

Baseball Data Analysis for DraftKings

Major Qualifying Project

Advisors:

Donald Brown, Randy Paffenroth

Written By:

Ardavasd Ardhaldjian, Hailey Delphia, Kaustubh Pandit, & River Yan

Sponsor:

DraftKings



WPI

A Major Qualifying Project

Worcester Polytechnic Institute

Submitted to the Faculty of the Worcester Polytechnic Institute
in partial fulfillment of the requirements for the Degree of
Bachelor of Science in Data Science, and Electrical and
Computer Engineering. This report
represents the work of one or more WPI undergraduate students
submitted to the faculty as evidence of completion of a degree
requirement. WPI routinely publishes these reports on the web
without editorial or peer review.

August 2021 - April 2022

Abstract

In this Major Qualifying Project (MQP), we partnered with the online sports betting company DraftKings to develop a process that assisted in setting odds for live propositional bets. With a focus on baseball, and more specifically the outcome of an at-bat, the goal of our project was to predict the likelihood of an at-bat outcome before assigning odds for that event. Working with data retrieved from the MLB API, our finalized dataset contained 16 features with over 80 thousand rows of data. Utilizing this dataset and machine learning models such as a Random Forest Classifier (RFC), K Nearest Neighbors (KNN), and Decision Tree Classifier (DTC) we were able to run various tests on model accuracy, which were used as the foundation for generating actual odds for each play.

Executive Summary

Sports betting has been around for thousands of years and some of its earliest traces can be dated back to the Ancient Greeks, during the Olympic games (Milton, 2017). Gambling since then has grown as an industry in many cultures and societies. In today's age, gambling and sports betting have moved to mainly online platforms, with many Sportsbooks beginning to offer various types of gambling opportunities on their websites. Some of these options include fantasy leagues, money line bets, and propositional (prop) bets for individual sports. One of such online companies, DraftKings, is one of the fastest-growing Sportsbooks in the United States. They offer their services, partially or in full, in over 43 states to 8 million users (Tatevosian, 2021).

DraftKings profits off money lines and propositional bets by intentionally setting unfair odds for every betting opportunity. This is essentially applying a premium so that the bookmaker can generate income from each game. This "premium", also known as the hold percentage, in sports betting is commonly known as the juice or *Vigour* of sports betting (Miller, 2019). Applying a premium is generally an easy process once the odds are set for a game, however, setting realistic odds for a sports game can be difficult with all of the player factors at hand. It is not the same process as looking at a regular die and knowing that each side has a 1 in 6 chance of occurring. On a sports team, there are individual players' statistics at play, coaching strategy, weather conditions, and even player injuries that could affect the actual odds of a bet.

To explore the issue of setting odds, our team is using the MLP-API, a database of baseball data, to extrapolate which variables affect the outcome of an at-bat in baseball. The MLP-API gave us access to numerous individual baseball player statistics (Ex. Batting Average, Earned Run Average), season games, and at-bat outcomes. This dataset was used to generate a customized Pandas DataFrame that our team could apply various machine learning algorithms to predict at-bat odds, a practice that DraftKings is trying to develop.

For our project, we tested out various machine learning algorithms, such as K-Nearest Neighbors (KNN), Logistic Regression, Random Forest Classifier (RFC), and Decision Trees. The outcomes of these tests were compared to find the best model to predict realistic probabilities for an at-bat. These probabilities were then converted into usable odds after applying the DraftKings premium.

After rigorous testing, we discovered that using KNN and decision trees, two highly effective classification algorithms, were our best methods. The generated models used 16 features to predict the outcome of an at-bat. Given that 70% of all at-bats in the MLB result in an out, this statistic set our baseline model that we worked to improve on during this project. Our goal was to beat an algorithm that solely predicted out and was correct 70% of the time. In addition, we wanted to go from retrieving model accuracies to producing accurate odds. And in the end, we were able to generate these desired odds using the KNN model.

Table of Contents

Abstract	2
Executive Summary	3
Table of Contents	5
List of Figures	7
List of Tables	9
Authorship	10
Table of Acronyms	13
1. Introduction	14
2. Background	15
2.1 DraftKings, Sportsbook, and Online Betting	15
2.1.1 Propositional Bets	15
2.1.2 Setting Lines	16
2.2 Baseball	19
2.3 MLB API, Libraries and Our Data	21
2.3.1 Our Dataset and Features	23
2.4 Machine Learning Models	25
2.4.1 K-Nearest Neighbors	26
2.4.2 Neural Network	27
2.4.3 Decision Trees	29
2.4.4 Random Forest Classifier and Regressor	30
2.4.5 Support Vector Machines	31
2.4.6 Logistic Regression	33
2.5 Confusion Matrices	34
2.5.1 Testing Versus Training	36
2.6 Accuracy Metrics	38
2.6.1 F1-Score, Precision, & Recall	38
2.6.2 Accuracy Score	39
2.6.3 Root Mean Square Error, Mean Square Error, & Mean Absolute Error	39
2.6.4 Standard Deviation	39
3. Methodologies	40

3.1 Working With Data	40
3.1.1 Cleaning the Data	40
3.1.2 Balancing the Data	41
3.1.3 Feature Engineering	41
3.2 Random Forest Testing	42
3.2.1 Feature Importance (Correlation Diagram)	42
3.2.2 Best Random Forest Depth	43
3.3 DataRobot / Akkio (AutoML) Testing	46
4. Results	49
4.1 Why Unbalanced Data Didn't Work Initially	49
4.2 Balanced Random Forest Classifier	50
4.2.1 Best Random Forest Depth	50
4.2.2 Model Evaluation	52
4.2.3 Number of Features Versus Accuracy	56
4.3 Pseudo KNN Results for Lines	58
4.3.1 Scatterplots	59
4.3.1.1: OBP vs. ERA	60
4.3.1.2: Batting Average vs. ERA	61
4.3.1.3: Batting Average vs. Strike Percentage	62
4.3.1.4: OBP vs. Strike Percentage	63
4.3.2 Comparisons with DraftKings Lines	64
4.3.3 Principal Component Analysis (PCA)	67
4.4 Decision Tree Betting Odds Generator	69
4.4.1 Decision Tree Splits Diagram	69
4.4.2 Decision Tree Results	70
4.5 Pseudo KNN and Decision Tree Result Comparison	72
5. Conclusion	73
5.1 Takeaways	73
5.2 Future Work	74
References	74

Table of Figures

Figure 1: Live prop bet of the 20-5-2021 Red Sox versus Yankees game taken from the DraftKings website regarding the outcome for Gleyber Torres' first plate appearance in the second inning. (MLB Odds: Draft Kings, 2022)	16
Figure 2: Live prop bet of the 20-5-2021 Red Sox versus Yankees game taken from the DraftKings website regarding the outcome for Enrique Hernandez's first plate appearance in the first inning. (MLB Odds: Draft Kings, 2022)	16
Figure 3: Example of money line Odds on DraftKings (MLB Odds: Draft Kings, 2022)	17
Figure 4: Prop Bet DraftKings Example (MLB Odds: Draft Kings, 2022)	18
Figure 5: Prop Bet DraftKings Example (MLB Odds: Draft Kings, 2022)	19
Figure 6: MLB API Python Dictionaries (toddrob99, 2020)	21
Figure 7: KNN Model Diagram (Harrison, 2019)	27
Figure 8: Neural Network Diagram (IBM Cloud Education, 2020)	28
Figure 9: Breakdown of a decision tree into its various leaf nodes. (Hanafy et al., 2021)	29
Figure 10: Process of voting in a Random Forest (Chakure, 2020)	30
Figure 11: Visualization of what a hyperplane looks like in relation to the given data depending on the dimensional space it is in (Gandhi, 2018)	31
Figure 12: Graphs of what a set of possible hyperplanes on data (left) can look like. These lines are drawn between two selected points with an equal distance apart from the actual points. The right graph shows what an optimal hyperplane would look like with perfect data (Gandhi, 2018).	32
Figure 13: The difference between what a small margin, versus a large margin is referring to in a SVM hyperplane (Gandhi, 2018)	33
Figure 14: Logistic Function Graph Diagram (Chávez, 2019)	34
Figure 15: Confusion Matrix Diagram (Narkhede, 2018)	35
Figure 16: Confusion Matrix Example (Kulkarni, 2020)	36
Figure 17: Training and Testing Confusion Matrices	37
Figure 18: Features Correlation Heatmap	42
Figure 19: Result Feature vs. Rest of Features Correlation Plot	43
Figure 20: Testing and Training Accuracies of Random Forest Depths	44
Figure 21: Training and Testing Confusion Matrices of Gini Criterion	45
Figure 22: Training and Testing Confusion Matrices of Entropy Criterion	45
Figure 23: Results of the best model type and the resulting prediction quality on an unbalanced dataset using Akkio	47
Figure 24: Results of the best model type and the resulting prediction quality on a balanced dataset using Akkio	48
Figure 25: Importance of features in comparison to the target variable as per DataRobot	48

Figure 26: Data Quality assessment of our dataset as per DataRobot	49
Figure 27: Testing vs. Training Accuracies of Random Forest Depths.....	51
Figure 28: Left - Balanced Model, Right - Unbalanced model	52
Figure 29: Figure 29a (Left) and Figure 29b (Right): This figure displays both the testing and training confusion matrices for our dataset when we increased the number of features from 4 to 16. The training matrix showcases the balanced nature of the data and its initial predictions, and the right testing matrix shows the predictions on the 20% of the data allocated for testing purposes after the model had learned from the 80% of the data on the left.	53
Figure 30: This heatmap shows the correlations between the various features within our dataset.	54
Figure 31: This figure shows the testing confusion matrix of a Random Forest Classifier utilizing 10 features with the highest correlations. A max depth of 10 was used with an 80/20 train/test split, and all correlations were greater than 3%.	54
Figure 32: This figure shows the testing confusion matrix of a Random Forest Classifier utilizing the Gini criterion, 100 trees with a max depth of 11, and all 16 of the dataset's features.	55
Figure 33: This figure shows the testing confusion matrix of a Random Forest Classifier utilizing the Gini criterion, 100 trees with a max depth of 5, and bat side and pitch hand as features.	56
Figure 34: This figure shows the testing confusion matrix of a Random Forest Classifier utilizing the Gini criterion, 100 trees with a max depth of 5, and bat side, pitch hand, batting average, and ERA as features.	57
Figure 35: This figure shows the testing confusion matrix of a Random Forest Classifier utilizing the Gini criterion, 100 trees with a max depth of 5. The features include atBats, batting avg, obp, bat side, ops games Played, slg, babip, atBats, era, strike Percentage, pitch hand, strikeoutsPer9Inn, runsScoredPer9, hitsPer9Inn, and walksPer9Inn.	58
Figure 36: Screenshot of Our Dataset	59
Figure 37: Screenshot of Our Normalized Dataset	60
Figure 38: OBP vs. ERA Scatterplot	61
Figure 39: Batting Average vs. ERA Scatterplot.....	62
Figure 40: Batting Average vs. Strike Percentage Scatterplot.....	63
Figure 41: OBP vs. Strike Percentage Scatterplot	64
Figure 42: Money Line for Enrique Hernandez vs. Gerrit Cole (Featured betting odds & lines: DraftKings Sportsbook, n.d.)	65
Figure 43: Money Line for Enrique Hernandez vs. Gerrit Cole (Featured betting odds & lines: DraftKings Sportsbook, n.d.)	66
Figure 44: PCA Scatterplot.....	68
Figure 45: PCA DataFrame	69
Figure 46: Decision Tree Split Visualization.....	69
Figure 47: Decision Tree Splits	70
Figure 48: DraftKings Generated Odds	70

Figure 49: Decision Tree Splits 71
Figure 50: DraftKings Odds Comparison (Rosario vs. Diekman)..... 72
Figure 51: DraftKings Odds Comparison (Riley vs. Guerra) 72
Figure 52: DraftKings Odds Comparison (Chio vs. Watkins)..... 72

Table of Tables

Table 1: Possible Outcomes for Out and Not Out	21
Table 2: Possible Outcomes of Out and Not Out.....	23
Table 3: Definition of Our Features from the Dataset	25
Table 4: Confusion Matrix Values Table.....	35
Table 5: Accuracy of Training and Testing Data.....	37
Table 6: Feature Testing Tuning.....	46
Table 7: Accuracy Comparison of Different Depths	51
Table 8: Created Money Lines for Different Hold Percentages	66
Table 9: Created Money Lines for Different Hold Percentages	67
Table 10: All Model Comparison with DraftKings on Generated Betting Odds	72

Authorship

Abstract	Hailey
Executive Summary	Ardavasd
Introduction	River
Propositional Bets (Section 2.1.1)	Hailey, River, Ardavasd
Setting Lines (Section 2.1.2)	Kaustubh, River, Ardavasd
Baseball (Section 2.2)	Hailey
MLB API, Libraries, & Our Data (Section 2.3)	River, Ardavasd
Machine Learning Models (Section 2.4)	Kaustubh
K-Nearest Neighbors (Section 2.4.1)	River
Neural Network (Section 2.4.2)	River
Decision Trees (Section 2.4.3)	Kaustubh
Random Forest Classifier (Section 2.4.4)	Hailey, Kaustubh
Support Vector Machines (Section 2.4.5)	Hailey
Logistic Regression (Section 2.4.6)	River
Confusion Matrices (Section 2.5)	Ardavasd
Accuracy Metrics (Section 2.6)	Kaustubh
Working with Data (Section 3.1)	Hailey
Cleaning the Data (Section 3.1.1)	Hailey
Balancing the Data (Section 3.1.2)	Hailey
Feature Engineering (Section 3.1.3)	River
Random Forest Testing (Section 3.2)	Ardavasd, River
Feature Importance (Section 3.2.1)	River
Best Random Forest Depth (Section 3.2.2)	Ardavasd
DataRobot / Akkio (Section 3.3)	Hailey, Kaustubh

Why Unbalanced Data Didn't Work (Section 4.1)	River
Balanced Random Forest Classifier (Section 4.2)	Ardavasd
Best Random Forest Depth (Section 4.2.1)	Ardavasd
Model Evaluation (Section 4.2.2)	Hailey
Number of Features Versus Accuracy (Section 4.2.3)	Kaustubh
Pseudo KNN Results for Lines (Section 4.3)	River
Decision Tree Betting Odds Generator (Section 4.4)	Ardavasd & River
Psuedo KNN and Decision Tree Result Comparison (Section 4.5)	Ardavasd & River
Conclusion (Section 5)	Kaustubh

Table of Acronyms

Word/Phrase	Acronym
Major Qualifying Project	MQP
Worcester Polytechnic Institute	WPI
Major League Baseball	MLB
Application Programming Interface	API
Random Forest Classifier	RFC
Random Forest Regressor	RFR
K-Nearest Neighbors	KNN
Support Vector Machine	SVC
Root Mean Squared Error	RMSE
Mean Squared Error	MSE
Mean Absolute Error	MAE
Standard Deviation	SD

1. Introduction

After first being established in 2012, DraftKings is now one of the United States' most popular online sports betting and daily fantasy sports corporations. DraftKings is one of the largest sports betting companies in the industry as there are around 8 million users currently utilizing their services (Wikimedia Foundation, 2022). One of the main features of DraftKings is their Sportsbook, where they provide multitudes of player prop bets for users to bet on. One type of player prop bets DraftKings offers are live prop bets. This is when current events during a certain game change the likelihood of the prop bets. For example, if Kike Hernandez from the Boston Red Sox is playing well and has hit the ball at every at-bat and hasn't received an out yet, his odds for hitting the ball at his next at-bat might slightly increase because of his strong performance so far. Live prop bets are becoming more popular in the sports betting world where bettors are paying extreme attention to games and see if anything during the game will increase or decrease a player's odds in a prop bet. Currently, DraftKings utilizes a third-party corporation to generate the money lines for live prop bets. However, DraftKings is trying to move away from their third-party vendor and starting to create their own live prop bets. The problem at hand for our group is to aid DraftKings with generating money lines for live player prop bets using machine learning methodologies.

Throughout this project, we used various skills from machine learning, computer science, and mathematics. The machine learning models we focused and tested on include Logistic Regression, K-Nearest Neighbors, Random Forest Classifier, Random Forest Regressor, Support Vector Machines, and a Multi-Layer Perceptron Classifier. To address the initial problem of developing live player prop bets, we first started off becoming familiar with machine learning and sports betting terminologies and methodologies. We then proceeded to extract our data through the MLB API and began creating a dataset. Next, we performed feature engineering on our finalized dataset to prepare for the modeling process. Following feature engineering, we began to implement machine learning models to try and classify the "result" variable using player baseball statistics. Lastly, we were then able to drift away from model predictions to constructing money lines. In the end after comparing the various models against one another, we

were able to determine that KNN was a promising model for creating money lines to aid DraftKings in generating their live player prop bets.

2. Background

2.1 DraftKings, Sportsbook, and Online Betting

DraftKings is an online Sportsbook provider based in Boston with offices located across the world. After being founded in 2012 by Jason Robins, Paul Liberman, and Matt Kalish, DraftKings has grown into a billion-dollar company within the last ten years. Their website has offerings across 15 professional sports in eight different countries, as well as varying games such as Sportsbook, and daily fantasy sports. (Who We Are, n.d).

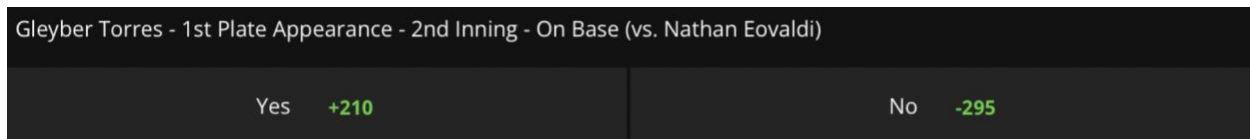
Similarly, to online Fantasy Sports, where people select players and essentially draft an imaginary team as a means to compete with others after they earn a score based upon their players' performance, Sportsbook has people betting on how certain players will perform. However, unlike Fantasy sports where these players' performance is evaluated over a period of time such as a month, or an entire season, Sportsbook looks at individual game outcomes, and often individual event outcomes within a certain game to wage bets upon. For example, a Sportsbook wager may involve whether a football team will have a score over or under a certain number, or how many touchdowns will be scored in a certain quarter of the match. For other sports such as baseball, these bets include on-base results of at-bats for various players, and even the type of hit a player will have, with options including a single, double, triple, home run, or an out. (Who We Are, n.d).

2.1.1 Propositional Bets

Propositional (prop) bets, or bets that focus on a question, are a key part of Sportsbooks and online betting. In terms of baseball, these questions usually involve strikeouts or at-bat outcomes, usually centered around specific home plate appearances. Some at-bats can be more focused, asking questions about whether or not a player will get on base (two choices: out and not out). However, other bets can center around the specific outcome of a hit (home run, triple, double, single), or whether the plate appearance will result in a walk, hit, or strikeout. These prop

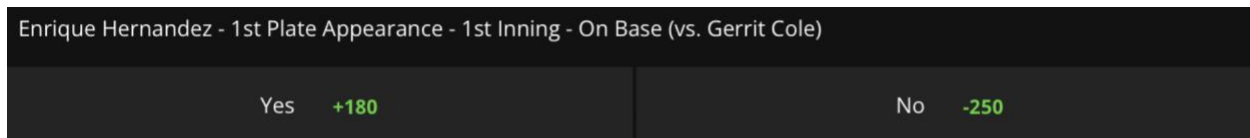
bets have what are called “derivative markets”, meaning their numbers are derived from those in other markets. (What is a Prop Bet?, n.d.). They are low-limit bets that have lines that can move fairly quickly depending on the current bets in place. (What is a Prop Bet?, n.d.).

For our project, our team focused on the out versus not out outcome of an at-bat in baseball. Below, a few screenshots taken from the DraftKings website can be seen that show this exact prop bet where the on-base outcome is staked. The “Yes” side corresponds to any result that is not an out, whereas the “No” side corresponds to an out.



Gleyber Torres - 1st Plate Appearance - 2nd Inning - On Base (vs. Nathan Eovaldi)	
Yes +210	No -295

Figure 1: Live prop bet of the 20-5-2021 Red Sox versus Yankees game taken from the DraftKings website regarding the outcome for Gleyber Torres' first plate appearance in the second inning. (MLB Odds: Draft Kings, 2022)



Enrique Hernandez - 1st Plate Appearance - 1st Inning - On Base (vs. Gerrit Cole)	
Yes +180	No -250

Figure 2: Live prop bet of the 20-5-2021 Red Sox versus Yankees game taken from the DraftKings website regarding the outcome for Enrique Hernandez's first plate appearance in the first inning. (MLB Odds: Draft Kings, 2022)

2.1.2 Setting Lines

Money lines are bets in which the bettors simply pick a team to win the game. The bets are evaluated by three outcomes which are a win, loss, and draw. A win is when the chosen team wins, and the Sportsbook pays out the bet and the winnings. A loss is when the chosen team loses, and the Sportsbook keeps the bet. A draw is more applicable on certain parts of the game instead of the whole game as baseball very rarely has ties and so a draw is more likely to be on bets such as the score of the game at a certain period in the game. Propositional bets are bets that are not related to the outcome of a sports match; therefore, a money line bet would not be an example of a prop bet. (Miller & Davidow, 2019).

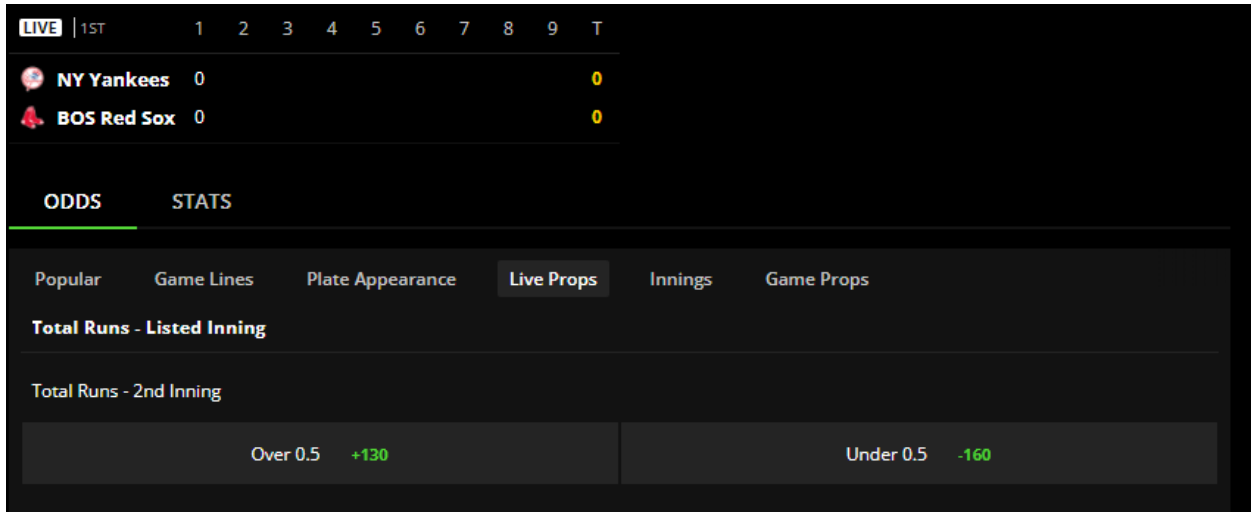


Figure 3: Example of money line Odds on DraftKings (MLB Odds: Draft Kings, 2022)

There are many ways of setting lines. American money line odds are one example of representing them. The way money lines are set using American odds is by having a money line of a “+” or a “-”. The Sportsbook looks at the data and the factors that can influence the outcome and determine the favorite and the underdog. The favorite will usually be negative on the money line while the underdog will be positive. To explain this in simpler terms, assuming that the money line is +110 on the underdog and -110 on the favorite; if a \$100 bet is placed on the underdog and the bet wins, then the return will be \$100 and the \$110 in winnings. Conversely, for the favorite odds, when a bet of \$110 is paid, then the return upon winning would be \$100, thus meaning that the payout is less (Miller & Davidow, 2019).

The above picture in reference to baseball shows the lines used in prop bet as discussed in the section above. The total runs in the 2nd inning prop bet have two possibilities that can be bet on which is whether the total runs in the 2nd inning is over or under 0.5. If one were to bet on the “over” which is the less likely outcome, then their payout would be \$230 which is \$100 (initial bet) + \$130 which is the winning. If one were to bet on “under”, the payout would be \$360 which is \$160 (initial bet) in addition to the \$100, which is the winning.

The following are two more examples of live prop bets that DraftKings provides through their platform. In Figure 4 we can see two live prop bets that DraftKings offered during the Yankees VS Boston Red Sox game on October 5th during the first inning. The option on top was

the odds on whether Aaron Judge would make a 1st place Appearance against Nathan Eovaldi. Compared to the prop bet below (Giancarlo Stanton) below, the numbers are grayed out. This signifies that at the moment the image was taken, DraftKings bettors were unable to place a bet. This is usually because the odds are being recalculated to accurately reflect the most current information available. It is not unusual for a live prop bet to be recalculated many times within a span of a few minutes.

Plate Appearance Result (2-Way)	Plate Appearance Result (3-Way)	Exact Plate Appearance Result	Pitch C
NY Yankees @ BOS Red Sox			
LIVE 1ST	1	2	3
4	5	6	7
8	9	T	
NY Yankees	0		0
BOS Red Sox	0		0
Aaron Judge - 1st Plate Appearance - 1st Inning - On Base (vs. Nathan Eovaldi)			
Yes +230	No -330		
Giancarlo Stanton - 1st Plate Appearance - 1st Inning - On Base (vs. Nathan Eovaldi)			
Yes +175	No -240		

Figure 4: Prop Bet DraftKings Example (MLB Odds: Draft Kings, 2022)

Another example for setting money lines in a game on DraftKings is illustrated in Figure 5 below. This is another screenshot from the Red Sox and Yankees playoff game. Here in this figure, the game has just begun, and the second inning is about to start. The top prop bet of the figure is a projection of the total runs that will occur in the second inning. On one hand, there is the bet a bettor can select which is “Over 0.5”. This means that the total runs by the end of the second inning will be over 0.5 runs. The money line for “Over 0.5” is +130, which means that it is about to happen. This propositional bet is also a live bet because since there were 0 runs scored in the first inning, the bet was updated to be more likely to have more runs in this inning since there weren’t any in the last inning. While on the other hand, the money line for “Under 0.5” is -160, which means that a bettor can bet on no total runs being scored in the second inning. Since the odds are negative, that means it is less likely to happen.

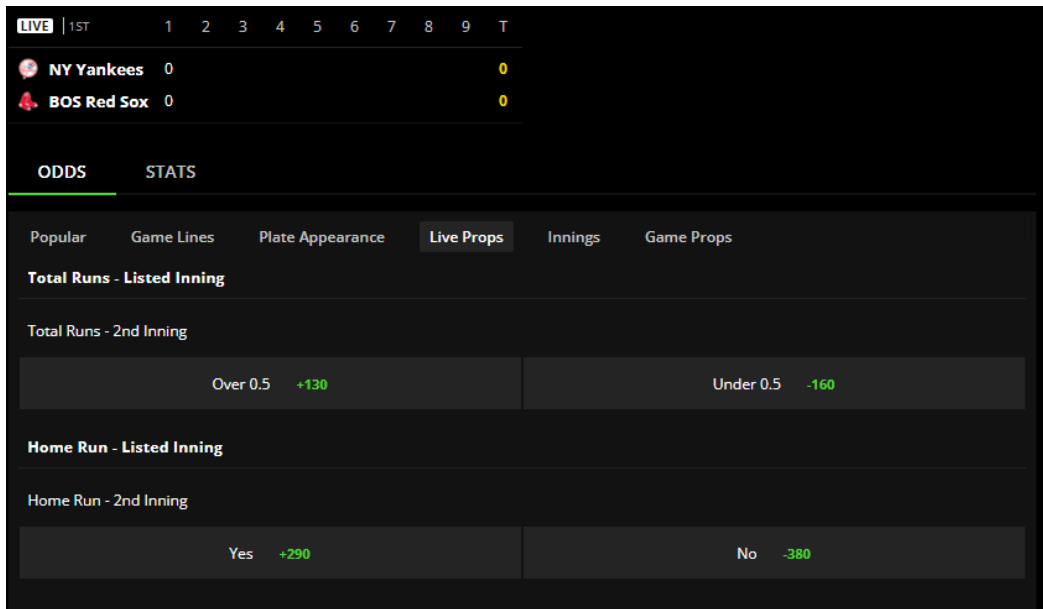


Figure 5: Prop Bet DraftKings Example (MLB Odds: Draft Kings, 2022)

The way that the money lines are set up is to make sure that the bookmakers make money either way. Usually, the odds would be considered fair if both the teams had the same corresponding negative and positive value, however, the difference in the two is called the “vig” or the “vigorish” (Miller & Davidow, 2019). By setting up the odds as such, the bookmakers stand to make money either way with people betting on both sides. We looked at the different types of odds such as decimal odds and fractional odds and settled upon money lines as they are the most prevalent in baseball and the USA in general.

2.2 Baseball

In baseball, a total of nine innings are played in a game, with each inning lasting until the defensive team (on the field) gets the offensive team (at-bat) out a total of three times. There are multiple ways to get out whilst batting in baseball including striking out, the ball a player hits or bunts being caught, being tagged by a live ball, or a play being made at a base a runner is advancing to. Striking out is when the batter obtains three strikes, where a strike is defined as the batter either missing a ball in the predefined strike zone or not swinging at this ball at all. Along

with this, if the player doesn't strike out and makes contact with the ball, if the ball they hit is caught, or thrown to the base they are running to and caught by that player before they get there, the batter is also out.

While each team has many players, there are a total of nine on the field at once for the defensive side, with the positions as follows: pitcher, catcher, first base, second base, shortstop, third base, right field, left field, and center field. For the offensive side, those same nine players on the opposing team hit in a predetermined order. Once three outs are made, the fielding team bats and the batting team becomes the fielding team. The order that the player's bat is determined by the coach and the order that the team's bat and field each inning is determined by a coin flip during the regular season. A coin flip is used to determine home (fielding first) and away (batting first) because it is often said that there is a home team advantage because if the home team is in the lead at the end of the game, they don't need to bat again and the game ends after the top of the ninth inning. Additionally, if the home team is behind, they have one final set of three outs to attempt to catch up to the away team's score.

In the case where the batter doesn't get out, many situations can occur. A batter can advance to the bases by putting the ball in play. This can be either hitting or bunting, walking, or advancing on a rule known as "drop third strike". In the first scenario, a batter can hit either a single, double, triple, home run, or a grand slam, which is a home run where the bases are loaded already. Along with this, a batter can advance to first base if they bunt, or put the ball in play with a static bat, often trying to keep the ball close to the plate. Both a hit and a bunt can result in an out if the batter doesn't beat the defensive player's throw to first if a play is made on the ball. The next case, known as a walk, is when the batter is pitched four balls outside of the strike zone before the pitcher pitches them three strikes. If this occurs, the batter is allowed to advance to first base without any chance of being called out. Finally, the drop third strike rule is a special rule in baseball where if first base isn't occupied, and the batter receives a called third strike that the catcher drops, they are allowed to run to first base in an attempt to get on. The dropped ball by the catcher becomes a live ball that must be thrown to first base and beat the runner advancing there to get them out, rather than the batter being called out automatically as usual on a third strike pitch. Below, there is a table that summarizes the relevant out versus not out batting outcomes for our project. (Alderson et al., n.d)

Out	Not Out
Strike Out	Hit
Fly Out	Walk
Ground out	Hit by pitch
Caught foul ball	Dropped third strike where the batter reaches first base

Table 1: Possible Outcomes for Out and Not Out

2.3 MLB API, Libraries and Our Data

Our main source of data came through the MLB API. To better understand the API, we extensively looked through the documentation and functions. The documentation of the functions can be outputted as formatted text or Python dictionaries. The functions include box score, game highlights, game pace, game scoring plays, last game, league leaders, line score, next game, player stats, roster, standings, and team leaders. The following pictures include examples of premade functions available to us to access Major League Baseball data.

Functions that return data in a Python dictionary

- `statsapi.boxscore_data` - generate a dict containing boxscore data for a given game
- `statsapi.game_highlight_data` - returns a python list of highlight data, including video links, for a given game
- `statsapi.game_pace_data` - returns a python dict of pace of game information for a given season (back to 1999)
- `statsapi.game_scoring_play_data` - returns a python dict of scoring play data for a given game
- `statsapi.league_leader_data` - returns python list of stat leader data for current or specified season
- `statsapi.lookup_player` - get a list of player data based on first, last, or full name, jersey number, current team id, position, etc.
- `statsapi.lookup_team` - get a list of teams' info based on the team name, city, abbreviation, or file code
- `statsapi.player_stat_data` - returns a python dict of a player's career or season stats, along with some biographical information
- `statsapi.schedule` - retrieve a list of games on a given date/range and/or team/opponent
- `statsapi standings_data` - returns a python list of standings data for a given league/date
- `statsapi.team_leader_data` - returns a python list of a team's leader data for a given stat

Figure 6: MLB API Python Dictionaries (toddrob99, 2020)

We mainly focused on the Python dictionaries as we worked with Jupyter Notebook. The game scoring play data is one of the main features we looked at, where it returns a Python dictionary of play-by-play data for a scoring play given a game. In addition, the lookup player method lets us find a player “based on first name, last name, full name, jersey number, current team ID, position, etc.” This was another function we used in a repeating loop to iterate through different players. Another important function we heavily implemented was “player_stat_data” (shown in Figure 6), where we were able to extract the important features used from the API to predict out or not out.

Moreover, to extract the data utilizing the MLB API, we utilized built-in functions of the API. First, we used the “schedule” function to specify which date range we wanted the plays of the games to be from. We did this because gathering all the plays is too much data to collect with a slow runtime. The inputs we mainly used for the “schedule” function were “start date” and “end date”.

Upon generating a timeframe of our dataset, we then used the “get” function to extract the plays of the games in between the dates we specified. To retrieve the plays of the games from the API, we had to specify the first input to be the plays of the games within the “get” function.

In the following step, we used the “get” function again to get the result of each play. The API then outputs the results of each of the plays that include either a flyout, forceout, grounded into a double play, groundout, lineout, popout, strikeout, double, hit by pitch, single, or walk. We then mapped each of the possible results of an at-bat to either a “0” for any type of out and a “1” if the at-bat resulted in anything other than an out. The results that were grouped in the “out” category and mapped to “0” were flyout, forceout, grounded into a double play, groundout, lineout, popout, or strikeout. On the other hand, the results of a given at-bat that were grouped in the “not out” category and mapped to “1” were single, double, hit by pitch, or walk.

Out (0)	Not Out (1)
Flyout	Single
Forceout	Double
Grounded into double play	Hit by pitch

Groundout	Walk
Lineout	-
Strikeout	-

Table 2: Possible Outcomes of Out and Not Out

After getting the results of each at-bat, we then created a DataFrame to store information about the at-bats from the MLB API. As part of the DataFrame, we first retrieved the information about the matchups of batters and pitchers for the at-bats. The matchup data was also retrieved from the “get” function where we specified the first input of the function to be “matchup”. For our initial dataset, we had three columns: batter ID, pitcher ID, and then result. Using the “get” function from the API, we were able to retrieve data for each of those three columns and append the columns together to the new data frame.

In addition to retrieving data from the MLB API, we used several different Python libraries to analyze the data we gathered over time. The main libraries that benefited our code and analysis were Pandas, Seaborn, Matplotlib, and Scikit-learn. Pandas was a popular library that we used initially to read in the CSV data, create data frames, and perform initial exploratory data analysis on the dataset. The next library we used was Seaborn, where we mainly utilized Seaborn for visualizations like scatterplots and other types of graphs. Similar to Seaborn, we used to generate graph visualizations like bar and line graphs. Lastly, we used scikit-learn to develop our models and evaluate our models as well. We used Scikit-learn to perform a train-test split on the data and implement the models used such as random forest, Logistic Regression, K-Nearest Neighbors, and Support Vector Machine. Scikit-learn also was used for evaluation where we used it for accuracy score, confusion matrices, classification reports, and other evaluation metrics including F1-score, recall, and precision.

2.3.1 Our Dataset and Features

To build our dataset we began by collecting batting and pitching statistics on all the players that MLB API kept information on. This was done using the “statsapi.player_stats” call function, which accesses both the season individual and career statistics of each player. Within

the scope of this project, emphasis was put on career statistics. Table 3 below includes a comprehensive list of all the features that were collected and stored in a DataFrame.

Feature Name	Data Type	Description
batter_id	integer	Six digit batter identification number corresponding to a specific player.
pitcher_id	integer	Six digit pitcher identification number corresponding to a specific player.
result	integer	Binary result of an at-bat where 1 corresponds to “not out” and 0 corresponds to “out”.
b_atBats	integer	Total number of at-bats a batter has had.
b_batting_avg	float	A batter’s batting average.
b_obp	float	A batter’s on-base percentage.
bat_side	integer	Binary integer where 1 corresponds to “right” and 0 corresponds to “left” for a player’s batting handedness.
b_ops	float	A batter’s on-base-plus-slugging percentage.
b_gamesPlayed	integer	Total number of games a batter has played.
b_slg	float	A batter’s slugging percentage.
b_babip	float	A batter’s batting average on balls in play.
p_atBats	integer	Total number of at-bats a pitcher has faced.
p_era	float	A pitcher’s estimated runs allowed.

p_strikePercentage	float	A pitcher’s strike percentage.
pitch_hand	integer	Binary integer where 1 corresponds to “right” and 0 corresponds to “left” for a pitcher’s handedness.
p_strikeoutsPer9Inn	float	A value corresponding to a pitcher’s strikeouts per nine innings.
p_runsScoredPer9	float	The number of runs scored against a pitcher per nine innings.
p_hitsPer9Inn	float	The amount of hits batters have against a pitcher per nine innings.
p_walksPer9Inn	float	The number of walks a pitcher allows per nine innings.

Table 3: Definition of Our Features from the Dataset

After storing every player’s career data in a pandas DataFrame, we used another call function to collect over 80,000 at-bat results from 2021 with pitcher and batter IDs from that play. We then used the batter and pitcher IDs as a join condition, to create a finalized DataFrame that included the result of an at-bat and 16 features describing the pitcher and batter in that play.

2.4 Machine Learning Models

Machine learning is a branch of artificial intelligence that uses the constant input of data to predict outcomes. The predictions become more accurate with more data input given to the models. Machine learning is split into three main types: supervised learning, unsupervised learning, and semi-supervised learning (Burns, 2021). For our research, we decided to proceed with supervised learning model methodologies.

Our reasoning for proceeding with supervised learning was that supervised learning requires labeled inputs and the desired output which we had. We had data that was labeled data

on baseball statistics, and we had a specific target output which was the result of a specific play in terms of “outs” versus “not outs”.

2.4.1 K-Nearest Neighbors

KNN, also known as K-Nearest Neighbors, is a supervised machine learning model that we mainly used for classification tasks. KNN performs classification by computing the distances between points and grouping together similar points into a cluster (Harrison 2019). The clusters of KNN are essentially created where it counts all the points near a certain point and sees which ones are in what class (As shown in Figure 7 below). The majority vote counts on what the class is and is then assigned to that point. This process is vital to setting our odds for an at-bat, which we will go further into detail later, where we implemented a KNN model from scratch to produce money lines. Based on the idea of counting the votes of class for each neighbor, we were able to utilize that idea to perform a similar task to count the votes of the result of an at-bat in our self-developed pseudo-KNN model.

The image below shows part of the process of the KNN model. There are two classes: A and B, where the red dot is the point that needs a classification. In the picture, if the k-value equates to 3, then the classification for the red dot would be Class B because there are more purple dots than yellow dots. If the k-value is 6, then the red dot would be in Class A because you count all the dots in the circle and whichever one has the most dots, the KNN will classify the new data point to the most popular class. This process shown in the diagram translates over to how we implemented our own KNN to produce money lines, where we counted up all the votes of “out” or “not out” in a given circle to show the probability of whether a certain at-bat will be “out” or “not out”. This greatly helped us to later generate money lines from these probabilities.

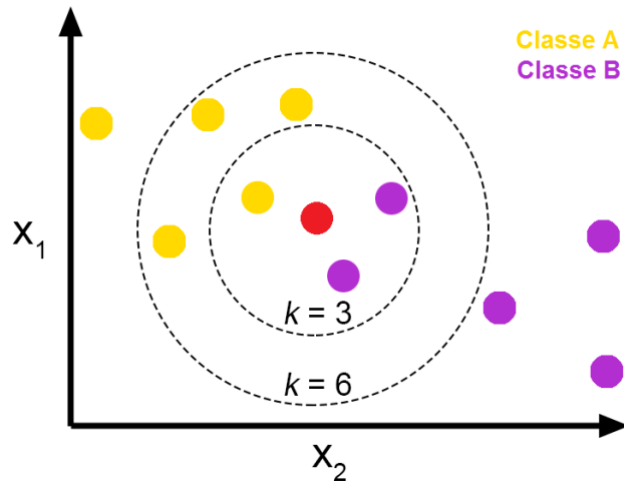


Figure 7: KNN Model Diagram (Harrison, 2019)

2.4.2 Neural Network

A neural network is a useful machine learning technique that consists of algorithms that can detect hidden patterns within a given dataset, leading to the classification of a given problem. Neural networks first start with an input and are passed through mathematical equations to eventually produce an output. There are three major parts of a simple neural network: the input layer, hidden layer, and output layer (As depicted in Figure 8 below).

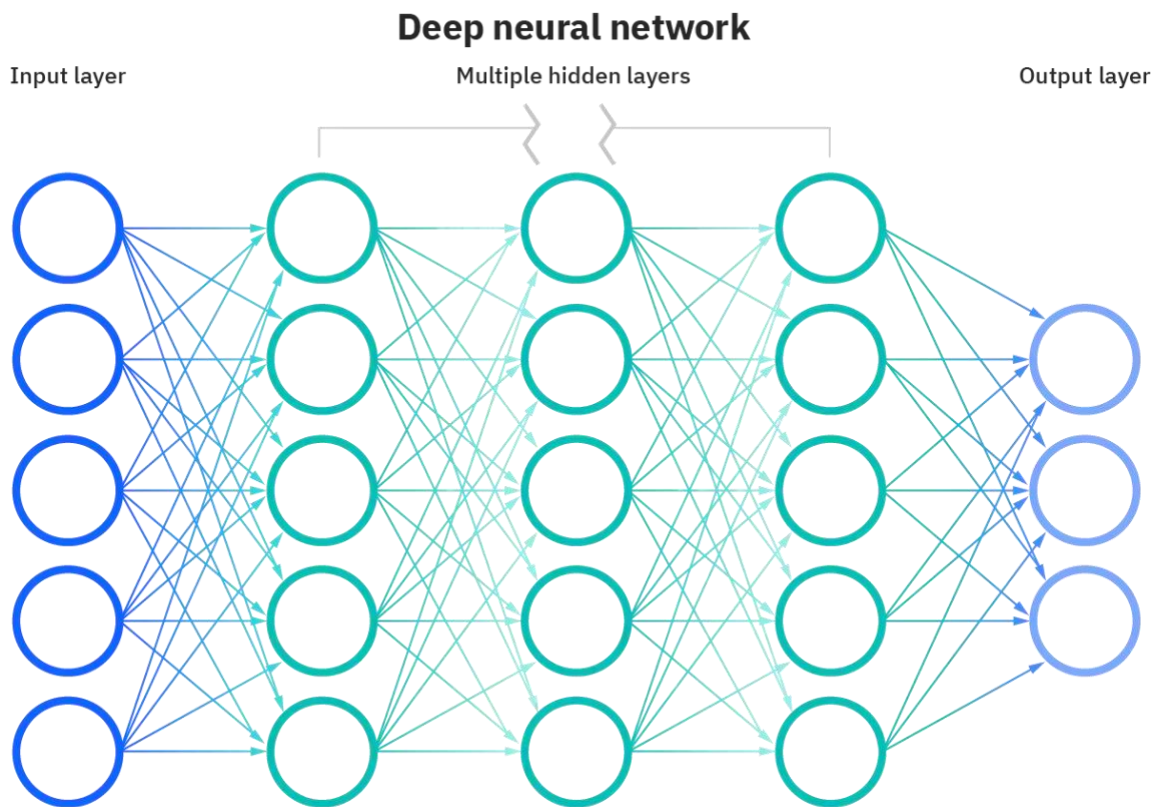


Figure 8: Neural Network Diagram (IBM Cloud Education, 2020)

Within the neural network, there are neurons that perform operations on the data. Each neuron processes the data through this operation:

$$Y = \sum_{i=0}^n (W_i * X_i) + B \quad (\text{Malik, 2021})$$

Shown by the equation above, a neuron processes the sum of the given input (X_i) and multiplies it by the weights (W_i) of each input. The weights of the input calculate how closely related two neurons are to each other. The last part of the equation is adding the bias term (B). Both the weights of each neuron and the bias term can be modified to allow the function to fit the data in the best way possible.

Neural networks were one of the classifiers we used next in hopes to find better results. More specifically we implemented a Multi-Layer Perceptron (MLP) classifier model to predict

whether an at-bat was “out” or “not out”. Our thinking behind picking an MLP classifier was utilizing the different layers to find the most optimal function for classification.

2.4.3 Decision Trees

One of the simplest methods used in machine learning is as a decision tree. A decision tree is a supervised machine learning method. A decision tree is the representation of probabilities of certain outcomes which is represented in the form of a tree in which each branch represents an outcome or a condition and is called a node. The decision tree splits data based on each node's classification or probability. The decision tree will keep on iterating based on whether the predetermined condition is met or not. Once the conditions are met, the leaves of the tree represent the classifications made by it (Sharma, 2021). As seen in Figure 9 below, the decision tree starts with a root node and iterates through the decision nodes until the leaf nodes are reached. The basic concept of decision trees is used both for classification and regression models in machine learning.

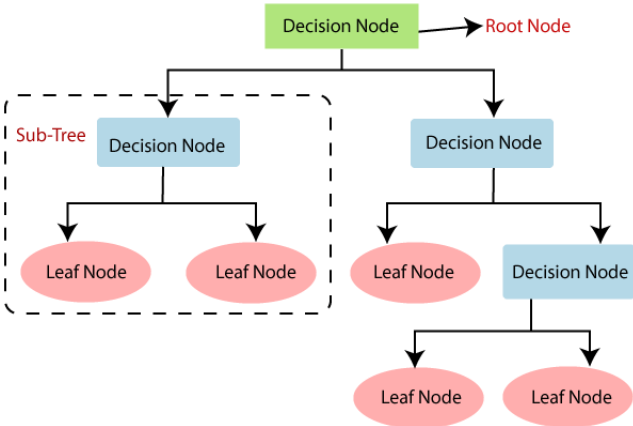


Figure 9: Breakdown of a decision tree into its various leaf nodes. (Hanafy et al., 2021)

The split criterion is usually done in two different manners, either by information gain or entropy. Information gain is the split at each node where the algorithm determines the subsequent split gives the most information to the algorithm or also called information gain. This split continues until the algorithm determines there is no more information to be gained and then all the nodes will be leaf nodes. Gini Impurity is another method for splitting the tree based on

how the tree would misclassify random new data as per the current split of the decision tree nodes (Sharma, 2021).

2.4.4 Random Forest Classifier and Regressor

Random forests are supervised machine learning models that make use of the decision trees discussed in the previous section. Rather than using one singular decision tree to predict what an outcome should be, a random forest utilizes the power of multiple different decision trees, using each tree's outcome to predict and then by a system of voting between the different models, makes a final prediction. According to Great Learning, "a random forest technique can focus both on observations and variables of a training data for developing individual decision trees and take maximum voting for classification and the total average for regression problem respectively" (Great Learning Team, 2020). This quote elucidates how Random Forest Classifiers work in classifying their outcomes and Random Forest Regressors work in predicting the outcomes. As seen in the figure below, the process for any Random Forest model begins with the random sampling of the data from the input data. Then, decision trees are built based on the random samples of data. The model then runs votes on the decision tree predictions and the tree with the most votes is then chosen to be the prediction (Chakure, 2020).

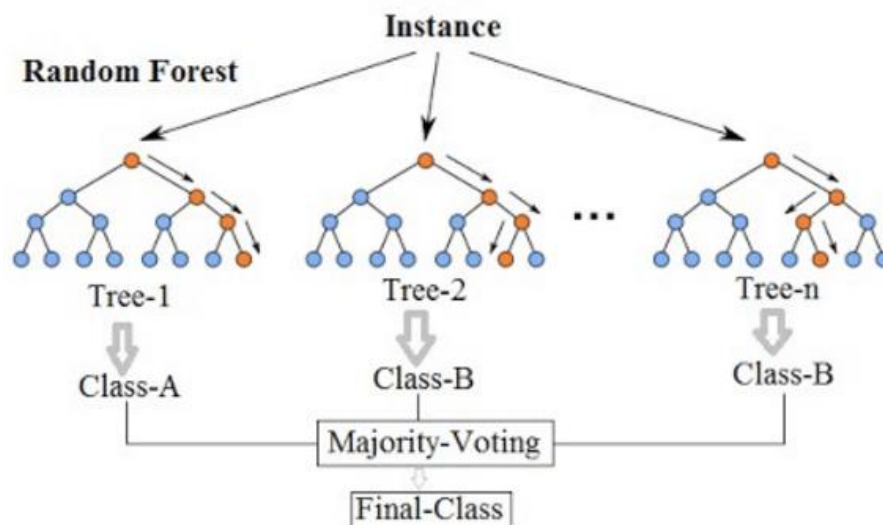


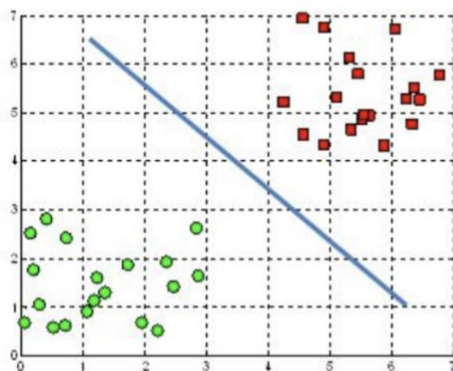
Figure 10: Process of voting in a Random Forest (Chakure, 2020).

Random Forest Classifiers work best with discrete values, such as data representing either a ‘0’ or a ‘1’, or in this project’s case, an ‘out’ or a ‘not out’ value. Random Forest Regressors, however, work best on data that has numerical outputs. For example, if we were trying to predict the likelihood that a batter would be out, rather than whether they were just out or not out. Conversely, a Random Forest Regressor would be preferential due to its specialization in numerical outputs. It could output a number representing this percent likelihood rather than just predicting one value (out) or another (not out).

2.4.5 Support Vector Machines

Another algorithm that we tested is the Support Vector Machine or SVM. This model can be used for both classification and regression models and utilizes something referred to as a hyperplane. Hyperplanes are, as Gandhi defined, “decision boundaries that help classify the data points” by splitting the data into classes depending on where the points fall between the line(s). Hyperplane dimensions depend upon the number of input features: two-dimensional hyperplanes, in the case of two input features, are lines, and three-dimensional hyperplanes, in the case of three input features, are planes (Gandhi, 2018).

A hyperplane in \mathbb{R}^2 is a line



A hyperplane in \mathbb{R}^3 is a plane

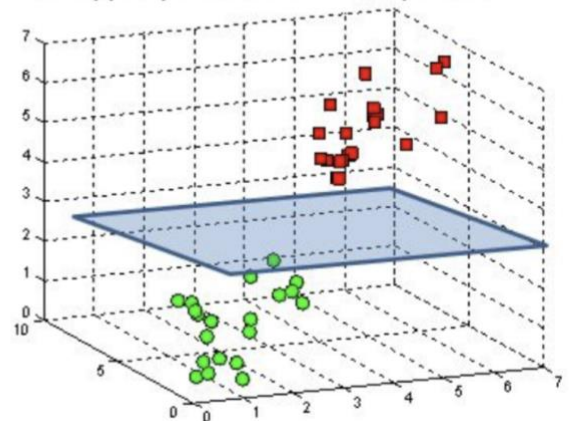


Figure 11: Visualization of what a hyperplane looks like in relation to the given data depending on the dimensional space it is in (Gandhi, 2018).

By utilizing these hyperplanes, the objective of an SVM model is to, “find a hyperplane in an n-dimensional space that distinctly classifies the data points.” This can be seen in Figure 12, where a series of various hyperplanes are drawn between data points, showcasing both the numerous possible hyperplanes that may be drawn, and what an ideal hyperplane would look like.

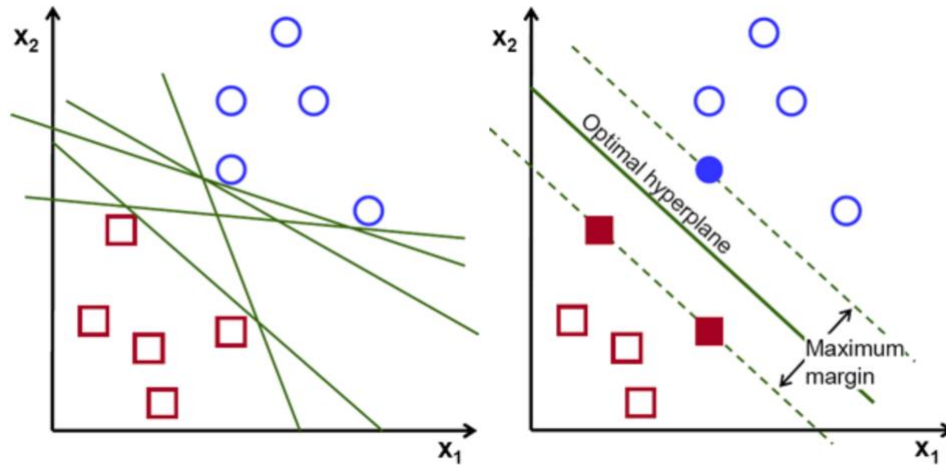


Figure 12: Graphs of what a set of possible hyperplanes on data (left) can look like. These lines are drawn between two selected points with an equal distance apart from the actual points. The right graph shows what an optimal hyperplane would look like with perfect data (Gandhi, 2018).

Along with trying to split the data into classes, it is best to find a line or plane that has the maximum margin (distance between data points) for each of the classes. These margins can be visualized utilizing Figure 13. The points closest to these hyperplanes are what are known as support vectors because they influence the position and orientation of it. (Gandhi, 2018).

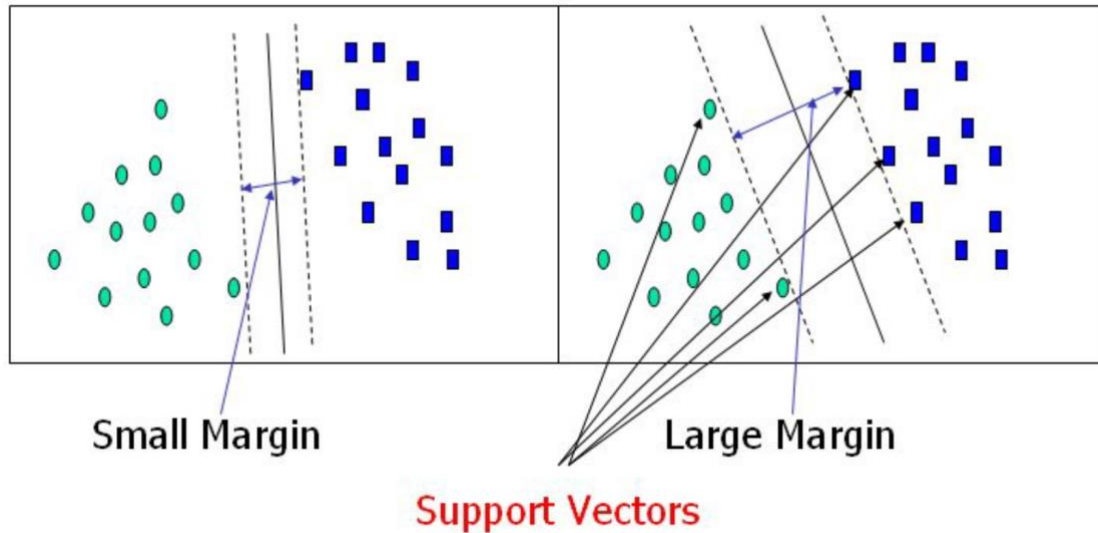


Figure 13: The difference between what a small margin, versus a large margin is referring to in a SVM hyperplane (Gandhi, 2018).

2.4.6 Logistic Regression

Following the documentation from sci-kit learn, we implemented several Logistic Regression models. The first step of the Logistic Regression model is that it uses the logit function to compute a probability for a value of the data. After that, the model can classify probabilities that are less than or equal to 0.5 to the “0” class and then probabilities that are greater than 0.5 to the “1” class. The Logistic Regression classifies the values of the data using the logistic function. The shape of this function turns into a sigmoid shape because the classes are 0 or 1 (As shown in Figure 14 below) (Jurafsky et al., 2020). Initially, we believed that the potential of Logistic Regression for classifying “out” or “not out” was promising. That was because there is a classification task to predict the result of an at-bat and Logistic Regression does a good job classifying data. However, after later implementing the model, we found out that it is always predicting “out”, which is something we were trying to avoid. This was an important step to take in the process of generating money lines because it has made us realize that we need to start to move on from model predictions to generating money lines.

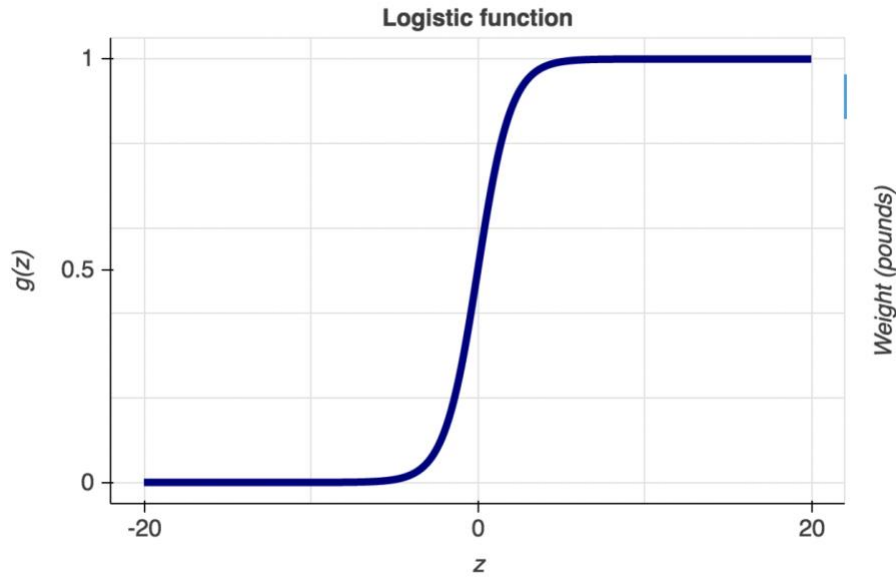


Figure 14: Logistic Function Graph Diagram (Chávez, 2019)

2.5 Confusion Matrices

To determine the effectiveness of many of our machine learning models, we used confusion matrices. A confusion matrix is used to measure the performance of a classification problem with two or more classes. One advantage to confusion matrices is that they are relatively easy to generate and understand compared to other data visualization methods (Narkhede, 2018). A disadvantage to using a confusion matrix is that it does not perform well on an unbalanced data set. This is a challenge we encountered during our project but were able to overcome.

There are four possible cells that a prediction could fall under in a binary classification problem when using a confusion matrix. Each of which is listed in Table 4 below, with definitions:

Confusion Matrix Values	
True Positive (TP)	Correct positive prediction
True Negative (TN)	Correct negative prediction

False Positive (FP)	Incorrect positive prediction
False Negative (FN)	Incorrect negative prediction

Table 4: Confusion Matrix Values Table

The following is an example of what a confusion matrix would look like for a binary classification problem (Narkhede, 2018)

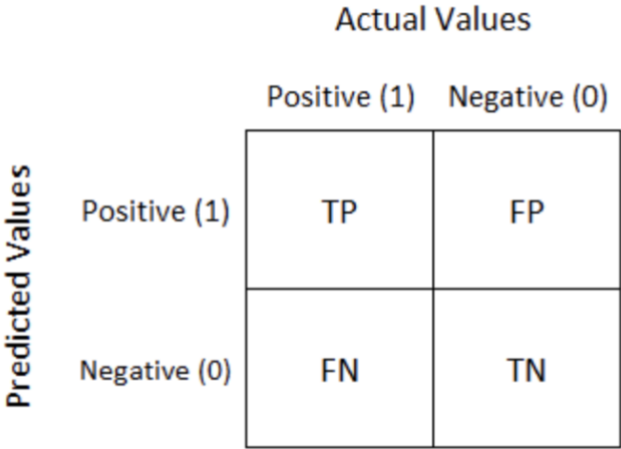


Figure 15: Confusion Matrix Diagram (Narkhede, 2018)

The accuracy of a classification problem can be calculated by getting the ratio of correct predictions (TP + FN) to total predictions (TP + FP + FN + TN). Multiplying that fraction by 100 will represent the accuracy as a percentage (Kulkarni, 2020).

One method for improving the quality of a confusion matrix is to add a heatmap over the numbers. This is particularly helpful when there are more than two classification outcomes possible. In a classification problem with 5 classes, there are 25 possible outcomes. This is because each Actual value can be misclassified as 4 more classes. With 25 numbers on the graph, it can be difficult to pick out what’s important. With a heatmap, it becomes much easier to visualize. The following is an example of a group classification problem (using the popular Iris dataset) where a darker shade of blue signifies a larger number.

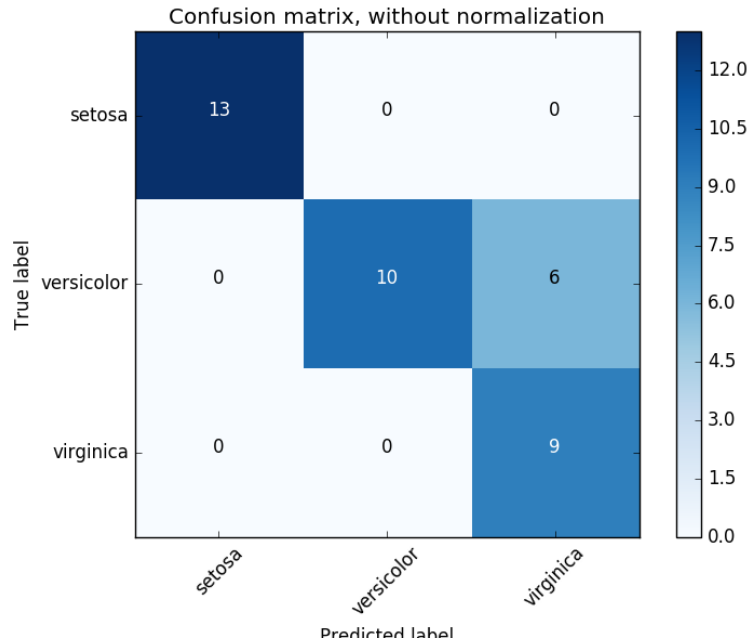


Figure 16: Confusion Matrix Example (Kulkarni, 2020)

In the context of our project, instead of positive and negative being the classification outcomes, we used out (encoded as 0) and not out (encoded as 1). A true positive in this case would indicate a correct prediction that the outcome of an at-bat is out, and a true negative would mean that we correctly predicted that the outcome of an at-bat would be not out.

2.5.1 Testing Versus Training

Throughout the machine learning process, an important step is the validation of an algorithm's accuracy. This step ensures that there's no bias in how a developer has set up their data science pipeline. There are many forms of algorithm validation. To name some popular ones: test/train split, K-Fold Cross Validation & Leave-one-out Cross-Validation (Grootendorst, 2019). In this process, we decided to use the test/train split method, where the model is created using a specified subset of the data and tested on the training subset of the data.

Now that we know what confusion matrices are, we can talk about how they're used to evaluate results on training and testing data within machine learning algorithms. In Figure 17 below, there are two confusion matrices displaying the predicted outcomes of the testing and training data using a random forest classifier with depth 5. For the sake of simplicity, the data

used has been balanced beforehand. There are a total of 36,891 samples in the Training data (sum of TP, FP, TN, FN) and 9,223 samples in the testing data meaning there are 46,114 samples in total. Using basic algebra, we can find that the test/train split used in this problem was 80/20. We can also compare the results from each graph to show if there are any differences.

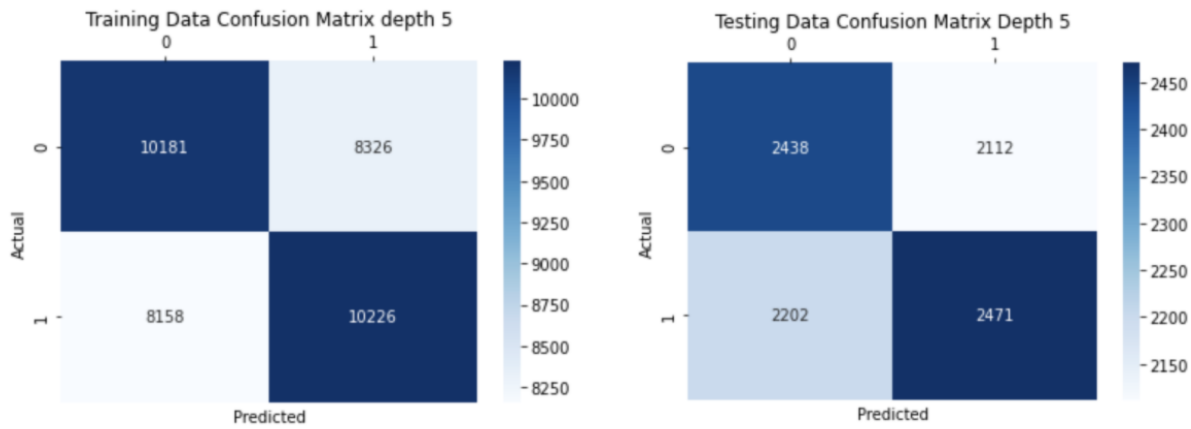


Figure 17: Training and Testing Confusion Matrices

Accuracy Rate (Accurate Prediction / Total Predictions)	
Training Data Accuracy	55%
Testing Data	53%

Table 5: Accuracy of Training and Testing Data

In the table above we can see that there are some differences in the accuracies of the testing data and training data. This is because, when you test a model on the data it was trained on the data you incorporate bias into the model. Generally speaking, the results on testing data are more reliable than the latter.

2.6 Accuracy Metrics

To evaluate the predictions and evaluate the tendency of the models in predicting the outcome of a specific at-bat. With all the different models being used, there are different metrics for accuracy for each of them. In this section, the metrics used to measure the performance of the models and which models they were used for are discussed.

2.6.1 F1-Score, Precision, & Recall

In looking at the predictions made by the different machine learning models, some of the most important metrics used to evaluate the performance of the models was the F1-score. F1-score comprises two different measures which are precision and recall. They both evaluate two different aspects of the predictions being made by the machine learning models.

Precision is calculated using the formula : $\frac{\# \text{ of true positives}}{\# \text{ of true positives} + \# \text{ of false positives}}$ (Korstanje, 2021). The way that the precision score can be understood is that it is the percentage of all the classified positive outcomes which are positive. If the model is not precise that means it predicts many positive outcomes that are not positive. If the model is precise, it means that the model might not predict all the positive outcomes but the percentage of positive outcomes that it predicts are more likely to be accurate.

Recall is calculated using the formula : $\frac{\# \text{ of true positives}}{\# \text{ of true positives} + \# \text{ of false negatives}}$ (Korstanje, 2021). Recall is the success rate of finding the true positives from all the positive outcomes. A high recall is basically when the model is successful at predicting the positive outcomes to a large extent while on the other hand, a low recall indicates the model is not good at predicting the majority of the positive outcomes.

The F1-score is a combination of the two and can help us evaluate the performance of the model. The formula for the F1-score is $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$ (Korstanje, 2021). The F1-score takes both into account and so a high F1-score high precision and recall, a low F1-score indicates low precision and recall. The team used the F1-score as an evaluation metric since it proved a good metric in evaluating the performance of the machine learning models we used on the data we had.

2.6.2 Accuracy Score

The accuracy metric we used to evaluate our models apart from the F1-score was the accuracy score which has the formula:

$$\frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{True Negative} + \text{False Positive} + \text{False Negative}}$$
 (Kumar, 2022). The accuracy score helped us understand how the model worked in predicting outcomes. We also got a more balanced understanding of the predictive ability as the F1-score was more so based on the positive outcomes whereas the accuracy score was based more on the overall predictive ability of the model.

2.6.3 Root Mean Square Error, Mean Square Error, & Mean Absolute Error

To evaluate the difference between the predictions and the actual values of a Random Forest Regressor, we used Root Mean Square Error (RMSE), Mean Square Error (MSE), & Mean Absolute Error (MAE). Not only did it show us the importance of adding more features, but it also allowed us to evaluate the fit of certain regression-based machine learning models towards our goal of predicting outs vs not outs and eventually setting lines based on them.

MAE is the mean absolute error in the prediction vs actual output. It measures the difference between the two and averages it out. It uses absolute values as a calculation which assumes all variation is a positive value and therefore evaluates the average residuals for the model (Trevisan, 2022). MSE is the mean squared error which is the square of the average between the training and testing data. This squared value accounts for negative values as well since it is squared as the difference between the prediction vs actual output. It measures the variance of the residuals between actual vs predicted values. RMSE is the root of the mean square error. It helps us evaluate the standard deviation of the residuals of the predicted vs actual values (Trevisan, 2022).

2.6.4 Standard Deviation

Standard deviation is another metric we used to evaluate the performance of our models.

The formula for standard deviation is $\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$ (Hargrave, 2022). In this case “N” is the

number of predictions being made, x_i being a single prediction and μ being the mean of all the predictions. We used the standard deviation to help us see how far away our predictions were on average from the actual values. While more helpful in the regressor machine learning models, it was also helpful in helping us evaluate the classification models we ran on our data by comparing the predictions vs the actual values.

3. Methodologies

3.1 Working With Data

After a few initial tests with our data, our team had a few obstacles within our dataset. First, our starter models all predicted out every single time due to there being a 75:25 ratio of out to not out outcomes. Because of this, we needed to balance our training dataset so that the models would focus more on the other data we provided it with and not just the perceived known likelihood of an outcome. Along with this, we ran into the issue of players with unreasonable ERAs and batting averages due to not having enough instances of an at-bat or games pitched. To combat this, we also cleaned the data and filtered out some of these players who didn't meet a certain numeric threshold.

3.1.1 Cleaning the Data

To process the large amount of data our group was able to obtain, some processing was required before the data could be used in our models to ensure the findings we were obtaining were as accurate as they could be. During the data collection process, we noticed that the MLB API was missing some data points on certain players. While collecting our data, these cells were replaced with the code “-1000”, signifying a null value. To begin the cleaning process, we listwise deleted any players that had a “-1000”, or null value from the player statistics DataFrame that we created. There were some cases where a pitcher would have no information in his pitching stats. This would likely mean the player had no at-bats, so it wouldn't be an accurate representation of a batting average. After this, players with less than 100 pitches and players with less than 100 at-bats were filtered out since they wouldn't have enough in-game experience for

their data to be reflective of traditional ERAs and batting averages. To determine these numbers, we tested by filtering out varying amounts of data. With this, we determined that filtering by each player having at least 1000 at-bats removed too much of our dataset, whereas 10 at-bats were not enough to accurately represent a player's abilities. Because of this, 100 was determined to be the best number to filter the at-bats by. This same ideology followed for the number of pitches. This ensured that a player who had, for example, one at-bat wouldn't be able to skew a result by having a batting average of 1 by getting on base once in that one at-bat. Before the cleaning process, we had collected a total of 83,843 pitcher/batter matchups with outcomes. After this process, we were left with 77,415 matchups. Some additional data cleaning and normalization were performed later in the project when working with PCA and certain predictive algorithms.

3.1.2 Balancing the Data

Next, for our plan to train the models on balanced data before testing them on our unbalanced data, our team needed to create a balanced dataset that we could use to train on. To do this, we took our 16-feature dataset that had 78,460 total plays and totaled the occurrences of both out (55111) and not-out (23349) occurrences. Then, we took a random 75% subsample of the not-out plays, and we were left with a 13,134 not-out play count. This 75% was decided upon so that when we tested our data, we would have some not-out occurrences that we had not trained with to evaluate the accuracy of our model better. Since when we matched a random subset of 13,134 of the "out" plays, there was nearly 75% of the out data not being used for training purposes, so there were ample amounts of out occurrence data to test our models on that we hadn't used in the initial training.

3.1.3 Feature Engineering

To process a large amount of data our group was able to obtain, some processing was required before the data could be used in our models to ensure the findings we were obtaining were as accurate as they could be. To begin this, we filtered out any "null", blank, or zero values from the datasets in columns where they shouldn't be, such as a 0 for a batting average. This would likely mean the player had no at-bats, so it wouldn't be an accurate representation of a

batting average. After this, players with less than 100 pitches and players with less than 10 at-bats were filtered out since they wouldn't have enough in-game experience for their data to be reflective of traditional ERAs and batting averages. These numbers were selected arbitrarily after it was determined that filtering out 1000 removed too many data points, 10 only removed barely any data points, and filtering out 100 removed about 6-7%. This ensured that a player who had, for example, one at-bat wouldn't be able to skew a result by having a batting average of 1 by getting on base once in that one at-bat.

3.2 Random Forest Testing

3.2.1 Feature Importance (Correlation Diagram)

To begin our random forest testing process, we needed to determine which features, from our 16 total, had the highest correlation to the outcome of an at-bat. To do this we implemented a heatmap of all the features plotted against each other (Displayed in Figure 18 below).

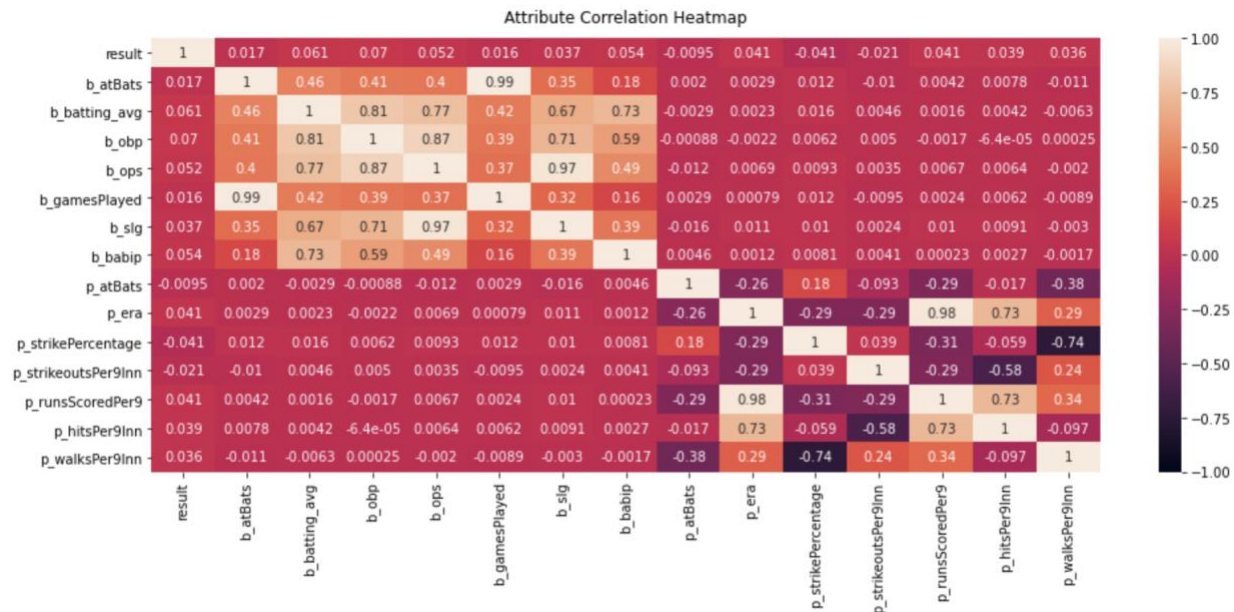


Figure 18: Features Correlation Heatmap

In Figure 17 above, all the features are plotted. However, to better visualize just the “result” feature against all the other features we developed Figure 19 below. This figure shows some of

the correlations between each feature and the “result” feature. In scikit-learn, the way the correlation is calculated is through the correlation coefficient. A high correlation of 1 means that the feature is closely related in predicting the “result” feature, while a low correlation of 0 means that the feature is not closely related in predicting the “result” feature.

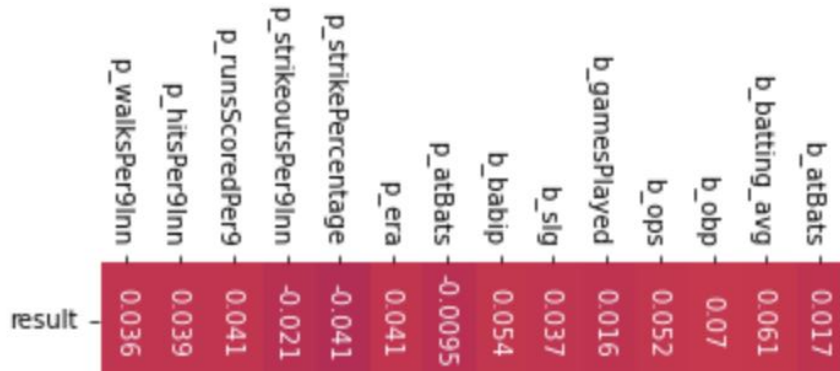


Figure 19: Result Feature vs. Rest of Features Correlation Plot

3.2.2 Best Random Forest Depth

While in the testing phase for the discovery of the best Random Forest Classifier, we worked with tweaking three main hyperparameters: The depth of trees being generated, the activation function used and the features selected for predicting.

Our first test used Gini as the criterion for selecting feature importance. The data set was manually balanced to extrapolate differences in regions, otherwise, the model would always predict out. To balance the data the total number of not-outs was kept constant, and an equal number of outs were selected randomly. Next, we tested the accuracy of the model at all depths between 1 and 30 on both the testing and training data using 8 pitching and 8 batting statistics for each player (16 in total).

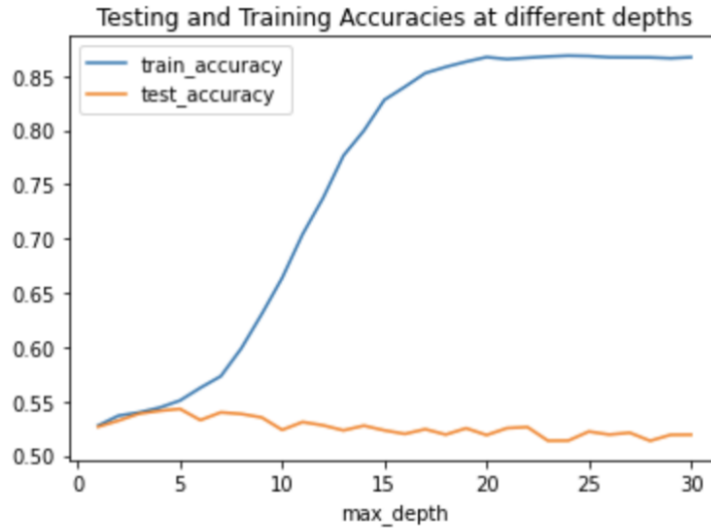


Figure 20: Testing and Training Accuracies of Random Forest Depths

From this graph we can see that the model performs best at depth 5. After a depth of 5, the testing set performs significantly worse and the training set overfits the data. After tuning for depth, our team tested the accuracy of the Random Forest Classifier with depth 5 using two different criteria: Gini and Entropy. The tables below display the training and testing results of these tests. Gini criterion's performance is significantly better (Shown in Figures 21 and 22 below).

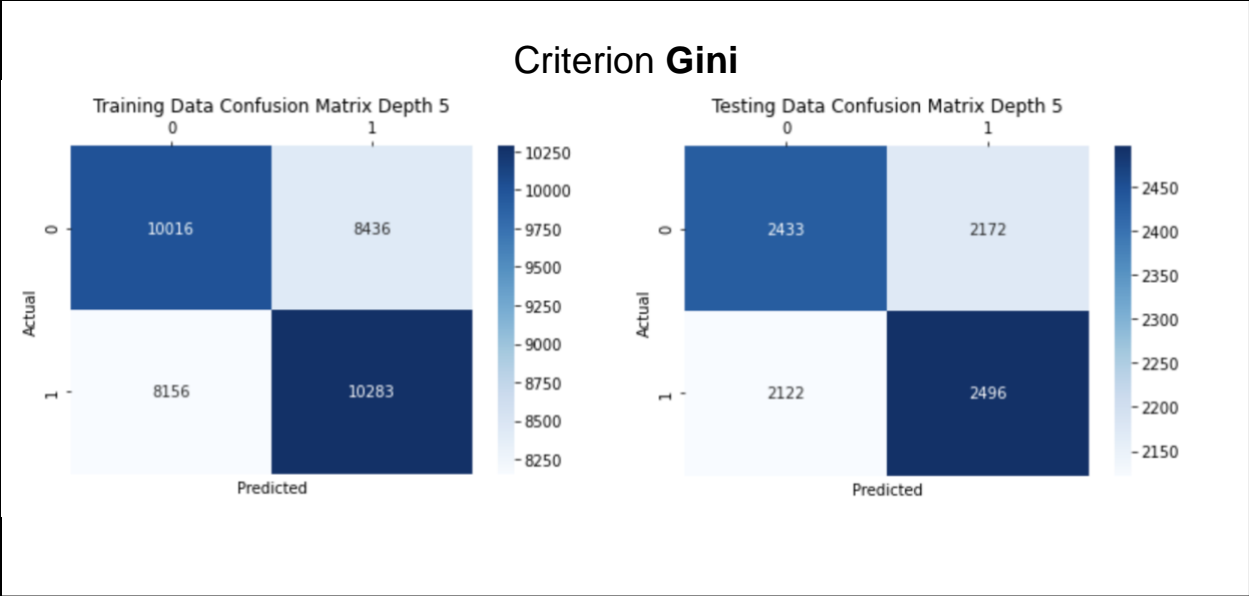


Figure 21: Training and Testing Confusion Matrices of Gini Criterion

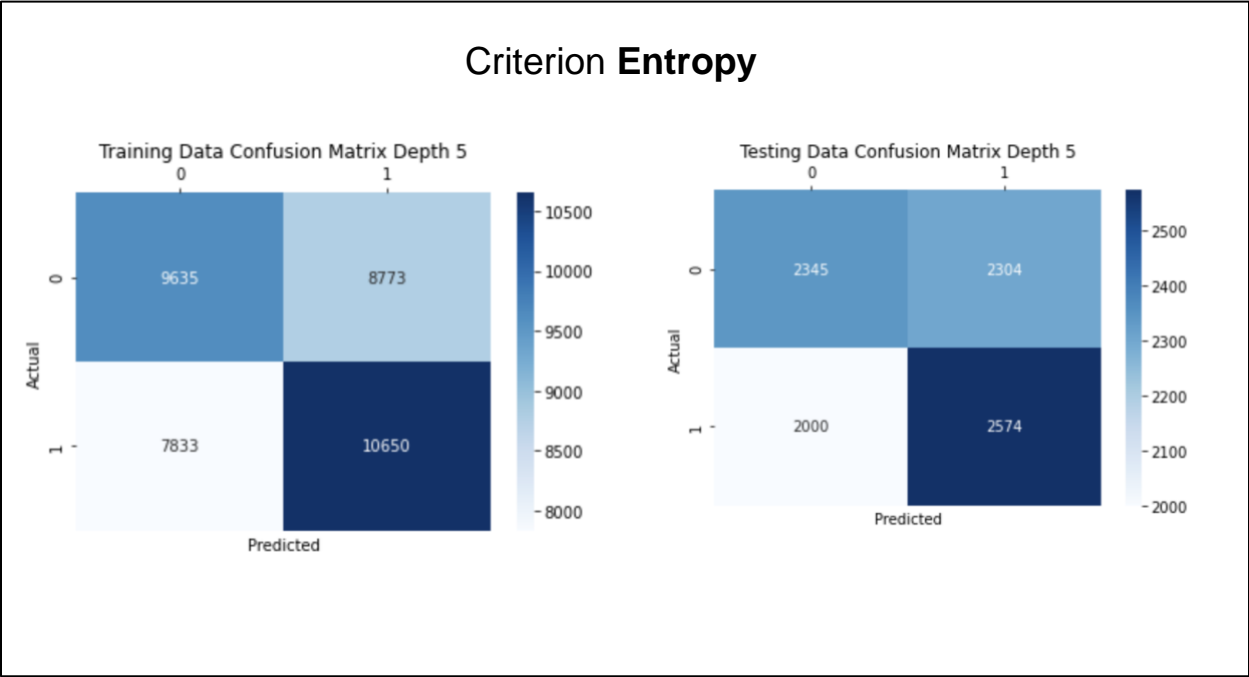


Figure 22: Training and Testing Confusion Matrices of Entropy Criterion

The final parameter that we tuned for while Random Forest Classifier testing, was the number of features used. We ran three tests to find the best combination of features. In all three

tests, the depth of the trees was 5 and the criterion used was Gini. The combination of features used with each test can be seen in the table below:

Predictive Feature Tuning	
Test #1	Pitching Hand, Batting Hand, ERA, Batting Average
Test #2	4 of the highest linearly correlated features to result of an at-bat (out/not out)
Test #3	All 16 features

Table 6: Feature Testing Tuning

While test #1 performed somewhat worse than test #2 and test #3, there were no significant differences in the accuracy results of test #2 and test #3. The Random Forest Classifier model that we determined was best at predicting the outcome of an at-bat had a depth of 5, used Gini as the criterion, and incorporated all 16 player statistics collected through the MLP API. These results helped us further predict the outcome of an at-bat which DraftKings can use to set lines.

3.3 DataRobot / Akkio (AutoML) Testing

To evaluate the effectiveness of the models we were testing, we decided to utilize a tool called AutoML to compare our prediction quality. To do this, AutoML looks at the data provided to it before running a bunch of different models on it to determine which had the best accuracy score. To run these tests, a website called Akkio was utilized. This website takes an input of a given dataset, as previously discussed, before returning information such as prediction quality, the best model to use, and the top fields it used to make that prediction.

First, this program was run using our unbalanced dataset with 16 features. It determined that the best model type would be a sparse neural network and had an accurate prediction of about 66%. However, since this accuracy prediction was in-line with the current accuracy predictions we were getting using the Random Forest Classifier, this model was not pursued.

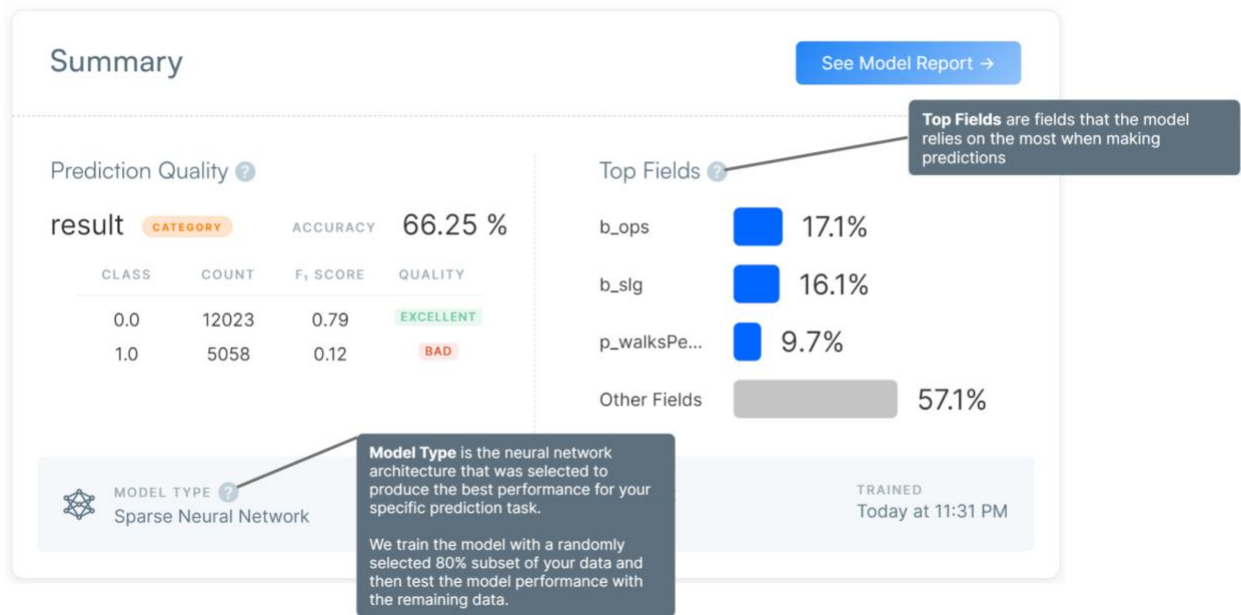


Figure 23: Results of the best model type and the resulting prediction quality on an unbalanced dataset using Akkio

Next, this program was run using our balanced dataset with 16 features. This balanced dataset was created per previous experimentation when we took a random subset of 75% of the not-out samples before taking an equal random sample of the out samples. With this dataset, it determined that the best model type would be a deep neural network with attention and had an accurate prediction rate of about 54%. However, like the last experiment, since this accuracy prediction was in-line with the current accuracy predictions we were getting using the Random Forest Classifier, this model was again not pursued.

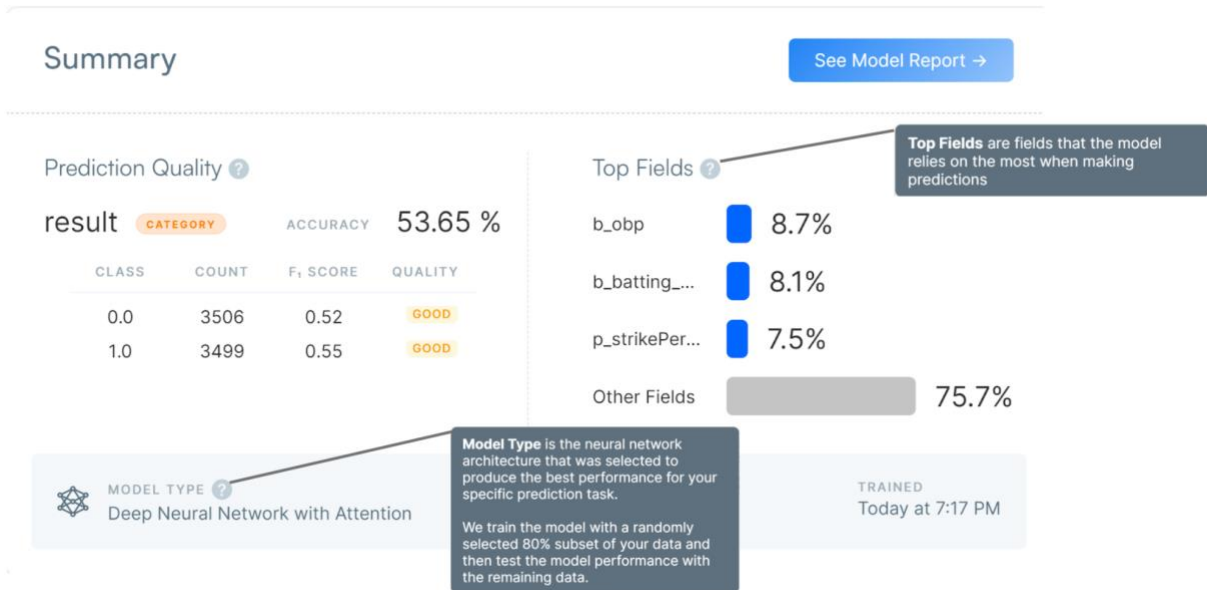


Figure 24: Results of the best model type and the resulting prediction quality on a balanced dataset using Akkio

We also used a program called DataRobot which helped us assess the quality of our data. DataRobot allowed us to set the target variable to the outcome of the play and then it assessed all the other features by correlation to the target variable.

Feature Name	Data Quality	Index	Importance	Var Type	Unique	Missing	Mean	Std Dev	Median	Min	Max
result		4	Target	Numeric	2	0	0.30	0.46	0	0	1
b_obp	i	7		Numeric	237	26	0.32	0.04	0.33	0	1
b_batting_avg	i	6		Numeric	227	26	0.25	0.04	0.26	0	1
b_ops	i	9		Numeric	430	26	0.74	0.12	0.75	0	2
b_slg	i	11		Numeric	346	26	0.42	0.08	0.43	0	1.33
b_babip	i	12		Numeric	202	72	0.30	0.04	0.31	0	1
p_era	i	14		Numeric	379	201	4.31	1.40	4.07	0	36
b_atBats	i	5		Numeric	574	26	1,905	1,733	1,332	0	11,114
p_runsScoredPer9	i	18		Numeric	373	201	4.67	1.43	4.42	0	36
b_gamesPlayed	i	10		Numeric	458	26	553	465	431	1	2,971
p_hitsPer9Inn	i	19		Numeric	404	201	8.51	1.57	8.45	0	40.50
p_strikeoutsPer9Inn	i	17		Numeric	432	201	8.69	1.71	8.67	0	27
p_strikePercentage	i	15		Numeric	29	201	0.64	0.02	0.64	0.27	0.90

Figure 25: Importance of features in comparison to the target variable as per DataRobot

We used this to inform us of the importance of specific features towards the final prediction. DataRobot also scanned our data for any possible target leakage which is when a certain feature indicates the outcome of the target variable.

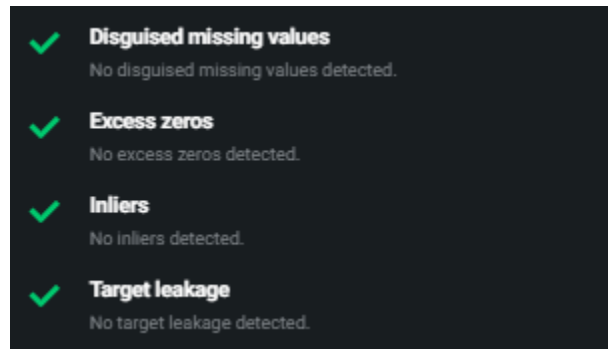


Figure 26: Data Quality assessment of our dataset as per DataRobot

DataRobot allowed us to evaluate our data for data quality issues and allowed us to handle any issues with our data as well. We didn't need to use this feature since we had cleaned and preprocessed our data before we ran machine learning models on them. However, to make sure we put it through DataRobot and were able to clarify its suitability to be analyzed by machine learning models.

4. Results

4.1 Why Unbalanced Data Didn't Work Initially

After implementing our classification models, we discovered a common theme in the result of each of the models. The common theme between the models was that the models were always predicting "out". This problem doesn't help generate money lines using the unbalanced data because if the model is always predicting "out", then it is impossible to generate money lines for results that are "not out". If the model doesn't predict "not out" then the probabilities of "out" and "not out" can't be calculated which is a vital step for developing betting money lines. This result is a key stepping point to eliminate the possibility of using solely unbalanced data and

helped us take a step further in transitioning from model accuracy to generating money lines. After realizing that the unbalanced data wasn't going to work by itself, we moved on to balanced data and implemented a Random Forest Classifier.

4.2 Balanced Random Forest Classifier

One of the main models that we experimented with was a Random Forest Classifier. The implementation of this model was done in two separate forms—both on a balanced version of the dataset and on the raw unbalanced form of the data. This distinction needed to be made due to the model's tendency to predict "out" heavily since that is the outcome of an at-bat about 70% of the time. In order to place more importance upon the features of our dataset rather than the frequency of out occurrences versus not out, we balanced our dataset so that the number of not out instances equaled the number of out instances before implementing our test-train split.

4.2.1 Best Random Forest Depth

With our balanced data set we ran a collection of Random Forest Classifier tests, with varying depths, activation functions, and test/train splits. The following test shows the best results from our testing using an 80/20 test/train split and Gini as the criterion to measure the quality of the splits and all 16 of the collected features.

Figure 27 shows the accuracy of this test on various decision tree depths from 1 to 30. Both the image on the left and the right, have the same hyperparameters. The difference stems from the randomness of the Random Forest Classifier. The two best accuracies recorded were 5 and 11.

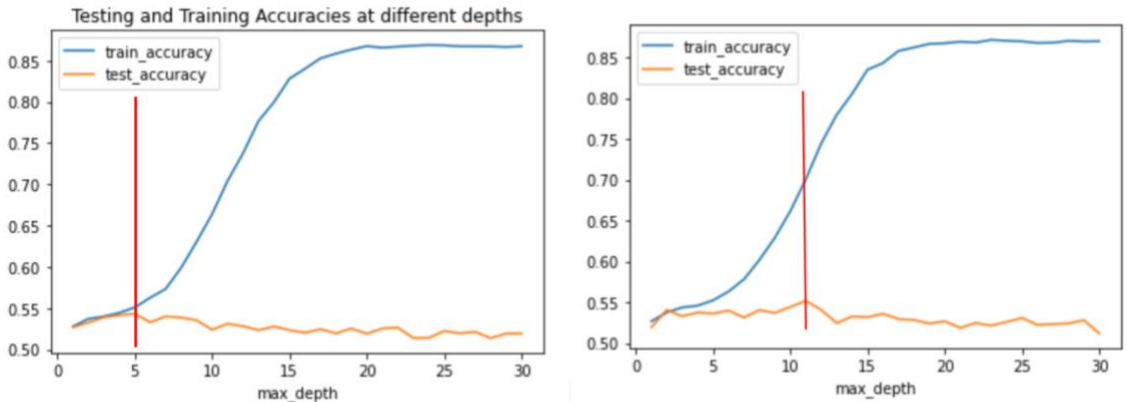


Figure 27: Testing vs. Training Accuracies of Random Forest Depths

Confusion Matrix		
Best Testing Accuracy	Depth = 5	Depth = 11
Training Accuracy	0.551	0.688
Testing Accuracy	0.543	0.549

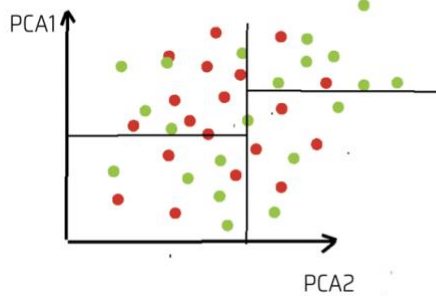
Table 7: Accuracy Comparison of Different Depths

Unfortunately, we were unable to generate money lines from our results the balanced data because the odds being generated did not accurately reflect the true unbalanced results from our dataset. One idea that we had, that we were unable to pursue given time restrictions was using the best hyperparameters uncovered above, to generate a decision tree model on the balanced dataset. However, instead of using the 16 features individually, we would use Principal Component Analysis (PCA) to reduce the dimensionality of our features to 2. This would allow us the ability to graph our results without losing information about our features, and accuracy.

The next step would be to use the decision tree splits generated on the balanced data, as a basis for classifying the unbalanced data. After this, we could use a process similar to KNN by

counting the number of not-outs and outs within a split region as a basis for generating the odds for an at-bat. For example, on the right image of Figure 28, there are four regions, as defined by the decision tree splits. On the upper right region, there is a red (aka Out) point circled in red. The true odds for this at-bat would be calculated by counting the number of red points (Outs) and green points (not outs) in that region. Before these odds are offered by a sportsbook, such as DraftKings, a 5% premium would be applied so that the sportsbook could ensure profit.

Use Balanced Dataset to generate a
Decision tree model



Use Balanced Model to generate
odds on unbalanced dataset

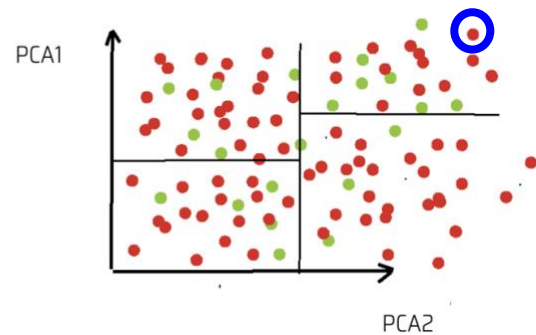


Figure 28: Left - Balanced Model, Right - Unbalanced model

4.2.2 Model Evaluation

Initially, as discussed in section 4.2.1, our team had settled upon using a max depth of 5 as a preventative measure to decrease the risk of overfitting. During our experimentation to determine how adding more features would increase the accuracy, we generated the below confusion matrices.

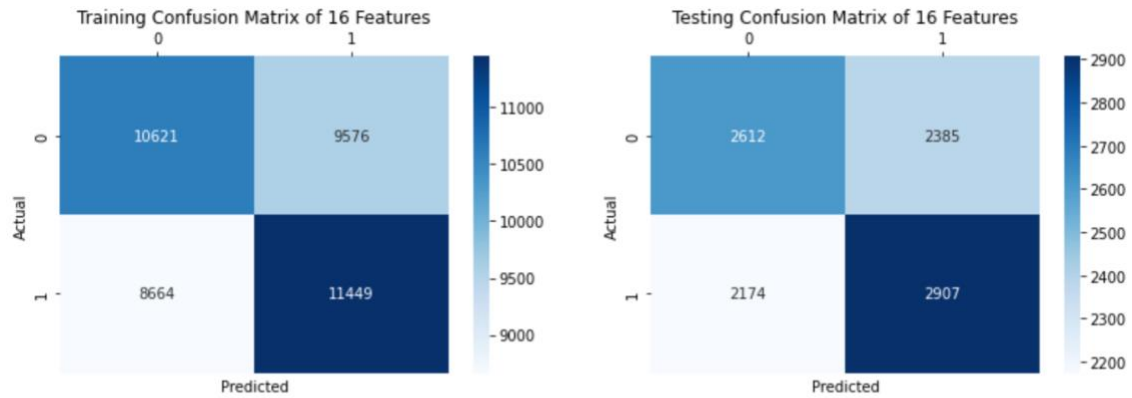


Figure 29: Figure 29a (Left) and Figure 29b (Right): This figure displays both the testing and training confusion matrices for our dataset when we increased the number of features from 4 to 16. The training matrix showcases the balanced nature of the data and its initial predictions, and the right testing matrix shows the predictions on the 20% of the data allocated for testing purposes after the model had learned from the 80% of the data on the left.

In our experiment, we ran the RFC ten times over the same dataset, generating different testing accuracy scores, training accuracy scores, and confusion matrices. The final testing accuracy score of 0.541 and the final training accuracy score of 0.557 was an average of the scores from all ten iterations in the experiment. The confusion matrices, however, are just the samples from the last iteration of the experiment, not an average.

Once we had a dataset with sixteen features to run our models on, we needed to determine the best possible set of features to utilize for our model. To achieve this, a correlation heatmap was created to determine the relationships between all the features.

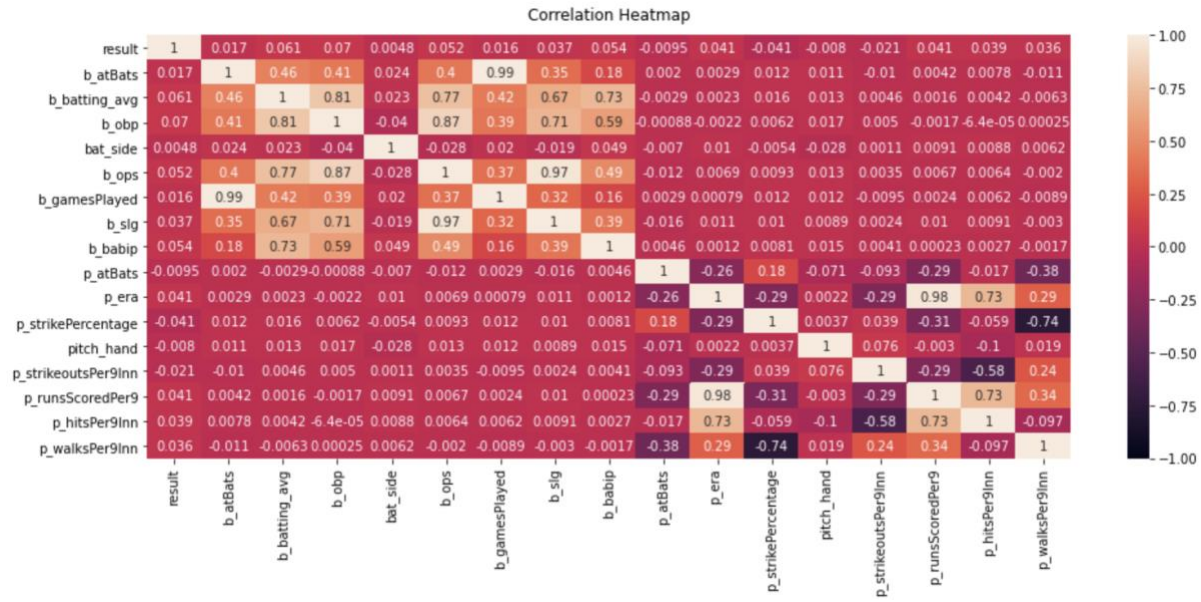


Figure 30: This heatmap shows the correlations between the various features within our dataset.

As expected, features such as batting average and a batter’s OBP had a very high correlation of positive 0.81, and a pitcher’s ERA and runs scored per 9 innings had a correlation of positive 0.98. On the other end, features such as pitcher’s walks per nine innings and strike percentage had a high negative correlation of -0.74. These logical correlations verified the validity of our dataset with the added columns of information.

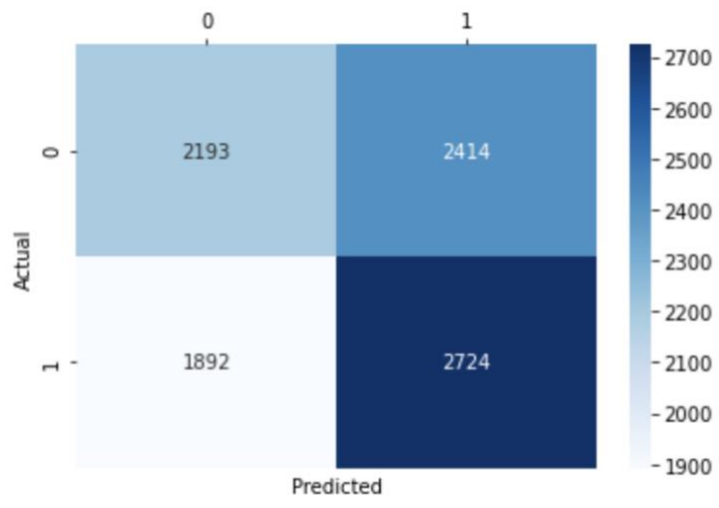


Figure 31: This figure shows the testing confusion matrix of a Random Forest Classifier utilizing 10 features with the highest correlations. A max depth of 10 was used with an 80/20 train/test split, and all correlations were greater than 3%.

With this heatmap and the various correlation coefficients we obtained; our team conducted experiments on which combination of the features would yield the best results. After selecting the ten features with the highest correlation coefficients ($> 3\%$), when a Random Forest Classifier model was used, it was producing an average accuracy of 0.537, with a standard deviation of about 0.004. This is shown in the confusion matrix of Figure 31 above. Out of the 9,223 total outcomes we tested on, the model correctly predicted 2,193 outs and 2,724 not-outs. This model also broke away from the biases we had previously seen where the classifier was leaning towards more heavily predicting out over not out. While an out result is the case nearly 70% of the time in at-bat outcomes, our model predicting not-out more led to it predicting not-out correctly 2724 times, which is where many of our prior models had fallen short. Therefore, our correct out and correct not-out predictions were more balanced, rather than mainly correctly predicting out to receive high accuracy. After comparing the accuracy scores of this experiment to the previously mentioned ones utilizing all 16 features, we determined that using all 16 features generated better results than solely using the features with the highest correlations, and we reverted to using all 16 features.

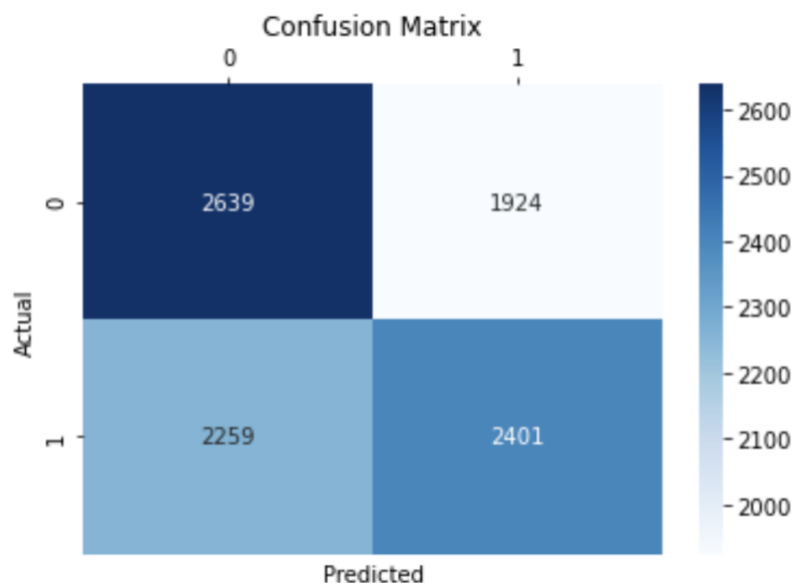


Figure 32: This figure shows the testing confusion matrix of a Random Forest Classifier utilizing the Gini criterion, 100 trees with a max depth of 11, and all 16 of the dataset's features.

As mentioned in section 4.2.1, tests on the max depth of the Random Forest Classifier were done to determine which depth was best regarding both the testing and training data. After this additional experimentation, our team was able to land on an accuracy of about 0.540 with a standard deviation of 0.005 when all sixteen features were used at a max depth of 11. Figure 32 above showcases this result.

4.2.3 Number of Features Versus Accuracy

As with any machine learning task, we used a variety of features in our task of predicting outs vs not-outs. We had a process of adding features to our dataset. We kept the depth constant at 5 using an 80/20 test/train split and Gini as the criterion to measure the quality of the split. This allowed us to change the factors apart from the depth and the split criterion, and experiment with adding more features to affect the prediction accuracy of our machine learning predictions. We started with 2 features with those being handedness of the pitchers and batter's bat side. We ran a Random Forest Classifier with these 2 features.

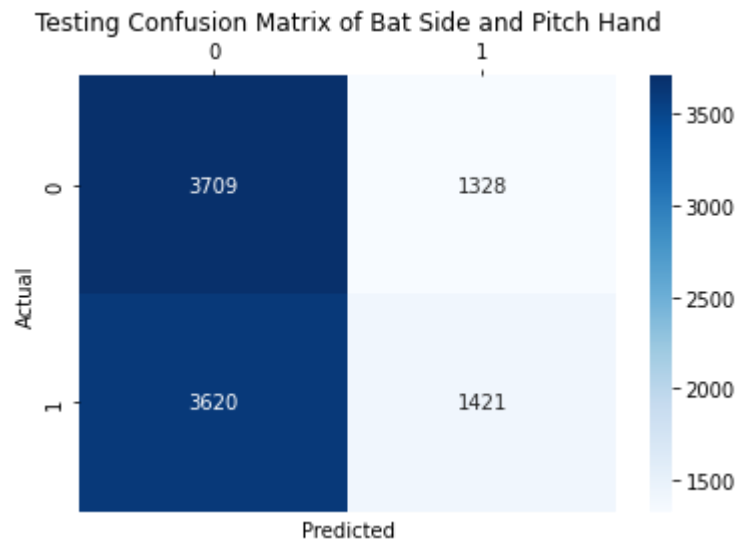


Figure 33: This figure shows the testing confusion matrix of a Random Forest Classifier utilizing the Gini criterion, 100 trees with a max depth of 5, and bat side and pitch hand as features.

As seen in this confusion matrix, the model with bat side and pitch hand tended to predict the outcomes mostly as outs. The model had an accuracy of 50.2% over 10 trial runs. The accuracy of the model is very similar to that of a coin flip with the probability of 50%. This classifier tends to classify most outcomes as out and so as a result we decided to improve upon our dataset and add more features that could improve our accuracy score and get a better classifying classifier.

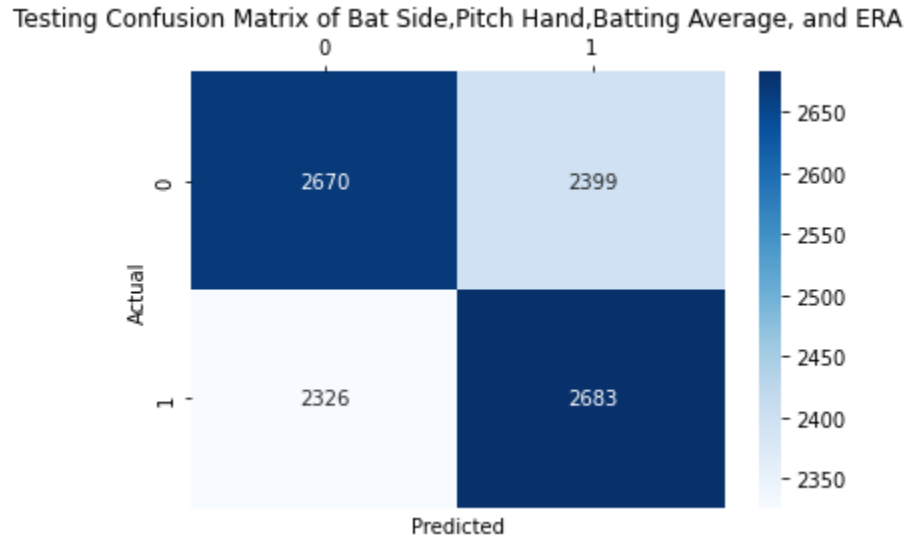


Figure 34: This figure shows the testing confusion matrix of a Random Forest Classifier utilizing the Gini criterion, 100 trees with a max depth of 5, and bat side, pitch hand, batting average, and ERA as features.

This confusion matrix shows that upon adding the two features of batting average and ERA, we were able to improve the accuracy with which the model predicts the out vs not out. In this case, the model was able to predict the real outcome with an accuracy of 53%. It was a whole 3% percent better than the model with just the bat side and pitch hand. This model is better at predicting a not-out outcome compared to the model with 2 features. This made us realize that by possibly adding more features, we could get an even higher accuracy score. We decided to add a lot more features to our Random Forest Classifier.

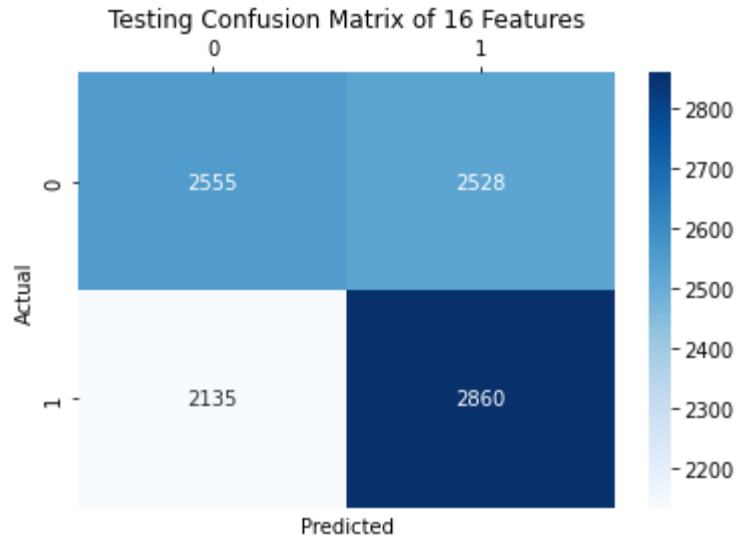


Figure 35: This figure shows the testing confusion matrix of a Random Forest Classifier utilizing the Gini criterion, 100 trees with a max depth of 5. The features include atBats, batting avg, obp, bat side, ops games Played, slg, babip, atBats, era, strike Percentage, pitch hand, strikeoutsPer9Inn, runsScoredPer9, hitsPer9Inn, and walksPer9Inn.

As seen in this figure above, adding more features caused the Random Forest Classifier to classify the outcome of a play around 1% better. The accuracy is 54% which is better than the classifier with 4 features. However, in comparison to the previous Random Forest Classifiers, this Random Forest Classifier is better at classifying “not outs”, however, it does tend to under classify outs compared to the other models with fewer features.

4.3 Pseudo KNN Results for Lines

To create the data for the Pseudo KNN model, we used the unbalanced data. After veering away from the unbalanced data in Section 4.1, we came back to the unbalanced data and discovered a new way that successfully uses the unbalanced data to generate odds and money lines. This was an important step as we went back to the unbalanced dataset from the balanced dataset as we developed a solution to incorporate the unbalanced data. Using the unbalanced data is key because the data isn’t skewed and won’t create any bias when developing a money line. The steps taken to concoct the Pseudo KNN model are described in the steps below.

4.3.1 Scatterplots

After experimenting with different models, we wanted to drift away from model accuracy and shift our focus towards creating money lines. This led us to implement our Pseudo K-nearest neighbor (KNN) model. For the first step of the Pseudo KNN model, we performed a filter on the dataset of 83,843 rows. We filtered out the number of at-bats that were below 100 at-bats. This is because we wanted to ensure that there were enough at-bats for every pitcher and batter to eliminate potential outliers. In the second step, we normalized our original dataset. This was done by using Min-Max Normalization to transform the data shown in Figure 36 below to Figure 37. The reason for doing this was that the different statistics we used had different ranges. Furthermore, we wanted to standardize the ranges of statistics, while maintaining the same distances for each statistic.

	batter_id	pitcher_id	result	b_atBats	b_batting_avg	b_obp	bat_side	b_ops	b_gamesPlayed	b_slg	b_babip	p_atBats	p_era	p_strikePercentage
0	641487	547943	1.0	1355.0	0.250	0.331	1.0	0.698	378.0	0.367	0.303	3729.0	3.20	0.65
1	571745	547943	1.0	1940.0	0.263	0.339	1.0	0.820	507.0	0.481	0.305	3729.0	3.20	0.65
2	572122	547943	1.0	5561.0	0.251	0.321	1.0	0.763	1480.0	0.442	0.272	3729.0	3.20	0.65
3	641343	547943	0.0	977.0	0.213	0.307	0.0	0.655	328.0	0.348	0.273	3729.0	3.20	0.65
4	664238	547943	0.0	716.0	0.204	0.301	1.0	0.685	277.0	0.384	0.265	3729.0	3.20	0.65
...
83838	435559	621368	0.0	5424.0	0.257	0.315	1.0	0.705	1584.0	0.390	0.272	456.0	4.80	0.63
83839	641432	676083	0.0	802.0	0.247	0.301	1.0	0.715	235.0	0.414	0.260	68.0	3.86	0.66
83840	643396	676083	0.0	1404.0	0.265	0.316	1.0	0.670	392.0	0.354	0.307	68.0	3.86	0.66
83841	666969	676083	0.0	604.0	0.237	0.280	1.0	0.722	173.0	0.442	0.301	68.0	3.86	0.66
83842	665120	621368	0.0	708.0	0.271	0.331	0.0	0.839	207.0	0.508	0.323	456.0	4.80	0.63

Figure 36: Screenshot of Our Dataset

	batter_id	pitcher_id	result	b_atBats	b_batting_avg	b_obp	bat_side	b_ops	b_gamesPlayed	b_slg	b_babip	p_atBats	p_era
0	0.848217	0.470330	1.0	0.113946	0.755814	0.717877	1.0	0.657385	0.117427	0.567925	0.629630	0.310808	0.169753
1	0.597653	0.470330	1.0	0.167060	0.806202	0.740223	1.0	0.805085	0.161334	0.783019	0.637037	0.310808	0.169753
2	0.599007	0.470330	1.0	0.495823	0.759690	0.689944	1.0	0.736077	0.492512	0.709434	0.514815	0.310808	0.169753
3	0.847700	0.470330	0.0	0.079626	0.612403	0.650838	0.0	0.605327	0.100408	0.532075	0.518519	0.310808	0.169753
4	0.929956	0.470330	0.0	0.055929	0.577519	0.634078	1.0	0.641646	0.083050	0.600000	0.488889	0.310808	0.169753
...
83835	0.905953	0.966794	0.0	0.167423	0.720930	0.664804	1.0	0.727603	0.172566	0.711321	0.551852	0.001627	0.308642
83836	0.915704	0.753051	0.0	0.118032	0.810078	0.779330	1.0	0.889831	0.127978	0.888679	0.696296	0.030490	0.334362
83837	0.680620	0.753051	1.0	0.084801	0.798450	0.670391	1.0	0.629540	0.146358	0.558491	0.700000	0.030490	0.334362
83838	0.108371	0.753051	0.0	0.483385	0.782946	0.673184	1.0	0.665860	0.527910	0.611321	0.514815	0.030490	0.334362
83842	0.933125	0.753051	0.0	0.055202	0.837209	0.717877	0.0	0.828087	0.059224	0.833962	0.703704	0.030490	0.334362

Figure 37: Screenshot of Our Normalized Dataset

4.3.1.1: OBP vs. ERA

Next, we then proceeded to create scatter plot visualizations to compare two statistics, one for a batter and one for a pitcher. The two statistics we closely examined were On-Base Percentage (OBP) and Earned Runs Average (ERA). These statistics were later used for generating the money lines. The distribution of the two features is displayed in Figure 38 below along with the corresponding result.

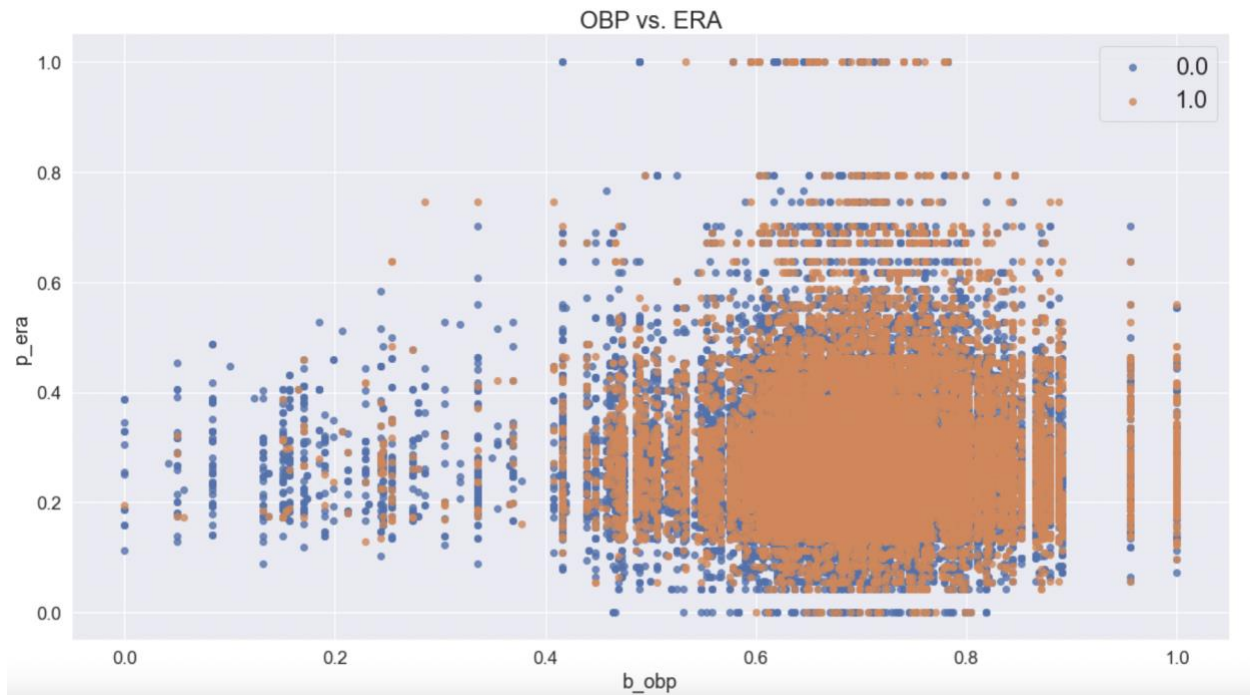


Figure 38: OBP vs. ERA Scatterplot

4.3.1.2: Batting Average vs. ERA

We then proceeded to examine other feature combinations as well. In Figure 39 shown below, we decided on using batting average for the batter and earned runs allowed (ERA) for the pitcher. In this scatter plot we also analyzed the points to be out or not an out.

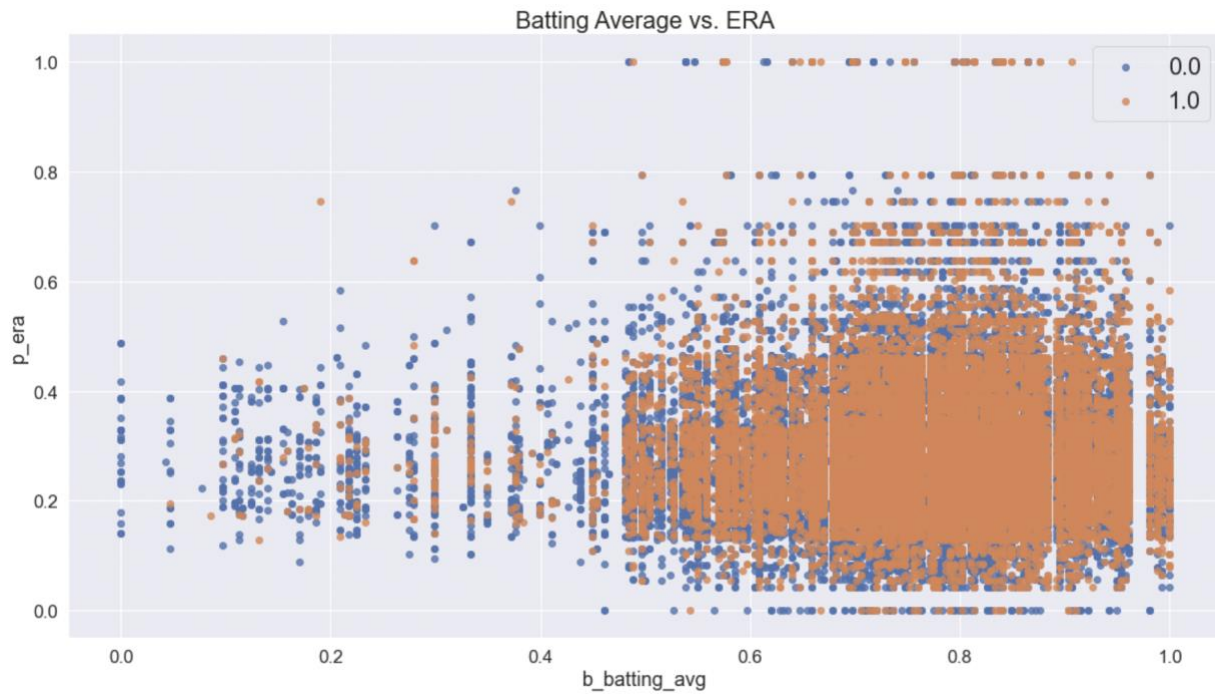


Figure 39: Batting Average vs. ERA Scatterplot

4.3.1.3: Batting Average vs. Strike Percentage

The next scatterplot we used was batting average and strike percentage. As shown in Figure 40 below, it had a slightly different appearance than the graph above where it was shifted upwards.

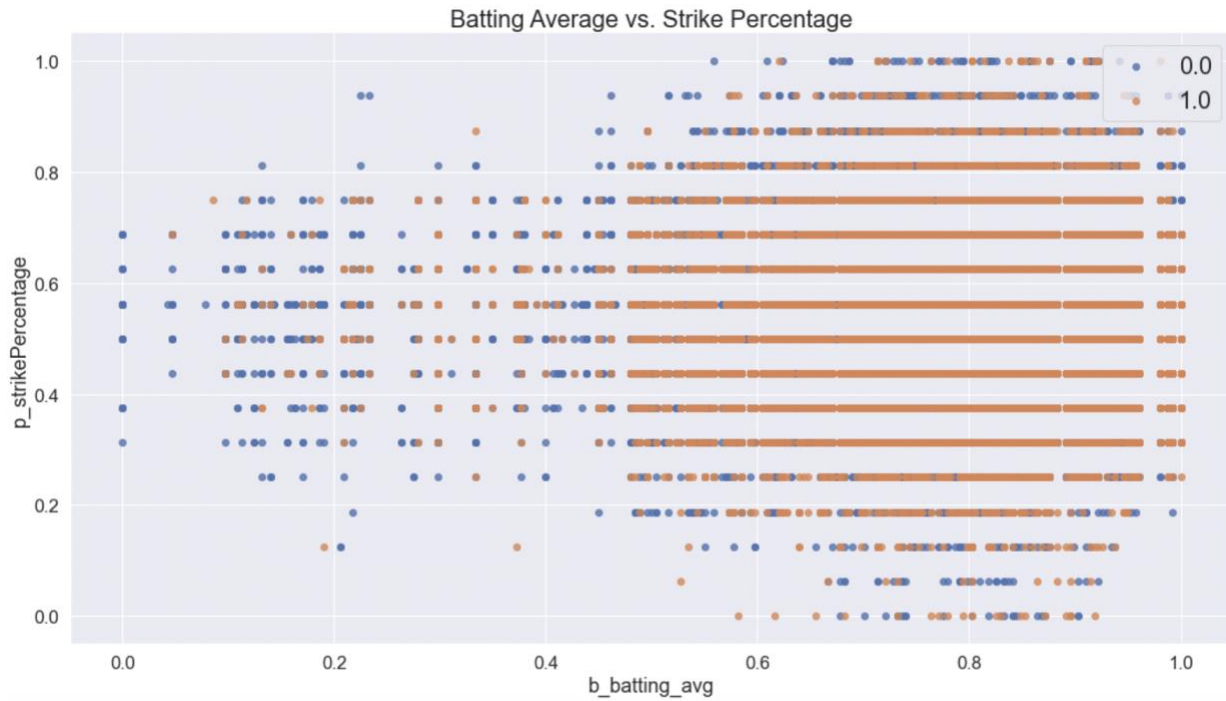


Figure 40: Batting Average vs. Strike Percentage Scatterplot

4.3.1.4: OBP vs. Strike Percentage

Lastly, we chose on-base percentage (OBP) and strike percentage as the next two statistics we wanted to explore (Displayed in Figure 41 below). This scatter plot had a relatively similar shape to the batting average and strike percentage scatter plot. This followed our thinking because on-base percentage and batting average are relatively similar statistics.

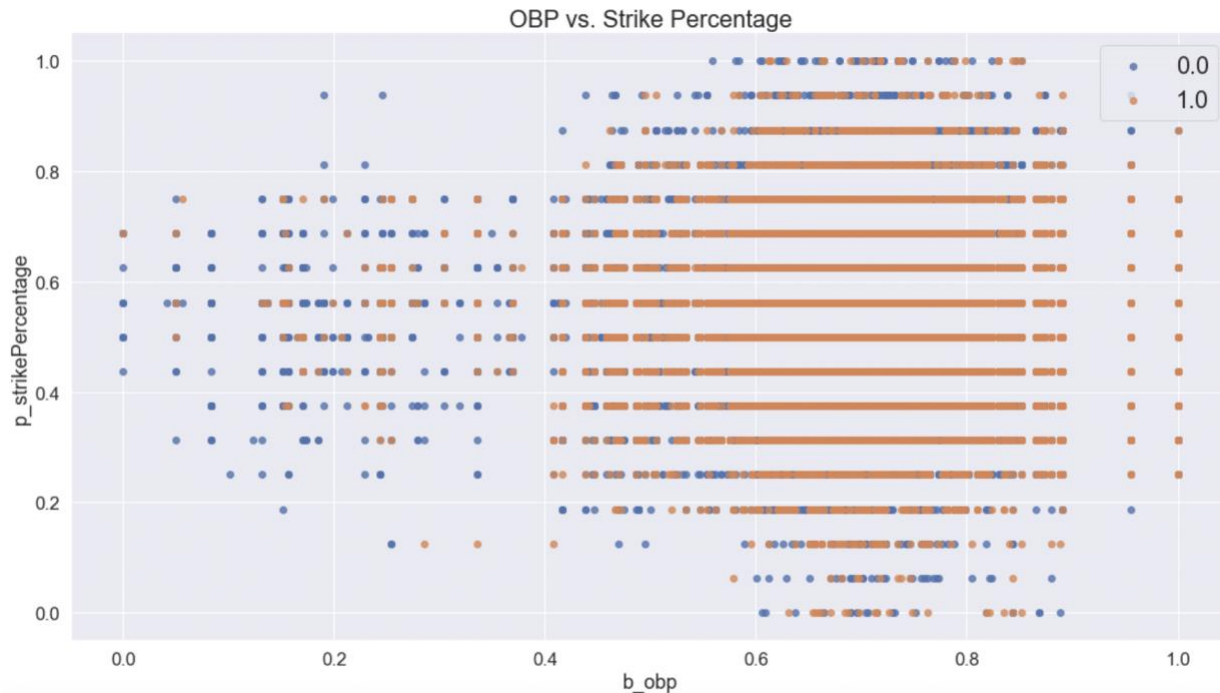


Figure 41: OBP vs. Strike Percentage Scatterplot

After examining the scatterplots, we picked a point on the scatter plot as the input used to calculate the money lines. Next, we then drew a circle around the point with a radius equaling 0.1. The value for the radius derives from the Z-transformed length of the OBP and ERA statistics. We then proceeded to and counted each point within that circle that was either “out” or “not out”. By counting each vote within the circle, we then were able to generate a probability for “out” or “not out”. These probabilities are then used in the money line equation from the Logic of Sports Betting (Miller, E., & Davidow, M., 2019). Using an initial hold percentage of 0.05, we were able to then generate the money line for “out” or “not out” for a given pitcher and batter matchup.

4.3.2 Comparisons with DraftKings Lines

Upon following the generation of the money lines, we wanted to compare the money lines we generated with our model with the DraftKings money lines. As shown in Figure 42 below, we took screenshots of the live prop bets DraftKings created for the Boston Red Sox and New York Yankees playoff matchup. In this screenshot, readers can see the matchup between

Enrique (Kike) Hernandez, a hitter for the Red Sox, against Gerrit Cole, a pitcher for the Yankees. Enrique Hernandez had an OBP of 0.337 for the 2021 season, while Gerrit Cole had an ERA of 3.23 for the 2021 season (*MLB stats, scores, History, & Records Baseball Reference*). This money line is for a 1st plate appearance for Enrique Hernandez in the first inning. A 1st plate appearance is equivalent to what we are looking for: “out” or “not out”. If there is a 1st plate appearance, that means that the batter either got a ground hit, home run, or walked. If there isn’t a 1st plate appearance, that means the batter either received a strikeout or fly out. Therefore, the odds that DraftKings set were +180 for “not out” and -250 for “out”.

Enrique Hernandez - 1st Plate Appearance - 1st Inning - On Base (vs. Gerrit Cole)	
Yes	+180
No	-250

Figure 42: Money Line for Enrique Hernandez vs. Gerrit Cole (Featured betting odds & lines: DraftKings Sportsbook, n.d.)

To compare the DraftKings money line results with our money line results from our model, we looked at different money line calculations based on different hold percentages. A hold percentage is the profit a casino or Sportsbook like DraftKings makes on average. Illustrated in Table 8 below, we experimented with the hold percentage values ranging from 0.01 to 0.09. The closest money line from our model to the DraftKings money line is highlighted in green where the hold percentage equates to 0.05. Conversely, the farthest money line from our model to DraftKings is highlighted in red, which was a hold percentage of 0.09. In this example of Enrique Hernandez and Gerrit Cole, our model’s money line was not too far off from the DraftKings money line from the screenshot above.

Enrique Hernandez (0.337 OBP) vs. Gerrit Cole (3.23 ERA) Table

Hold %	Not Out Moneyline	Out Moneyline
0.01	+227	-247
0.02	+217	-255
0.03	+207	-265
0.04	+198	-275
0.05	+189	-287
0.06	+181	-299
0.07	+173	-312
0.08	+166	-326
0.09	+159	-342

Table 8: Created Money Lines for Different Hold Percentages

In another comparison, we compared Gleyber Torres and Nathan Eovaldi. Gleyber Torres is a batter for the Yankees, while Nathan Eovaldi is a pitcher for the Red Sox. Gleyber Torres had a 2021 season OBP of 0.331, while Nathan Eovaldi had a 2021 season ERA of 3.75 (*MLB stats, scores, History, & Records*). Similar to the screenshot example above, the prop bet is asking whether or not the result of the at-bat is “out” or “not out”. The odds for “not out” is +210, while the odds for “out” is -295 (Shown in Figure 43 below).

Gleyber Torres - 1st Plate Appearance - 2nd Inning - On Base (vs. Nathan Eovaldi)	
Yes	+210
No	-295

Figure 43: Money Line for Enrique Hernandez vs. Gerrit Cole (Featured betting odds & lines: DraftKings Sportsbook, n.d.)

In conjunction with the Enrique Hernandez and Gerrit Cole table of hold percentage comparisons, we generated a similar table for Gleyber Torres and Nathan Eovaldi. Displayed in Table 9 are the money lines for “out” and “not out” for each of the hold percentages from 0.01 to

0.09. As shown in the green highlighted text, the hold percentage of 0.08 was closest to the money line produced by DraftKings while the red highlighted text was the furthest away from the DraftKings money line. In this comparison, the “not out” money line wasn’t too close to the DraftKings money line but the “out” money line from our model was extremely close to the DraftKings money line.

Gleyber Torres (0.331 OBP) vs. Nathan Eovaldi (3.75 ERA) Table

Hold %	Not Out Moneyline	Out Moneyline
0.01	+208	-225
0.02	+199	-233
0.03	+190	-241
0.04	+182	-249
0.05	+174	-259
0.06	+167	-269
0.07	+160	-280
0.08	+153	-292
0.09	+147	-305

Table 9: Created Money Lines for Different Hold Percentages

4.3.3 Principal Component Analysis (PCA)

To follow up on creating money lines through our version of a KNN, we used a Principal Component Analysis (PCA) model. The money lines through KNN used all of the 16 features of the dataset, 8 for batters and 8 for pitchers. We wanted to further examine the money lines by creating a PCA where we used two components. The reason for using a PCA was to perform dimensionality reduction on the numerous information. While reducing the dimension of the dataset, the PCA model also keeps the most important variables that impact our target variable,

which was the result of an at-bat. As displayed in Figure 44 below, the two principal components are plotted against each other with the result of the at-bat as well.



Figure 44: PCA Scatterplot

After generating the scatterplot for the two principal components, we used a similar procedure to generate the money lines from the KNN method above. After picking a given point on the scatterplot, we counted the number of votes for “out” or “not out” within the circle of that point (Shown in Figure 45 below). After that, we used the same methodology to generate probabilities and produce money lines for “out and “not out” as shown in the DataFrame figure below. This DataFrame is a small screenshot of the larger dataset, where we generated lines for all 77,415 rows of the dataset.

	pca1	pca2	result	odds_out	odds_not_out	out_count	not_out_count
0	0.788582	-1.676962	1	198.226	-303.712	6415	2561
1	-0.924717	-1.676376	1	198.115	-303.498	5983	2390
2	-0.988990	-1.676025	1	198.058	-303.388	5861	2342
3	2.301168	-1.692179	0	229.188	-366.195	2173	739
4	2.307517	-1.660483	0	228.697	-365.158	2203	751
5	-1.885200	1.264318	0	157.912	-230.172	2712	1383
6	-1.920515	1.267665	0	158.464	-231.125	2602	1322
7	-0.975015	1.274100	0	160.625	-234.870	4329	2168
8	2.543903	-1.660767	0	233.217	-374.751	1784	595
9	2.283080	-1.663811	0	231.440	-370.964	2292	771
10	0.788582	-1.676962	1	206.115	-319.103	5464	2090

Figure 45: PCA DataFrame

4.4 Decision Tree Betting Odds Generator

4.4.1 Decision Tree Splits Diagram

To illustrate how the decision tree betting odds generator process worked, we created diagrams as shown in Figure 46 below. In this process, the data was split into the decision tree cutoffs as shown in the lines below. After visualizing the cutoffs, we then proceeded to count each matchup for a class within the given sections to determine the probability of each result. From there, we then proceeded to convert the probabilities to betting odds.

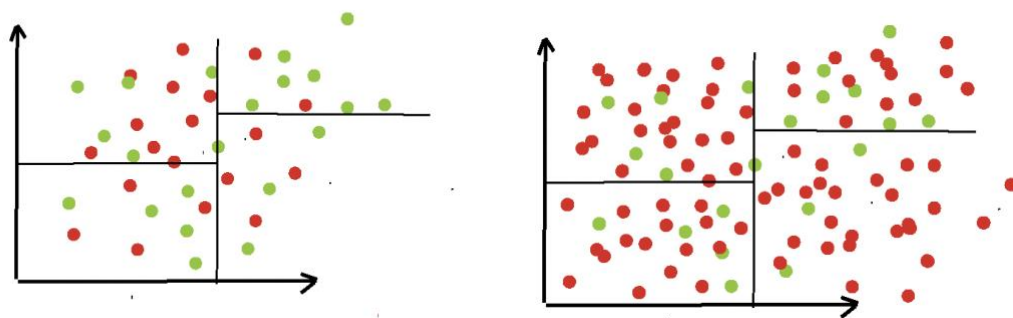


Figure 46: Decision Tree Split Visualization

4.4.2 Decision Tree Results

We ran two similar tests to generate betting odds using the decision tree classifier. In both examples, the criterion for measuring the quality of split is “Gini”. In the first test, the minimum number of samples required to be at a leaf node of the model was set to 5000. As a reminder, the decision tree model was generated on a manually balanced subset of the data. The model generated can be seen in the images below where features 2, 3 are on-base percentage, batting hand of the batter, and feature 13 is p_runsScoredPer9 which is the number of runs scored against a pitcher per nine innings.

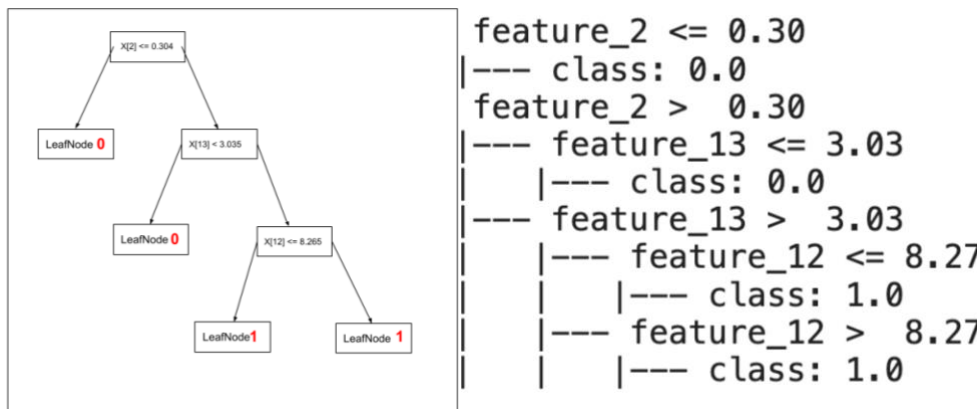


Figure 47: Decision Tree Splits

The following is an example of odds generated from the model above for pitcher and batter matchups. Yes, indicates that the batter (in this case Enrique Hernandez) will make it on first base and No means the batter will not make it to first place. Later, in the results, there will be a section comparing these odds with the odds set on the Draft Kings website.

Enrique Hernandez vs. Gerrit Cole (Matchup Betting Odds, Using 5% Hold Percentage)

DraftKings Odds	Yes +180	No -250
------------------------	-----------------	----------------

Figure 48: DraftKings Generated Odds

For the next iteration of using decision tree split nodes to generate odds on prop bets, we used a nearly identical hyperparameter criterion as the previous decision tree. The only change is

the criterion for a decision tree node to continue requiring a split. In this result, a tree node would continue to split until the number of samples in a leaf was less than or equal to 1000. This resulted in a decision tree model with a larger depth and imbalance. Since the number of splits and features used is much larger than in the previous example, it is not easily visualized. The following is a simplified version of what the decision tree model generated looks like.

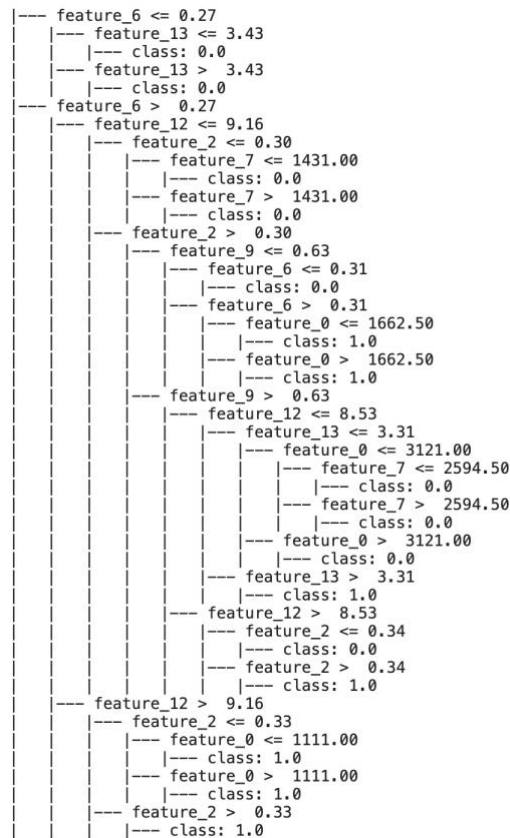


Figure 49: Decision Tree Splits

The following figures are some examples of odds generated from the model above for pitcher and batter matchups in the MLB produced by the updated Decision Tree model. Figure 50 below shows the odds generated from our Decision Tree model for Amed Rosario versus Jake Diekman.

Amed Rosario vs. Jake Diekman (Matchup Betting Odds, Using 5% Hold Percentage)

Decision Tree Odds	Yes +172	No -256
---------------------------	-----------------	----------------

Figure 50: DraftKings Odds Comparison (Rosario vs. Diekman)

Another matchup we looked at was Austin Riley and Javy Geurra, where the generated odds are shown in Figure 51 below.

Austin Riley vs. Javy Guerra (Matchup Betting Odds, Using 5% Hold Percentage)

Decision Tree Odds	Yes +193	No -294
---------------------------	-----------------	----------------

Figure 51: DraftKings Odds Comparison (Riley vs. Guerra)

Lastly, we examined Ji-Man Chio and Spencer Watkins and generated the betting odds depicted in Figure 52 below.

Ji-Man Chio vs. Spencer Watkins (Matchup Betting Odds, Using 5% Hold Percentage)

Decision Tree Odds	Yes +142	No -204
---------------------------	-----------------	----------------

Figure 52: DraftKings Odds Comparison (Chio vs. Watkins)

4.5 Pseudo KNN and Decision Tree Result Comparison

To compare the two betting odds generators, we took a closer look between the Pseudo-KNN model and the Decision Tree model. To compare our models, we analyzed them against the odds that were generated by DraftKings. As shown in Table 10 below, the Pseudo-KNN came closest to the DraftKings generated odds. On the other hand, the odds that were furthest away from the DraftKings odds was the Decision Tree model that had a minimum sample leaf size of 1000.

Enrique Hernandez vs. Gerrit Cole (Matchup Betting Odds, Using 5% Hold Percentage)

DraftKings Odds	Yes +180	No -250
Pseudo-KNN Odds	Yes +189	No -287
Decision Tree Odds (Min Sample Leaf of 5000)	Yes +191	No -290
Decision Tree Odds (Min Sample Leaf of 1000)	Yes +202	No -312

Table 10: All Model Comparison with DraftKings on Generated Betting Odds

After completing some of the experiments with the Pseudo-KNN a Decision Tree models, we would recommend to further pursue the Psuedo-KNN model based on the knowledge we know. This is because the Pseudo-KNN model didn't have any major limitations, while the Decision Tree model did. With the Decision Tree model, a major limitation was that based on the number of splits there is only that number of odds generated per split. For example, if there is a split of four, there will only be four odds generated. When more betting odds need to be generated, this limitation is not ideal where it can only produce a few betting odds. However, to address this issue, increasing the number of splits of the Decision Tree will allow for more distinct odds to be produced. When increasing the number of splits, one needs to be conscious of the amount of data that they have access to. This becomes relevant when the number of points in leaf nodes becomes dangerously low. Finding the right balance when building the decision tree model is critical.

5. Conclusion

Initially, the lines were based on the pseudo-KNN model due to which we were able to generate lines with the dimensionally reduced features for batting and pitching features for all 77,415 plays in our dataset. We were able to get lines that were very close to the example DraftKings lines we had for reference. However, we also wanted to generate lines based on the decision trees model machine learning model and then compare them to the lines we obtained by using the pseudo-KNN model. We decided to also implement a decision tree model which would provide a comparative set of lines to evaluate our pseudo-KNN lines against.

5.1 Takeaways

As we finished up our project, we had a few takeaways from our analysis. We had takeaways on everything from the data processing to the solution to the problem. Our data was very unbalanced after we extracted it from the MLB API, when we ran machine learning models on the data we had, the models tended to predict outcomes skewed towards predicting outcomes.

However, when we balanced the data and then ran the models predicting on unbalanced data, our models performed better in predicting outs and not-outs thus reducing the skewness of the predictions the models made.

We were working with Random Forest Classifiers with balanced data to accurately predict the outcome of a specific play. We then planned on using these predictions to set lines. However, we decided to take a step back and use a simple scatter plot to run a KNN analysis. This simpler approach allowed us to use the unbalanced dataset to generate odds based on the features. We took an approach of simplifying the data and filtering instead of balancing in this case and were able to generate lines like the DraftKings money lines.

However as mentioned above, we realized upon simplifying our approach from Random Forest classifier to a decision tree split model, that there is a tradeoff between accurate odds and simplicity of the tree and further splitting of the data manually using more trees and bigger trees will give us more accurate odds at the cost of possibly overfitting.

5.2 Recommendations & Future Work

As with any experiment and analysis, there is a scope for improvement in our work. This section discusses the possible improvements that can be made towards the objective of setting odds for “out” versus “not out”. In this section, we look at some ways that our work can be improved and built upon. In terms of future work, as mentioned before the lines generated were based on the KNN and Decision Tree machine learning models. We did not reach a point in our analysis in which we were able to use the Random Forest machine learning model to set lines. Additionally, machine learning models apart from Random Forest Classifiers and KNN could also be used to generate lines from the dataset. Our dataset is very comprehensive and has a lot of data points and so the data can be used with more complex models and possibly better data cleaning and sampling.

In our analysis of the data, we were unable to use non-baseball factors for our predictions. Some of the non-baseball factors we discussed included weather, sunlight, current, and past events, fan support, and even political indicators. The weather with the wind, humidity, and precipitation at the time of a certain play can all affect the pitch played to a batter. The

sunlight similarly affects the temperature and visibility conditions during a play and can affect the quality of the pitch played.

Current and past events, political indicators, and fan support can be used as an indicator of the atmosphere in the game. While they can be good indications of factors as they affect player performance, it is hard to numerically quantify them and so we were not able to use these other factors in our analysis. These factors are intangibles that make a difference to player performance to a high extent. For this, we thought that a game state dataset could be useful to the project as it would consider the factors such as game events before a specific on-bat and other in-game and out-of-game events into account. However, due to not being able to numerically quantify the intangibles and not being able to find a consistent data source for the game state, we could not create the game state dataset.

Finally, the last future work that we think could be worked upon is the complexity of the Decision Tree models we used. We based our decision tree model's splits of data based on the number of samples left in the leaf and since we had so much data, our splits were limited in number thus giving us fewer odds and bigger group splits. To counter this, a bigger tree with more leaf nodes, and possibly more trees together to form a random forest model could be used. However, the issue was the high number of features and the amount of datapoints that we did have made it hard to do so. As a result, our main recommendation would be to improve upon our decision tree model and provide a better comparison of odds with the pseudo-KNN model.

In conclusion our main recommendations are to use other machine learning models to classify the outcomes, use other external factors in the classification problem, and finally to use more complex trees and maybe even random forest models to create more splits to generate more accurate odds.

References

- Acharya, S. (2021, June 15). *What are RMSE and Mae?* Medium. Retrieved March 19, 2022, from <https://towardsdatascience.com/what-are-rmse-and-mae-e405ce230383>
- Alderson, S., Antonetti, C., Bernabe, S., Daniels, J., Dipoto, J., Gorman, B., Mozeliak, J., Schuerholz, J., & Torre, J. (n.d.). *2019 Official Baseball Rules 2019 official ... - mlb.com*. Major League Baseball. Retrieved March 24, 2022, from [https://content.mlb.com/documents/2/2/4/305750224/2019 Official Baseball Rules FIN AL .pdf](https://content.mlb.com/documents/2/2/4/305750224/2019_Official_Baseball_Rules_FIN_AL.pdf)
- Burns, E. (2021, March 30). *What is machine learning and why is it important?* TechTarget. Retrieved March 19, 2022, from <https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML>
- Chávez, G. (2019, December 26). *Understanding logistic regression step by step*. Medium. Retrieved March 22, 2022, from <https://towardsdatascience.com/understanding-logistic-regression-step-by-step-704a78be7e0a>
- Chakure, A. (2020, November 6). *Random Forest Classification and its implementation*. Medium. Retrieved March 24, 2022, from <https://medium.com/swlh/random-forest-classification-and-its-implementation-d5d840dbead0>
- Featured betting odds & lines: DraftKings Sportsbook*. Bet Online Legally and Safely with DraftKings Sportsbook. (n.d.). Retrieved March 2, 2022, from https://Sportsbook.DraftKings.com/featured?_gl=1%2A100r6ey%2A_ga%2AMjA1NjQzNzYzLjE2NDYwMTc5ODM.%2A_ga_QG8WHJSQMj%2AMTY0NjI3OTYzMi4yLjAuMTY0NjI3OTYzNC4w&_ga=2.200248685.212265551.1646279633-205643763.1646017983
- Gandhi, R. (2018, June 7). *Support Vector Machine — Introduction to Machine Learning Algorithms*. Towards Data Science. Retrieved March 18, 2022, from <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>
- Great Learning Team. (2020, February 19). *Random forest Algorithm in Machine*

- learning: An Overview* [Review of *Random forest Algorithm in Machine learning: An Overview*]. Great Learning; Great Learning.
<https://www.mygreatlearning.com/blog/random-forest-algorithm/>
- Grootendorst, M. (2019, September 26). *Validating your machine learning model*. Towards Data Science. Retrieved March 19, 2022, from <https://towardsdatascience.com/validating-your-machine-learning-model-25b4c8643fb7>
- Hanafy, Mohamed & Ming, Ruixing. (2021). *Machine Learning Approaches for Auto Insurance Big Data*. *Risks*. 9. 42. 10.3390/risks9020042.
https://www.researchgate.net/figure/Structure-of-a-general-decision-tree_fig1_349499774
- Hargrave, M. (2022, March 17). *What is the standard deviation?* Investopedia. Retrieved March 23, 2022, from <https://www.investopedia.com/terms/s/standarddeviation.asp>
- Harrison, O. (2019, July 14). *Machine learning basics with the K-Nearest Neighbors algorithm*. Medium. Retrieved March 18, 2022, from <https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761>
- IBM Cloud Education. (2020, August 17). *What are neural networks?* IBM. Retrieved March 24, 2022, from <https://www.ibm.com/cloud/learn/neural-networks#:~:text=Neural%20networks%2C%20also%20known%20as,neurons%20signal%20to%20one%20another.>
- Jurafsky, D., & Martin, J. H. (2020). Chapter 5: Logistic Regression. In *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. essay, Pearson, from <https://web.stanford.edu/~jurafsky/slp3/5.pdf>
- Korstanje, J. (2021, August 31). *The F1 score*. Medium. Retrieved March 19, 2022, from <https://towardsdatascience.com/the-f1-score-bec2bbc38aa6>
- Kulkarni, A. (2020). *Foundations of data imbalance and solutions for a data democracy*. ScienceDirect Topics. Retrieved March 22, 2022, from <https://www.sciencedirect.com/topics/engineering/confusion-matrix>
- Kumar, B. (2022, January 7). *Scikit learn accuracy_score*. Python Guides. Retrieved March 19, 2022, from <https://Pythonguides.com/scikit-learn-accuracy-score/>

Learn. scikit. (n.d.). Retrieved April 6, 2022, from <https://scikit-learn.org/stable/>

Malik, F. (2021, March 4). *Neural networks bias and weights*. Medium. Retrieved March 24, 2022, from <https://medium.com/fintechexplained/neural-networks-bias-and-weights-10b53e6285da>

Miller, E., & Davidow, M. (2019). How Odds are Listed. In *The logic of sports betting* (pp. 21–21). essay, Ed Miller.

Milton, J. (2019, July 19). *History of sports betting*. Retrieved March 20, 2022, from <https://www.bigonsports.com/history-of-sports-betting/>

MLB stats, scores, History, & Records. Baseball. (n.d.). Retrieved March 2, 2022, from <https://www.baseball-reference.com/>

Narkhede, S. (2018, May 9). *Understanding Confusion Matrix*. Medium. Retrieved March 19, 2022, from <https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62>

Neural network models. (2022). SciKit Learn https://scikit-learn.org/stable/modules/neural_networks_supervised.html

Sharma, A. (2021, March 1). *Decision tree algorithm for classification: Machine Learning 101*. Analytics Vidhya. Retrieved March 22, 2022, from <https://www.analyticsvidhya.com/blog/2021/02/machine-learning-101-decision-tree-algorithm-for-classification/>

Sportsbook / Baseball Odds / MLB Odds. (2022). DraftKings <https://sportsbook.draftkings.com/live>

Tatevosian, P. (2021, June 18). *2 reasons why DraftKings can grow more easily than you think. The Motley Fool*. Retrieved March 20, 2022, from <https://www.fool.com/investing/2021/06/18/2-reasons-why-DraftKings-can-grow-more-easily-than/>

toddrob99. (2020, March 16). *MLB-Statsapi*. GitHub. Retrieved March 18, 2022, from <https://github.com/toddrob99/MLB-StatsAPI/wiki>

Trevisan, V. (2022, January 11). *Comparing robustness of Mae, MSE and RMSE*. Medium. Retrieved March 23, 2022, from <https://towardsdatascience.com/comparing-robustness-of-mae-mse-and-rmse->

[6d69da870828#:~:text=Usually%20the%20metrics%20used%20are,the%20average%20over%20all%20observations](#)

What Are Prop Bets? | How To Place A Prop Bet And Find The Best Odds. (n.d.). The Lines.

Retrieved March 18, 2022, from <https://www.thelines.com/betting/prop-bets/>

Who We Are | About DraftKings. (n.d.). www.DraftKings.com.

<https://www.DraftKings.com/about/who-we-are/>

Wikimedia Foundation. (2022, March 12). *DraftKings*. Wikipedia. Retrieved March 21, 2022,

from <https://en.wikipedia.org/wiki/DraftKings>

(n.d.). *Akkio*. computer software. Retrieved from <https://www.akkio.com/>.