of binary techniques to reduce the feature set. We then employ post processing methods to determine a measure of variable importance to find what inputs are the most informative for our model. Finally, we implement the prediction method in a live game environment, presenting the results of the different construction of the model and discussing the difficulties associated with the time limitations.

Using Multi-Class Machine Learning Methods to Predict Major League Baseball Pitches

by
Glenn Daniel Sidle

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Applied Mathematics

Raleigh, North Carolina

2017

APPROVED BY:

John Griggs	Negash Medhin
Kevin Flores	Hien Tran Chair of Advisory Committee

DEDICATION

To the Pittsburgh Pirates.

BIOGRAPHY

Glenn Sidle was born in southeastern Connecticut in the fall of 1991. Once he graduated from high school in North Stonington, CT, he went through a brief period of time where he wanted to be, among other things, a writer, a high school teacher, and a Naval Officer. Attending college at Duquesne University in Pittsburgh, PA, he found a new home and discovered the excitement and frustration of research. After graduating with a B.S. in Mathematics, he made his way south and enrolled at North Carolina State University in Raleigh, NC in the Applied Mathematics Ph.D. program. Finding a balance between being a jock and being a nerd, he focused on applying machine learning to new applications in sports analytics. He took a job at Vadum, Inc. in Raleigh in the fall of 2016 and is excited to continue working in mathematics and machine learning.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Hien Tran, for all the help and guidance he has given over the past four years, as well as his willingness to let me spend years working on sports analytics research. Also, I want to thank Dr. John David, who was the first person to ever introduce me to the world of machine learning and who encouraged me to pursue topics that I was passionate about. And I will always be grateful to Dr. Stacey Levine, my advisor at Duquesne, who sparked my passion for research and problem solving.

I would like to thank my parents, my brothers, my aunt and uncle, and the rest of my family for their support and the motivation to research something that I can explain at Thanksgiving. I am, and always will be, indebted to my friends for their round-the-clock help, advice, and availability whenever I need them.

And finally, I would like to thank you, the reader, for whatever reason you may have come across my work, I hope you enjoy it.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1

INTRODUCTION

Major League Baseball (MLB) is a massive organization with close to 10 billion dollars in revenue in 2015 [10], and is only getting bigger. One of the biggest draws to a baseball game is scoring, and as pitching skill has increased over the past few years, it has become more and more important for a runner to get on base. Of the 31 teams that advanced to the postseason over the past three seasons, 25 of those teams ranked in the top half of the league in on-base-percentage (OBP), and 21 of those were in the top ten teams in OBP [17]. OBP is a measure not only of a batter's ability to get a hit, but also draw a walk or, in rare cases, get an intentional walk, get hit by a pitch, or have a third strike dropped. As statistical analysis has become more and more common in baseball, OBP is replacing batting average as a true mark of a batter's ability [31]. Being able to anticipate the next type of pitch that will be thrown would help a batter decide to swing or not, given the tendency of some pitches to result in balls or strikes.

While we employ machine learning in an effort to predict the type of pitch that will be thrown, the initial classification of a pitch is a difficult problem alone. MLB Advanced Media (MLBAM) employs a neural network algorithm that automates this task, and we consider seven unique pitch types as classified by MLBAM.

In this work, we seek to provide an accurate prediction of what type of pitch will be thrown next. Because each pitcher has a unique style of throwing and not every pitcher throws every type of

pitch, we create individual models for each pitcher. Being able to predict from up to seven different pitch types is not a trivial problem, however, as many machine learning methods were designed originally for binary type prediction. Extending these methods from two classes to more involves an exponential increase in complexity and computation time, as well as being susceptible to large class imbalances. We measure our success not only in overall accuracy but also relative to the best naive guess, as well as the ability of the method to beat the naive guess for the set of pitchers.

1.1 Prior Work

Much of the previous work has focused on a binary split between baseball pitches into the categories of fastball and not fastball. In [23], Guttag et al., looked at 359 pitchers, training the model with data from 2008 and testing with data from 2009 and comparing their models predictive accuracy against a naive guess (which we will discuss in Chapter 4). They achieved an average accuracy of 70%, with varying degrees of success depending on the pitcher's initial naive guess and amount of available data.

With feature selection, this binary prediction has achieved close to 80% accuracy [27], but has many problems associated with it. There is no unifying definition of a "fastball." While two and four seam fastballs are the most obvious members of the category, pitches such as sliders, sinkers, and cutters can often be included as well, despite having very different appearances and behavior when thrown. Our initial work involved using at least five different types of pitches, but we eventually extended our prediction to incorporate seven pitch classes. Guttag et al., also modified the binary approach to consider other types of pitches against the others, and in doing so addressed up to six different types of pitches: fastballs, changeups, sliders, curveballs, split-finger fastballs, and cutters [23] . To our knowledge no other binary approaches have been modified this way to consider pitch types other than fastballs as the "positive" class.

Bock et al., [7] is the only publication in a scientific journal that has used a multi-class approach to pitch prediction, predicting up to four unique pitch types. The authors of the paper employed a one-versus-all (which we will discuss more in Chapter 3) support vector machine based method using a linear and radial basis kernel functions. Using a data set of 402 pitchers from the 2011, 2012, and 2013 seasons, they used the cross validation accuracy as a measure of predictability. Across all pitchers, they report an average cross-validation accuracy of 74.5%. The authors then examine the correlation between the predictability of the pitcher and some standard statistics, looking to forecast pitcher success. They examined the ERA and FIP of each pitcher, eventually determining there was "no significant relationship" between predictability and performance.

The true accuracy reported for blind prediction in [7] was listed as less than 61%, with the out

of set sample taken only from a subset of pitchers who pitched in the 2013 World Series. Because the focus of the paper was on using predictability to gauge long term pitcher performance, the out of sample accuracy was only briefly mentioned. The authors used a range of 14 pitchers from the World Series who faced as few as five batters and as many as 60, resulting in possibly skewed or class imbalanced datasets. While this out of sample accuracy is a proof of concept, the number of pitchers and size of the datasets used do not give a well-tested measure of the model's ability to predict pitches it hasn't seen before.

Another multi-class prediction method was proposed in [53] where the author employed a decision tree based method. This work again only examined four different pitch types and also restricted the results to a proof of concept based on the prediction performance for two pitchers. Woodward discusses the results for Justin Verlander and Clayton Kershaw, stating that overall prediction for Verlander is 50%, well below the naive guess for Verlander. He does, however, examine the performance of the decision tree classifier measured against totally random guessing. Our work incorporates many of the ideas proposed in [7] and [53], but we extend the number of classes and the research to optimize our methods in a much more thorough manner.

While multi-class machine learning methods have not been applied to the problem of pitch prediction outside of [7] and [53], they have been implemented to predict or classify other types of problems. Multi-class support vector machines are used on a number of standard datasets from the UCI machine learning repository in [29] and [51]. Both [29] and [51] are comparison papers, giving an overview of the different formulations of the multi-class support vector machine method, tested on datasets with a minimum of four classes and a maximum of 17 classes.

In [25], the multi-class support vector machine is demonstrated on massively multi-class datasets, i.e., classifying an image of a flower into one of roughly 20,000 types of flowers. While the support vector machine models did not outperform the method the authors were introducing, it is an excellent demonstration of the extendability of the support vector machine classification on datasets of up to more than 96,000 classes.

As linear discriminant analysis is not the most common machine learning classification method, even for binary problems, few multi-class journal articles have been published. A survey paper by Li et al., [32] gives an overview of the method and its extension to multi-class classification, again using many of the same benchmark datasets from the UCI repository as [29] and [51] as well as others. In this paper, linear discriminant analysis is used on datasets ranging from three to one hundred unique classes.

Finally, classification trees were again initially designed for binary classification but have been extended into the multi-class domain. In [16], Dietterich et al., use the multi-class decision tree as a baseline method for error correcting, applying the method to datasets ranging from six to 26 classes.

Holmes et al., introduce alternating decision trees in [28], applying the classifier to binary datasets as well as multi-class datasets also ranging from three to 26 classes. The successes in implementing the different machine learning methods in previous works gives us a good starting point for the pitch prediction problem, where we encounter pitchers with pitch classes ranging from two to seven pitch types.

1.2 Overview and Contributions

In Chapter 2 of this work, we first detail the PITCHf/x system and the data we mined from the repository provided by the MLB. We introduce a novel feature set, including not only observable measurements or historical trends but a combination of the two, balancing the risk of uncertainty and the extra information gained from the PITCHf/x system. Chapter 3 provides an overview of the machine learning methods we use for prediction, our parameter optimization algorithm, and an explanation of the committee approach, which has been very lightly used in previous work for multi-class problems. We also introduce the use of the DIRECT method for parameter optimization for the classifiers we use.

We present our results from each method in Chapter 4 and determine the best method based on accuracy and efficiency, demonstrating better than state of the art results from all the methods examined. As well as presenting the overall results, we also show a breakdown by relievers versus starters and compare our improvement against a naive guess to standard metrics of pitcher performance. In Chapter 5 we detail the decision directed acyclic graph implementation of the multi-class methods. We also employ feature selection in an innovative approach, using the unique architecture of the directed graph implementation to use binary feature selection methods. In the same chapter we also use post-processing methods to measure levels of variable importance and consider whether the results match our expectations.

In Chapter 6, we discuss the implementation of the pitch prediction in a live, real-time game environment, using open-source Python code. We also introduce the possible implementation of the prediction method into an iOS compatible application for the iPad tablet. Finally, in Chapter 7, we give conclusions and detail contributions from our work and discuss future improvements that can be made.

CHAPTER

2

FEATURE SELECTION

Baseball is one of the most popular sports in the world and is played in countries around the globe. Because of its unique method of gameplay, baseball is easily broken into discrete events. Each pitch and its subsequent result can be observed and recorded in real time. The MLB is considered to be the highest level of baseball in the world, and as such has some of the most advanced technology available to measure and record different data about each game.

2.1 PITCHf/x

The PITCHf/x system was introduced in 2006 in all 30 MLB stadiums. Three tracking cameras are used to track every pitch from the moment the pitcher releases it to the moment the ball crosses the plate or is hit by the batter. The system records the speed, spin rate, break angle, break length, and trajectory of the pitch among many other characteristics, and stores the data alongside other game situational information such as runners on base, the time the pitch is delivered, and the name of the batter [44]. Previous work in classifying pitches has used the pitch characteristics in order to determine what type of pitch was thrown, similar to the MLBAM's neural network algorithm [5]. These classification approaches have the advantage of being able to take into account these attributes of the pitch, but in order to predict the pitch the decision must be made before the

pitcher releases the ball.

In order to download the data, we used the MAMP software stack. MAMP is an open-source software that runs on Apple computers, its name coming from the operating system Mac OSX, the web server Apache, the database management system MySQL, and the programming languages PHP, Perl, and Python [33]. MAMP uses a structured query language (SQL) to scrape data off the .xml pages found on gd2.mlb.com. This data was downloaded to a comma separated value (.csv) file that can be opened in Excel. We used this structured data in MATLAB.

All PITCHf/x data is stored at gd2.mlb.com and available to the public. Figure 2.1 shows what an example at-bat looks like from the .xml version of the data on the site itself (the data is from the last at-bat from the 2016 World Series) with all the individual information provided for each pitch.

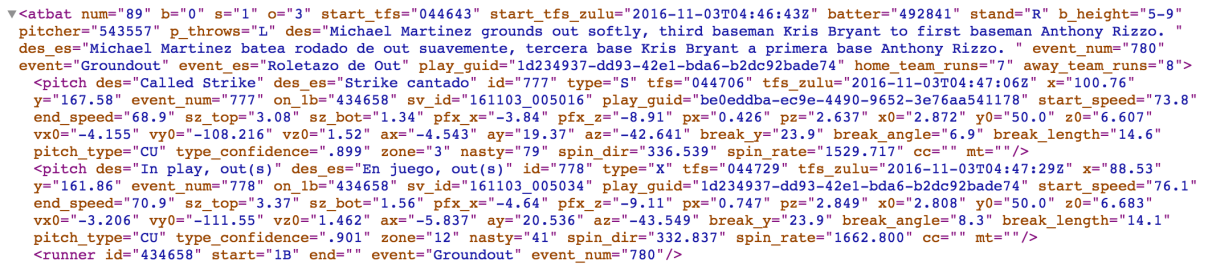


Figure 2.1: A sample at-bat screenshot from the gd2.mlb.com site.

We used data from the 2013, 2014, and 2015 regular seasons, which amounted to nearly 2.1 million total unique pitches. We included both starting pitchers, who throw just over 100 pitches per start, making up to 30 or 35 starts per season as well as relievers and closers, who play in many more games but throw fewer pitches per appearance [12]. We restricted our data set only to pitchers who threw at least 500 pitches in both the 2014 and 2015 seasons, which left us with 287 total unique pitchers, 150 starters and 137 relievers as designated by ESPN. Because each pitch that is thrown is classified by MLBAM's neural network method, each pitch is also assigned a confidence level, and in order to ensure the fidelity of our data we restricted the data to pitches with a type confidence level above 80%. The average size of the data set for each pitcher was 4,682 pitches, with the largest 10,343 pitches and the smallest 1,108 pitches.

The MLBAM neural network classifier has fifteen unique labels that it can assign a given pitch. Based on prior knowledge of the types of pitches thrown, we combine some of them into a single class of pitch, leaving seven unique pitch types that we consider in our training and testing sets.

Table 2.1 shows the PITCHf/x label, the common name, and the label we use.

Table 2.1: The different types of pitches considered.

Class Number	PITCHf/x Symbol	Name	Label
1	FA, FF, FT, FS, SF	Fastball	**FF**
2	FC	Cutter	**CT**
3	SI	Sinker	**SI**
4	SL	Slider	**SL**
5	CB, CU, KC, SC	Curveball	**CU**
6	CH	Changeup	**CH**
7	KN, EP	Knuckleball	**KN**

2.2 Training and Testing Sets

Most of the authors in previous literature split data into training and testing sets based on different seasons. Our first split of data used all of 2013 and 2014 as the training set for each pitcher and all of 2015 as the testing set. But, because pitchers change their behavior over time [6], we needed to take that into account. During the offseason, pitchers will decide to work on or change pitch types that they use, or develop entirely new pitch types and scrap old ones. To account for the possible changes in pitcher behavior, we decided to include the first quarter of each pitcher's 2015 season as part of the training set, and use the remainder of the 2015 season as the testing set. Table 2.2 shows the properties of the training and testing sets as well as the amount of pitchers throwing the respective number of pitches in our training and testing sets.

2.3 Input Features

To create a set of inputs, or feature set, for the pitch prediction methods, we first extracted information about the game situation at the time the pitch is thrown, such as the players on base, the time of day, and the inning. We also included information about prior events in the game, from the previous pitch or the previous at bat. In doing so, we found 23 situational features for each pitch, nearly all of which were categorical. Figure 2.2 shows the 17 unique location zones we divided the plane that stretches across the front of home plate in order to determine a categorical measure of pitch location.

Table 2.2: Properties of the training and testing sets for individual pitchers.

Value	Training Set	Testing Set
Average Size	3,486	1,196
Maximum Size	8,005	2,524
Minimum Size	723	375
Pitch Types Thrown	Pitchers (Training)	Pitchers (Testing)
2	9	14
3	42	82
4	132	130
5	89	53
6	13	6
7	2	2

Despite having this information, we wanted to add more data into the feature set, and so we generated features using the full data set, not just from a certain inning or game. We found the pitcher's historical tendencies over the past 5, 10, and 20 pitches per game, his historical tendency over the entire data set, and his historical tendencies towards the specific batter he is facing, where his tendency is the percent of each pitch type he has thrown. We also added the individual batter's

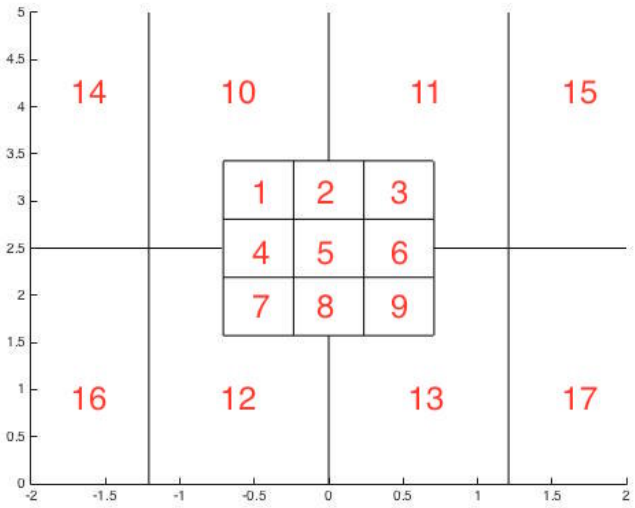


Figure 2.2: The zones used to determine the location where the pitch crosses home plate.

historical performance against each type of pitch. Because each pitcher threw a different number of unique pitch types, not all the datasets are the same size. At the most, a pitcher could have 103 features associated with each pitch and at the minimum he could have 63. The average pitcher had 81 features. The groups of features are listed in Table 2.3.

2.3.1 Expert Feature Selection

As discussed in [26], the decision of what features to include as inputs in any machine learning method will make or break the ability of the method to solve, classify, predict, or whatever else one may want to do. Early in their paper [26], Guyon and Elisseeff present a "checklist" that one should use in feature selection. If there are many features available, this selection may involve reducing the number used as inputs (discussed later in Chapter 5), or if one feels there is not enough informative features, the selection may involve the creation of new inputs, referred to as synthetic feature generation. The first item on the checklist is "Do you have domain knowledge?" recommending that if the answer is yes, then it is best to first construct a set of *ad hoc* features, which is exactly what we did.

While the author is not a major league baseball player, he has developed significant domain knowledge through extensive observation and reading of current literature and analysis to create the synthetic features previously discussed. These features were picked in an effort to mitigate the uncertainty caused by the nature of the PITCHf/x system and to gain the most benefit from the large amount of historical data available.

The PITCHf/x system is a visual radar system that can identify many important features of a pitch, giving input to the MLBAM neural network for pitch classification. While the system is very accurate, there is still a level of uncertainty that comes with the features it measures. We address the uncertainty involved in the pitch type classification by only including the pitches classified with over 80% confidence, which also helps to reduce any uncertainty in the specific statistics of each pitch. We employ features that are based on the PITCHf/x classification to use the historical data available (such as each pitcher's overall historical tendency), but we also use game situational features that are independent of any computational classification.

Using the features numbered between 20 and 103 as shown in Table 2.3 allows us to employ an extended data set that gives insight into historical performance and tendencies. Features 1 through 19, with the exception of the previous pitch type, however, are all observable features that have little to no uncertainty associated with them at all. The combination of these different types of features strikes a balance between *ad hoc* game knowledge, definite observable features, and features that give deeper insight but at the cost of higher uncertainty.

Table 2.3: Feature groups for each pitch. Numbers are given if the pitcher throws all seven pitch types. Tendency refers to the percentage of each pitch type.

Number	Feature Group	Type of Variable
1	Inning	Categorical
2	Top or Bottom	Binary
3	Outs	Categorical
4	Order Position	Categorical
5	Total At-Bat	Categorical
6	Score Spread	Categorical
7	Time of Day	Categorical
8	Batter Handedness	Binary
9	Strikes	Categorical
10	Balls	Categorical
11-13	On Base	Binary
14	Base Score	Categorical
15	Previous At-Bat Result	Categorical
16	Previous Pitch Result	Categorical
17	Previous Pitch Type	Categorical
18	Previous Pitch Location	Categorical
19	Pitch Number	Categorical
20-23	Previous Pitch Speed, Break Angle, Break Length, Break Height	Continuous
24-30	Previous 5 Pitch Tendency	Continuous
31-37	Previous 10 Pitch Tendency	Continuous
38-44	Previous 20 Pitch Tendency	Continuous
45-52	Previous 5 Pitch Strike Tendency	Continuous
53-60	Previous 10 Pitch Strike Tendency	Continuous
61-68	Previous 20 Pitch Strike Tendency	Continuous
69-75	Pitcher Historical Tendency	Continuous
76-82	Pitcher Tendency vs. Batter	Continuous
83-89	Batter Strike Tendency	Continuous
90-96	Batter In-Play Tendency	Continuous
97-103	Batter Ball Tendency	Continuous

2.4 Contributions

Prior to this work, feature selection was discussed in [7, 23, 53]. All three contain relatively similar basic game situation information and all three include some sort of historical data analysis, but each differ in the specific types of history they examine. In [7], the pitcher's specific movements and physical behavior associated with each pitch is used and in [53] and [23], the only historical data that is included is focused solely on the batter at the plate for each pitch. Each of them look at a short in-game window prior to the current pitch, but at most go four pitches back.

In our work, we also use similar game situation features, but also include somewhat intangible ones such as the time of day, e.g., if a pitcher wants to use the setting sun or the night lights in an effort to disguise a pitch. Our feature selection examines in-game windows of varying length, not only looking at the type of each pitch but whether or not the pitches are strikes or not, giving insight into whether a pitcher may feel that he is on a hot or cold streak with a particular pitch. Finally, the inclusion of overall historical tendencies and successes of pitcher and batter is an effort to replicate the information that both players may have about the other as the pitch is thrown. The combination of observational and synthetic features gives deeper insight into the game situation and increases the predictive accuracy.

CHAPTER

3

METHODS

As noted in the title of [40], machine learning (ML) and its use in artificial intelligence (AI) has gone through an explosive period of growth in the last few years. Machine learning methods are used in many every day products and services that we use, from online product ordering giants like Amazon to entertainment providers like Netflix, as well as in Google searches and Apple's Siri, a voice-activated assistant [50], and even medical diagnoses have been improved using machine learning techniques [27]. Machine learning has begun to supplement standard statistical analysis in many sports, being used to predict, not just analyze.

Machine learning methods are divided into two main different categories based on the availability of information about the data being worked with. Methods that deal with the problem of classifying or clustering similar groups of unlabeled data together are called unsupervised learning methods, while methods that use data with an associated label are supervised learning methods. Supervised learning requires *a priori* information about the data, what class or label or value match up with it. Supervised learning methods can be either classification methods or regression methods. Classifiers output a discrete class assignment, while regression methods output a continuous value along some interval. Supervised learning requires at the minimum a split in the data between training and testing sets [47].

Unsupervised learning is a much more complex problem, in which unlabeled feature vectors are

processed in an effort to find similarities between vectors and group them together. This is referred to as clustering, where the goal is to accurately categorize the features given. Semi-supervised learning, as the name would suggest, is a middle ground between the two, where some unlabeled data can be given as part of the training set, clustering algorithms can be used to help group unclassified observations in the training data in order to provide more information for the function applied to the unlabeled testing set [47].

In this work, we examine three different supervised learning methods, specifically classification methods: Linear Discriminant Analysis (LDA), Support Vector Machines (SVM), and Classification Trees (CT).

3.1 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is an extension of Fisher's linear discriminant. R.A. Fisher wrote a paper in 1936 [21], detailing a new way to find the separation between two distinct classes of observations, y_0 and y_1, given a feature set $\hat{X}$.

3.1.1 Fisher's Linear Discriminant

To find Fisher's linear discriminant, we first assume that each class y_0 and y_1 have respective mean-covariance pairs $(\vec{\mu}_0, \Sigma_0)$ and $(\vec{\mu}_1, \Sigma_1)$. In order to find the separation between the two classes, we have to find a projection hyperplane $\vec{w}$, defining the linear combinations $\vec{w} \cdot \vec{x}$ with mean-covariance pairs $(\vec{w} \cdot \vec{\mu}_i, \vec{w}^T \Sigma_i \vec{w})$ for $i = 0, 1$. Fisher defined the separation between the classes with the formula

$$S = \frac{\sigma^2_{\text{between}}}{\sigma^2_{\text{within}}} = \frac{(\vec{w} \cdot \vec{\mu}_1 - \vec{w} \cdot \vec{\mu}_0)^2}{\vec{w}^T \Sigma_1 \vec{w} + \vec{w}^T \Sigma_0 \vec{w}} = \frac{(\vec{w} \cdot (\vec{\mu}_1 - \vec{\mu}_0))^2}{\vec{w}^T (\Sigma_0 + \Sigma_1) \vec{w}}. \tag{3.1}$$

We define the between class scatter matrix as $S_B = (\vec{\mu}_1 - \vec{\mu}_0)(\vec{\mu}_1 - \vec{\mu}_0)^T$, and the within class scatter matrix as $S_W = \Sigma_0 + \Sigma_1$, so we can rewrite the objective function as

$$S = \frac{\vec{w} S_B \vec{w}^T}{\vec{w}^T S_W \vec{w}}, \tag{3.2}$$

and we can minimize S by taking the derivative with respect to $\vec{w}$ and setting it equal to zero

$$\frac{\mathrm{d}S}{\mathrm{d}\vec{w}} = \frac{(2S_B \vec{w}) \vec{w}^T S_W \vec{w} - (2S_W \vec{w}) \vec{w}^T S_B \vec{w}}{(\vec{w}^T S_W \vec{w})^2} = \vec{0}. \tag{3.3}$$

To find the minimum, we solve

$$\begin{aligned}(S_B\,\vec{w})\vec{w}^{\,T}S_W\,\vec{w}-(S_W\,\vec{w})\vec{w}^{\,T}S_B\,\vec{w}&=\vec{0}\\ \frac{(S_B\,\vec{w})\vec{w}^{\,T}S_W\,\vec{w}}{\vec{w}^{\,T}S_W\,\vec{w}}-\frac{(S_W\,\vec{w})\vec{w}^{\,T}S_B\,\vec{w}}{\vec{w}^{\,T}S_W\,\vec{w}}&=\vec{0}\\ S_B\,\vec{w}-\frac{(S_W\,\vec{w})\vec{w}^{\,T}S_B\,\vec{w}}{\vec{w}^{\,T}S_W\,\vec{w}}&=\vec{0}.\end{aligned}$$

So for $\lambda=\frac{\vec{w}^{\,T}S_B\,\vec{w}}{\vec{w}^{\,T}S_W\,\vec{w}}$, we find the generalized eigenvalue problem

$$S_B\,\vec{w}=\lambda S_W\,\vec{w}, \tag{3.4}$$

which can be rewritten as the standard eigenvalue problem

$$S_W^{-1}S_B\,\vec{w}=\lambda\,\vec{w}. \tag{3.5}$$

For any vector $\vec{x}$, we find that $S_B\,\vec{x}$ points in the same direction as $\vec{\mu}_1-\vec{\mu}_0$, then

$$S_B\,\vec{x}=(\vec{\mu}_1-\vec{\mu}_0)(\vec{\mu}_1-\vec{\mu}_0)^T\,\vec{x}=\alpha(\vec{\mu}_1-\vec{\mu}_0),$$

where $\alpha=(\vec{\mu}_1-\vec{\mu}_0)^T\,\vec{x}$. The solution to the eigenvalue problem is $\vec{w}=\alpha S_W^{-1}(\vec{\mu}_1-\vec{\mu}_0)$ [49], and so the maximum separation is found by

$$\vec{w}\propto(\Sigma_0+\Sigma_1)^{-1}(\vec{\mu}_1-\vec{\mu}_0), \tag{3.6}$$

which is the optimal projection hyperplane for the separation of the two classes.

3.1.2 Binary Classificiation

Linear discriminant analysis is an extension of Fisher's linear discriminant, that relies on the assumption that both covariance matrices are the same, i.e., $\Sigma=\Sigma_0=\Sigma_1$. Using this property in equation (3.6) gives $\vec{w}\propto\Sigma^{-1}(\vec{\mu}_1-\vec{\mu}_0)$, and the assumption that the best separation would be the hyperplane between the projections of the two means, $\vec{w}\cdot\vec{\mu}_0$ and $\vec{w}\cdot\vec{\mu}_1$, leads to the inequality

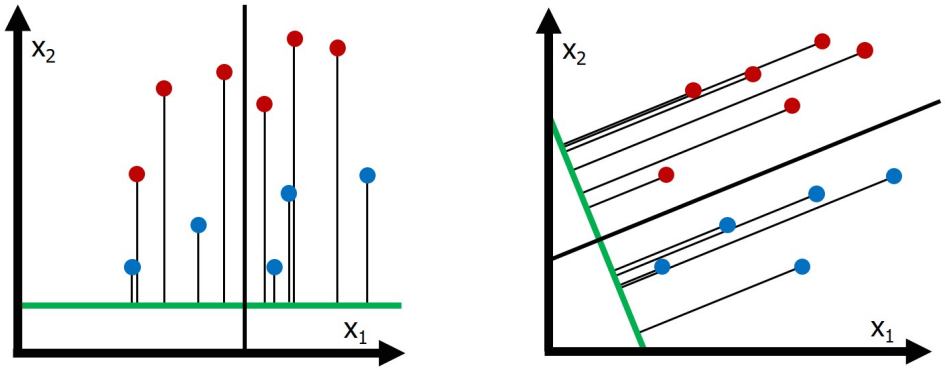
$$\vec{w}\cdot\vec{x}>c, \tag{3.7}$$

where

$$\vec{w} = \Sigma^{-1}(\vec{\mu}_1 - \vec{\mu}_0), \tag{3.8}$$

$$c = \frac{1}{2}\left(\vec{\mu}_1^T \Sigma^{-1} \vec{\mu}_1 - \vec{\mu}_0^T \Sigma^{-1} \vec{\mu}_0\right). \tag{3.9}$$

The decision of which class an unknown observation $\vec{x}$ belongs in depends on whether or not the inequality in (3.7) is satisfied or not, i.e., which side of the separation c the projection falls on. Figure 3.1 shows a good and bad example of finding the projection hyperplane $\vec{w}$ for a binary classification.



(a) (b)

Figure 3.1: Example of a bad (a) and good (b) hyperplanes found for LDA.

The classification of an unknown observation can also be written as

$$\hat{y} = \arg\min_{y=0,1} \sum_{k=0}^{1} \hat{P}(k|\vec{x}) C(y|k), \tag{3.10}$$

where $\hat{y}$ is the predicted class, $\hat{P}(k|\vec{x})$ is the posterior probability of class k for observation $\vec{x}$, and $C(y|k)$ is the cost of misclassifying an observation [35].

LDA employs a regularization method with two parameters, γ and δ. γ is used to combat bias in the eigenvalues of the covariance matrix Σ [46]. Let X be the data matrix and $\hat{X}$ be the centered data

($\hat{X} = X - \mu$ where μ is the mean of the data), then we define

$$D = \text{diag}(\hat{X}^T \hat{X}). \tag{3.11}$$

We use γ to define the regularized covariance matrix $\hat{\Sigma}$ as

$$\hat{\Sigma} = (1-\gamma)\Sigma + \gamma D. \tag{3.12}$$

Letting $\vec{\mu}_k$ be the mean vector for the observations in class $k = 0, 1$ and C be the correlation matrix of X, then we define the regularized correlation matrix $\hat{C}$ as

$$\hat{C} = (1-\gamma)C + \gamma I, \tag{3.13}$$

where I is the identity matrix. With D, $\hat{\Sigma}$, and $\hat{C}$ defined, we rewrite the linear term found in equation (3.7) as

$$(\vec{x} - \vec{\mu})^T \hat{\Sigma}^{-1}(\mu_k - \vec{\mu}) = \left[(\vec{x} - \vec{\mu})^T D^{-1/2}\right]\left[\hat{C}^{-1} D^{-1/2}(\vec{\mu}_k - \vec{\mu})\right]. \tag{3.14}$$

δ is used as a threshold dependent on the second term in square brackets. The Delta Predictor value of each feature (discussed later in Chapter 5) is determined by $\left[\hat{C}^{-1} D^{-1/2}(\mu_k - \mu)\right]$, and so δ is used to eliminate features with a Delta Predictor for each class that is less than the parameter, i.e.,

$$\left[\hat{C}^{-1} D^{-1/2}(\vec{\mu}_k - \vec{\mu})\right] \leq \delta \tag{3.15}$$

for all classes k [35].

3.1.3 Multi-Class Classification

To expand LDA classification from two classes to a multi-class problem, we consider N total classes, each with a unique mean μ_i, but still with the same covariance Σ. Letting $\vec{\mu}$ be the mean of the means, we find the between class covariance by

$$\Sigma_b = \frac{1}{N} \sum_{i=1}^{N} (\vec{\mu}_i - \vec{\mu})(\vec{\mu}_i - \vec{\mu})^T, \tag{3.16}$$

and class separation is defined as

$$S = \frac{\vec{w}^T \Sigma_b \vec{w}}{\vec{w}^T \Sigma \vec{w}}. \tag{3.17}$$

We employ a "one-against-one" method of classification, which will be explained further in section 3.2.4.

3.2 Support Vector Machines

Support Vector Machines (SVMs) are a linear classification tool designed for optimizing prediction accuracy while avoiding overfitting on training data [47].

3.2.1 Linearly Separable Case

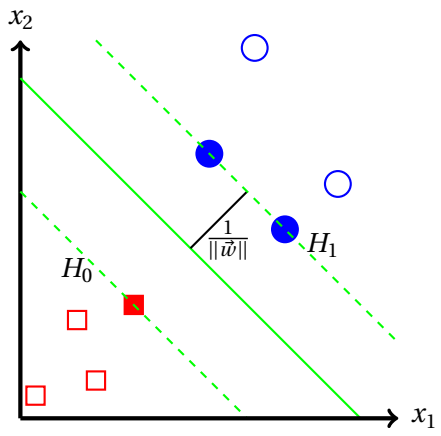


Figure 3.2: An example of a linearly separable binary problem with SVM.

SVMs were originally designed, like LDA, for binary classification and prediction. Any unknown observation $\vec{x}$ belongs to one of two classes, y_0 or y_1, and so the SVM method finds a separation hyperplane

$$g(x) = \vec{w} \cdot \vec{x} + b$$

between the two classes. Because this separation hyperplane is not necessarily unique, the goal is to find the optimal hyperplane that maximizes the distance between the two classes. The decision boundaries run through training data points on either side (the *support vectors*) of the respective classes. Given a feature vector $\vec{x} \in X \in \mathbb{R}^{M \times N}$ and some weight vector $\vec{w}$ and a separating hyperplane

$H = \vec{w} \cdot \vec{x} + b$, classification is determined by the decision hyperplane

$$g(x) = \begin{cases} \vec{w} \cdot \vec{x} + b \geq 1 & \text{if } \vec{x} \in y_1 \\ \vec{w} \cdot \vec{x} + b \leq -1 & \text{if } \vec{x} \in y_0 \end{cases} \tag{3.18}$$

which, using the class indicator values $y_1 = 1$ and $y_0 = -1$, can also be written in combination as

$$y_i(\vec{w} \cdot \vec{x} + b) \geq 1 \text{ for } i = 0, 1. \tag{3.19}$$

We define the hyperplanes H_0 and H_1

$$H_0 : \ \vec{x} \cdot \vec{w} + b = -1, \tag{3.20}$$

$$H_1 : \ \vec{x} \cdot \vec{w} + b = +1 \tag{3.21}$$

such that any points that lie along H_0 or H_1 are the defined support vectors. Since the distance from origin to H_0 is $\frac{|-1-b|}{||\vec{w}||}$ and from the origin to H_1 is $\frac{|1-b|}{||\vec{w}||}$, the distance between the two that we are trying to maximize is $\frac{2}{||\vec{w}||}$. Thus, in order to maximize the distance, we have to minimize $||\vec{w}||$. Combining that with (3.18), the optimization problem is

$$\min \frac{1}{2}||\vec{w}||^2, \tag{3.22}$$

$$\text{subject to } \ y_i(\vec{w} \cdot \vec{x} + b) \geq 1, \text{for } i = 1, \ldots, N. \tag{3.23}$$

Because $||\vec{w}||$ has a square root, it is much simpler to optimize $\frac{1}{2}||\vec{w}||^2$, which becomes a quadratic problem that can be solved using Lagrange multipliers [24]. The form of the Lagrangian is

$$L = \frac{1}{2}||\vec{w}||^2 - \sum_{i=0}^{N} \alpha_i [y_i(\vec{w} \cdot \vec{x}_i + b) - 1], \tag{3.24}$$

where α_i denotes the Langrange multipliers. Minimizing L requires four Karush-Kuhn-Tucker (KKT) conditions [47] to be satisfied

$$L_{\vec{w}} = 0, \tag{3.25}$$

$$L_b = 0, \tag{3.26}$$

$$\alpha_i \geq 0, \ \ i = 1, \ldots, N, \tag{3.27}$$

$$\alpha_i [y_i(\vec{w} \cdot \vec{x}_i + b) - 1] = 0, \ \ i = 1, \ldots, N. \tag{3.28}$$

In combination with (3.24), we find

$$\vec{w} = \sum_{i=1}^{N} \alpha_i y_i x_i, \tag{3.29}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0. \tag{3.30}$$

The Langrangian duality can be written in the Wolfe dual representation form, that is

$$\max \ \frac{1}{2}||\vec{w}||^2 - \sum_{i=0}^{N} \alpha_i [y_i(\vec{w} \cdot \vec{x}_i + b) - 1], \tag{3.31}$$

$$\text{subject to } \ \vec{w} = \sum_{i=1}^{N} \alpha_i y_i x_i, \tag{3.32}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0, \tag{3.33}$$

$$\alpha_i \geq 0, \tag{3.34}$$

which is equivalent to

$$\max \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j, \tag{3.35}$$

$$\text{subject to } \sum_{i=1}^{N} \alpha_i y_i = 0, \tag{3.36}$$

$$\alpha_i \geq 0. \tag{3.37}$$

When $\alpha_i > 0$, then $\vec{x}_i$ is a support vector and the optimal separation hyperplane is

$$\vec{w} = \sum_{i=1}^{n} \alpha_i y_i \vec{x}_i, \tag{3.38}$$

where $n \leq N$ is the total number of support vectors. To find b, from (3.28), for any nonzero Lagrange

multiplier α_i with corresponding support vector $\vec{x}_i$ and label y_i, we find

$$\alpha_i[y_i(\vec{w}\cdot\vec{x}_i+b)-1]=0 \tag{3.39}$$

$$y_i(\vec{w}\cdot\vec{x}_i+b)-1=0 \tag{3.40}$$

$$y_i\vec{x}_i\cdot\vec{w}+y_ib=1 \tag{3.41}$$

$$b=\frac{1}{y_i}(1-y_i\vec{x}_i\cdot\vec{w}) \tag{3.42}$$

$$b=y_i-\vec{x}_i\vec{w} \tag{3.43}$$

$$b=y_i-\sum_{j=1}^{N}\alpha_jy_j\vec{x}_j\cdot\vec{x}_i. \tag{3.44}$$

Having found $\vec{w}$ and b, the classification function is

$$g(w)=\text{sign}(\vec{w}\cdot\vec{x}+b) \tag{3.45}$$

$$=\text{sign}\left(\sum_{i=1}^{n}\alpha_iy_i\vec{x}_i\cdot\vec{x}+b\right). \tag{3.46}$$

3.2.2 Nonseparable Case

While SVMs work very well for data that is linearly separable, the reality is that most data is not so easily separated. When a linear hyperplane cannot be drawn between the two classes, it is the nonseparable case, and requires a reformulation of the problem. In this instance, we can still use SVMs to classify unknown points, but need to account for any error. To do so, we know that H_0 and H_1 have the forms

$$\vec{w}\cdot\vec{x}_i+b=\pm1, \tag{3.47}$$

and the distance between them is the margin of size $\frac{2}{\|\vec{w}\|}$. As shown in Figure 3.3, any training data $\vec{x}_i$ with associated class label y_i belongs to one of three cases:

- $\vec{x}_i$ is outside the margin, correctly classified, and therefore satisfies the inequality $y_i(\vec{w}\cdot\vec{x}_i+b)\geq1$.
- $\vec{x}_i$ is correctly classified but within the margin, satisfying the compound inequality $0\leq y_i(\vec{w}\cdot\vec{x}_i+b)<1$.
- $\vec{x}_i$ is incorrectly classified, satisfying the inequality $y_i(\vec{w}\cdot\vec{x}_i+b)<0$.

To be able to handle all three cases, we introduce a slack variable ξ_i in the constraint:

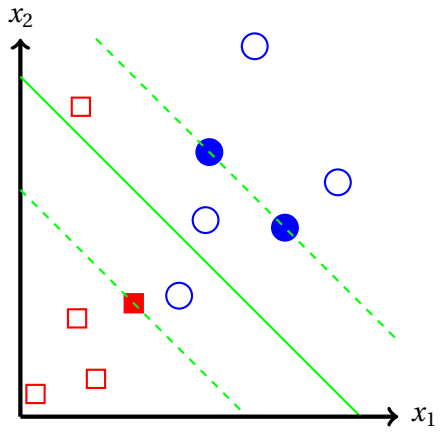


Figure 3.3: An example of a nonseparable binary problem with SVM.

$$y_i(\vec{w}\cdot\vec{x}_i+b)\geq 1-\xi_i, \tag{3.48}$$

where the first case has $\xi_i = 0$, the second case has $0 < \xi_i \leq 1$, and the third case has $\xi_i > 1$. The optimization problem now becomes

$$\min\frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{N}\xi_i, \tag{3.49}$$

$$\text{subject to } \; y_i(\vec{w}\cdot\vec{x}+b)\geq 1-\xi_i \text{ for } i=1,\ldots,N, \tag{3.50}$$

$$\xi_i \geq 0 \text{ for } i=1,\ldots,N, \tag{3.51}$$

where C is an error parameter that determines how much emphasis is placed on maximizing the margin of the hyperplane versus reducing the number of misclassified data points. The Langrangian method still works for this optimization. The new form is

$$L=\frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\mu_i\xi_i - \sum_{i=0}^{N}\alpha_i[y_i(\vec{w}\cdot\vec{x}_i+b)-1+\xi_i], \tag{3.52}$$

with the respective KKT conditions

$$L_{\vec{w}} = 0 \text{ or } \vec{w} = \sum_{i=1}^{N} \alpha_i y_i \vec{x}_i, \tag{3.53}$$

$$L_b = 0 \text{ or } \sum_{i=1}^{N} \alpha_i y_i = 0, \tag{3.54}$$

$$L_{\xi_i} = 0 \text{ or } C - \mu_i - \alpha_i = 0, \ \ i = 1, \ldots, N, \tag{3.55}$$

$$\alpha_i [y_i(\vec{w} \cdot \vec{x}_i + b) - 1 + \xi_i] = 0, \ \ i = 1, \ldots, N, \tag{3.56}$$

$$\alpha_i \geq 0, \ \ i = 1, \ldots, N, \tag{3.57}$$

$$\mu_i \geq 0, \ \ i = 1, \ldots, N, \tag{3.58}$$

as well as the corresponding Wolfe dual representation

$$\max \ \frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \mu_i \xi_i - \sum_{i=0}^{N} \alpha_i [y_i(\vec{w} \cdot \vec{x}_i + b) - 1], \tag{3.59}$$

$$\text{subject to } \ \vec{w} = \sum_{i=1}^{N} \alpha_i y_i x_i, \tag{3.60}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0, \tag{3.61}$$

$$C - \mu_i - \alpha_i = 0, \ \ i = 1, \ldots, N, \tag{3.62}$$

$$\alpha_i \geq 0, \, \mu_i \geq 0, \ \ i = 1, \ldots, N. \tag{3.63}$$

Using the equality constraints substituted into the Lagrangian (3.52), we find the maximization with new conditions

$$\max \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j, \tag{3.64}$$

$$\text{subject to } \ \sum_{i=1}^{N} \alpha_i y_i = 0, \tag{3.65}$$

$$0 \leq \alpha_i \leq C. \tag{3.66}$$

As in the separable case, we use the KKT conditions to solve for b. Combining the two equality statements $C - \mu_i - \alpha_i = 0$ and $\mu_i \xi_i = 0$, we find $\xi_i = 0$ if $\alpha_i = C$. Thus, using any value satisfying $0 < \alpha_i < C$ and the fact that $\xi_i = 0$, we use $\alpha_i [y_i(\vec{w} \cdot \vec{x}_i + b) - 1 + \xi_i]$ to compute b. We find that the

solution for b is the same as the separable case, and the decision function

$$g(w) = \text{sign}(\vec{w} \cdot \vec{x} + b), \tag{3.67}$$

$$\text{where } b = y_i - \sum_{j=1}^{N} \alpha_j y_j (\vec{x}_j \cdot \vec{x}_i), \tag{3.68}$$

is also the same as in the separable case, with the only additional constraint being that the Langrange multipliers α_i are bounded above by C [47].

3.2.3 Kernel Function

In an effort to prevent the need for any error in the classification, we can try to make the data linearly separable to use with SVM. This technique depends on using the kernel "trick," a method using a function, denoted $K(x_i, x_j)$, to map the original input data into a higher dimensional space, ideally taking it from the nonseparable case (and thus requiring the slack error variable discussed previously) to a linearly separable case [11, 27]. There are a variety of functions that can be used in this mapping, an example of which is shown in Figure 3.4, but a few are the most commonly used:

$$\text{Linear: } K(\vec{x}_i, \vec{x}_j) = \vec{x}_i \cdot \vec{x}_j, \tag{3.69}$$

$$\text{Polynomial: } K(\vec{x}_i, \vec{x}_j) = (\gamma \vec{x}_i \cdot \vec{x}_j + b)^p, \tag{3.70}$$

$$\text{Sigmoid: } K(\vec{x}_i, \vec{x}_j) = \tanh(\gamma \vec{x}_i \cdot \vec{x}_j + b), \tag{3.71}$$

$$\text{Gaussian: } K(\vec{x}_i, \vec{x}_j) = \exp(-\gamma ||\vec{x}_i - \vec{x}_j||^2). \tag{3.72}$$

Because all of the training data is used in the optimization functions as inner products $\vec{x}_i \cdot \vec{x}_j$, then these functions $K(\vec{x}_i, \vec{x}_j)$ can be written as the dot product of the functions $\Phi(x_i) \cdot \Phi(x_j)$ that maps data from the original input dimension to some higher dimensional space, i.e.,

$$\Phi: \mathbb{R}^m \rightarrow \mathbb{R}^n, \tag{3.73}$$

where $n > m$. Replacing every inner product in the optimization with $K(\vec{x}_i, \vec{x}_j)$, then the differentiation happens in the higher space, where the data is linearly separable. From prior literature [11], we find the the Gaussian kernel (also known as the radial basis function, or RBF) works the best. Because we need to be able to use $K(\vec{x}_i, \vec{x}_j)$ in any dimension, we can show that the RBF is infinitely

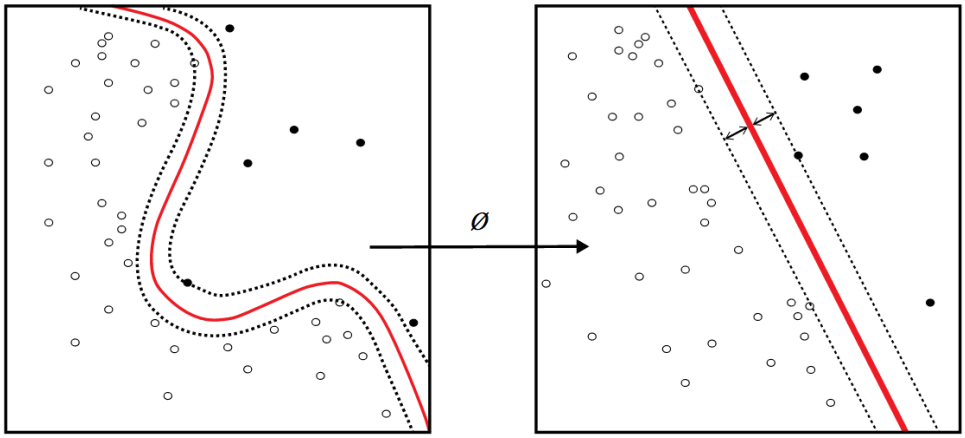


Figure 3.4: Example of a kernel function mapping.

dimensional [27]. Without loss of generality, we assume $\gamma > 0$ and $\vec{x} \in \mathbb{R}^m$, then

$$K(\vec{x}_i, \vec{x}_j) = \exp(-\gamma ||\vec{x}_i - \vec{x}_j||^2) \tag{3.74}$$

$$= \exp(-\gamma(\vec{x}_i - \vec{x}_j)^2) \tag{3.75}$$

$$= \exp(-\gamma \vec{x}_i^2 + 2\gamma \vec{x}_i \cdot \vec{x}_j - \gamma \vec{x}_j^2) \tag{3.76}$$

$$= \exp(-\gamma \vec{x}_i^2)\exp(\gamma \vec{x}_j^2)\exp(2\gamma \vec{x}_i \vec{x}_j) \tag{3.77}$$

$$= \exp(-\gamma \vec{x}_i^2)\exp(\gamma \vec{x}_j^2)\left(1 + \sqrt{\frac{2\gamma}{1!}}\vec{x}_i \sqrt{\frac{2\gamma}{1!}}\vec{x}_j + \sqrt{\frac{4\gamma^2}{2!}}\vec{x}_i \sqrt{\frac{4\gamma^2}{2!}}\vec{x}_j + \cdots\right) \tag{3.78}$$

$$= \Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j), \tag{3.79}$$

where

$$\Phi(x) = \exp\left(1, \sqrt{\frac{2\gamma}{1!}}x, \sqrt{\frac{4\gamma^2}{2!}}x^2, \ldots\right). \tag{3.80}$$

To apply the kernel trick to a nonseparable SVM [27], we replace every inner product in the

training algorithm with $K(\vec{x}_i, \vec{x}_j)$, so the maximization becomes

$$\max \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j K(\vec{x}_i, \vec{x}_j), \tag{3.81}$$

$$\text{subject to } \sum_{i=1}^{N} \alpha_i y_i = 0, \tag{3.82}$$

$$0 \leq \alpha_i \leq C, \tag{3.83}$$

and the final decision solution is

$$g(w) = \text{sign}\left(\sum_{i=1}^{N} \alpha_i y_i K(\vec{x}_i, \vec{x}_j) + b \right), \tag{3.84}$$

$$\text{where } \; b = y_i - \sum_{j=1}^{N} \alpha_j y_j K(\vec{x}_i, \vec{x}_j). \tag{3.85}$$

3.2.4 Multi-Class SVM

The extension of binary SVMs to a multi-class method has led to two common approaches to multi-class classification. For a problem with N distinct classes, the one-vs-all (OVA) method creates N SVMs, and places the unknown value in whatever class has the largest decision function value.

For our model, we used the one-vs-one (OVO) method, which creates $N(N-1)/2$ SVMs for classification. For a set of p training data, $(x_1, y_1), \ldots, (x_p, y_p)$ where $x_l \in \mathbb{R}^n, l = 1, \ldots, p$ and $y_l \in \{1, \ldots, N\}$ is the class label for x_l, then the SVM trained on the i^{th} and j^{th} classes solves

$$\min_{w^{ij}, b^{ij}, \xi^{ij}} \frac{1}{2} (w^{ij})^T w^{ij} + C \sum_{\ell} \xi_{\ell}^{ij}, \tag{3.86}$$

$$\text{where } (w^{ij})^T \phi(x_\ell) + b^{ij} \geq 1 - \xi_{\ell}^{ij} \text{ if } y_\ell = i, \tag{3.87}$$

$$\text{or } (w^{ij})^T \phi(x_\ell) + b^{ij} \leq -1 + \xi_{\ell}^{ij} \text{ if } y_\ell = j, \tag{3.88}$$

$$\xi_{\ell}^{ij} \geq 0, \tag{3.89}$$

where $\Phi(x)$ is the radial basis function used to map the training data $x_{i,j}$ to a higher dimensional space and C is the penalty parameter [29]. After all comparisons have been done, the unknown value is classified according to whichever class has the most votes from the assembled SVMs.

Figure 3.5 shows a basic representation of the differences between the OVA and OVO methods. Computationally, both methods are fairly equivalent, and so since the middle triangle formed by

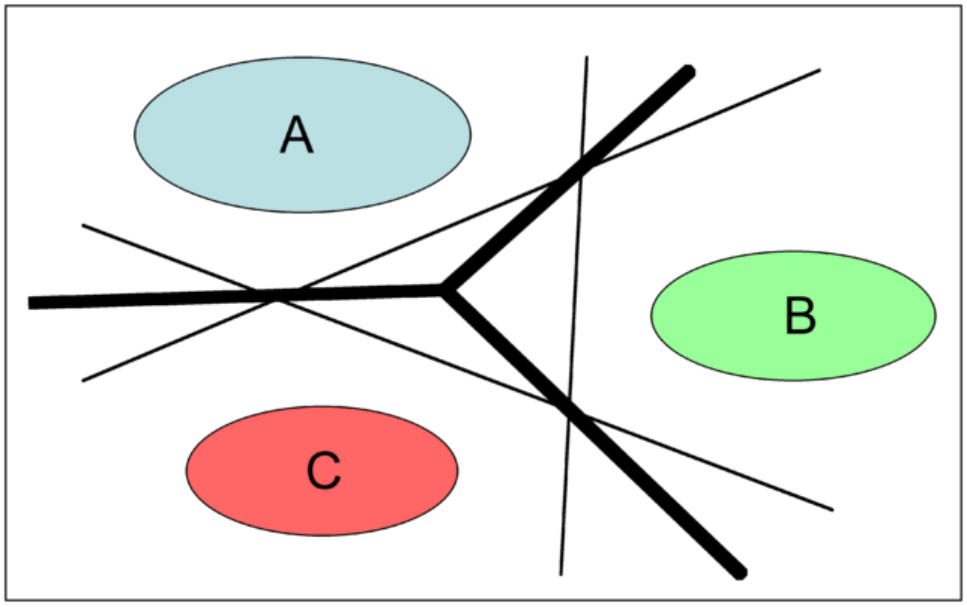


Figure 3.5: A visual representation of the one-vs-all method (thin lines) compared to the one-vs-one method (thick line) from [2].

the OVA method leaves a gap where the classification algorithm can fail to place an unknown value, but since the OVO method does not have any blind spots, we used it for our classification.

3.3 Classification Trees

While linear discriminant analysis and support vector machines are geometrically-based methods, decision trees do not rely on optimizing the distance between classes or the projections of classes. Introduced by Breiman et al., classification and regression trees (CART) are binary decision trees built using both continuous and categorical data [9]. Both classification and regression trees are constructed (or grown) the same way, examining all possible splits of all possible features, and then making a binary decision, yes or no, if the condition is satisfied or not. Figure 3.6 shows a basic example of what a single decision tree may look like for pitch prediction.

Decision trees can be thought of as analogous to the way a human being makes decisions, weighing the most important variable first, then making subsequent decisions on other features of the data [43]. These decisions, since they split the data at some threshold, can be thought of as geometric partitions, but are not necessarily visualized geometrically as easily as SVMs and LDA can be. Instead of attempting to minimize distance or margin error, trees attempt to minimize what

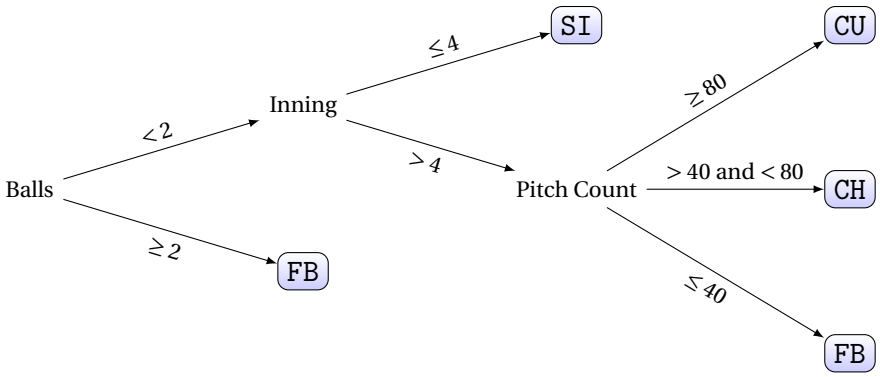


Figure 3.6: Basic Classification Tree for Pitch Selection.

is known as total impurity.

To build a classification tree, all features are considered at first. The algorithm examines each feature and finds which one, when split, will reduce the impurity of the tree nodes the most. Once the feature is selected, then the threshold that reduces impurity the most is found, and the process is repeated until either some impurity threshold is determined or all the end nodes (leaves) are pure, i.e., they only contain one class of data. Impurity can be found by different measures, but the one we employ is the measurement used in the MATLAB implementation of classification trees, the Gini diversity index (gdi), which can be represented as

$$I = 1 - \sum_{i=1}^{N} p^2(i), \tag{3.90}$$

where $p(i)$ is the fraction of each class $i = 1, \ldots, N$ as a total part of the number of observations at that node. Using the gdi to determine the impurity is the same idea as judging the total accuracy of the tree by randomly selecting an answer from the distribution of each class at the end leaf node [36]. During the training process, the impurity threshold is generally set to be small enough so that all leaf nodes are pure. An example of two different splits with two different levels of impurity are shown in Figure 3.7.

Trees can be pruned. If a validation step is included in the training process, then for any parent node of two separate leaves, if the validation error is the same whether the leaves are included or if the parent node itself is used as a leaf, then the excess leaves are discarded and the parent node is used as the final leaf [43].

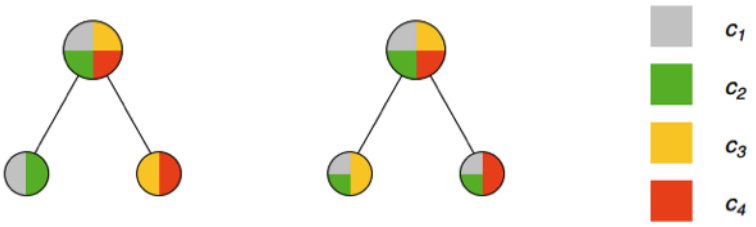


Figure 3.7: Two sample splits for a decision tree with four unique classes. The split on the left is better, as total impurity is lower due to having higher proportions of fewer classes present in each leaf, minimizing the gdi. Taken from [30].

3.3.1 Random Forests

Introduced by Brieman in 2001, random forests are used to reduce error by grouping together large numbers of classification trees. Random forests are an extension of the ensemble approach for decision making, which involves growing multiple trees on the full data set and letting a majority vote determine the class (we will discuss this in more depth later in the chapter). Random forests differ in that each tree is grown on a random subset of the training data without replacement. The use of these random subsets is known as bootstrap aggregation (or bagging). Parameter selection may be done randomly as well, selecting from some number K of the best features and the best splits to grow the tree [8].

For some testing set data $\mathbf{X}$ with corresponding labels $\mathbf{Y}$, we denote the ensemble set of random forest classifiers $h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_K(\mathbf{x})$, each trained with random selections from $\mathbf{X}$. Breiman defines the margin function

$$mg(X,Y) = av_k I(h_k(\mathbf{X}) = Y) - \max_{j \neq Y} av_k I(h_k(\mathbf{X}) = j), \tag{3.91}$$

where $I(\cdot)$ is the indicator function. This margin function is a measure of how much the number of average votes for the correct class Y is greater than the number of average votes for any other (incorrect) class. The greater the margin function, the greater the confidence in classification there is. Breiman defines the generalization error as

$$PE^* = P_{\mathbf{X},Y}(mg(\mathbf{X},Y) < 0), \tag{3.92}$$

where the $\mathbf{X}$, Y subscripts mean that the probability is over the $\mathbf{X}$, Y space [8]. In random forests, the classifiers $h_k(\mathbf{X}) = h(\mathbf{X}, \Theta_k)$ where Θ_k is the random sequence subset of $\mathbf{X}$ used to train the classifier.

Given a large amount of trees, then from the Strong Law of Large Numbers and the structure of the trees themselves, we find:

Theorem 3.3.1. *As the number of trees N increases, for almost surely all sequences $\Theta_1,\ldots,\Theta_N$, then PE* converges to*

$$P_{\mathbf{X},Y}(P_\Theta(h(\mathbf{X},\Theta)=Y)-\max_{j\neq Y}P_\Theta(h(\mathbf{X},\Theta)=j)<0). \tag{3.93}$$

The proof is shown in [8]. This theorem shows why the random forest approach does not overfit as more and more trees are added, but instead limit the generalization error. The resistance to overfitting is a large strength of the random forest formulation, as opposed to a single decision tree, which can be overfit as more and more training data is added and the tree grows larger and larger.

3.4 DIRECT Algorithm

To save time and skip performing a brute force grid search to determine optimal parameters for the SVM and LDA methods, we used DIRECT. The DIRECT algorithm was developed in an effort to combat issues faced by Lipschitzian optimization: a nontrivial generalization to dimensions of $N>1$ and the need for an estimate of the Lipschitz constant. DIRECT is easily extended into multiple dimensions, and requires no initial parameter estimate, just an error measurement function f.

The algorithm begins by transforming the domain of the optimization problem to the unit hypercube

$$\Omega=\{x\in\mathbb{R}^N : 0\leq x\leq 1\},$$

where N is the number of parameters being estimated. DIRECT marks the center of this space c_1 and find $f(c_1)$, the error function value at the center. Next, it divides the hypercube into potentially optimal hyper-rectangles (leading to the name, DIviding RECTangles) by evaluating the error function at the points $c_1\pm\delta e_i$ for $i=1,\ldots,N$, where δ is one-third the length of the side of the hypercube and e_i is the ith unit vector. The algorithm initializes by choosing to leave the best function values in the largest space, defining

$$w_i=\min\big(f(c_1+\delta e_i),f(c_1-\delta e_i)\big), \tag{3.94}$$

for $1\leq i\leq N$. DIRECT then divides the dimension where w_i is smallest into thirds, leading to $c_1\pm\delta e_i$ as the new centers of the new hyper-rectangles, repeating the split for each subsequently larger w_i to find an initialization, an example which is shown in Figure 3.8.

After testing the center c_j of each hyper-rectangle j with dimension measure d_j, those which

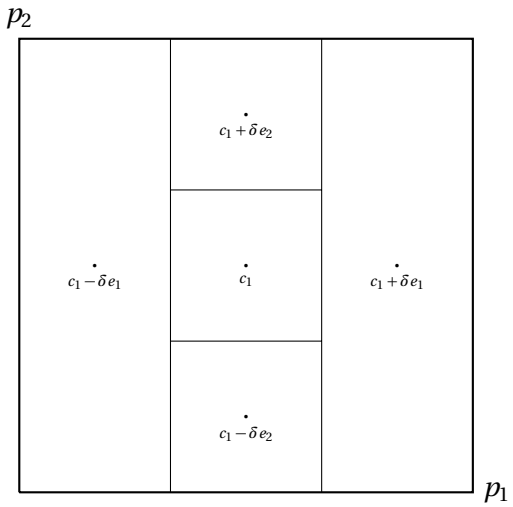


Figure 3.8: An example initialization of the DIRECT method for two parameter optimization.

minimize the function are identified as potentially optimal, which are then split and tested themselves. These potentially optimal rectangles may be found using a few observations defined in [20]:

- If hyper-rectangle i is potentially optimal, then $f(c_i) \leq f(c_j)$ for all hyper-rectangles that are of the same size as i, i.e., $d_i = d_j$.
- If $d_i \geq d_k$ for all k hyper-rectangles, and $f(c_i) \leq f(c_j)$ for all hyper-rectangles such that $d_i = d_j$, then hyper-rectangle i is potentially optimal.
- If $d_i \leq d_k$ for all k hyper-rectangles, and i is potentially optimal, then $f(c_i) = f_{\min}$.

Once a potentially optimal hyper-rectangle is identified with center c_i, DIRECT continues to divide it into small hyper-rectangles along the j^{th} dimension, evaluating at the points $c_i \pm \delta_i e_j$, using a determination similar to the initialization, i.e.,

$$w_j = \min\big(f(c_i + \delta_i e_j), f(c_i - \delta_i e_j)\big), \tag{3.95}$$

for $j \in I$, where I is the set of all dimensions of maximal length of hyper-rectangle i. The algorithm continues as in Figure 3.9 until either a maximum number of function evaluations is reached, a maximum number of iterations is reached, or the error value is within a certain distance of a known global minimum [20].

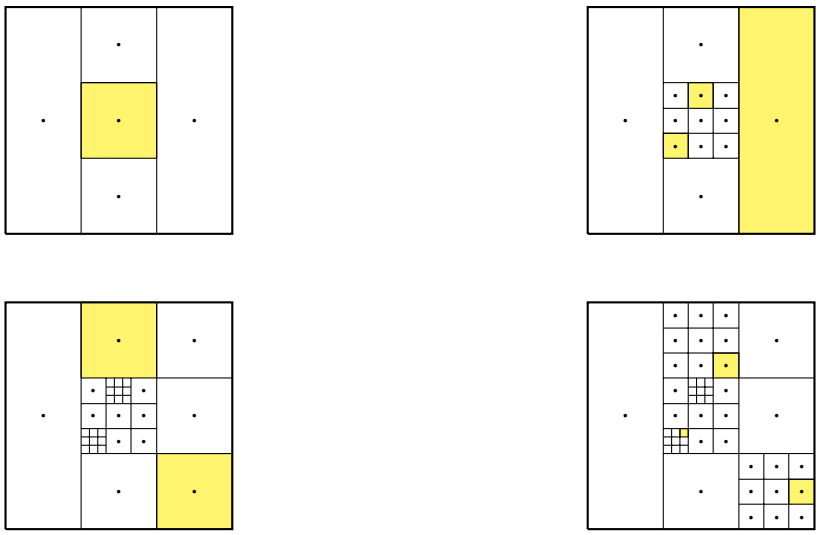


Figure 3.9: An example of four iterations of the DIRECT algorithm. Possible optimal rectangles are highlighted in yellow for each iteration.

Cross validation error is a method of model validation that uses a majority portion of the data to fit the model and the remaining fraction to test it. Cross validation is repeated at least twice on different splits of the data (called n-fold cross validation for n different splits) to determine a theoretical level of accuracy for the model. For DIRECT, we use the error of the five-fold cross-validation estimate as the value we are trying to minimize over a two dimensional rectangle of parameters C and γ for the SVM and the regularization parameters γ and δ for LDA. Some experiments were attempted using different class weights as parameters, but the computation time was prohibitively increased for little to no improvement in results.

3.5 Committee Approach

A reoccurring topic in machine learning is the use of a committee in developing a model for prediction or classification, also referred to as the ensemble approach. Committees occur commonly when using neural network or decision tree based algorithms [48], and can either be applied as a total aggregate approach, or can be combined into subsets of all of the trained classifiers [54]. In order to create a committee, either different types of methods have to be used (such as combining neural networks with decision trees or decision trees with SVMs), or, if the same type of method is used, the architecture has be changed in some way that will result in different results trained on the

same subset. This can be accomplished by varying the hyper parameters for the individual models, or changing the error threshold for the committee members, among other ways [1]. The committee approach is recognized to work better than individual models for three reasons [15]: statistically, computationally, and representationally:

1. Statistical reason: Any machine learning method is searching a possible hypothesis space H in an attempt to find the best hypothesis. If, however, the amount of training data is too small compared to the size of H, then the method could find multiple hypotheses in H that have the same accuracy on the training data. By constructing multiple different classifiers and using them in an ensemble, this approach "averages" out the votes and therefore reduces the risk of finding the wrong class.

2. Computational reason: In order to find the best way to classify, many methods perform a local search that can possibly get stuck in a local optima (the best example being neural networks using gradient descent training). Even when there is enough training data to avoid the statistical pitfall, by combining multiple different classifiers, the various local minima each gets stuck in can be "averaged" to minimize error.

3. Representational reason: When the actual classification function cannot by represented by any hypothesis in H, then the weighted sums of the hypotheses found by the individual members of the committee can expand the search space.

A visual representation of the three reasons for the success of the ensemble method is shown in Figure 3.10. Just as there are a wide variety of machine learning classifiers, there are a variety of ways methods can be combined in an ensemble committee, as detailed in [15]:

1. The Bayesian Voting ensemble method uses the sum over all hypotheses function $h(x) \in H$, each weighted by its posterior probability $P(h|S)$, where S is the training sample.

2. The Training Set Manipulation method of ensemble construction is a relatively straightforward approach, simply giving each member of the committee a different random subset of the training data.

    - The most common way of doing this is taking a set fraction of the original dataset drawn randomly with replacement, called bootstrap aggregation, or bagging, giving each member of the committee an equal vote.
    - By constructing the training sets with disjoint subsets of the training data, the members of the committee are trained similarly to the way cross-validation is done, leading to the name cross-validated committees, which again have equal votes.

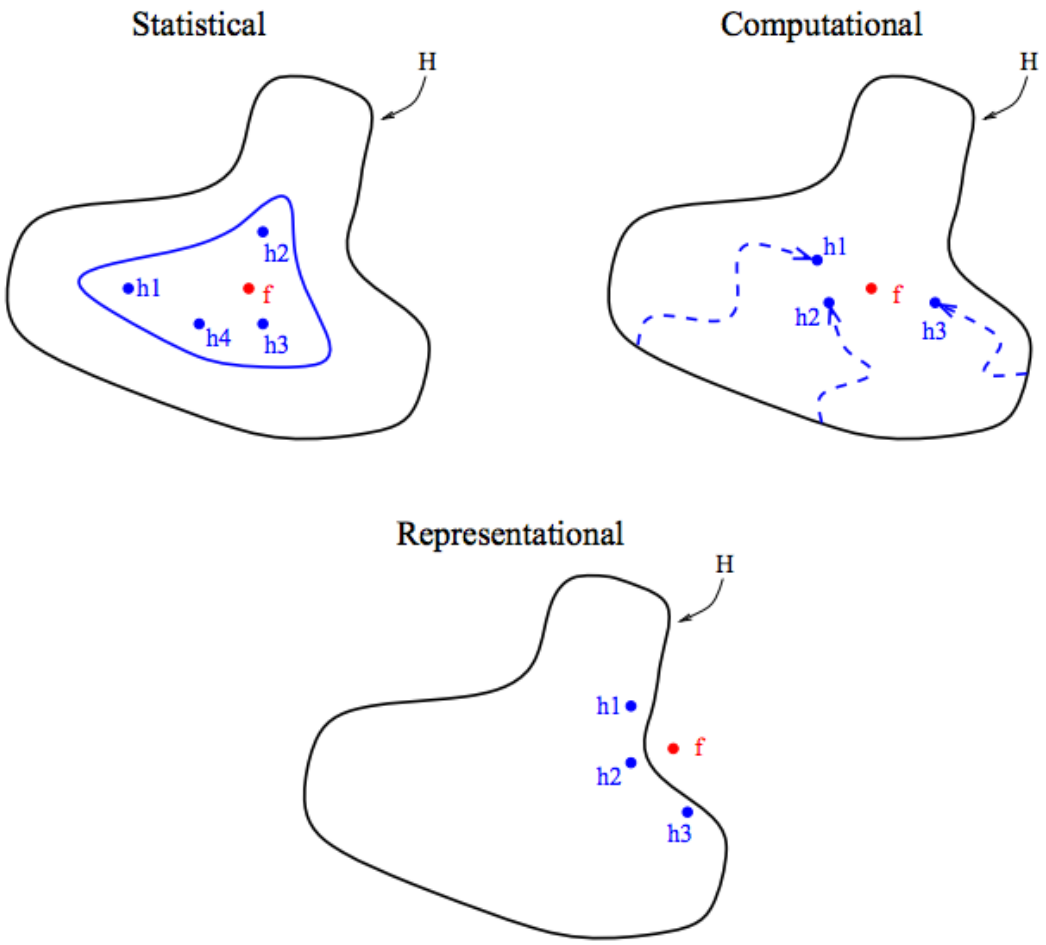


Figure 3.10: A visual representation of the three reasons why an ensemble method works better than a single one, taken from [15]. Each committee member is represented in blue, with the true classifier labeled in red.

- A third way of manipulating the training data is the ADABOOST algorithm, which operates similarly to bagging but associates a weight with each committee member's hypothesis.

3. The Input Feature Manipulation method operates similarly to training set manipulation, but instead of using different subsets of the data as training, uses different subsets of the features themselves.

4. The Output Target Manipulation method uses derived labels with error-correcting output to determine different committee members that have been trained on different output subsets.

5. The Randomness method uses random initializations to create different members of the ensemble, usually applied to neural network algorithms that, because of the backpropagation training method, end up with different weights given different initializations.

The use of a committee approach for Classification Trees and SVMs is not novel, having been used for binary SVM classification in [3, 4], but to our knowledge it has not been extended to multi-class prediction outside of a survey paper [22], and certainly not in the realm of pitch prediction.

We decided to apply bagging to our methods, allowing each classifier an equal vote in the end decision, so our committee of ten models decides according to the pitch type that has the most votes. For our committee approach, we run DIRECT ten times on random permutations of the training set, getting ten different C and γ values for SVM and ten different regularization values (γ and δ) for LDA. We use 100 trees in each iteration of TreeBagger, as trying to optimize the number of trees in each random forest was computationally inefficient and resulted in no accuracy increase. In the case of a tie, five extra models are trained, using a random value between the minimum and maximum values of the ten original parameters.

3.6 Contributions

While the individual machine learning methods we use are well-defined, and the committee method has been used before, the combination of the DIRECT method of parameter optimization with the variety of learning methods bundled into the ensemble committee is a new approach. The redundancy of the random forests inside a committee is also, to our knowledge, an innovative method of improving classification and prediction accuracy, as we will show in Chapter 4.

CHAPTER

4

PREDICTION RESULTS

We implemented all of our experiments in MATLAB, using the statistics and machine learning toolbox for random forests and linear discriminant analysis and the libsvm software [11] for support vector machines. Due to the committee method, we used the parallel computing toolbox on a remote server to train each of the ten committee members at the same time to reduce computation time.

To establish a value for comparison, we found the best "naive" guess accuracy, similar to that used in [23]. We define this naive best guess to be the percent of time each pitcher throws his preferred pitch from the training set in the testing set. Consider some pitcher who throws pitch types 1, 2, 4, and 5 with distribution $P = \{p_1, p_2, p_4, p_5\}$ where $\sum p_i = 1$, and his preferred training set pitch is $\max(P_{\text{train}}) = p_2$, then the naive guess for the pitcher is $P_{\text{test}}(p_2)$. For example, since Jake Arrieta threw his sinker the most in the training set (26.31%), we would take the naive guess as the percentage of the time he threw a sinker in the testing set, which gives a naive guess of 34.03%. For the random forest method, we predicted 48.33% of his pitches correctly, so we beat the naive guess by 14.30% in his case. For all 287 pitchers, the average naive guess was 54.38%.

4.1 Comparison of Methods

Because of the way the committees were built for each pitcher, the LDA parameters γ and δ and the SVM parameters C and γ varied across each member of the ensemble. Table 4.1 shows the average values for each parameter across all of the pitchers tested.

Table 4.1: Average parameters values for all committee members for all pitchers.

LDA Parameters		SVM Parameters	
γ	δ	C	γ
0.0882	0.0932	43.1814	0.0688

Table 4.2 shows the prediction results from each individual method (the random forest method is labeled 100 CT). Results for every individual pitcher tested are given in Appendix A. The number of pitchers we predicted better than naive is given, as well as the percentage of the 287 total pitchers that number represents. The average prediction accuracy is shown, given along with the overall average improvement over the naive guess, denoted $\bar{P}_I$, the average improvement for those pitchers who did beat the naive guess, denoted as $\bar{P}_B$, and the average amount the pitchers who did not beat the naive guess failed by, denoted by $\bar{P_W}$. Given the number of pitchers N with respective prediction value P_i and naive guess G_i, the number who did better than the naive guess, N_B, the number who did worse than the naive guess N_W, we find

$$\bar{P}_I = \frac{\sum_i^N P_i - G_i}{N},$$
$$\bar{P}_B = \frac{\sum_i^{N_B} P_i - G_i}{N_B},$$
$$\bar{P}_W = \frac{\sum_i^{N_W} P_i - G_i}{N_W}.$$

We also give the average range of accuracy between the most and least accurate members of each committee as well as the average time for each pitcher's model to be trained and tested.

As shown in Table 4.2, the random forests of classification trees outperformed both LDA and

Table 4.2: Comparison of prediction results for each method.

Value	**LDA**	**SVM**	**100 CT**
# of Predictions > Naive	263	251	282
% of Predictions > Naive	91.64	87.46	98.26
Prediction Accuracy (%)	65.08	64.49	66.62
$\bar{P}_I$ (%)	10.70	10.11	12.24
$\bar{P}_B$ (%)	13.26	12.38	12.52
$\bar{P}_W$ (%)	-9.08	-5.40	-1.15
Range of Committee (%)	1.52	3.02	2.22
Time (s)	22.75	2,383.8	72.05

SVM by a wide margin. Basing the judgement solely on how many pitchers were predicted better, the random forests were near-perfect, leading the average prediction accuracy and improvement to also be higher. LDA outperforms the random forests only when we examine the average improvement for those pitchers who we are able to beat the naive guess for, but conversely also has much worse performance for the pitchers we do not beat the naive guess for. At this stage, we undertook further comparative analysis to determine if the random forests were the best method overall.

4.2 Results Analysis

4.2.1 Results by Type of Pitcher

Because our data set featured both starters and relievers, we wanted to examine if there was any difference between the prediction accuracy for starters against relievers. Due to the fact that starters will play many more innings (and therefore throw more pitches), the average starter had 6,390 pitches over the three seasons, and the average reliever threw 2,812 pitches over all three seasons. The average naive guess for starters was 51.73%, while for relievers it was 57.28%, indicating that relievers relied on a preferred pitch more than starters did, which makes sense given the nature of a reliever's job: pitch for a short amount of innings, get a lot of strikes and outs, and do so using a pitch they have the most control over. Table 4.3 shows the difference in the random forests prediction accuracy for starters versus relievers for all three methods considered.

As shown in Table 4.3, starters were more difficult to predict than relievers. This result makes sense, as starters generally have more time to pitch in each game, so they have more flexibility to use pitches they have less control over. A starter has more flexibility, more time in the game to salvage a bad at-bat or giving up runs, whereas a reliever is brought in explicitly to try to mount a comeback

Table 4.3: Comparison of prediction results for starters and relievers.

	LDA		**SVM**		**100 CT**	
Value	Starters	Relievers	Starters	Relievers	Starters	Relievers
# of Predictions > Naive	134	130	125	126	146	136
% of Predictions > Naive	89.33	94.89	83.33	91.97	97.33	99.27
Prediction Accuracy (%)	60.08	70.43	59.89	69.53	61.90	71.79
$\bar{P}_I$ (%)	8.35	13.15	8.16	12.25	10.17	14.51
$\bar{P}_B$ (%)	10.86	15.69	10.29	14.47	10.48	14.73
$\bar{P}_W$ (%)	-7.90	-13.92	-2.53	-11.92	-1.12	-1.30

or preserve a lead, and so therefore has more incentive to throw the pitch he is most comfortable with and has the most accuracy with.

Once again, the random forests had a higher prediction accuracy and improvement than both LDA and SVM. The only metric where the random forests were not the highest was in the average positive improvement, beaten slightly by LDA, but the slight gain is overwhelmingly outweighed by the drastic average loss for those pitchers where LDA did not beat the naive guess.

4.2.2 Individual Results

In Table 4.4, we show the ten starters and ten relievers with the highest prediction accuracy based on the results from the random forest method. We compare the results for these pitchers with results from LDA and SVM to examine another metric of measurement between the three methods.

Of the ten starters shown, nine were in the top ten highest naive guesses (and the tenth was the eleventh highest naive guess), and of the ten relievers shown, eight were in the top ten highest naive guesses, giving a good measure of how well each method can handle very imbalanced data sets. Of the 20 pitchers shown, the random forests were either the highest prediction accuracy or tied for the highest accuracy 15 out of the 20 times, giving us confidence in selection the random forests as the best performing method.

To give an example of the kind of results we found for each pitcher, Tables 4.5 and 4.6 give the full breakdown of the predictions for the two pitchers who throw all seven types of pitches: Jeremy Guthrie and Odrisamer Despaigne. From this point forward in this chapter, we focus on the results given by the random forest method since it had the best overall performance.

Both pitchers have relatively similar distributions across the pitches they throw (with the exception of Guthrie only throwing 4 knuckleballs in the whole testing set), but their results demonstrate two possible types of results of prediction. In Guthrie's case, the random forests were very biased towards predicting fastball, and so more than 80% of his fastballs were predicted correctly and only

Table 4.4: 10 highest predicted starters and 10 highest predicted relievers by the random forests, training and testing set sizes, naive guesses, and comparison to LDA and SVM results.

Starters						
Pitcher	Training	Testing	Naive	LDA	SVM	100 CT
R.A. Dickey	7786	2385	88.89	89.31	89.94	89.64
Kevin Gausman	2492	1386	88.31	88.89	88.67	88.82
Lance Lynn	7497	2091	86.90	86.85	87.33	87.33
Taijuan Walker	1490	1905	83.94	84.72	83.20	85.20
Bartolo Colon	6293	1944	83.95	83.44	83.90	84.57
Jake Odorizzi	4262	1831	85.53	80.17	80.17	84.27
Alfredo Simon	5080	2185	81.33	79.82	79.91	82.11
Ubaldo Jiminez	6077	2153	80.12	79.01	78.36	80.03
James Paxton	1809	823	71.81	78.25	77.64	78.86
Juan Nicasio	4446	762	75.33	78.48	75.85	78.35
Relievers						
Pitcher	Training	Testing	Naive	LDA	SVM	100 CT
Koji Uehara	2072	472	99.30	99.30	99.30	99.30
Sam Freeman	1053	476	92.86	92.44	87.61	93.91
Zach Putnam	1063	490	87.78	90.02	84.52	91.04
Jake McGee	2334	384	89.84	90.63	91.15	90.36
Brad Brach	1840	800	86.63	90.00	89.75	89.75
Tony Cigrani	3069	468	88.03	88.03	89.32	89.74
Jonathan Papelbon	2067	649	83.20	89.06	88.44	89.52
Zach Britton	1937	674	31.16	87.98	88.43	89.02
Kenley Jansen	2513	588	85.88	86.39	86.54	87.07
Jeremy Jeffress	974	760	76.45	81.71	81.71	84.74

Table 4.5: **100 CT** pitch-specific model predictions for Jeremy Guthrie, overall accuracy 37.94%.

		Predicted Pitch Type								
		FF	CT	SI	SL	CU	CH	KN	% Thrown	% of Each Correct
Actual Type	FF	432	1	4	3	10	69	0	31.30	83.24
	CT	111	9	1	0	3	29	0	9.23	5.88
	SI	175	1	23	1	4	75	0	16.83	8.24
	SL	53	0	2	5	5	10	0	4.52	6.67
	CU	223	2	8	2	41	65	0	20.57	12.02
	CH	164	1	1	0	2	119	0	17.31	41.46
	KN	4	0	0	0	0	0	0	0.24	0.00

Table 4.6: **100 CT** pitch-specific model predictions for Odrisamer Despaigne, overall accuracy 53.30%.

		Predicted Pitch Type								
		FF	CT	SI	SL	CU	CH	KN	% Thrown	% of Each Correct
Actual Type	FF	330	5	77	2	17	2	0	28.60	76.21
	CT	55	42	19	2	1	8	3	8.59	32.31
	SI	108	5	312	2	19	10	0	30.12	68.42
	SL	31	2	23	10	2	6	2	5.02	13.16
	CU	36	2	43	3	54	3	1	9.38	38.03
	CH	72	1	34	0	2	33	2	9.51	22.92
	KN	63	3	30	3	5	3	26	8.78	19.55

one other pitch type, changeup, had a higher accuracy than the actual percent each type was thrown. For Despaigne, the random forests did much better at predicting more than just a single pitch type correctly.

We show Guthrie and Despaigne since they are the only two pitchers who throw all seven pitch types, but the outputs for each pitcher can be represented with a similar error matrix.

4.2.3 Results by Count

To give us an idea of whether or not our predictions were behaving the way we would expect them to, we examined the breakdown of prediction results given the count of the at-bat. Depending on the count, pitchers and batters behave very differently. A neutral count has the same number of balls and strikes, a pitcher-favored count has more strikes than balls, and a batter-favored count has more balls than strikes. Of the twelve possible ball-strike combinations, three are neutral, three are pitcher-favored, and six are batter-favored (although one could argue a count of 2-2 is tilted more towards the pitcher, since he could throw another ball without walking the batter, but the batter

cannot take one more strike). The average prediction accuracy is shown in Table 4.7.

Table 4.7: Average Prediction accuracy for each pitch count for the **100 CT** method. Pitcher favored counts are shown in bold, batter-favored counts in italics.

Count (B-S)	**100 CT**
0-0	71.48
0-1	64.77
0-2	62.27
1-0	70.01
1-1	61.15
1-2	58.94
2-0	74.78
2-1	67.43
2-2	59.84
3-0	83.00
3-1	75.62
3-2	67.53

The results we found for each count is encouraging, as we would expect that a pitcher facing a batter-favored count would behave more predictably, relying on pitches he is most comfortable with, and therefore has thrown more. In a pitcher-favored count, the pitcher has more flexibility and ability to use a pitch he may not have as much control over, and so will most likely be less predictable. Table 4.8 shows the number of pitchers who we predicted 100% of the time for each given count, by starters and relievers.

On the very batter-favored count of 3-0, we were able to predict 104 pitchers (36.24% of 287 pitchers) totally correct, i.e., for every pitch they threw on a 3-0 count, we predicted them all exactly.

4.2.4 Correlation with Standard Statistics

In an effort to determine if the prediction success correlated with any standard measure of pitcher success, we looked at the ten pitchers whose prediction improved over the naive guess the most, and those ten who improved the least (or were worse than the naive guess). We compared the improvement over the naive guess to the pitchers' wins-above-replacement (WAR) and fielding-independent-pitching (FIP) statistics. FIP is an extension of a pitchers' earned run average (ERA) that examines only outcomes over which the pitcher had control. A good pitcher will have a low FIP

Table 4.8: Number of pitchers predicted 100% accurate for each count.

Count (B-S)	Starters	Relievers	Total
0-0	0	0	0
0-1	0	1	1
0-2	0	0	0
1-0	0	2	2
1-1	0	1	1
1-2	0	1	1
2-0	1	13	14
2-1	0	5	5
2-2	0	1	1
3-0	40	64	104
3-1	6	27	33
3-2	1	4	5

and a high WAR. The comparisons are shown in Table 4.9. FIP and WAR statistics were found via baseball-reference.com.

Table 4.9: 100 CT Prediction Improvement Compared to FIP and WAR.

Most Improved					**Least Improved**				
Pitcher	**Team**	**% Imp.**	**FIP**	**WAR**	**Pitcher**	**Team**	**% Imp.**	**FIP**	**WAR**
Jim Johnson	ATL	79.45	3.73	0.56	Marco Estrada	TOR	-1.74	4.40	3.60
T.J. McFarland	BAL	72.84	4.47	-0.30	Jordan Lyles	COL	-1.37	3.79	0.30
Luis Avilan	LAD	70.24	3.66	0.29	Jared Hughes	PIT	-1.30	3.81	1.20
Jesse Hahn	OAK	67.84	3.51	1.00	Jake Odorizzi	TB	-1.26	3.61	3.60
Hector Santiago	MIN	63.40	4.77	1.80	Ubaldo Jiminez	BAL	-0.09	4.01	2.60
Jaime Garcia	STL	63.04	3.00	3.90	Koji Uehara	BOS	0.00	2.44	1.30
Joe Kelly	BOS	61.95	4.18	1.00	Lance Lynn	STL	0.43	3.44	3.50
Kyle Gibson	MIN	59.88	3.96	3.20	Kevin Gausman	BAL	0.50	4.10	1.30
Tanner Roark	WSH	58.06	4.70	0.70	Jake McGee	COL	0.52	2.33	1.00
Zach Britton	BAL	57.86	2.01	2.50	Bartolo Colon	NYM	0.62	3.84	1.00
Average		65.46	3.80	1.47	**Average**		-0.37	3.58	1.94

In Table 4.9, we examined those pitchers who improved over the naive guess the most rather than just the pitchers with the highest overall prediction accuracy because some pitchers highly

favor one type of pitch, sometimes throwing it upwards of 90% of the time, and so even with only a small improvement they would be one of the highest predicted pitchers. A look back at Table 4.4 will show many of those pitchers with high naive guesses also appearing in the "Least Improved" column in Table 4.9. Many of the pitchers appearing in the "Most Improved" column completely abandoned their formerly preferred pitch in the testing set, and therefore had a naive guess of 0.

After examining the standard metrics next to the prediction improvement, we found a small correlation between the ability of the classification trees to improve on the naive guess and the overall pitcher performance. On average, the pitchers who were hardest to beat the guess had a WAR almost 0.5 higher than those who were most improved on and around 0.2 less FIP. While not a huge difference in performance metrics, these results suggest that the harder it is to predict a pitcher, the better he is in a game.

4.3 Contributions

Our results are better than the start of the art presented in [7], even more so considering that we examine up to seven pitch types as opposed to only four. All three of the methods we examined outperformed the out-of-sample results given in [7], with the random forest method giving the best performance, LDA giving the next best, most efficient, and most consistent (i.e., lowest variability within the committee) performance, and SVM performing the worst overall. The use of multiple random forests together in an ensemble method was able to improve on the performance of LDA without costing too much in computation time, and therefore are the most promising method to examine further for prediction.

CHAPTER

5

DDAGS AND FEATURE ANALYSIS

One of the most important part of any machine learning method is the feature selection (as discussed in Chapter 2) and the subsequent reduction in the number of features to avoid overfitting to the training data, leading to poor out of sample performance. To explore feature reduction within our methods, we decided to use a pair of feature reduction techniques prevalent throughout the literature, such as [27]. An issue, however, arose due to these techniques being built for binary classifiers. In order to overcome this obstacle, we needed to employ a multi-class classifier that still behaved like a binary one, which led us to Decision Directed Acyclic Graphs.

5.1 Directed Acyclic Graphs

Directed Acyclic Graphs (DAGs) are directed graphs of finite size with no cycles, i.e., a graph that has a specified number of edges and vertices, where each edge points to a single vertex, and there is no way to cycle back to a vertex once you have followed an edge from it. DAGs are used in variety of mathematical settings, but have been implemented in machine learning using an approach analagous to a decision tree. Referred to as a decision directed acyclic graph (DDAG), it is a graph where some form of a decision is placed at each vertex that has at least one edge pointing away from it, leading to nodes at the end, which are vertices that only having edges pointing towards them [45].

We use the definition of DDAGs from [39]:

Definition 1. *Decision Directed Acyclic Graphs (DDAGs): Given a space X and a set of boolean functions $F = \{f : X \to \{0,1\}\}$, the class DDAG($F$) of Decision DAGs on N classes over F are functions which can be implemented using a rooted binary DAG with N leaves labeled by the classes where each of the $K = N(N-1)/2$ internal nodes is labeled with an element of F. The nodes are arranged in a triangle with the single root node at the top, two nodes in the second layer and so on until the final layer of N leaves. The ith node in layer $j < N$ is connected to the ith and $(i+1)$st node in the $(j+1)$st layer.*

To evaluate some DDAG graph G on an input x, start at the root node's binary function. The input exits the root node either to the right or left, then the next node's binary function operates on the input, again exiting either to the left or right, continuing down the graph until the input reaches a class labeled node, also called a leaf. A DDAG can be considered equivalent to operating on a list where each binary function at the internal nodes eliminates one possible class from the list that is initialized with all classes. The evaluation on the DDAG ends when only one class remains in the list [39].

Similar to the one-vs-one and one-vs-all methods, a DAG implementation can be used for support vector machines [29, 39]. In this instance, the support vector machines are used as the decisions for the interior vertices, and the class labels are the end vertices. Because each vertex is only making a binary decision, to implement the DAGSVM construction, each binary combination of classes has to have a classifier trained for it, resulting in $N(N-1)/2$ total classifiers trained.

The DAG implementation is not limited solely to SVMs. To examine whether the alternate formulation of the multiclass problem changed our results, we also employed binary LDA and random forest classification as intermediate decision vertices. Again, this required training $N(N-1)/2$ classifiers of each type to populate the graph. In an attempt to fully enhance classification, we employed the DIRECT parameter optimization method for each of the binary classifiers in the DAGLDA and DAGSVM constructions.

Because the DDAG formulation is similar to a decision tree, we can visualize it in a similar way. Figure 5.1 gives a graphical representation of how we employed the directed graph method for each pitcher, starting with the binary classifier for the "first" and "last" type of pitch thrown, and progressively working towards the middle pitches separation.

For example, if a pitcher throws the five pitches given as an example in Figure 5.1, then for an unknown pitch, the inputs will first be fed into the 1v5 (FF vs CH) binary classifier, and if the pitch is classified as 1, then the unknown features will be used as inputs into the 1v4 binary classifier. If that classifier gives a result of 4, the pitch proceeds to the 2v4 classifier, and assuming it is classified as 2

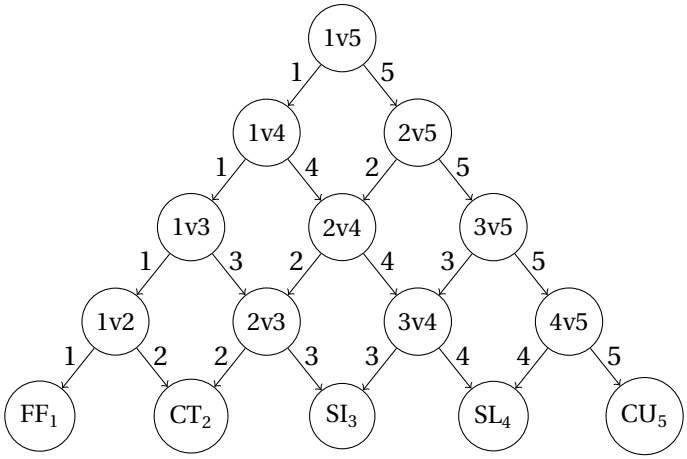


Figure 5.1: A DAG visualization for the classification of five pitch types. A binary classifier is trained for each pair of pitch types in the ellipses.

by this binary decision then to the 2v3 classifier, and if it is classified as 3 from that binary model then the final answer is 3, or sinker. This example is shown visually in Figure 5.2.

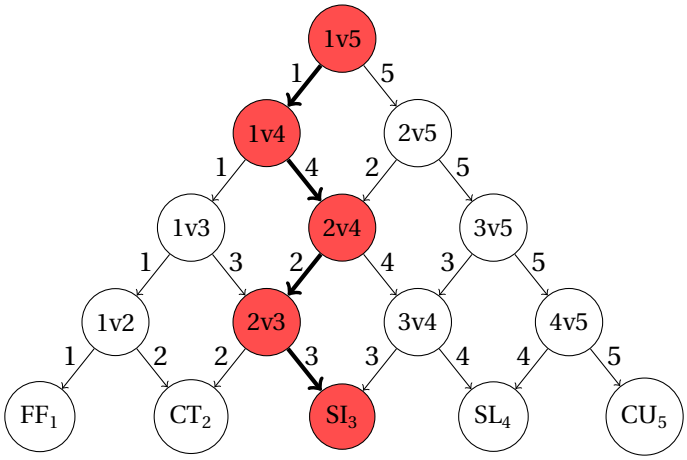


Figure 5.2: An example of how the DAG classifier labels an unknown input.

While the DDAG implementation was worth exploring as an alternative formulation of the multi-class classification problem (the results of which are shown in Table 5.2), we also utilized the unique

interior binary structure of the DAG to employ preprocessing feature selection methods. Because these methods are designed for and only work with binary classifiers, the $N(N-1)/2$ classifiers trained in the construction of the DAG are able to undergo individual feature reduction at each individual node, allowing for each binary decision to be optimized on its own, and not risk losing important data that may be disregarded due to the correlations between the other classes.

Our work focused on two specific binary methods of feature selection and reduction. Because these are preprocessing techniques, we had to be able to find the most amount of information possible from each data set before the classifiers were trained. Similar to [27], we used receiver operating characteristic (ROC) curves as one method of feature reduction for each binary classifier. We also examined the F-score method as detailed in [13] due to its use in previous work as an SVM feature selection strategy.

5.2 Feature Reduction

One of the most commonly cited issues in machine learning is the so-called "curse of dimensionality." This problem arises most often when there are more features associated with a given observation than the number of observations themselves. Due to a high number of features, without an equivalently high number of observations, the features can be thought of as sparse since the likelihood that every possible combination will not be seen in training, and can lead to overfitting of the training data. Overfitting results when the classifier is trained too tightly on the training set, and cannot adapt to or accurately classify out of sample observations from the testing set. By reducing the number of features given for each data point, then the classifier is more likely to be able to predict the testing set [34].

While the PITCHf/x data we gathered for our experiments does not necessarily run the risk of overfitting (the highest any ratio of the number of features to the number of training observations would be is roughly 15%), feature reduction techniques can still be useful in providing a more adaptable framework for out of sample prediction.

5.2.1 F-Scores

As described in [13], F-scores are used to measure the discrimination between two classes within a certain feature. For some set of feature vectors $\mathbf{x_j}$, $j = 1, \ldots, n$, where the number of instances of 1 is

n_1 and the number of instances of -1 is n_{-1}, then the F-score of the i^{th} feature is

$$F(i) = \frac{(\bar{x}_{1_i} - \bar{x}_i)^2 + (\bar{x}_{-1_i} - \bar{x}_i)^2}{\frac{1}{n_1 - 1}\sum_{j=1}^{n_1}(x_{1_{j,i}} - \bar{x}_{1_i})^2 + \frac{1}{n_{-1} - 1}\sum_{j=1}^{n_{-1}}(x_{-1_{j,i}} - \bar{x}_{-1_i})^2}, \tag{5.1}$$

where $\bar{x}_i, \bar{x}_{1_i}, \bar{x}_{-1_i}$ are the means of the i^{th} feature of the entire set, the set of 1's, and the set of -1's, respectively; $x_{1_{j,i}}$ is the i^{th} feature of the j^{th} instance of 1, and $x_{-1_{j,i}}$ is the i^{th} feature of the j^{th} instance of -1.

Once the F-scores of each feature are found, we determine a threshold to eliminate features with a score less than. We tested thresholds of 0.01, 0.005, and 0.001 for all of the pitchers examined. Table 5.1 shows the average amount of features remaining given each thresholding level for pitchers throwing various pitchtypes. Figure 5.3 shows the F-score plots for Cleveland Indian's pitcher Corey Kluber for all six binary decisions made in the DDAG implementation as well as the thresholding levels in different colors.

As seen in Figure 5.3, each binary split can have very different amounts of features remaining to be used as input data, which can then affect whether some binary decisions are more accurate than others within the DDAG construction. Table 5.2 shows the results of the different levels of thresholding we implemented.

5.2.2 ROC Curves

First developed during WWII for object detection on battlefields, receiver operating characteristic (ROC) curves were shortly thereafter introduced in many application areas as a method of visualizing the traditional confusion matrix of a classifier [18]. ROC curves are generated by plotting the true positive rate (TPR, the sensitivity) of a classifier against its false positive rate (FPR, 1 - specificity) as the threshold (the value of the feature) increases. We define TPR and FPR as

$$\text{TPR} = \frac{tp_1}{t_1}, \tag{5.2}$$

$$\text{FPR} = \frac{fp_{-1}}{t_1}, \tag{5.3}$$

where tp_1 is the number of correct predicted 1's, fp_{-1} is the number of incorrectly predicted -1's, and t_1 is the total number of true 1's. Because these curves are on a scale of 0 to 1, all the features used, including categorical features, must be normalized. To decide whether or not to keep a feature, we examine the area under the curve (AUC). It is common practice to eliminate features with an AUC less than 0.5, which is the equivalent of simply guessing randomly [42]. An example ROC curve is

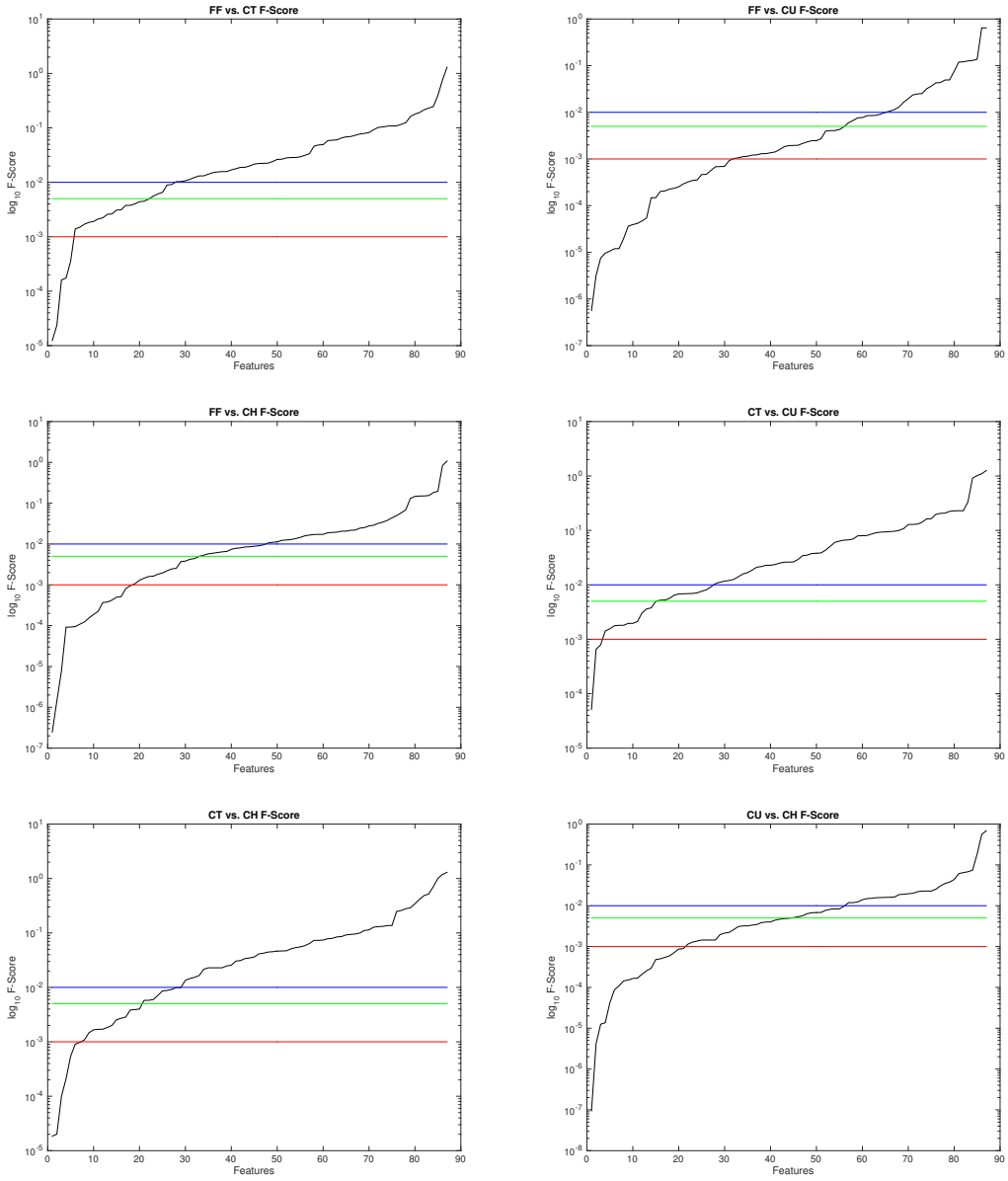


Figure 5.3: F-Score plots for the six binary splits of pitches thrown by Corey Kluber. F-Scores are shown in $\log_{10}$. Thresholding levels of 0.01, 0.005, and 0.001 are shown in blue, green, and red, respectively.

Table 5.1: Average features remaining given different thresholding measures for F-Scores and ROC AUC.

Pitches Thrown	Binary Decisions	# Features	**AUC<.5**	**FS<.001**	**FS<.005**	**FS<.01**
2	1	63	25.13	42.13	19.75	11.13
3	3	71	35.81	51.4	34.1	25.01
4	6	79	39.39	58.9	40.01	30.91
5	10	87	44.53	66.74	47.62	37.83
6	15	95	45.91	70.35	49.96	39.59
7	21	103	41.43	77.48	51.90	39.17

shown in Figure 5.4 for Corey Kluber for the fastball vs. curveball decision made in the DDAG. The two highest and two lowest AUCs are shown.

Most pitchers had ROC behavior similar to Kluber when the data was split up for the individual decisions. Because the binary decisions were being made about each pair of pitch types, generally the least informative features were the historical or game-time information about the pitches that were not being considered, resulting in their elimination from the training set. Table 5.2 shows the results of eliminating the features with an AUC < 0.5 for each DDAG implementation.

5.2.3 Reduced Subset Results

Due to issues in the way the binary subsets were constructed, we eliminated five pitchers from the DAG implementation. These errors were caused by the pitchers throwing so few pitches of a certain type that the training or testing sets had none of that type of pitch in them. We examined how many features were, on average, left for the pitchers who threw each number of pitch types (since the size of the dataset changes for the different numbers of pitches thrown) and show the results in Table 5.1. The subsequent reduced subset prediction results are shown in Table 5.2.

For both LDA and SVM, we again used the DIRECT method for parameter optimization of γ and C for SVMs and δ and γ for LDA. We used DIRECT to optimize for each individual binary classifier on the interior of the tree, which caused a dramatic spike in computation times for both methods as shown in Table 5.2 in the standard DDAG implementation, but was lessened out by the feature reduction techniques.

Once again, the random forest method has the best overall performance, improving on the standard implementation prediction results for both the regular DAG without any feature reduction and the DAG with the ROC curve feature reduction. Past that, however, as fewer and fewer features are used (since as the F-score threshold goes up, less features are used in the classifier), the random forest performance declines. Since random forests already only use a subset of the data for the

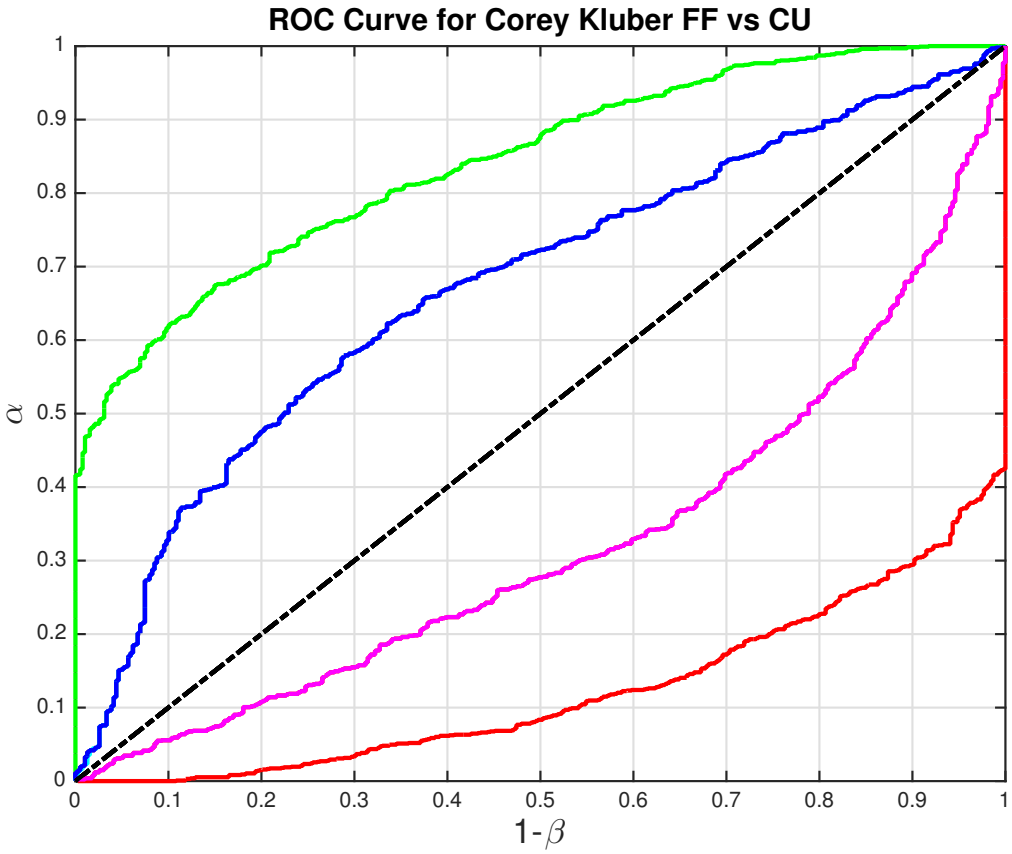


Figure 5.4: Highest two AUCs and lowest two AUCs for Corey Kluber for Fastball vs. Curveball split. The highest AUC, 0.8414, for his historical tendency to throw a fastball, is shown in green; the second-highest, 0.7123, for the percent of the last five pitches that were curveballs in blue; the second-lowest, 0.322, for his historical tendency to throw a changeup in magenta, and lowest, 0.1219, for his historical tendency to throw a slider in red. The black line in the middle has an AUC of 0.5.

Table 5.2: Average values comparing different preprocessing techniques for the reduced subset of 282 pitchers.

SVM					
Value	**DAG**	**AUC<.5**	**FS<.001**	**FS<.005**	**FS<.01**
# Better	251	268	255	262	265
% Better	89.00	95.04	90.43	92.91	93.97
Acc.	64.86	63.67	65.35	65.70	65.51
Imp.	10.59	9.40	11.08	11.43	11.24
PI	12.58	10.10	12.92	12.76	12.52
NL	-4.69	-3.19	-4.38	-5.41	-7.99
CR	3.19	3.85	2.88	2.81	2.73
Time	11,499	5,950.9	6,839.2	4,236.2	3,265.7
LDA					
Value	**DAG**	**AUC<.5**	**FS<.001**	**FS<.005**	**FS<.01**
# Better	261	274	260	260	259
% Better	92.56	97.16	92.20	92.20	91.84
Acc.	64.90	64.30	65.39	65.21	65.11
Imp.	10.63	10.03	11.12	10.94	10.84
PI	13.07	10.54	12.97	12.80	12.68
NL	-11.63	-0.90	-10.06	-9.92	-9.35
CR	1.68	2.14	1.76	1.52	1.42
Time	963.1	812.6	440.6	340.6	337.2
100 CT					
Value	**DAG**	**AUC<.5**	**FS<.001**	**FS<.005**	**FS<.01**
# Better	278	277	274	268	258
% Better	98.58	98.23	97.16	95.04	91.49
Acc.	66.91	66.90	66.89	66.20	65.22
Imp.	12.64	12.63	12.62	11.93	10.95
PI	12.90	12.92	12.99	12.81	12.36
NL	-1.73	-1.13	-2.00	-2.97	-3.13
CR	2.42	2.45	2.36	2.42	2.39
Time	116.1	111.8	122.1	105.5	100.8

construction of each tree, as well as practicing a type of feature selection in the pruning method, reducing the number of features overall is not going to boost performance. Regardless of the selection method, however, random forests were by far the fastest method since there was no parameter optimization employed.

We noticed the opposite trend with SVMs, noting in fact the performance generally increased as

fewer features were used in the training set. This is most likely due to the kernel trick and its effect on finding the optimal hyperplane, as reducing the number of features usually leads to less risk of overfitting and better performance on out of sample observations.

5.3 Variable Importance

While ROC curves and F-scores are used as pre-processing techniques to eliminate those features that provide the *least* amount of information, post-processing techniques can be used to determine what features are the *most* important. Post-processing techniques are dependent on a model being made already, and so we had to use the models created for the results in Chapter 4 to find measures of variable importance: the permuted variable delta error (PVDE) for the random forests and the delta predictor (Δ P) for LDA.

Before we expanded our dataset to include all 287 pitchers and all seven pitch types, our preliminary experiments were run on a smaller, less comprehensive dataset that only had 110 pitchers and incorporated five pitch types. We employed variable importance methods on the models created in that instance, including for SVM, using the central difference derivative based sensitivity equation

$$\frac{\partial g_j}{\partial x_i} \approx \frac{g_j(x_i+h)-g_j(x_i-h)}{2h}, \tag{5.4}$$

where g_j is the decision value for the j^{th} class and x_i is the i^{th} feature. This derivative approximation did not work well due to a variety of factors, most notably the change of dimensionality caused by using the kernel function and the multiple types of input (i.e., continuous vs. binary vs. categorical), and so an accurate measure of variable importance for the SVMs could not be found.

The delta predictor for LDA, as previously described in Chapter 3, can be used in conjunction with the δ parameter to reduce the number of features used in the testing set. The delta predictor is found once the regularized linear term

$$(\vec{x}-\mu)^T\hat{\Sigma}^{-1}(\mu_k-\mu)=\left[(\vec{x}-\mu)^T D^{-1/2}\right]\left[\hat{C}^{-1}D^{-1/2}(\mu_k-\mu)\right] \tag{5.5}$$

is established, where the terms are found as detailed previously. The right-most term of the right-hand side of the equality is the vector of delta predictors, i.e.,

$$\Delta P=\left[\hat{C}^{-1}D^{-1/2}(\mu_k-\mu)\right]. \tag{5.6}$$

For a feature set with N variables, this is an N-by-1 vector where the magnitude of each entry is the measure of variable importance [35]. The MATLAB implementation of discriminant analysis

provides the vector for each model constructed.

MATLAB also gives a convenient flag that can be turned on when constructing random forests to include a variety of measures of variable importance, including the number of splits for each predictor, two different permuted variable margin differences, and the one we use as our measure of variable importance for the random forests: permuted variable delta error (PVDE).

The PVDE is found during the construction of each random forest for each variable by first finding the expected error ($E_{O\,i}$) against a hold-out validation set, similar to the cross-validation used for the parameter optimization. The values for a particular variable x_i are then randomly permuted across every observation in the subset of the training data used for the tree construction, and the expected error value ($E_{P\,i}$) is found against the same holdout set. The PVDE for each variable is found by

$$\text{PVDE}_i = E_{O\,i} - E_{P\,i}, \tag{5.7}$$

again output as an N-by-1 vector where the magnitude is the measure of variable importance to the model [36].

Table 5.3 gives the ranks of the delta predictors and permuted variable delta error for each input feature group (with 29 feature groups, total), respectively. The ranks were found by first averaging the values for each pitcher over all ten models created, then sorting those averages by magnitude, and then averaging each rank across each variable in the group. Once the group ranks were found, we sorted the averaged group ranks to find the overall importance.

Some of the variable importance results are consistent with what we would expect, but many others are not, especially the disparity between the rankings of the two methods. As seen in Table 5.3, the LDA and random forest models agree on the relative importance of a few of the inputs (such as the strike and ball count going into each pitch) but differ widely on others. For example, the LDA models rank the batter handedness as the 25th most important variable group while the random forests consider it to be the 7th most important, with that feature group having the largest absolute difference between the two method. The next largest difference between the two methods comes for the Previous 10 Pitch Tendency group, which the LDA model ranks as the 4th most important but the random forests rank as the 17th most. To further analyze the variable importance results, we again broke the information down between starters and relievers, shown in Table 5.4.

There are few wide disparities in the importance for starters and relievers for each model type, but the small differences are fairly consistent with what we would expect. For the LDA mode, the top or bottom of the inning's importance increases by five for relievers, which is reasonable since a reliever will change his pitch selection if he knows his team is guaranteed another at bat or not

Table 5.3: Variable Importance for LDA Delta Predictor and CT Permuted Variable Delta Error for all pitchers. 1 means highest importance, 29 means lowest importance.

Feature Group	Delta Predictor	PVDE
Inning	13	16
Top or Bottom	20	29
Outs	26	27
Order Position	15	18
Total At-Bat	7	6
Score Spread	25	21
Time of Day	19	25
Batter Handedness	23	7
Strikes	3	2
Balls	2	3
On Base	28	28
Base Score	29	19
Previous At-Bat Result	27	24
Previous Pitch Result	22	10
Previous Pitch Type	1	4
Previous Pitch Location	18	8
Pitch Number	10	1
Previous Pitch Stats	5	5
Previous 5 Pitch Tendency	6	13
Previous 10 Pitch Tendency	4	17
Previous 20 Pitch Tendency	8	14
Previous 5 Pitch Strikes	12	11
Previous 10 Pitch Strikes	9	15
Previous 20 Pitch Strikes	14	20
Pitcher Historical Tendency	24	26
Pitcher Tendency vs. Batter	21	23
Batter Strike Tendency	17	22
Batter In-Play Tendency	11	12
Batter Ball Tendency	16	9

(i.e., the top of the ninth inning, if his team is up, a reliever will pitch to prevent runs and prevent his team from going back to the plate). For the random forest importance, the largest difference is in the batter in-play tendency, with the reliever models assigning a higher importance, suggesting that the relievers are more adverse to throwing a pitch that a batter may put in play at all, as that increases the risk of runs being scored.

Table 5.4: Variable Importance for LDA Delta Predictor and CT Permuted Variable Delta Error broken down by starts and relievers.

	Delta Predictor		PVDE	
Feature Group	Starters	Relievers	Starters	Relievers
Inning	12	13	16	16
Top or Bottom	21	16	29	29
Outs	27	26	27	27
Order Position	15	18	18	18
Total At-Bat	8	7	6	6
Score Spread	26	24	21	21
Time of Day	18	20	24	25
Batter Handedness	20	25	7	7
Strikes	3	2	5	2
Balls	2	3	2	5
On Base	28	27	28	28
Base Score	29	29	20	19
Previous At-Bat Result	25	28	25	24
Previous Pitch Result	24	22	11	10
Previous Pitch Type	1	1	3	3
Previous Pitch Location	17	21	8	8
Pitch Number	10	11	1	1
Previous Pitch Stats	5	5	4	4
Previous 5 Pitch Tendency	6	6	13	13
Previous 10 Pitch Tendency	4	4	17	17
Previous 20 Pitch Tendency	7	8	12	14
Previous 5 Pitch Strikes	11	12	10	12
Previous 10 Pitch Strikes	9	9	15	15
Previous 20 Pitch Strikes	14	15	19	20
Pitcher Historical Tendency	23	23	26	26
Pitcher Tendency vs. Batter	22	14	23	23
Batter Strike Tendency	19	17	22	22
Batter In-Play Tendency	13	10	14	9
Batter Ball Tendency	16	19	9	11

5.4 Contributions

Feature selection and reduction is an important topic in any machine learning application. To the best of the author's knowledge, the use of the DDAG architecture to allow the use of binary feature reduction techniques such as F-Scores and ROC curve analysis for a multi-class method is novel. Because measuring the discrimination between multiple classes is a difficult and ill-defined task, using the binary separation techniques allows for useful information to be found and previously employed reduction methods to be used. Other works in the pitch prediction domain such as [7], [53], and [23] have mentioned what features are the most important in their work, but have not broken the full analysis down in the detail shown in Table 5.3. Since the models used previously have all focused on SVMs, the use of the LDA Delta Predictor and the random forest Permuted Variable Delta Error as measures of importance for pitch prediction is also novel.

CHAPTER

6

LIVE IMPLEMENTATION

At the start of this research, one of the reasons we examined different machine learning methods of prediction was to determine what would work best in real time in a live game environment. The previous experiments were all done in a "bulk" setting, i.e., predicting all of the testing set all at once. While this gives a way to measure the effectiveness of each method, the construction of the testing datasets was not reflective of the way a dataset would be built during an actual baseball season. Any live prediction training set could only be updated with after each game, and would only show historical pitcher or batter tendencies up to the day before a game was played.

The data for the live predictions was parsed appropriately, creating pitcher preferences and batter performance measures up until the day being predicted. We examined the games in September 2016, creating models for each pitcher for not only predicting the type of pitch thrown, but also the speed of the pitch and the location of the pitch (as determined by the zones detailed in Chapter 2). Models were created for every pitcher who pitched in September, as long as he had pitched at some point after the All-Star break (mid-July) and before September 1st.

Along with the rebuilt datasets, the other difficulty involved in implementing the prediction in real time was the programming language the prediction method was written in. MATLAB is a powerful software originally designed for manipulating matrices and expanded to include the statistics and machine learning toolboxes we employed in the original results, but it is not open-

source. Because it requires a license for each user in an academic, government, or business setting, it is not necessarily flexible enough for the design of the live prediction environment [19]. These limitations required us to find a different language to use, which led us to employ Python. Because a large amount of machine learning work done by mathematicians is reliant on MATLAB and its licenses in academia, we feel it is important to introduce a different language and its capabilities in general as well as for our application.

6.1 Conversion to Python

Python is high-level dynamic programming language that was first introduced in 1991 with the updates Python 2.0 and Python 3.0 following in 2000 and 2006, respectively. Guido van Rossum, a Dutch developer, came up with the language as a "hobby" programming project [41]. Designed to support multi paradigms, Python allows for object-oriented and structure programming, among others. The language was built to be extendable, resulting in a plethora of open-source packages that can be downloaded and called with an "import" command to increase the versatility of the code.

The guiding philosophy behind the Python language can be summed up in "The Zen of Python" included in the Python Developer's guide. This philosophy [38] contains such single-sentence guidelines as:

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.

Python can be used to scrape and download data from the web, similar to the SQL process we discussed in Chapter 2. To create an up-to-date, Python-compatible dataset, we used the *BeautifulSoup* package to download and parse the .xml pages from gd2.mlb.com.

Similar to the statistics and machine learning toolbox in MATLAB, Python uses the *scikit-learn* package to build and apply different machine learning methods. Because we use the ensemble method, we use the subpackage *sklearn.ensemble* to maintain a similar methodology to the testing done in MATLAB. We constructed our models using the RandomForestClassifier structure from *scikit-learn*. These random forests are built similarly to the TreeBagger class, using the Gini diversity index as a measure of impurity and employing pruning to reduce the size of the trees.

As discussed previously, we limited the training data to only what was available prior to the day tested. Datasets for each pitcher, as well as up-to-the-day summaries for each batter, were constructed to be used. An example of an individual pitch and subsequent output for pitch type, speed, and location is given in Figure 6.1.

We decided to test our live prediction method on the tail end of the 2016 baseball season, between September 1st and October 2nd, the last day of the regular season. There was a large amount of data available to test on, and the characteristics of the data are shown in Table 6.1.

Table 6.1: Details of the testing dataset from 9/1/2016 to 10/2/2016 used for the live predictions.

Value	Amount
Days	32
Total Pitches	111,425
Pitcher Appearances	3,892
Unique Pitchers	544
Pitches/Day	3,482
Pitches/Appearance	28.63
Pitch Type	# Thrown
FF	58,102
CT	4,588
SI	6,974
SL	17,075
CU	13,272
CH	11,222
KN	192

Instead of simulating every pitch one-by-one, we designed a bulk testing method that stayed

```
Who's Pitching? Jake Arrieta
453562
Model 0 made
Model 1 made
Model 2 made
Model 3 made
Model 4 made
Model 5 made
Model 6 made
Model 7 made
Model 8 made
Model 9 made
Who's Batting? Andrew McCutchen
457705
Height: 69
Stance (1 for R, 0 for L): 1
Inning: 4
Outs: 2
Spread: -1
Time of Day: 3
Balls: 2
Strikes: 1
On First: 0
On Second: 1
On Third: 0
Prev AB Result: Strikeout
Prev Pitch Result : Ball
Prev Pitch Type: SI
Prev Zone (Guess?): 10
Prev Pitch Speed: 89
Fastball at 73.1725 mph in zone 11
Who's Batting?
```

Figure 6.1: Example of the Python interface for live pitch prediction on the command line in OSX.

true to the up-to-the-day training set but also accounted for the game situational data. Part of the live prediction datasets includes additional features: the pitcher's historical tendency given the current count he is facing. While not used in the MATLAB results, we include the new synthetic features here in hope they will help raise prediction accuracy even higher. The results from testing from the first day of September until the end of the 2016 regular season are shown in Table 6.2. For the sake of efficiency, the results shown use 20 trees per random forest created. Because we attempted to create models to predict both the speed and the location, we have included those results, which we will discuss later. ΔSpeed is the average difference between the predicted speeds and the actual speeds, and the zone accuracy is simply the percentage of locations we predicted

correctly.

Table 6.2: Results of the live prediction method for September 1st, 2016 through October 2nd, 2016.

	Prediction Acc.	ΔSpeed	Zone Acc.
Value	58.69%	14.0649	12.29%

Overall, the pitch type accuracy did a satisfactory job, as the amount of training data varied wildly for the pitchers we examined over the course of the 32 days. The random forests did not do as well as the MATLAB implementation, but that could be due to a few factors, the reduced number of trees and training set size among them. The error matrix for the entire testing set is shown in Table 6.3 in a similar style as the previous individual pitcher results were.

Table 6.3: Python live pitch predictions for September 1st through October 2nd, 2016, with overall accuracy 58.69%.

		Predicted Pitch Type								
		FF	CT	SI	SL	CU	CH	KN	% Thrown	% of Each Correct
Actual Type	FF	53322	354	803	1839	1154	610	20	52.14	91.77
	CT	2766	1356	200	16	133	114	3	4.12	29.56
	SI	787	168	5386	362	189	82	0	6.26	77.23
	SL	12681	60	1521	2640	92	81	0	15.32	15.46
	CU	10153	364	651	183	1726	195	0	11.91	13.00
	CH	8668	197	990	296	275	796	0	10.07	7.09
	KN	18	0	0	0	0	0	174	0.17	90.62

We also examined the progression of the prediction over the course of the testing set, to see if the addition of training data would make a difference in the prediction accuracy (i.e., would the accuracy go up as time went on). The daily results, plotted alongside the number of pitchers that day, and compared to the overall accuracy, are shown in Figure 6.2.

There is no correlation between the time passed and the prediction accuracy, but the accuracy mostly follows along with the number of pitchers throwing that day. This is consistent with expectations, as more pitchers will generally mean a better training set overall, leading to better results for the given day.

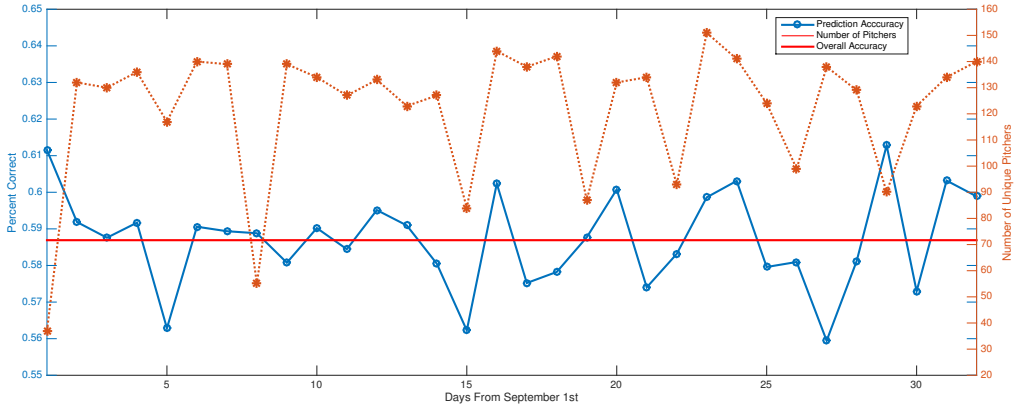


Figure 6.2: Progression of daily prediction accuracy from September 1st to October 2nd.

6.1.1 Live Prediction Limitations

While the pitch type prediction results are somewhat consistent with what we had found with the MATLAB approach, the speed and location predictions do not achieve the same accuracy as the pitch type. This is consistent with expectations, however, as the feature set was developed specifically to predict the pitch type, and not the speed or location of the pitch, which would involve a different set of informative features. We have included the histogram of the error between the predicted pitch speed in Figure 6.3 and the histogram overlay of the predicted zone versus the actual zone for the pitches thrown in Figure 6.4.

Again, because the feature set was not designed for predicting the speed or the location of the pitch, these results, while disappointing, are not surprising. The upside of these results, however, is that we have a better idea of the types of modifications necessary for improving these alternate models in the future.

6.2 iOS App Development

Due to the agreement between the MLB and Apple, the iPad tablet can now be found in every dugout around the league [37]. We decided the next step in the real-time, in-game prediction was to work on a mobile application that would work on iOS, using the capabilities of Python as the workhorse and the iPad interface to communicate back and forth with a remote server the Python was running on.

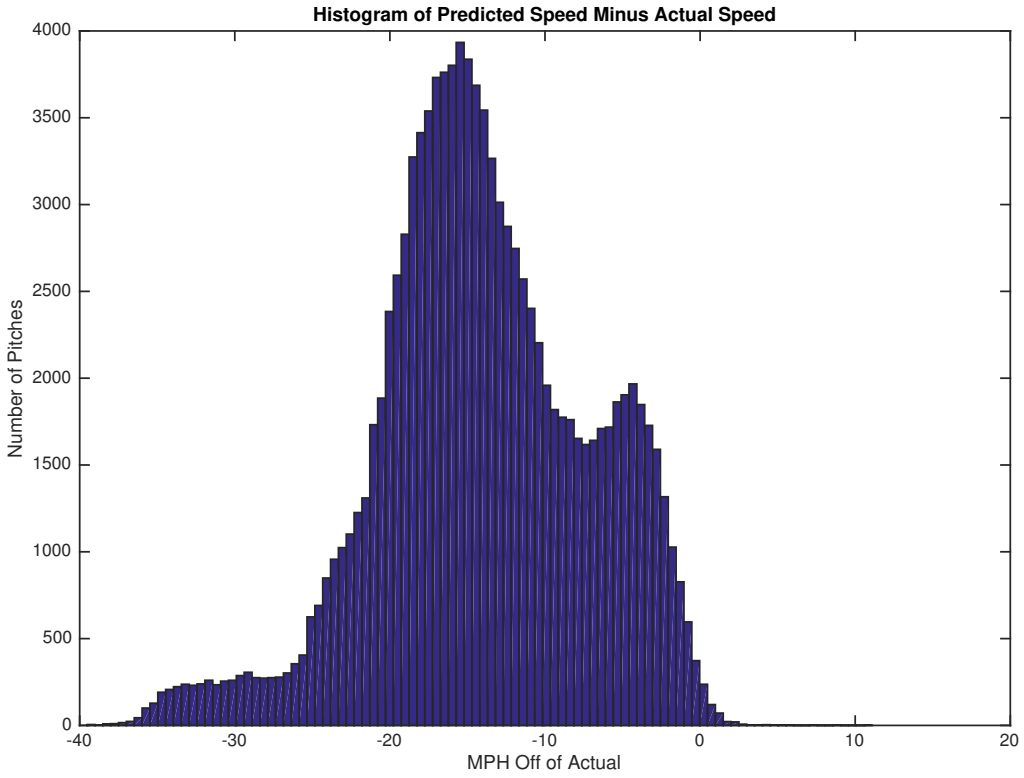


Figure 6.3: Histogram of error between predicted pitch speed and actual pitch speed.

Because an iPad does not necessarily have the hardware necessary to build and run the random forest classifiers, we decided to use a Representational State Transfer (REST) Application Program Interface (API). This REST API allows the iPad to communicate with the server running the Python code, transferring data back and forth in a JSON format, designed to transmit data in a minimalist style [52]. This JSON data is relayed in the same way as it would be in the interface shown in Figure 6.1, allowing for real-time predictions.

We worked with a graduate student in the Computer Science department, Dustin Lampright, who developed the GUI for the iOS application and the REST interface built into the app. A prototype GUI for the application is shown in Figure 6.5. The application development was pursued until the author started a full-time job, but will hopefully be resumed in the future either by the author or a fellow graduate student.

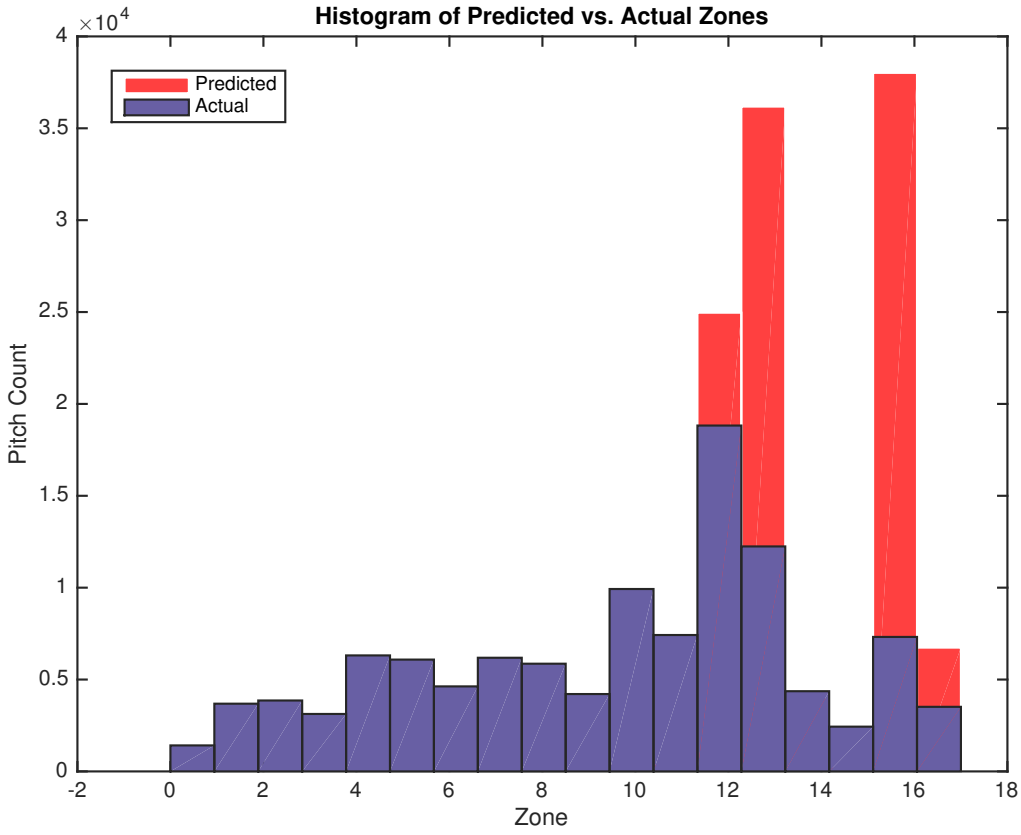


Figure 6.4: Histogram overlay between the predicted zones and actual zones.

6.3 Contributions

Besides occasionally hearing an announcer take a guess during a windup, the author believes no prior attempts have been made at mathematically predicting the type of pitch as a game is going on. The conversion to Python and subsequent development of the input environment to allow for real-time prediction was a large step forward in the use of open-source machine learning packages to enhance sports analytics, leading to a prediction method that, given time and more improvement, may perform better, helping hitters get on base and teaching pitchers how to change their routines and increase strikeout rates.

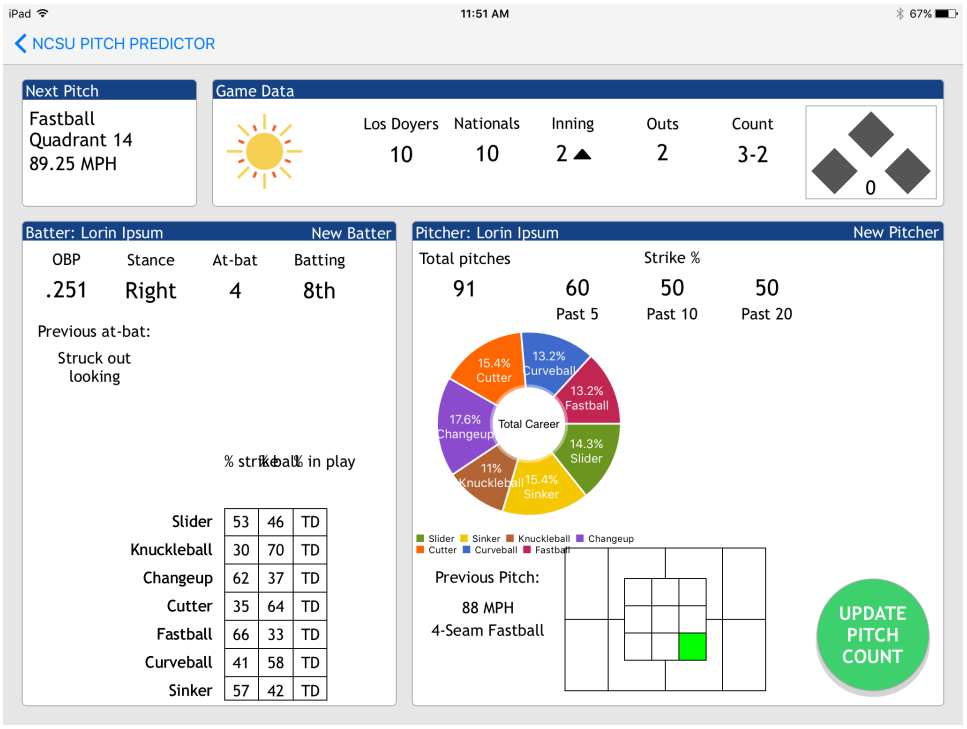


Figure 6.5: GUI for the pitch prediction iOS application.

CHAPTER

7

CONCLUSION AND FUTURE WORK

7.1 Summary of Contributions

Our work in pitch prediction has centered around the difficulties involved in taking a classifier designed for a binary problem and extending it to multi-class prediction. We highlight some of our key contributions here:

- The use of expert domain knowledge in the development of a feature set for the input into our models, balancing the uncertainty caused from using the PITCHf/x data with the extra insight those features give.
- Comparing various multi-class machine learning methods in an attempt to find which is the best and the most computationally efficient gives us an advantage over previous work that relied solely on a single classification method for prediction.
- The use of the DIRECT optimization method for parameter optimization combined with the ensemble approach for the prediction method is a previously unused optimization method and a very lightly used approach for the multi-class problem.
- Using the the DDAG construction for the multi-class classification combined with the committee approach and DIRECT optimization is a novel method for the extension of the decision

graph architecture.

- To use binary feature selection techniques, our use the of the DDAG structure is a new approach that allows us to utilize the spread out construction of the graph architecture for feature reduction.
- Because previous work centered around the use of SVMs as the pitch predictor, utilizing the LDA and random forest variable importance measures is a different approach to feature analysis.
- Converting the code to Python and implementing a live, in game style of prediction is an innovative approach that may be used in real MLB situations.
- All of the models tested, but especially the random forests, represent a large margin of improvement over the naive guess, which would give a batter extra knowledge at the plate, theoretically helping to improve on base percentage or even batting average.

7.2 Pitch Prediction Future Work

Even though the results from the random forest method outperform the best naive guess, we do not have a human accuracy to measure against. The random forests essentially replicate what the best hitters in the MLB already do, using context of the game and prior knowledge about the pitcher to anticipate the type of incoming pitch. While the limitations of machine learning will prevent 100% accuracy from ever being achieved, the method can be improved in a few ways.

We have shown that feature generation is very important in creating the model and can be improved with some feature reduction techniques and variable importance measures. The same intuition and expert knowledge that led to the generation of the current synthetic features can be used in an attempt to find and include more into the model. Some possible examples include data about the individual ballpark being played in (some ballparks are regarded as more hitter- or pitcher-friendly), who is catching (as some pitchers have a strong preference for one catcher or another), how many days rest a pitcher is throwing on, or even how long it has been since he was last injured. We feel that the best chance of improvement lies in selecting better and more informative features. As discussed in Chapter 6, the live prediction for speed and location does not perform as well as the pitch type prediction, and so the best way to improve those predictions may rely on reducing the size of or changing the input feature set.

Some relatively recent developments in the subfield of machine learning called deep learning may be one way to improve the features used and therefore the prediction accuracy. Recurrent

neural networks (RNNs) can be used with time-dependent data (as pitches over the course of a game are) as a feature extraction method, finding the most informative combination for the top-level classifier (in this case our random forest method). RNNs are multi-layered neural networks that can examine past, present, and future data in the training stage to find the best features for prediction, and can be optimized using different architectures, such a long short term memory (LSTM) or gated feedback RNNs [14]. The author's employment has sparked a need and desire to learn, implement, and improve deep learning techniques, and so he is hopeful that they will lead to a second sports statistics revolution: as opposed to Bill James' effort to help managers and coaches and fans learn more from combinations of the data, machine learning may lead to the data teaching them how to play the game.

BIBLIOGRAPHY

[1] Abbott, D. "Benefits of Creating Ensembles of Classifiers". *TDAN.com* (2001).

[2] Aisen, B. *A Comparison of Multi-Class SVM Methods*. 2006. URL: http://courses.media.mit.edu/2006fall/mas622j/Projects/aisen-project/ (visited on 04/28/2016).

[3] al., A. Vinayagam et. "Applying Support Vector Machines for Gene ontology based gene function prediction". *BMC Bionformatics* (2004).

[4] al., B. Moslem et. "Combining multiple support vector machines for boosting the classification accuracy of uterine EMG signals". *Electronics, Circuits and Systems (ICECS)* (2011).

[5] Attarian Adam, e. "A Comparison of Feature Selection and Classification Algorithms in Identifying Baseball Pitches". *Int. MultiConference of Engineers and Computer Scientists 2013, Lecture Notes in Engineering and Computer Science* (2013), pp. 263–268.

[6] Berg, T. "How Cubs ace Jake Arrieta went from one of MLB's worst starters to one of its best." *USA Today* (2015).

[7] Bock, J. R. "Pitch Sequence Complexity and Long-Term Pitcher Performance". *Sports* (3 2015), pp. 40–55.

[8] Breiman, L. "Random Forests" (2001).

[9] Breiman, L. e. a. *Classification and Regression Trees*. Brooks/Cole, 1984.

[10] Brown, M. *MLB Sees Record Revenues For 2015, Up $500 Million And Approaching $9.5 Billion*. Ed. by Magazine, F. 2015. URL: http://www.forbes.com/sites/maurybrown/2015/12/04/mlb-sees-record-revenues-for-2015-up-500-million-and-approaching-9-5-billion/#2f00c62f2307 (visited on 04/25/2016).

[11] Chang, C.-C. & Lin, C.-J. "LIBSVM: A library for support vector machines". *ACM Transactions on Intelligent Systems and Technology* **2** (3 2011). Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm, 27:1–27:27.

[12] Charts, S. *Pitches Per Start Leaders: 2015 MLB Season*. 2015. URL: http://www.sportingcharts.com/mlb/stats/pitching-pitches-per-start-leaders/2015/ (visited on 09/28/2016).

[13] Chen, Y.-W. & Lin, C.-J. "Combining SVMs with Various Feature Selection Strategies". *Department of Computer Science, National Taiwan University* (2005).

[14] Chung, J. "Gated Feedback Recurrent Neural Networks". *Proceedings of the International Conference on Machine Learning* (2015).

[15] Dietterich, T. G. “Ensemble Methods in Machine Learning”. *Multiple Classifier Systems: First International Workshop, MCS 2000 Cagliari, Italy, June 21–23, 2000 Proceedings*. Springer Berlin Heidelberg, 2000, pp. 1–15.

[16] Dietterich, T. G. & Bakiri, G. “Solving multiclass learning problems via error-correcting output codes”. *Journal of artificial intelligence research* **2** (1995), pp. 263–286.

[17] ESPN. *MLB Standings 2013-2015*. 2015. URL: `http://espn.go.com/mlb/standings/_/season/2013/group/overall` (visited on 04/25/2016).

[18] Fawcett, T. “An Introduction to ROC Analysis”. *Pattern Recognition Letters* **27** (8 2006).

[19] Feldman, P. *Eight Advantages of Python Over Matlab*. 2016. URL: `http://phillipmfeldman.org/Python/Advantages_of_Python_Over_Matlab.html`.

[20] Finkel, D. “DIRECT Optimization Algorithm User Guide”. *Center for Research in Scientific Computation, NCSU* (2003).

[21] Fisher, R. “The Use of Multiple Measurements in Taxonomic Problems”. *Annals of Eugenics* **2** (7 1936), pp. 179–188.

[22] Galar, M. e. a. “An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes”. *Pattern Recognition* **44** (8 2011).

[23] Ganeshapillai, G. & Guttag, J. “Predicting the Next Pitch”. *Proceedings of the MIT Sloan Sports Analytics Conference* (2012).

[24] Guestrin, C. “SVMs, Duality, and the Kernel Trick”. *Machine Learning-10701/15781* (2007).

[25] Gupta, M. R. et al. “Training highly multiclass classifiers.” *Journal of Machine Learning Research* **15**.1 (2014), pp. 1461–1492.

[26] Guyon, I. & Elisseeff, A. “An Introduction to Variable and Feature Selection”. *Journal of Machine Learning Research* (2003).

[27] Hoang, P. “Supervised Learning in Baseball Pitch Prediction and Hepatitis C Diagnosis”. *NC State University, Ph.D. Thesis* (2015).

[28] Holmes, G. et al. “Multiclass alternating decision trees”. *European Conference on Machine Learning*. Springer. 2002, pp. 161–172.

[29] Hsu, C.-W. & Lin, C.-J. “A Comparison of Methods for Multi-class Support Vector Machines”. *Department of Computer Science, National Taiwan University* (2005).

[30] Lettmann & Stein. *ML Chapter III: Decision Trees*. 2005-2015.

[31] Lewis, J. *On Base Percentage Vs. Batting Average: Which Is More Important?* Ed. by Report, B. 2008. URL: http://bleacherreport.com/articles/40132-on-base-percentage-vs-batting-average-which-is-more-important (visited on 04/25/2016).

[32] Li, T. et al. "Using discriminant analysis for multi-class classification: an experimental investigation". *Knowledge and information systems* **10**.4 (2006), pp. 453–472.

[33] *MAMP and MAMP PRO*. 2016. URL: https://www.mamp.info/en/.

[34] Marimont, R. B. & Shapiro, M. B. "Nearest Neighbor Searches and the Curse of Dimensionality". *IMA Journal of Applied Mathematics* **24**.1 (1979), pp. 59–70.

[35] MathWorks. *Discriminant Analysis*. 2016. URL: http://www.mathworks.com/help/stats/discriminant-analysis.html (visited on 06/29/2016).

[36] MathWorks. *fitctree.m documentation*. 2016. URL: http://www.mathworks.com/help/stats/fitctree.html (visited on 06/29/2016).

[37] Perez, A. "MLB, Apple strike agreement: iPads in dugouts". *USA TODAY* (2016).

[38] Peters, T. "The Zen of Python". *PEP 20* (2004).

[39] Platt, J. C. et al. "Large Margin DAGs for Multiclass Classification." *nips*. Vol. 12. 1999, pp. 547–553.

[40] Potts, A. "Rise of the machines". *The Economist* (2015).

[41] Rossum, G. van. "A Brief Timeline of Python". *The History of Python* (2009).

[42] Serrano Antonio J, e. a. "Feature selection using ROC curves on classification problems". *Proceedings of the 2010 International Joint Conference on Neural Networks* (2010).

[43] Shalizi, C. "Classification and Regression Trees". *36-350, Data Mining* (2009).

[44] Sportvision. *PITCHf/x Technology*. 2014.

[45] Takahashi, F. & Abe, S. "S.: Optimizing Directed Acyclic Graph Support Vector Machines". *In Proc. of ANN in Pattern Recognition*. 2003, pp. 166–170.

[46] Thatcher, T. *Discriminant Analysis Documentation*. 2016. URL: https://media.readthedocs.org/pdf/discriminantanalysis/latest/discriminantanalysis.pdf (visited on 08/22/2016).

[47] Theodoridis, S. *Pattern Recognition*. Elsevier, Inc., 2009.

[48] Tresp, V. “Committee Machines”. *Handbook for Neural Network Signal Processing*. Ed. by Hu, Y. H. & Hwang, J.-N. CRC Press, 2001.

[49] Veksler, O. *CS434a/541a: Pattern Recognition*.

[50] Weinberger, M. “Apple is facing a crisis of salesmanship”. *Business Insider* (2016).

[51] Weston, J., Watkins, C., et al. “Support vector machines for multi-class pattern recognition.” *ESANN*. Vol. 99. 1999, pp. 219–224.

[52] Wong, M. *Building a Simple REST API for Mobile Applications*. 2015. URL: `https://www.sitepoint.com/building-simple-rest-api-mobile-applications/`.

[53] Woodward, N. “A Decision Tree Approach to Pitch Prediction”. *The Hardball Times* (2014).

[54] Zhou Zhi-Hua, e. a. “Ensembling neural networks: Many could be better than all.” *Artificial Intelligence* **137** (1 2002).

APPENDIX

APPENDIX

A

FULL PITCHER RESULTS

Table A.1: Results from all 287 pitchers for all three methods tested.

Pitcher	Naive	LDA	SVM	100 CT
A.J. Burnett	29.99	59.88	57.93	59.60
A.J. Ramos	45.70	62.03	61.39	59.87
Aaron Harang	67.26	68.87	67.02	69.64
Aaron Loup	47.04	72.78	70.93	67.59
Aaron Sanchez	80.36	79.04	80.17	81.68
Adam Warren	39.75	51.20	52.15	50.76
Addison Reed	66.51	74.31	73.70	76.91
Al Alburquerque	61.07	71.73	72.00	73.20
Alex Torres	64.05	71.43	68.10	74.29
Alex Wood	26.73	56.46	56.60	62.56
Alfredo Simon	81.33	79.82	79.91	82.11
Allen Webster	60.27	76.00	71.47	72.80
Andrew Cashner	65.10	69.90	69.45	69.45
Andrew Miller	42.18	42.18	73.60	76.61
Anibal Sanchez	50.46	53.29	50.75	53.75
Anthony DeSclafani	61.74	63.28	63.78	66.52

Table A.1: (continued)

Pitcher	Naive	LDA	SVM	100 CT
Antonio Bastardo	76.73	76.86	73.07	79.08
Aroldis Chapman	74.02	76.28	75.44	78.05
Bartolo Colon	83.95	83.90	83.44	84.57
Blaine Boyer	64.16	77.58	76.25	78.17
Blaine Hardy	37.75	64.12	62.82	63.98
Blake Treinen	62.43	78.42	77.87	76.50
Brad Boxberger	64.10	73.25	72.41	74.22
Brad Brach	86.63	90.00	89.75	89.75
Brad Hand	71.63	71.63	71.72	74.20
Brandon Gomes	58.59	73.54	73.84	76.23
Brandon Maurer	35.26	66.49	66.67	67.54
Brett Anderson	53.42	61.12	60.37	62.77
Brett Cecil	45.27	64.02	63.45	60.42
Brett Oberholtzer	59.58	12.29	55.00	65.21
Brian Duensing	32.96	48.13	50.94	51.12
Brian Matusz	59.97	71.99	73.48	73.15
Bryan Morris	39.70	55.25	56.34	50.61
Bryan Shaw	81.39	82.41	79.33	83.44
Bud Norris	59.96	68.22	69.35	66.98
Burke Badenhop	61.45	75.42	72.35	75.98
C.J. Wilson	52.96	58.75	58.54	56.10
Carlos Carrasco	57.01	60.59	56.81	61.36
Carlos Martinez	56.58	63.19	59.77	62.76
Carlos Torres	35.79	63.38	61.20	62.21
Carlos Villanueva	44.96	62.40	62.40	66.40
Casey Fien	56.67	65.82	70.76	69.12
Casey Janssen	48.95	69.00	68.07	73.66
CC Sabathia	26.25	46.42	48.12	46.99
Cesar Ramos	41.07	58.06	53.55	58.23
Chad Bettis	65.73	64.69	60.61	69.44
Charlie Morton	72.36	78.05	76.93	77.15
Chase Anderson	59.41	64.99	63.13	65.50
Chris Archer	52.93	61.73	61.29	61.73
Chris Bassitt	30.99	48.88	48.19	49.56

Table A.1: (continued)

Pitcher	Naive	LDA	SVM	100 CT
Chris Capuano	43.59	64.29	67.03	64.10
Chris Hatcher	65.55	70.02	68.90	72.04
Chris Sale	51.61	56.72	55.53	54.92
Chris Tillman	62.52	63.04	63.47	64.55
Chris Young	57.78	65.48	63.85	66.17
Christian Bergman	47.07	58.93	55.68	56.55
Clay Buchholz	42.96	48.90	46.32	47.81
Clayton Kershaw	54.30	62.81	63.07	64.63
Cody Allen	62.26	62.26	75.00	77.00
Colby Lewis	40.53	56.22	54.19	53.38
Cole Hamels	51.12	58.99	58.29	57.30
Collin McHugh	33.88	46.28	50.00	51.34
Corey Kluber	31.36	37.79	41.17	42.31
Craig Breslow	50.15	60.23	59.36	58.92
Craig Kimbrel	69.48	78.71	78.18	77.38
Dale Thayer	72.89	74.89	76.44	82.00
Dallas Keuchel	55.35	58.54	59.97	57.84
Dan Haren	41.59	55.30	55.50	56.07
Dan Jennings	53.06	76.03	76.20	76.53
Dan Otero	62.18	70.76	67.45	70.37
Daniel Webb	66.05	75.33	73.21	74.27
Danny Duffy	67.32	67.32	68.32	69.03
Danny Farquhar	42.81	59.20	59.74	60.66
Danny Salazar	70.12	63.21	60.12	71.62
Darren ODay	44.10	64.46	64.20	63.04
David Buchanan	64.81	67.08	64.69	67.88
David Hale	61.92	68.93	63.43	68.07
David Holmberg	60.52	66.49	63.12	62.60
David Phelps	56.84	59.86	59.55	60.25
David Price	49.18	54.64	53.43	52.03
David Robertson	56.18	71.30	70.08	68.70
Dellin Betances	51.39	69.05	69.94	69.74
Dillon Gee	54.50	57.43	55.63	58.56
Doug Fister	32.57	66.28	61.13	65.47

Table A.1: (continued)

Pitcher	Naive	LDA	SVM	100 CT
Drew Hutchison	65.27	69.24	67.42	69.19
Drew Pomeranz	69.22	76.89	74.37	74.46
Drew Smyly	57.11	61.65	60.22	61.05
Drew Storen	37.03	59.81	60.44	58.70
Edinson Volquez	25.01	51.45	51.45	51.87
Edward Mujica	74.80	74.80	81.00	83.40
Edwin Jackson	63.00	69.81	71.90	74.52
Erasmo Ramirez	53.74	60.64	58.67	59.19
Eric Stults	45.66	51.45	48.75	48.36
Ervin Santana	54.62	65.14	66.24	66.84
Esmil Rogers	57.18	59.33	59.09	65.31
Felix Doubront	53.18	58.11	56.94	57.98
Felix Hernandez	15.99	42.12	42.54	42.77
Fernando Abad	61.57	34.89	53.36	64.93
Fernando Rodney	59.15	74.69	76.57	77.44
Fernando Salas	59.10	69.25	68.83	68.27
Francisco Liriano	46.59	58.27	58.98	60.02
Francisco Rodriguez	38.22	65.94	64.49	68.66
Franklin Morales	73.80	77.48	77.00	77.80
Garrett Richards	35.05	50.59	51.36	50.45
Gerrit Cole	67.13	67.50	68.09	67.79
Gio Gonzalez	63.67	66.19	67.62	66.95
Glen Perkins	65.72	74.25	73.08	75.92
Greg Holland	47.14	70.82	72.86	71.02
Hector Noesi	53.15	64.41	59.46	64.64
Hector Rondon	57.60	66.79	68.58	67.82
Hector Santiago	0.00	54.45	50.22	63.40
Hisashi Iwakuma	40.14	43.08	48.89	51.35
Ian Kennedy	61.18	59.82	63.57	62.54
Ian Krol	59.63	34.56	64.38	68.87
J.A. Happ	68.72	71.57	72.67	70.99
J.J. Hoover	69.77	72.79	72.91	75.33
J.P. Howell	58.58	70.15	69.78	68.66
Jacob deGrom	61.40	63.47	60.05	63.89

Table A.1: (continued)

Pitcher	Naive	LDA	SVM	100 CT
Jaime Garcia	0.00	62.58	55.75	63.04
Jake Arrieta	29.15	49.40	50.52	48.33
Jake Diekman	34.72	78.40	61.47	75.09
Jake McGee	89.84	90.63	91.15	90.36
Jake Odorizzi	85.53	80.17	80.17	84.27
Jake Peavy	48.71	54.47	56.31	57.81
Jake Petricka	23.13	82.28	38.43	28.92
James Paxton	71.81	78.25	77.64	78.86
James Shields	37.10	53.70	53.44	54.84
Jared Hughes	83.55	78.35	76.19	82.25
Jarred Cosart	50.58	52.56	52.21	59.30
Jason Grilli	65.75	74.66	73.52	76.48
Jason Hammel	50.82	59.34	58.37	59.60
Jason Vargas	53.83	63.10	62.30	61.49
Jean Machi	53.05	68.70	67.76	68.54
Jeanmar Gomez	66.79	71.89	70.34	72.24
Jeff Locke	62.35	65.45	63.21	63.06
Jeff Samardzija	57.47	57.91	59.45	61.11
Jered Weaver	39.18	48.81	47.29	48.57
Jeremy Affeldt	65.93	65.93	78.19	75.98
Jeremy Guthrie	31.30	39.32	40.11	37.94
Jeremy Hellickson	52.74	61.29	61.12	61.53
Jeremy Jeffress	76.45	81.71	81.71	84.74
Jerome Williams	33.01	51.43	50.59	49.06
Jesse Chavez	40.42	51.16	51.72	52.10
Jesse Hahn	0.00	65.75	64.80	67.84
Jeurys Familia	66.44	69.10	67.01	71.06
Jim Johnson	0.00	83.12	79.84	79.45
Jimmy Nelson	39.02	48.94	45.14	52.23
Joakim Soria	52.70	71.64	67.38	70.89
Joaquin Benoit	43.63	65.85	65.72	63.28
Joba Chamberlain	47.87	66.92	62.41	66.92
Joe Kelly	0.00	64.78	58.32	61.95
Joe Smith	30.88	65.37	63.42	64.95

Table A.1: (continued)

Pitcher	Naive	LDA	SVM	100 CT
John Axford	68.19	74.65	76.05	74.90
John Danks	41.89	41.89	51.31	50.47
John Lackey	64.62	64.62	65.80	66.51
Johnny Cueto	48.63	57.45	55.54	58.60
Jon Lester	46.94	52.45	55.23	53.06
Jon Niese	53.03	14.92	44.44	57.67
Jonathan Broxton	64.44	76.25	78.19	74.03
Jonathan Papelbon	83.20	89.06	88.44	89.52
Jordan Lyles	64.90	63.73	60.59	63.53
Jordan Zimmermann	59.74	65.89	65.33	64.71
Jorge De La Rosa	68.17	69.93	67.77	70.22
Jose Fernandez	52.16	52.16	58.86	63.60
Jose Quintana	58.04	60.64	60.48	60.12
Josh Collmenter	66.39	69.89	69.05	71.51
Josh Fields	76.29	80.21	79.14	80.39
Josh Tomlin	47.54	59.38	58.00	55.85
Juan Nicasio	75.33	78.48	75.85	78.35
Julio Teheran	62.93	65.42	65.19	67.45
Jumbo Diaz	74.01	77.37	71.39	78.83
Junichi Tazawa	78.72	78.72	11.74	82.02
Justin De Fratus	57.44	66.82	65.40	68.72
Justin Grimm	58.41	68.10	64.92	71.27
Justin Masterson	36.77	60.45	59.47	61.14
Justin Verlander	58.11	57.15	55.65	59.62
Justin Wilson	78.67	80.73	79.94	83.25
Kelvin Herrera	77.47	77.47	81.25	82.03
Ken Giles	59.78	75.67	75.55	76.28
Kenley Jansen	85.88	86.39	86.56	87.07
Kevin Gausman	88.31	88.89	88.67	88.82
Kevin Jepsen	67.32	70.37	70.37	73.66
Kevin Quackenbush	74.35	79.30	79.43	81.90
Kevin Siegrist	75.27	77.09	79.76	79.01
Koji Uehara	99.30	99.30	99.30	99.30
Kyle Gibson	0.00	60.88	55.38	59.88

Table A.1: (continued)

Pitcher	Naive	LDA	SVM	100 CT
Kyle Hendricks	61.83	68.18	65.88	68.04
Kyle Kendrick	25.96	47.01	43.87	52.55
Kyle Lobstein	19.67	49.17	47.23	48.34
Kyle Lohse	43.32	56.20	56.09	56.52
Lance Lynn	86.90	86.85	87.33	87.33
LaTroy Hawkins	72.15	78.69	81.84	78.93
Liam Hendriks	66.86	71.92	69.24	74.89
Luis Avilan	0.00	76.02	74.83	70.24
Luke Gregerson	42.93	71.72	70.88	75.42
Madison Bumgarner	48.72	56.58	56.25	57.08
Marc Rzepczynski	58.98	80.34	79.37	80.58
Marco Estrada	58.43	41.48	51.22	56.69
Mark Buehrle	54.23	56.79	56.73	58.36
Mark Melancon	57.85	73.69	74.33	74.20
Martin Perez	56.80	59.67	56.68	60.98
Masahiro Tanaka	47.42	50.72	54.78	54.57
Mat Latos	49.05	60.41	61.47	61.70
Matt Belisle	61.30	72.21	73.77	72.47
Matt Cain	50.74	55.44	56.03	57.21
Matt Garza	67.69	67.17	68.78	70.69
Matt Shoemaker	57.45	63.99	60.55	65.19
Matt Thornton	69.05	79.29	80.48	78.81
Max Scherzer	59.97	62.75	62.07	65.62
Michael Pineda	35.67	56.95	53.45	56.68
Michael Wacha	59.59	62.87	60.06	62.68
Miguel Gonzalez	58.47	60.20	57.92	61.74
Mike Bolsinger	37.24	50.74	44.65	52.46
Mike Dunn	63.32	71.43	73.08	72.94
Mike Fiers	57.11	58.79	58.33	61.19
Mike Leake	46.67	53.03	51.54	53.68
Mike Morin	39.00	63.50	60.00	68.75
Nathan Eovaldi	63.58	61.71	61.97	65.49
Neal Cotts	51.15	73.68	70.31	71.79
Nick Martinez	52.25	61.32	64.13	60.51

Table A.1: (continued)

Pitcher	Naive	LDA	SVM	100 CT
Odrisamer Despaigne	8.59	51.12	50.00	53.30
Oliver Perez	46.07	67.23	62.17	64.04
Pat Neshek	38.08	75.20	75.04	75.20
Pedro Strop	42.61	69.13	70.43	71.88
Phil Hughes	53.72	58.82	58.88	56.36
R.A. Dickey	88.89	89.31	89.94	89.64
Randall Delgado	53.63	60.99	57.14	61.32
Rick Porcello	61.10	63.55	62.62	63.61
Ricky Nolasco	55.62	67.89	68.92	65.24
Robbie Ray	70.06	73.94	68.60	72.47
Robbie Ross Jr.	58.59	60.08	58.05	63.19
Roberto Hernandez	21.15	51.28	45.83	50.11
Roenis Elias	51.04	60.35	60.66	60.74
Ross Detwiler	34.02	65.50	61.52	66.34
Rubby De La Rosa	60.83	64.85	61.32	64.56
Ryan Vogelsong	56.19	56.19	56.54	60.63
Sam Dyson	75.14	75.14	78.45	78.59
Sam Freeman	92.86	92.44	87.61	93.91
Santiago Casilla	53.87	66.23	66.39	64.58
Scott Carroll	64.90	71.21	72.22	70.45
Scott Feldman	37.81	46.78	49.54	48.16
Scott Kazmir	16.90	39.84	44.11	58.69
Sergio Romo	62.73	68.32	66.30	65.22
Shane Greene	46.90	57.82	54.46	60.55
Shawn Kelley	58.06	77.93	71.31	78.10
Shawn Tolleson	66.07	74.40	72.50	73.93
Shelby Miller	67.14	65.04	62.95	67.82
Sonny Gray	60.06	62.50	61.44	63.24
Stephen Strasburg	61.89	64.92	67.27	65.52
Steve Cishek	39.43	59.43	63.57	57.86
T.J. McFarland	0.00	73.26	75.79	72.84
Taijuan Walker	83.94	84.72	83.20	85.20
Tanner Roark	0.00	60.48	41.48	58.06
Tim Hudson	47.89	54.26	50.28	53.78

Table A.1: (continued)

Pitcher	Naive	LDA	SVM	100 CT
Tim Lincecum	40.51	53.93	54.20	53.79
Tom Koehler	55.89	62.19	61.16	61.77
Tom Wilhelmsen	53.95	64.12	63.42	66.95
Tommy Hunter	60.59	71.00	71.38	73.05
Tommy Kahnle	49.02	68.19	69.50	69.28
Tommy Milone	56.09	62.19	62.06	60.22
Tony Cingrani	88.03	88.03	89.32	89.74
Tony Sipp	50.27	67.39	66.31	65.05
Tony Watson	75.95	78.02	78.14	79.12
Travis Wood	65.52	65.93	66.53	67.85
Trevor Bauer	56.15	61.33	56.06	62.38
Trevor Cahill	14.69	40.17	41.04	39.31
Trevor May	57.02	65.43	61.65	65.05
Trevor Rosenthal	76.95	76.72	75.82	78.53
Tyler Clippard	62.40	75.19	74.31	76.07
Tyson Ross	50.65	64.27	64.61	63.62
Ubaldo Jimenez	80.12	79.01	78.36	80.03
Vance Worley	65.17	69.40	65.30	70.73
Vidal Nuno	33.82	56.43	55.65	57.12
Wade Davis	55.66	64.11	61.90	62.29
Wade Miley	56.05	57.99	55.71	59.24
Wei-Yin Chen	64.61	65.11	64.66	65.99
Will Smith	52.15	73.48	69.19	66.41
Wily Peralta	65.64	67.93	69.27	70.77
Yohan Flande	48.20	58.37	57.75	64.95
Yordano Ventura	55.64	62.47	60.03	61.46
Yovani Gallardo	49.06	49.06	53.96	55.32
Yusmeiro Petit	45.48	59.63	57.14	61.07
Zach Britton	31.16	87.98	88.43	89.02
Zach Duke	51.76	67.67	65.32	64.28
Zach McAllister	76.47	37.95	46.02	78.66
Zach Putnam	87.78	90.02	84.52	91.04
Zack Greinke	54.68	58.15	59.23	59.50