Using PitchFX Data to Predict Player Performance

Aidan Boyd
University of Notre Dame
Notre Dame, IN
aboyd3@nd.edu

Samuel Grieggs
University of Notre Dame
Notre Dame, IN
sgrieggs@nd.edu

Ryan Karl
University of Notre Dame
Notre Dame, IN
rkarl@nd.edu

ABSTRACT

In 2003 Michael Lewis's book *Moneyball: The Art of Winning an Unfair Game* [15] revolutionized the sport of baseball by revealing to the world how the Oakland Athletics baseball team leveraged statistical analysis to build a competitive team despite having the 3rd lowest team salary in the league. For example, by re-evaluating their strategy in this way, the 2002 Athletics, with approximately $44 million in salary, were competitive with larger market teams, such as the New York Yankees, who spent over $125 million in payroll that season. Since then, there has been a baseball analytics revolution [2]. In 2006, the MLB installed stereoscopic cameras into every stadium and started using them to record flight path characteristics for every pitch. However, there is also a good deal of randomness in the outcomes of individual pitches [10, 28]. For this project, we investigated how techniques from data science, specifically linear regression, SVM-based regression, AdaBoost regression, and Neural Network regression can be used to build a predictive model to analyze pitchers' performance. After performing a comprehensive evaluation of these methods, we found that our approach provides good predictions for player statistics in the current season, and also can accurately predict future performance.

KEYWORDS

Data Sets, Baseballs, Neural Networks

ACM Reference Format:
Aidan Boyd, Samuel Grieggs, and Ryan Karl. 2020. Using PitchFX Data to Predict Player Performance. In . ACM, New York, NY, USA, 9 pages. 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Over the past 15 years, baseball player recruitment strategy has witnessed a paradigm shift, that has moved away from relying on the skills of scouts (former players or coaches who have decades of experience in playing the game) to find and evaluate players. Instead, modern teams utilize Sabermetrics, a unique combination of techniques from data science and statistics, to analyze what qualities in a player can offer the team the highest chance of a winning season at the lowest monetary cost.

The success of these methods has been repeated in other sports. For instance, Sabermetric models have been successfully applied to European club Soccer as recently as the 2018-19 UEFA Champions League, and Liverpool F.C. owner John W. Henry has publicly credited a model developed by Cambridge physicist Ian Graham, as a major part of the team's successful strategy when selecting what players to hire for Liverpool to win the 2018-19 UEFA Champions League Cup. [17] In this project, we intend to utilize various data science methods to build an ensemble method to predict a baseball players future performance statistics (also sometimes referred to as metrics), such as a player's expected Earned Run Average (ERA), for the upcoming season based on the physical data collected from the PitchFX system's stereoscopic cameras (such as ball speed, ball spin angle, etc.). We are interested in building a regression model, because the majority of our data is numeric (i.e. one player in the dataset has an ERA of 4.12 for the 2017 season), so this process will involve minimizing the error (MSE, etc.) between the model's outputted expected stat and the ground truth data. We will evaluate our method on several canonical datasets and offer predictions for the previous season which we can compare to the ground truth data to determine the usefulness of these methods and our ensemble overall.

2 RELATED WORK

Since the publication of *Moneyball* [15], data analytics research has flourished within the sports domain. A variety of surveys cover the history of modern sports analytics [21] and noteworthy state-of the art results in the field as a whole [5]. A useful source describing common problems that have well developed solutions (by utilizing linear methods such as linear regression, naive Support Vector Machines, or similar variants) and/or are the subject of active research (such as measuring a player's value to the team, ranking teams, and predicting game scores) is Miller et al. [18]. Currently, the sports of Soccer (also called Football) and Baseball receive the most attention from sports analytics researchers, and many comprehensive surveys of recent results for each sport exist [12, 26]. Within the baseball domain, groundbreaking work was done by Bill James in the 1980s that demonstrated that slugging and on-base percentages are better indicators of a players' success than traditional metrics such as speed and hit percentage, and historically do not require paying

players as high of a salary [7] . However, James' work was conducted several decades before many modern data science methods had been developed, and focused on hitters, rather than pitchers. Similarly, work by Tango et al. [30] improved on James results, by devising new metrics to measure players' ability, and developing distributions to aid in evaluation. While they devoted some analysis to pitchers, their main focus was on developing strategies for drafting an entire team, and as a result their analysis of pitcher performance is somewhat limited (only linear regression methods were considered).

Notably, there has been an interest in leveraging Markov Chains and Bayesian models to predict the performance of individual hitters and teams as a whole, [1, 4, 31], but these techniques have not been adapted to predict pitcher performance. With all of this previous work, only the existing statistics on player performance in previous seasons were used to train models (such as a player's ERA for earlier seasons). Our approach differs significantly, because (1) we make use of nonlinear methods (neural networks), and (2) utilize physical and spatial data collected using the PitchFX systems to make our predictions. This allows our system to train on data whose features potentially contain far more knowledge about the pitchers' abilities.To the best of our knowledge, the only existing work that applied relatively modern data science methods specifically to analyzing pitcher performance was Kim et al. [10]. While their work was notable for developing several new metrics to compare pitcher performance and analyze starting pitcher ability, the size of their data set was limited, as they only focused on Korean players. Also, their work primarily utilized arithmetic mean, weighted average and principal component analysis with linear regression to analyze models, and we expect that their results could be further improved upon if more complex methods were applied that have historically been more successful than these basic methods, such as Neural Networks, AdaBoost etc..

Only recently has the StatCast dataset been used in formal sports analytics studies [13]. Note, some previous work has used the dataset to to test new visualization techniques [11, 22], and predict the type of pitch thrown based on its physical characteristics [9], but these tasks are orthogonal to our own goal of predicting future pitcher performance metrics. Our approach differs significantly from the above mentioned studies, because (1) we make use of nonlinear methods (neural networks) combined with methods used successfully in the past (linear regression) when building an ensemble method, and (2) we utilize physical and spatial data collected using the PitchFX system to make our predictions, instead of simply using previous seasons' stats as was done in previous studies. This allows our system to more effectively predict future player performance, in spite of the complex, at times nonlinear relationships between the data and the players' performance statistics, and to train on new data whose features potentially contain far more knowledge about the pitchers' abilities.

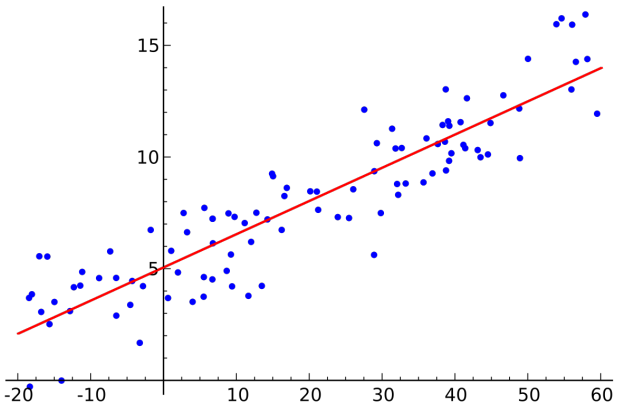

Figure 1: Linear Regression Example

3 SOLUTION/METHOD

To solve the problem of building an accurate method to predict pitcher performance, we have focused on four main approaches: linear regression, SVM-based regression, the ensemble regression method AdaBoost, and techniques based on Neural Networks, which we briefly summarize below.

Linear Regression

This concept was discussed in detail during class, so we provide only a brief overview of the technique. Linear regression is a technique for finding the relationship between multiple continuous variables in a dataset. For example, in two dimensions, by assuming that at least one variable is independent and the other(s) are dependent variables, this method models a statistical relationship between these variables by constructing a line that best fits the data. This line can be used to estimate the overall prediction error between all data points, and tries to minimize the distance between the points and the regression line. This concept can be easily extended to multidimensional space. An example of linear regression is shown in Figure 1.

SVM-based Regression

Again, this concept was discussed in detail during class, so we provide only a brief overview. SVM-based regression involves finding a boundary (i.e. a hyperplane in a multidimensional space) that passes between clusters of data to, and separates them into classes. If a data point is viewed as an n-dimensional vector, we would separate such points with a (n-1)-dimensional hyperplane, and try to find a boundary that correctly classifies the training data, and also maximizes

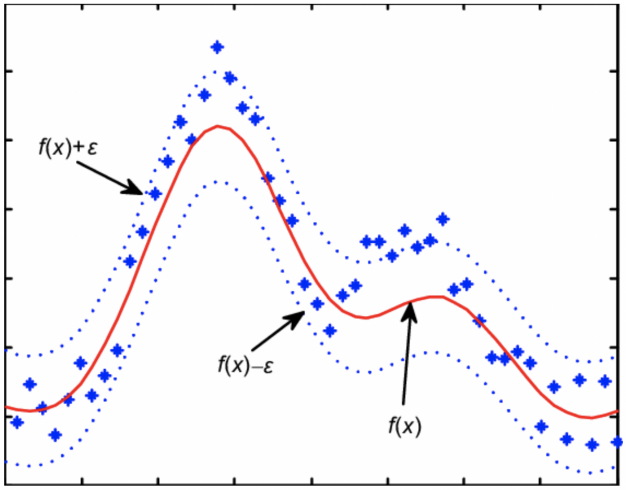

Figure 2: SVM Regression Example

Input :
- A training set $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m))$.

Initialization :
- Maximum number of iterations T;
- initialize the weight distribution $\forall i \in \{1, \ldots, m\}, D^{(1)}(i) = \frac{1}{m}$.

for $t = 1, \ldots, T$ **do**
- Learn a classifier $f_t : \mathbb{R}^d \to \{-1, +1\}$ using distribution $D^{(t)}$
- Set $\epsilon_t = \sum_{i: f_t(\mathbf{x}_i) \neq y_i} D^{(t)}(i)$
- Choose $a_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
- Update the weight distribution over examples

$$\forall i \in \{1, \ldots, m\}, D^{(t+1)}(i) = \frac{D^{(t)}(i) e^{-a_t y_i f_t(\mathbf{x}_i)}}{Z^{(t)}}$$

where $Z^{(t)} = \sum_{i=1}^{m} D^{(t)}(i) e^{-a_t y_i f_t(\mathbf{x}_i)}$ is a normalization factor such that $D^{(t+1)}$ remains a distribution.

Output : The voted classifier $\forall \mathbf{x}, F(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} a_t f_t(\mathbf{x})\right)$

Figure 3: The AdaBoost Algorithm

distance to the points closest to it (i.e. the margin). Note that by using a technique called the kernel trick, we can map inputs into a higher-dimensional feature space to further improve performance. An example of SVM-based regression is shown in Figure 2.

AdaBoost Regression

AdaBoost is a popular boosting technique which combines multiple "weak regressors" into a single "strong regressor". A weak regressor is simply a regressor with poor performance, but at a minimum is better than random guessing. AdaBoost can be applied to any regression algorithm that builds on top of existing regressors to improve overall performance.

Instead of training several weak regressors and combining the results, AdaBoost quantifies the best training set to choose for each new regressor that is trained based on the results of the previous regressor. Specifically, determines how much weight should be given to each regressor's answers when combining all of the results. Each weak regressor is be trained on a random subset of the total training set, and afterwords each is assigned a "weight" to each training example, to determine the probability that an example should appear in the next training set. The higher the assigned weight, the greater the likelihood it will be included in the training set. Following training, AdaBoost increases the weight on the incorrect examples so they will make up a larger part of the following regressor's training set, to increase the chances that the next regressor will perform better on them. The regressors are trained one at a time, and after each regressor is trained, the probabilities of each of the training examples appearing in the training set is updated for the next regressor.

The weight is typically assigned to grow exponentially as the error approaches 0, so better regressors are given exponentially more weight. Similarly, the regressor weight grows exponentially negative as the error approaches 1. Note that AdaBoost also incorporates the regressor's effectiveness into consideration when updating the weights, so if a weak regressor is mistaken on an input, we don't take that as seriously as a strong regressor's mistake. A more formal description of the AdaBoost algorithm is shown in Figure 3, and more discussion can be found in Section 5.

Neural Networks

A Neural Network (NN), is a type of machine learning model that draws inspiration from a biological neural network, often exemplified by the human brain. The Neural Network itself is not an algorithm but a kind of structure to combine many machine learning algorithms together and deal with complex and large inputs. In contrast to the structure of traditional machine learning algorithms, which have an input layer and output layer, Neural Networks add hidden layer(s) of neurons into the frame which greatly improve the accuracy. The structure of a Neural Network is shown in Figure 4.

Many parameters besides the data affect the behavior of the Neural Network. These are called hyperparameters, to distinguish them from ordinary parameters of the model. One of the most important of these is the learning rate, which determines the speed at which the Neural Network learns from its input. If this parameter is set too high, then the Neural Network's weights and thresholds can diverge. In the opposite case, the network can fail to learn quickly enough to evolve significantly based on its given training data. As our group learned the hard way, this can lead to problems ranging from a total lack of meaningful output to programs

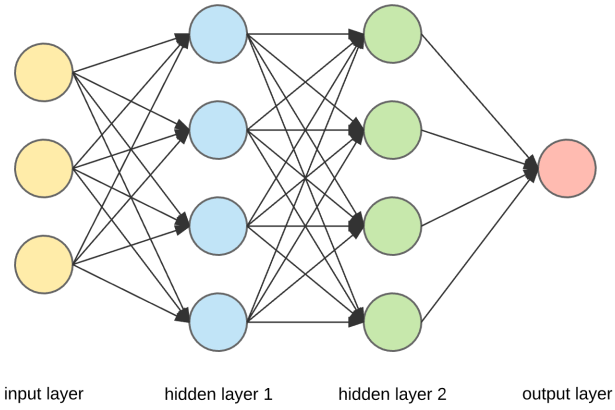

Figure 4: Neural Network Structure

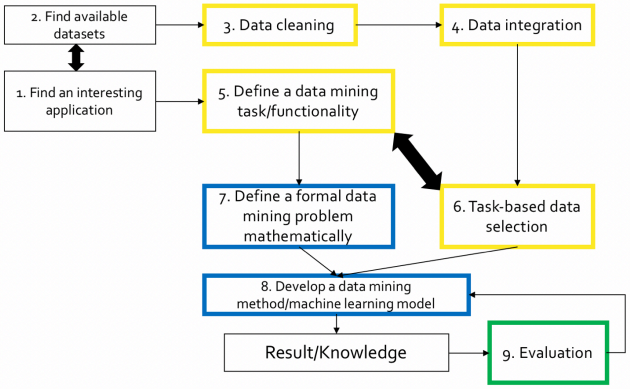

Figure 5: The Nine Components of Data Science

crashing due to numerical overflow. A similar hyperparameter is the momentum, which prevents the network from targeting a local minimum or maximum (as opposed to global minimums/maximums).

Another important hyperparameter is the choice of activation function. The step function described above is the one most analogous to biological neurons. However, other functions may be more suited to a particular type of work. Most activation functions will be (at least stepwise) continuous, and will have output in an interval of $[0, 1]$ or $[-1, 1]$.

4 PROBLEM DEFINITION

More formally, we built a regression model over data collected from the Fangraphs and MLB Advanced Media Statcast databases. Each data object will consist of one player's pitchFX data for one season. The attributes will consist of statistical metric scores that summarize the flight path data of all of the pitches a particular player threw in a particular season. The label for each data set will be statistics that measure the player's performance such as ERA, xFIP etc.. A full example of a data object would be Clayton Kershaw's 2016 season. We would use the mean, minimum, maximum, and standard deviation of 13 flight path measures for 10 different types of pitches. After training our models, our ideal functionality is that we can input an unlabeled test data object and our models will attempt to predict performance statistics for the season. We also trained the model to attempt to predict a player's performance for the NEXT season. This gives it utility as a tool to identify players that could potentially become much better in the future, and to identify players who have gotten lucky and over performed their skill level. More details about the data objects are included in the *Data Integration* subsection within Section 5.

5 METHODOLOGY

Below, we summarize our methodology in the context of the nine components of data science research shown in Figure 5. Please note that all of the code for each of these steps can be found at https://github.com/RyanKarl/PitchFX_Classifier.

Find an Interesting Application. We chose to apply data science techniques to predicting pitcher performance, by using regression to model pitchers' future success. Statistics have been successfully used in the past to help team managers decide what batters to draft to form a competitive team for the lowest price, and we are interested in learning if similar methods can be applied to analyze pitchers' performance.

Find Available Datasets. There are several canonical datasets associated with Sabermetrics research, that have been collecting data on team and player performance for over a decade. We primarily rely on two datasets when building our model: Fangraphs and MLB Advanced Media Statcast (Statcast). More information about each dataset is included below:

Fangraphs [16] contains a wealth of statistics on minor and major league players starting from the 2006 baseball season. In total over 1,000 players' stats are collected in the database. Importantly, this dataset tracks how far players advanced from the Minor Leagues to the Major Leagues. However, there are over 40 different statistics listed for every player, and so as part of our cleaning process we chose a subset of this dataset to avoid having too much noise during our calculations.

Statcast [19] is a state-of-the-art tracking technology that allows for the collection and analysis of a massive amount of baseball data, by using pitch tracking hardware installed in each Major League stadium. Statcast is a combination of two different tracking systems – a Trackman Doppler radar and high definition 20,000 frames per second Chyron Hego cameras, that record pitch speed, ball spin rate, pitch movement

angle, exit velocity, among a total of 28 different statistics about each player. In the first three seasons of Statcast (2015-17), over 2.1 million pitches and nearly 400,000 balls in play were tracked. Statcast currently reports measurements (raw numbers from the on-field action) and metrics (combinations of raw measurements into useful numbers). [19]

Data Cleaning. The datasets discussed above contain data with very high dimensionality, and nontrivial amounts of duplicate/noisy data. Fortunately, the python package Pybaseball [14] provides a number of scripts and commands to download the datasets into a Pandas [19] dataframe, remove any duplicate or noisy data, and perform basic calculations that can be displayed in charts using Matplotlib. Pybaseball is a very useful tool because it automates most of the Visualization, Cleaning and Integration process. Users simply specify in the command line how they would like data to be cleaned/displayed, and the package will produce the required results automatically (such as what data to remove from analysis, what method to use to normalize the data i.e. Z-Score, Min-Max etc, or what chart to use to display the data i.e. Bar Chart, Box Plot, etc.). However, in order to format our data in a manner amenable to our deep learning based and AdaBoost approaches, we did write a simple script to convert how the data is structured. The script essentially performs text manipulation on .csv files to format attribute columns in a way that preserves the relationship between the data objects, their attributes, and their labels, but formats their indices so we can use tools from sklearn [24] to train our models. There were also some issues in which the various datasets interfaced by Pybaseball would have players listed with slightly different names. For instance, New York Mets pitcher Zack Wheeler is incorrectly named Zach Wheeler in the Statcast playerid lookup table. In order to rectify this issue, we added manual corrections as the issues occurred. Unfortunately, much of the data of interest did not start getting collected until the 2015 season, so our experiments were run using data from 2015-2019. This leaves us with 1958 Pitcher Seasons.

Data Integration. To integrate the datasets, we again wrote a python script to perform text manipulation and remove duplicates or otherwise noisy data. After receiving .csv files as input, our script combines the data into two .CSV files that we can operate over easily using Pandas. We use the player's name and the year as a primary key to link the pitchFX data with the feature that we are trying to predict.

Each data object consisted of a player's season. That is, a measure of the distribution of all the pitches that they had thrown. The pitchFX system automatically classifies each pitch into one of thirteen pitch types: Slider, Four-seam Fastball, Curveball, Two-seam Fastball, Changeup, Cut Fastball, Knuckle Curve, Sinker, Pitch out, Fastball, Eephus, Screwball, and knuckleball. Eephuses and Screwballs are extremely rare, so we did not include them. Additionally, Knuckleball pitchers are rare and an somewhat of a statistical anomaly, since most of them only throw knuckleballs, and the pitch has a much more random and less controllable movement, so we removed all of the pitchers that threw a knuckleball from our dataset. For each type of pitch we collected: Release Speed, which is the velocity of the baseball towards the plate at the point of release, Release position X, and Z, which are coordinates of the pitch's release point, vertical and horizontal break (listed as pfx_x and pfx_z), the balls fight velocity and acceleration in the X, Y, and Z planes, effective speed, which is supposed to simulate what a radar gun would pick up, and spin rate. Now, these features are measured for each individual pitch thrown, so we take the mean, standard deviation, max, and min for all of the pitches thrown of a particular type. This gives us 52 features per pitch. As previously stated, we are looking at 10 types of pitches, so that leaves us with 520 features.

Define a Data Mining Task. Our data mining task was to build a regression model over data collected from the Fangraphs and MLB Advanced Media Statcast databases, to correctly predict how a player performs, and later predict player performance in future seasons. For linear regression, we used a simple ordinary least squares linear regression approach. For SVM-based regression, we use the basic extension from the Vapnik's SVM regressor described in more detail in the tutorial [29]. For our AdaBoost based approach, we used the Multi-class AdaBoost algorithm [6] that combines several weak classifiers with accuracies slightly over 50% into a single strong classifier. For our deep learning based approach, we trained a multilayer perceptron [23] with 3 layers and 676, 5, and 1 node respectively, to build a regression model for pitchers' performance.

Task Based Data Selection. After this, we selected the attributes from our cleaned and integrated dataset that we felt were the strongest indicators of a pitcher's overall ability. The metrics that we wish to predict are Earned Run Average (ERA) which is the total number of runs given up over a 9-inning game, xFielding Independent Pitching (xFIP) which is intended to estimate a pitcher's ERA with an average defense, Strikouts per 9 innings pitched(k/9), Hits allowed per 9 innings pitched (H/9), Batting Average against (BA), Batting Average on Balls in Play against (BABIP), which is generally assumed to be outside a pitcher's control, and ground ball percentage (GB%).

Define a Formal Data Mining Problem Mathematically. Here we mathematically define the frameworks we use; some notes are inspired by [8]. For linear regression, in a two

dimensional space we would solve $y = mx + b$ for m and b to estimate the relationship between x and y after minimizing error (this can be easily extended to multidimensional space). SVM-based regression can be summarized as attempting to find the $min(\frac{w^T w}{2})$ such that $y_i(wx_i + b) \geq 1$ for any (x_i, y_i). For AdaBoost (the full algorithm for a binary label framework is shown in Figure 3.) we take a training set containing m samples where all x inputs are an element of the total set X, such that (x_1, y_1) is the first training sample and (x_m, y_m) is the m-th training sample and where y outputs are an element of a set comprising of only two values, -1 and 1, we initialize all weights of the samples to 1 divided by the number of training samples, i.e. set $D_1(i) = 1/m$ for $i = 1, \ldots, m$ where D is the weights of samples and i is the i-th training sample. After this, for $t = 1$ to T classifiers, we fit it to the training data and select the classifier with the lowest weighted error ϵ. More formally, we train weak learner using distribution D_t, calculate the weak hypothesis $h_t : X \in -1, +1$, select h_t with low weighted error, and calculate the minimum error for the model $\epsilon = Pr_{i,D_t}[h_t(x_i) \neq y_i]$. Then we choose $\alpha_t = 1/2 * ln(1 - \epsilon/\epsilon)$ where α is the weight for the classifier, and update, for $i = 1, \ldots, m$, $D_{t+1}(i) = Dt(i)e^{(-\alpha_t * y_i * h_t(x_i)}/Z_t$ where Z_t is the normalization factor. Note that we can calculate ϵ by calculating the sum of the error rate, where weight for training sample i and y_i not being equal to our prediction h_j (which equals 1 if there is an error and 0 if it is correct). The formula to calculate this is is $\epsilon = \sum_{n=1}^{\infty} w_i I(y_i \neq h_j(x_i))/\sum_{n=1}^{\infty} w_i$.

Our Deep Learning model contains many processing nodes that are interconnected into layers. These nodes are fed data that moves in one direction through the network. Each node will receive data from several nodes connected to it in the previous layer and send data to several nodes that it connects to in the following layer. A node assigns a weight (a numerical value) to each connection from the previous layer, and each time it receives a piece of data along this connection, it multiplies it by the corresponding weight. After this, it adds all of the products generated together into an output value. If the value computed is less than a specified threshold then the node does not transmit any data to the following layer, but if the value is greater than the threshold, the node transmits the value to all of the nodes it connects to in the following layer (other activation functions that determine how the node transmits information exist, but the step function described here is the biologically inspired, intuitive model). Note that we used the ReLU activation function with the Adam optimizer and a learning rate of 0.01. Between the first input layer and the final output layer, there are typically many intermediate layers called "hidden layers", which allow the system to better tune its final output. Generally speaking, Neural Networks learn better and become more accurate as the number of hidden layers increases [20].

When a Neural Network is being trained, all of its weights and thresholds are initially set to random values. Training data is fed to the bottom layer (the input layer) and it passes through the succeeding layers, until it finally arrives, radically transformed, at the output layer. During training, the weights and thresholds are continually adjusted until training data with the same labels consistently yield similar outputs. Neural Networks learn via backpropagation, a feedback process where the output of the final layer of a Neural Network is compared against the output it was supposed to produce. By analyzing the difference between the two values, the network can modify the weights of each connection between the nodes by traversing the network topologically from the final output layer to the input layer [27]. The goal of this process is to reduce the difference between the two until they are almost identical, so the network responds accurately when posed a problem [25]. The equation to calculate the error δ in the output layer L from the j activation function is $\delta_j^L = \frac{\partial C}{\partial a_{jL}} \sigma'(z_j^L)$. where C is the cost, a is the response, σ' is the activation function, and z is the weighted input. The equation to calculate the error δ^L in terms of the error in the next layer δ^{L+1}, is $\delta^l = ((w^{l+1})^T \delta^{l+1}) \times \sigma'(z^l)$, where T denotes the transpose. The equation to calculate the rate of change of the cost with respect to any bias in the network is $\frac{\partial C}{\partial b_{lj}} = \delta_l^j$. The equation to calculate the rate of change of the cost with respect to any weight in the network is $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l1} \delta_l^j$.

Develop a Data Mining Model. Our data mining model takes in pitch data (ball trajectory, etc.) for a pitcher over the course of the season and can predicts their performance statistics (ERA, etc.) from that season. Additionally we will train a model to predict the next season's performance as well. This would give a suite of metrics that could be used for player evaluation independent of many of the factors outside of the pitcher's control like defense or luck. To do this, we took advantage of an ensemble of data science techniques and evaluated them on a case by case basis to find the best type of model for each statistic and type of pitch.

Evaluation. To evaluate our methods, we computed the Mean Absolute Error, Mean Squared Error and R^2 (also called the coefficient of determination) values for each technique. The error measures are straightforward (covered in class), but the coefficient of determination is a more complex measurement, that calculates the proportion of the variance in the dependent variable that is predictable from the independent variable(s). Normally, it ranges from 0 to 1, where an R^2 of 0 means that the dependent variable cannot be predicted

from the independent variable, and an R^2 of 1 means the dependent variable can be predicted without error from the independent variable (i.e. the closer R^2 is to 1 the better). However, if a nonlinear function is used to fit the data, negative values can arise when the mean of the data provides a better fit to the outcomes than do the fitted function values [3].

6 EXPERIMENTS

To train and test our approach, it was decided to use disjoint data. As previously described there are two settings for our experimentation. The first setting is when we take a season's data and try to predict the labels for that same season. The second setting is when we take a season's data to predict the following season's labels, effectively attempting to predict future season performance based on the current season data. The data we used was the 2015 to the 2019 season data. This gave us five years worth of data and labels. In the first setting we took the all of the 2015 - 2018 data and their associated labels as the training data and then we used the 2019 data and labels as the test data. For the second setting, we took the 2015 data and assigned it the 2016 labels, the 2016 data was assigned the 2017 labels, and the 2017 data was assigned the 2018 labels. Because we don't have the 2020 labels, the test data that we used was the 2018 data with the 2019 data. We assumed that each player and their performance for that season were independent, and as such in both settings the test set is subject disjoint to the training set.

To generate models to effectively predict the player statistics outlined above in section 5, we decided to model each pitch type independently. The reasoning for this is that we are looking at 10 different pitch types; however, standard pitchers generally only throw three or four of these pitch types. If we modeled all pitch types together this feature vector would be very sparse as the pitches that weren't thrown would be populated with zeros. Modeling these pitches independently meant this sparsity was not an issue. As well as modeling each pitch type independently, each label we wish to predict is also modeled independently ERA pitch type models would be separate from xFIP models.

The models that we employed for this work all perform differently under various conditions, as such we did not pick one best global model. Instead, each independent pitch type has it's own best method. That means one pitch model may be the SVM Regressor whereas another would be AdaBoost. The three metrics we investigated to determine the performance of the models were the Mean Squared Error (MSE), Mean Absolute Error (MAE) and the R^2 measure.

To ensure that the best model was selected, we ran the training and validation on 10 different subsplits of the data. For each split 67% was used for training and the remaining 33% was used for validation. Note the training and validation are not subject disjoint as these are just splits within the corpus of training data. This mimicked cross-validation, however, as will be seen, we could not use cross-validation due to feature selection.

To ensure that all features were on the same scale, we employed min-max scaling on all features independently for each pitch type to reduce the scale of each feature to between 0 and 1. This also ensured that when performing feature selection, the features with the largest range don't dominate the selection even though they may not be the most variant. For each split the min-max scaler was trained on the 67% and this was then applied to the 33%. Once the data was scaled, again using the scaled 67%, PCA was performed. We set the constraint that we limit the features for each pitch type independently to 10 features or 90% of the feature variance (whichever is less features). This reduced the number of features for each pitch from 52 to 10 or less. The PCA that was fit to the 67% was then applied to the validation. Then, for each method we evaluated the three metrics mentioned above when tested on the validation set. To decide what the best method was for each pitch type, a simple majority voting scheme was used where each of the three metrics are a vote. If one method performs best on any two of the three metrics it is deemed the best. If each metric results in a different best method, the method that performed best for MAE is selected as the tiebreaker. The result of this is 70 independent models for each setting (140 total), one for each of the 10 pitch types for each of the 7 labels we want to estimate.

An interesting thing to note here is that the best model for virtually all pitches was either the linear regressor, the SVM regressor or the AdaBoost regressor and the neural network was not the best for almost any pitch. We observed this phenomenon for all pitches when predicting all performance metrics, and include box plots for our results when calculating the MAE, MSE, and R^2 for all models when predicting ERA as an illustrative example in standard box plots shown in Figures 6-8, where each data point is the score (MAE, etc.) for a pitch (Note all such plots generally had a similar trend). Interestingly, Linear, SVM, and AdaBoost Regression share noticeable overlap, but the performance of the NN is considerably worse.

Our hypothesis is that there simply is not enough data or time spent crafting a good network for this data driven method to effectively learn patterns in the data. Once the best method was found for each pitch type for each metric we wanted to estimate, we retrained the best method using all of the training and validation data together. This to ensure we have the best model to test with.

For testing, given the previously trained models, if a pitcher only throws three or four of the pitch types, these models are the only ones employed in prediction. Each of the models

Metric	League Average	MAE (Present)	MAE(Future)	MSE (Present)	MSE (Future)	R^2 (Present)	R^2 (Future)
ERA	4.51	0.962	**0.933**	1.664	**1.547**	0.0135	**0.02877**
xFIP	4.51	**0.593**	0.5939	0.54968	**0.537**	0.07889	**0.1004**
K/9	6.17	**1.31**	1.38	**2.735**	2.9679	**0.19743**	0.173
H/9	8.451	1.1416	**1.0338**	2.1	**1.77**	0.02679	**0.0942**
BA	.252	0.0251	**0.0232**	0.000977	**0.00085**	0.0158	**0.07**
BABIP	.300	0.0242	**0.0214**	0.00099	**0.000716**	**0.001509**	-0.0566
GB%	.44	**0.0439**	0.045	**0.0029**	0.003	**0.2307**	0.16996

Table 1: Table outlining all average results. Best result for a given metric is bolded between present and future. Both settings are attempting to predict the same labels.

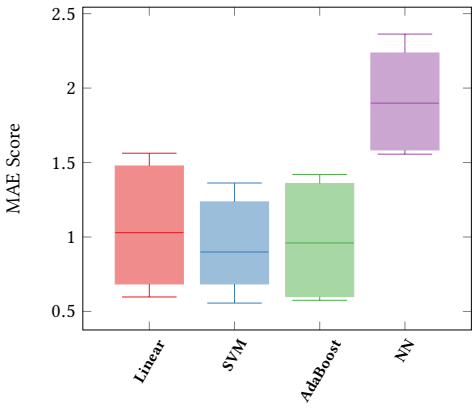

Figure 6: Comparison of average MAE scores for each method for ERA.

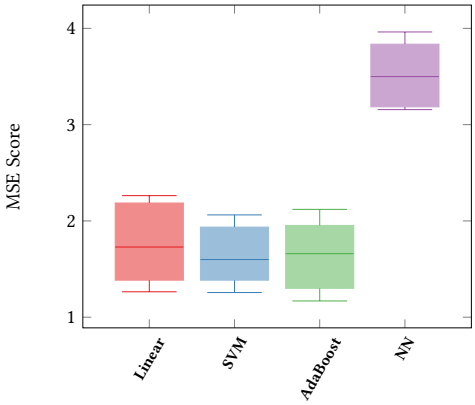

Figure 7: Comparison of average MSE scores for each method for ERA.

will output a score for the designated label (ERA, xFIP, GB% etc.) so to reduce this to a single score we take the mean of the outputs from all used models. The assumption here is that each pitch has an equal weight in the performance of a pitcher. This may not be true, however, without further investigation we cannot determine how much each pitch type should be weighted relative to the others. Currently, the methodology of weighting each pitch type equally means

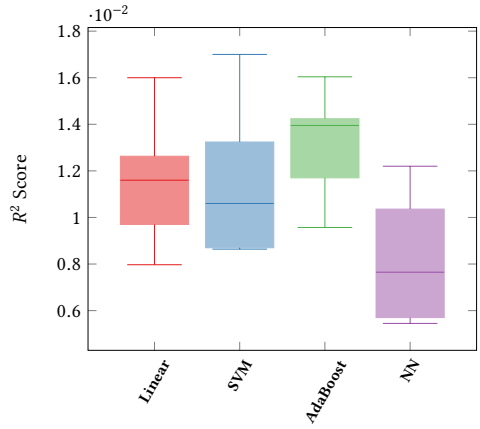

Figure 8: Comparison of average R^2 scores for each method for ERA.

pitchers are not rewarded or penalized for throwing specific pitches.

7 DISCUSSION

Table 1 outlines all results obtained by the models when trying to predict the 2019 labels. From these results there are multiple observations to be made. Firstly, in both settings, the relative MAE to the league average is pretty good. Our models predicted the players xFIP to a mean difference of 0.6, which is less than 15% error. For something like professional baseball where individual events are hard to predict, this is impressive.

Secondly, as can be seen in the table, the results for the present setting performs very similarly to the predicting the future setting. This was surprising as you would think current season data would be better at predicting current season labels, however, in our investigation taking this seasons data to predict next season's labels works just as well. This indicates that across seasons in the MLB, the average stats do not change monumentally. Individual pitchers may perform better or worse year in and year out, but globally the statistics are relatively constant. This was an interesting finding as it shows the predictability of baseball and how, given more time and effort than a semester project, valuable knowledge

can be mined from this data in terms of predicting player's future performance as well as current performance.

Finally, we found when investigating individual predictions that our models performed very well for average performing players. The methods we employed for regression are relatively simplistic and as such tended to under-fit the data. This meant average players were easier to predict than exceptionally good or bad players. In terms of outliers, our models underestimated the best players and overestimated the worst players. This is somewhat expected behavior and in terms of a league wide prediction algorithm is acceptable because most players fall around the average mark.

8 CONCLUSION AND FUTURE WORK

In conclusion, we find that our methodology is quite effective at predicting player performance from pitch flight paths. Perhaps more surprisingly, we were often even more effective at predicting the pitchers performance for the next season. This methodology offers astounding potential as a tool for player evaluation from the amateur to the professional level. The pitch flight data seems to be a very useful tool for predicting performance. More sophisticated models could evaluate all of the pitches thrown instead of just looking at the distribution. Furthermore, a recurrent model could be used with individual pitches being treated as time series data to better evaluate how the path of the pitches change over the course of games and the season overall. Finding weights for pitches to determine whether a pitch type represents good or bad players would also boost model performance.

REFERENCES

[1] Bruce Bukiet, Elliotte Rusty Harold, and José Luis Palacios. 1997. A Markov chain approach to baseball. *Operations Research* 45, 1 (1997), 14–23.

[2] Jason Chang and Joshua Zenilman. 2013. A study of sabermetrics in Major League Baseball: The impact of Moneyball on free agent salaries. (2013).

[3] Brian Everitt and Anders Skrondal. 2002. *The Cambridge dictionary of statistics*. Vol. 44. Cambridge University Press Cambridge.

[4] Rob Gray. 2002. Behavior of college baseball players in a virtual batting task. *Journal of Experimental Psychology: Human Perception and Performance* 28, 5 (2002), 1131.

[5] Maral Haghighat, Hamid Rastegari, and Nasim Nourafza. 2013. A review of data mining techniques for result prediction in sports. *Advances in Computer Science: an International Journal* 2, 5 (2013), 7–12.

[6] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. 2009. Multi-class adaboost. *Statistics and its Interface* 2, 3 (2009), 349–360.

[7] Bill James. 2003. The Historical Baseball Abstract, 4 e éd.

[8] Haebichan Jung. 2018. Adaboost: Breaking Down the Math (and its Equations) into Simple Terms. (2018).

[9] Lucas Kelly. 2019. Machine Learning Our Way to the Gold Glove Award.

[10] Hyeon-Gyu Kim and Jea-Young Lee. 2017. Suggestion of starting pitcher ability index in Korea baseball-Focusing on the sabermetrics statistics WAR. *The Korean Data & Information Science Society* 28, 4 (2017), 863–874.

[11] Joongsik Kim, Moonsoo Ra, Hongjun Lee, Jeyeon Kim, and Whoi-Yul Kim. 2019. Precise 3D baseball pitching trajectory estimation using multiple unsynchronized cameras. *IEEE Access* (2019).

[12] Kaan Koseler and Matthew Stephan. 2017. Machine learning applications in baseball: A systematic literature review. *Applied Artificial Intelligence* 31, 9-10 (2017), 745–763.

[13] Marcos Lage, Jorge Piazentin Ono, Daniel Cervone, Justin Chiang, Carlos Dietrich, and Claudio T Silva. 2016. StatCast Dashboard: Exploration of spatiotemporal baseball data. *IEEE computer graphics and applications* 36, 5 (2016), 28–37.

[14] James LeDoux. 2018. pybaseball. https://github.com/jldbc/pybaseball

[15] Michael Lewis. 2004. *Moneyball: The art of winning an unfair game*. WW Norton & Company.

[16] Mitchel Lichtman. 2019. FanGraphs Baseball Metrics. https://www.fangraphs.com

[17] Ana Paulina Maupome. 2019. Moneyball, Liverpool's reason behind Jürgen Klopp's hiring. https://en.as.com/en/2019/06/03/football/1559596265_282692.html

[18] Thomas W Miller. 2015. *Sports analytics and data science: winning the game with methods and models*. FT Press.

[19] LP MLB Advanced Media. 2019. StatCast. http://www.mlb.com/glossary/statcast

[20] Michael A Nielsen. 2015. *Neural networks and deep learning*. Vol. 25. Determination press USA.

[21] Bahadorreza Ofoghi, John Zeleznikow, Clare MacMahon, and Markus Raab. 2013. Data mining in elite sports: a review and a framework. *Measurement in Physical Education and Exercise Science* 17, 3 (2013), 171–186.

[22] Jorge P Ono, Carlos Dietrich, and Claudio T Silva. 2018. Baseball Timeline: Summarizing Baseball Plays Into a Static Visualization. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 491–501.

[23] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic Differentiation in PyTorch. In *NIPS Autodiff Workshop*.

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

[25] Tariq Rashid. 2016. *Make your own neural network*. CreateSpace Independent Publishing Platform.

[26] Robert Rein and Daniel Memmert. 2016. Big data and tactical analysis in elite soccer: future challenges and opportunities for sports science. *SpringerPlus* 5, 1 (2016), 1–13.

[27] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature* 323, 6088 (1986), 533.

[28] Huang-Chia Shih. 2017. A survey of content-aware video analysis for sports. *IEEE Transactions on Circuits and Systems for Video Technology* 28, 5 (2017), 1212–1231.

[29] Alex J Smola and Bernhard Schölkopf. 2004. A tutorial on support vector regression. *Statistics and computing* 14, 3 (2004), 199–222.

[30] Tom M Tango, Mitchel G Lichtman, and Andrew E Dolphin. 2007. *The book: Playing the percentages in baseball*. Potomac Books, Inc.

[31] Tae Young Yang and Tim Swartz. 2004. A two-stage Bayesian model for predicting winners in major league baseball. *Journal of Data Science* 2, 1 (2004), 61–73.