Baseball, Exploratory Data Analysis and Machine Learning:

A Beginning Look at ML with America's Past-time

Kyle Miller

April 15, 2020

1 Abstract

Baseball's vast and detailed data sets are prime candidates for Exploratory Data Analysis (EDA) and Machine Learning (ML) tools. With the rise of ML in the public consciousness it is understandable to ask what can these tools do for us? Can I develop a model that can beat the odds and make me some money in Vegas? Or perhaps, as a fan of the game, you wish to simply understand the sport better. While, unfortunately, there is no golden bullet here, there is much to be learned about both ML and baseball. This paper will walk you through the basics of different modeling types and processes, using the python package SciKit Learn (Sklearn). This paper explores most major ML models in an attempt to answer two main questions: Can we predict the outcome of a baseball game? Can we predict the outcome of a season? The answer to these questions are, in order: not much better than guessing, and actually pretty well. More mathematically, we'll see that our best models predict the outcome of a baseball game with $\sim$1.5% greater accuracy than a naive guess and the outcome of a season can be predicted with $\sim$86% accuracy.

This paper is a beginner's guide, constructed for a term project in a beginning machine learning course at Utah Valley University, so in addition to procedure and results this paper will also make note of common tools, jargon and pitfalls, meant to help the young data scientist in their journey. These supplements can be found in the attached appendices. There is a lot to learn with machine learning and it can definitely be overwhelming, but it is also fun - a real treasure hunt. And what better way to learn than by taking a dive into America's Past-time? All data sets and code, in the form of Jupyter Notebooks, can be found in the attached zip file.

2 Introduction

Major League Baseball (MLB) has been collecting statistics in various forms since perhaps the very inception of the game.[1] There are massive repositories of data, even granular stats like the order of balls and strikes in a specific at bat. These statistics have, for the majority of the sport's history, been not much more than a set of talking points for enthusiasts and broadcasters. But since the advent of Sabermetrics in the early 2000's, statistical analysis has been used to build entire teams.[1] At this point in the game, using statistical models in baseball is commonplace. We will not be attempting to analyze the minutia of the game in this paper, though it may certainly be worth your time. Instead, for this project we will be focusing on the basics of the EDA and ML pipeline. This paper is meant to be a beginner's guide for the interested data scientist (or baseball fan) who wishes to get a small sense of what these tools can do.

For the interested, I come to this project as someone who falls into both of those categories. I enjoy data analysis, and I love baseball (Seattle Mariners fan, born and raised). I know I am not alone in this. I first began this project as a personal quest to see if I could train a ML model that could predict the outcome of a game better than a coin flip, and if successful, also predict the final score. I was unsuccessful in that goal. Frankly, I went into this project a bit naive - I overestimated what could be accomplished - primarily because I was unfamiliar with what ML tools really do. That is why this paper will include brief, but detailed, explanations of each ML model and how it works. To be successful as a data scientist one must understand the assumptions, strengths and weaknesses of each modeling tool.

While the results to my initial questions are not earth-shattering, along the way I did discover that I could model the overall season wins of a team with decently high accuracy. This was a thrilling moment, but also includes one of the pitfalls that I discuss in appendix B. Machine Learning and data science, like being a Mariners fan, is fun, often frustrating, but ultimately rewarding. I hope you learn something from this journey.

3 Predicting Game Outcomes

My initial goal with this project was to see if I could create a model that could predict the outcome of an MLB game with accuracy 'better than a coin flip.' In this section I will show why this question is an unsatisfactory benchmark and the metric I finally settled upon. I will walk you through the process and results of attempting to fit the data with multiple ML models. For this question the Neural Network (NN), Naive Bayes(NB) and the Principal Component Analysis (PCA) to Logistic Regression Pipeline performed the best for reasons that I will attempt to make clear in this section.

3.1 Hypothesis & EDA

After downloading the results of every 2019 MLB game from Baseball-Reference I first set out to do some basic statistics on the dataset - how many home wins were there, what was the distribution of team inning percentage, etc?[2] It turns out that for the year in question, there was in fact a home field advantage. Of the 2429 games played, the home team won 52.9% of the time. If the home team had a higher winning percentage than the away team, they won even more - 60.6% of the time. If the away team had the higher winning percentage they won on average 55.2% of the time. This means that for the entire season the team with the higher winning percentage, measured as wins divided by total games played (at the time of the game), was favored to win 58% of the time.

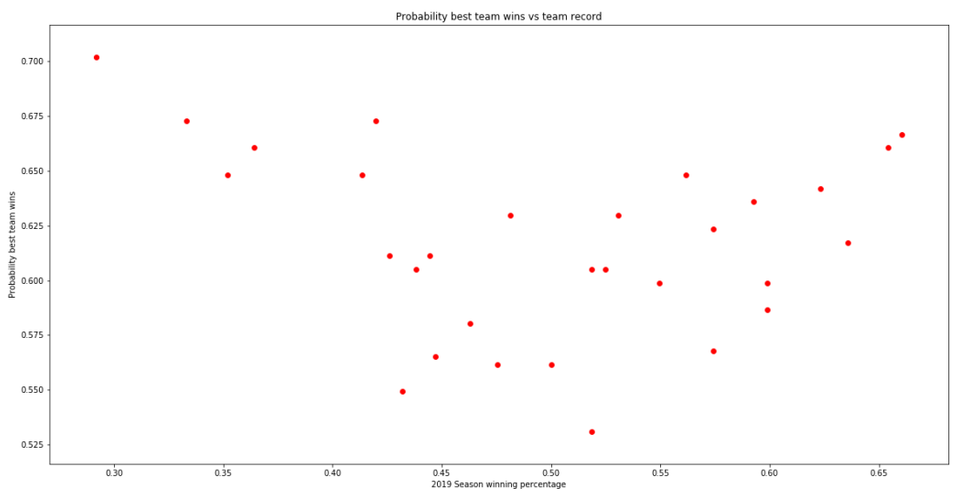

Figure 1: The greater the disparity between team win percentage, the greater the probability the team with the higher win percentage wins the game.

These results led me to realize that my initial question, or goal, was uninformed. The data showed quite clearly that to believe the naive predicted outcome of a baseball game should be a 50-50 coin flip is too naive to withstand scrutiny. A much better metric for the success of a game outcome model should be whether it predicts better than this naive bias: "Winning teams will continue to win." In other words, if a team has a higher win rate than another team they should win.[1] This is a naive prediction, because of course we expect two teams that have very similar win rates to be basically equal - perhaps a 50-50 coin flip. But there are also teams with very high and very low win percentages and we would expect those teams to be much better or worse than the teams they are playing against. This plays out in the statistics and means that the real baseline for a successful model should be, can this model achieve accuracy of greater than 58%? See fig.1

[1] I am aware of other research which chose as a baseline the published odds by baseball bookies for gambling purposes.[3] I think this a fair metric, but because I have no intent of using this research for betting I think this is satisfactory place to start.

3.2 Procedure

All work for this section can be found in the Jupyter Notebook title "pipeline_team_stats.ipynb"

1. Import the file "2019_all_delta_extra.csv" into pandas
   This file was created from the datasets downloaded from baseball-reference. I took all games played for the year by every team and cross referenced them with each other in order to reduce the number of entries by half. I then took each teams statistics going into that game and created a delta value with reference to the home team. For example a run allowed delta of 14 means that the home team has given up 14 more runs than the away team during the season.

2. Create the sample set and the output classes ('W' or 'L') - basically your X and Y values

3. Scale the data and label the Y values. For this I used the sklearn tools StandardScaler and LabelEncoder. The StandardScaler takes all input features and transforms them into a normalized distribution and the LabelEncoder converts the W/L labels into a 1 and 0. While the Scaling isn't always strictly necessary, there are some models, like the Neural Network that require it.

4. I then conducted a PCA analysis on the sample set to see which features were most prominent in determining the variance of the data. Of the seven features included, the delta win percentage plays the biggest role.

5. I then decided to test different pipelines using PCA as a starting point. I tested PCA→Logistic Regression, PCA→NN and PCA→Decision Tree. Additionally I test the NN, Decision Tree and a Naive Bayes by themselves.

6. A main point to remember before beginning model building is to properly split your data into a training and test set. I used the train_test_split function built into sklearn. For validation I also used the cross_val_score function in sklearn. The main difference being that cross validation gives you a better impression of the models performance across the entire dataset while splitting your data into a training and test set only tells you the performance for that particular split. If the data is well randomized and large both should work basically the same.

7. After finding additional stats on retrosheet: team batting average (AVG), on base percentage (OBP), slugging percentage (SLG), batter fielder wins (BFW) and opposing team stats, I decided to include that data in the same models to see if that produced significant differences in model accuracy.

3.3 Results

There was not a significant difference in model accuracy between the sample set that included team averages in the features vs the sample set that only included team deltas. The best performing models were the PCA, Logistic Regression pipeline and the PCA, NN pipeline. Interestingly, while the PCA, Logistic Regression model worsened with more features, the PCA, NN pipeline and the Naive Bayes both showed improvement. In the end, the best models performed with 59.5% accuracy. This is a 1.5% improvement on the heuristic that winning teams will continue to win. It is important to keep in mind, however, that this improvement (though slight) was accomplished without including any account for time of game, pitching match-up, roster or any other game specific stat. It was entirely from average team performance statistics. All models performed similarly with NB performing best and Decision Tree (DT) performing slightly worse. An explanation of model types in Appendix A may be of some use for the uninitiated.

3.3.1 PCA

PCA was always my first step because I wanted to get a sense for the feature impact on my samples. The home team stats seemed to always weigh the largest, and unsurprisingly, the team winning percentage played the biggest role. From the following images one can see that the sample set with fewer features was able to be shifted greater than the sample set with more features. In either case though, it is clear that PCA is not able to significantly separate wins from losses.

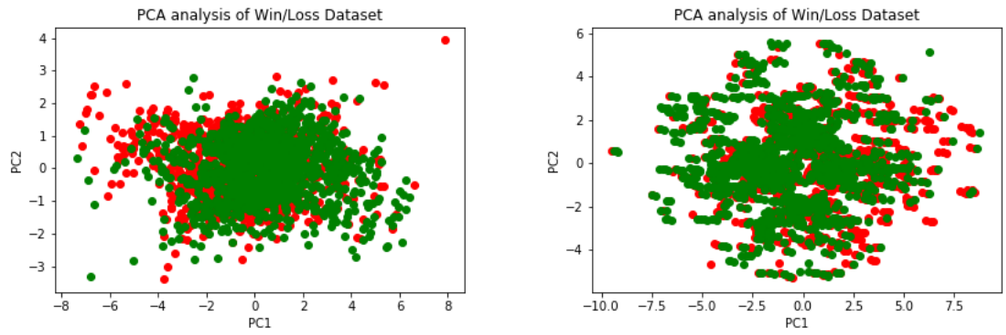

Figure 2: PCA on the smaller feature set (left) separates the wins and losses somewhat, providing a better starting point for a logistic regression or Nueral Network. With more features included (21) the PCA now does almost nothing, actually reducing the accuracy of the PCA-LR pipeline from 59.5% accuracy down to 59%

3.3.2 ROC Performance of the PCA to NN Pipeline

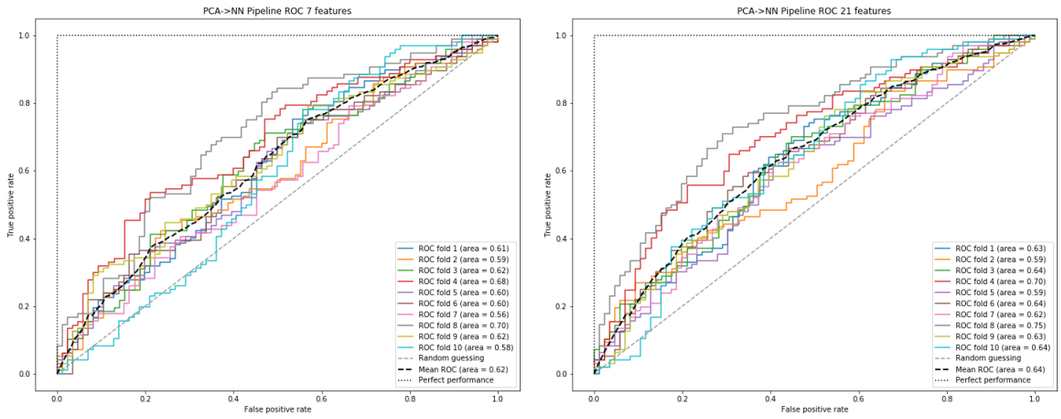

Figure 3: Adding features improves the ROC chart of the PCA, NN Pipeline by a couple percent

3.3.3 Model Accuracy Results

All models were tested with 10 fold cross validation. It should be noted that due to the random nature of the neural network initialization process, the CV value could vary up to 1% for the NN, average values have been supplied.

	PCA (all), Logistic	PCA, NN	Naive Bayes	Decision Tree
Accuracy	58.5%	58.5%	58%	57.8%
	PCA (2), Logistic	LDA, NN	PCA, Naive Bayes	PCA, Decision Tree
Accuracy	59.5%	57.8	58.4%	58.1%

Table 1: For 7 features these accuracy scores were obtained using 10 fold cross validation. The best performing model was the PCA(2) to Logistic Regression Pipeline

	PCA (all), Logistic	PCA, NN	Naive Bayes	Decision Tree
Accuracy	59%	59.7%	59.5%	57.8%
	PCA (2), Logistic	LDA, NN	PCA, Naive Bayes	PCA, Decision Tree
Accuracy	59.2%	59%	59.5%	58.3%

Table 2: For 21 features these accuracy scores were obtained using 10 fold cross validation. The best performing model is now the PCA(2) to NN Pipeline, essentially tied with Naive Bayes.

4 Predicting Season Outcomes

Up to this point we have only worked with data from a single season. For this portion of the project I wanted to examine multi-year trends and see how these ML models fared with a season long approach. I gathered data for this section via the Retrosheet website using the webscraping python library Beautiful Soup.[4] The data spanned the 2012-2019 seasons and initially contained data just related to runs scored (RS), runs allowed (RA) and win percentage. Later I added a run differential column as well as team player averages (as discussed in the previous section.)

4.1 Hypothesis EDA

My intuition and bias for this analysis was the following: "Teams that score more runs than other teams will win more games." To test this hypothesis I began my research with basic EDA comparing season wins to runs scored, runs allowed, and run differential. This initial exploration showed promise quickly in the form of a graph that clearly follows a linear trend. Which brings up a good point, when doing EDA, keeping it easy and visual can help you quickly know if you're on the right track or not.

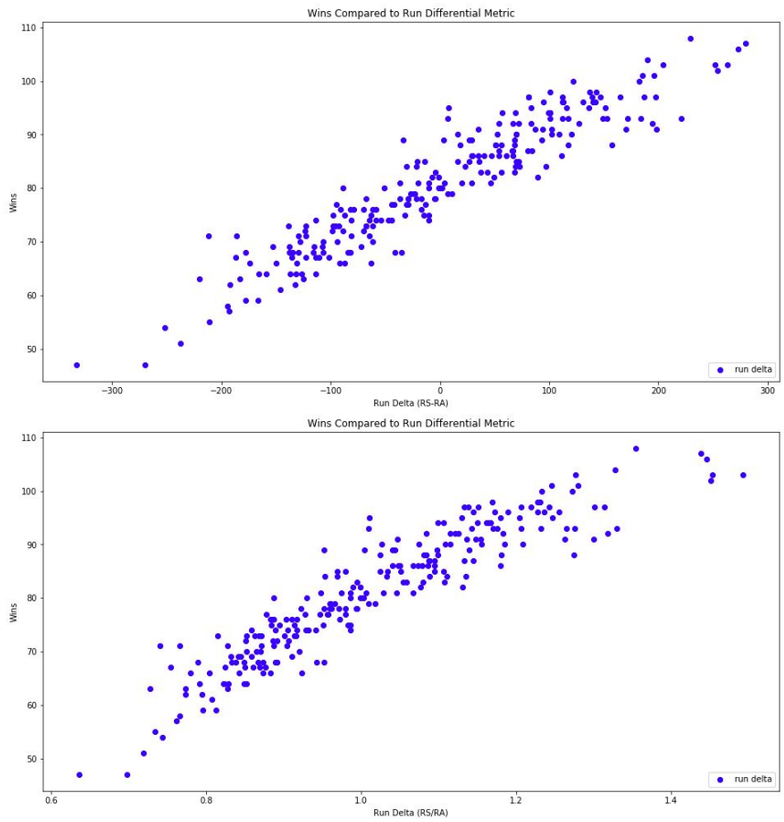

Figure 4: There is a clear and strong correlation between run differential and total season wins. Both versions of run differential are usable. Because runs scored (RS) divided by runs allowed (RA) has more long term stability over a season it is the stat that I used throughout this project

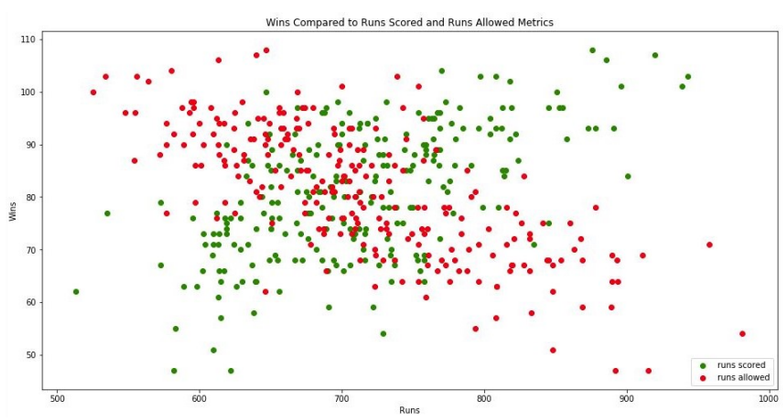

Figure 5: While RS and RA do correlate to season wins, it is not as strong a predictor as run differential

4.2 Results

Using a linear regression model on just the run differential feature produced a model with 84.8% accuracy. There was slightly higher accuracy in predicting National League games, perhaps because there were fewer outliers. These initial results were promising so I decided to add features to the sample set.

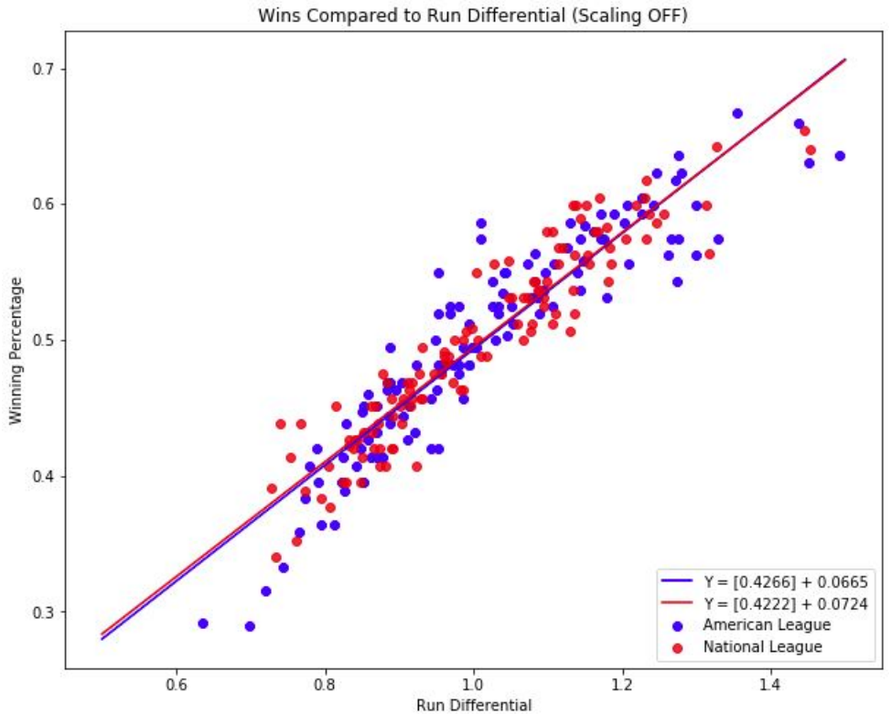

Figure 6: Both leagues exhibit a strong linear relationship between run differential and season wins.

When adding the additional team average features to the sample set the model accuracy improves to 86.2%. This is only a modest improvement, but isn't too surprising given the results of PCA. (see fig.7) While this value indicates a model that is significantly better than the first experiment in this project, it still isn't in the range that one might consider an 'A' level. That said, from the following histograms you can see that the model has on average very low variance from actual results and greater than 80% of predictions will be within 5 games of the actual result.(see fig.8)

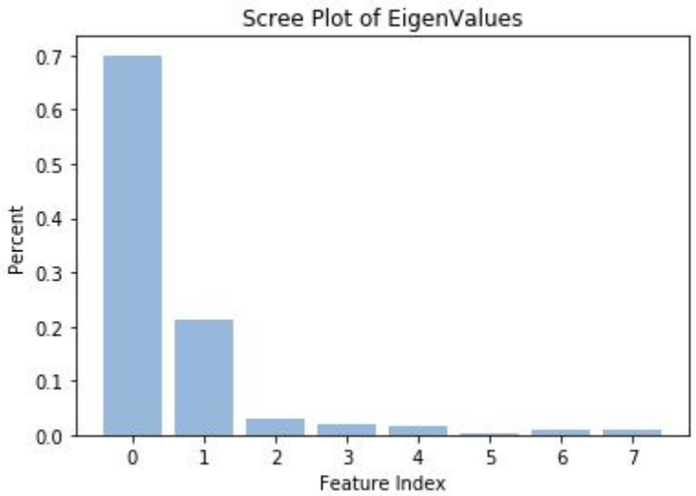

Figure 7: Run differential plays the largest role in the variance of this data by far, accounting for nearly 70% of the correlation

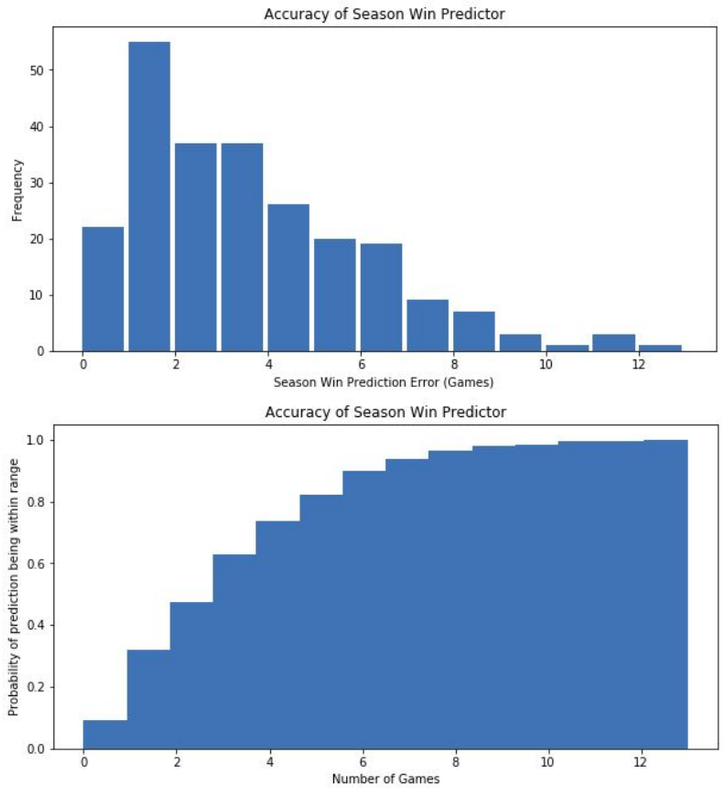

Figure 8: Approximately 80% of seasons will be predicted within 5 wins

Finally, I correlated the player average stats to both team wins and run differential. The purpose of this analysis was to provide some actionable data for teams wishing to improve the number of wins they have in a season. These results are not new as this correlation between on base percentage and wins has been known since the early days of sabermetrics.[5] (see fig.9)

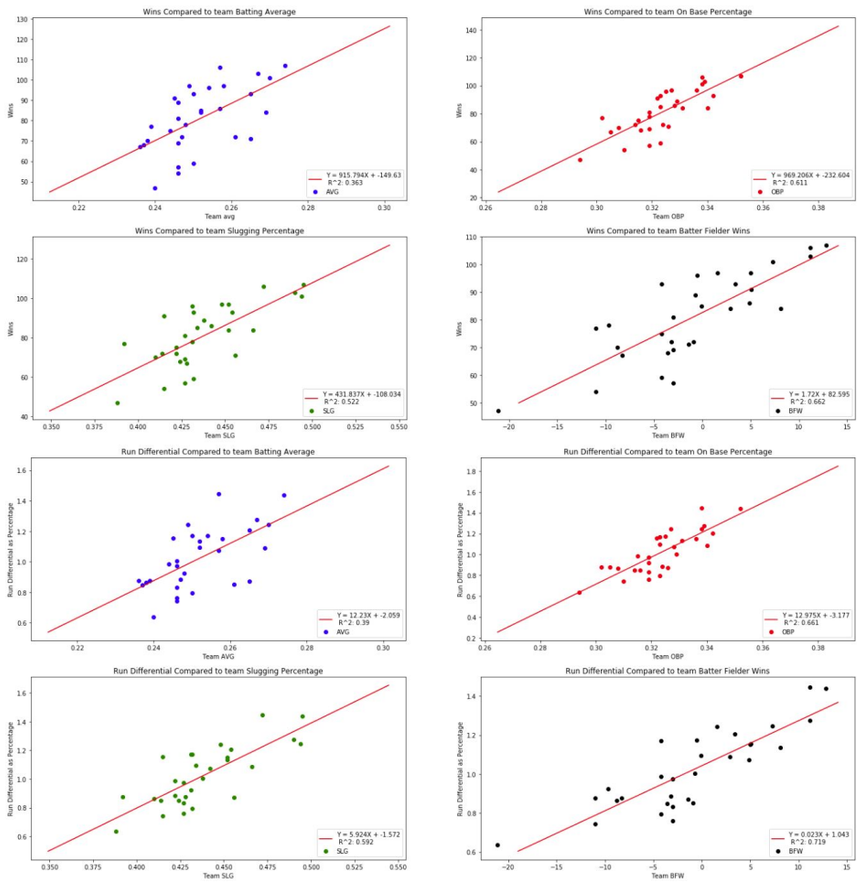

Figure 9: Player OBP and BFW correlate the most strongly with both team wins and run differential. Teams wishing to improve their season wins should focus on acquiring players with these stats.

5 Conclusion

Machine learning, EDA and statistical analysis are very useful tools and have been widely used in the baseball domain. For this research project I demonstrated several ML modeling tools meant to predict the outcome of a baseball game with varying degrees of success. Logistic Regression, Neural Network and Naive Bayes performed the best overall and only showed modest improvement by increasing the feature set. The best average accuracy obtained on this problem was 59.5% which is 1.5% better than the heuristic: 'Winning teams will continue to win.' Needless to say one should not be using any of those models for gambling purposes.

For the second phase of this project I looked at overall wins in season, using data from the 2012-2019 seasons. This endeavor was much more successful and resulted in a Linear Regression model that, depending on the number of features provided, received an accuracy score as high as 86.2% measured in the form of R^2. Furthermore, a strong relationship was established between the OBP and BFW of a team and the run differential of that team. Ultimately the run differential of a team is the most highly correlated metric with wins (an idea widely established by sabermetrics.) Teams wishing to improve their standings should focus on acquiring players that improve a team's average in the areas of OBP and BFW.

6 Key Stakeholders

Researcher, Author: Kyle Miller
Mentor: Dr. George Rudolph

References

[1] Wikipedia. Sabermetrics. "https://en.wikipedia.org/wiki/Sabermetrics".

[2] Baseball Reference. Baseball statistics. "https://www.baseball-reference.com/bullpen/Baseball_statistics".

[3] Roger D. Pharr. Predicting mlb-game outcomes with machine learning. "https://towardsdatascience.com".

[4] Retrosheet Inc. "https://www.retrosheet.org/gamelogs/index.html".

[5] Michael Lewis. *Moneyball: the art of winning an unfair game*. 2003.

[6] Anaconda. https://www.anaconda.com/distribution/.

[7] Google. https://colab.research.google.com/notebooks/intro.ipynb.

[8] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[9] NumPy. https://docs.scipy.org/doc/.

[10] The pandas development team. pandas-dev/pandas: Pandas, February 2020. URL https://doi.org/10.5281/zenodo.3509134.

[11] Beautiful Soup. https://github.com/wention/BeautifulSoup4.

[12] Microsoft. What is a machine learning model? https://docs.microsoft.com/en-us/windows/ai/windows-ml/what-is-a-machine-learning-model.

[13] Prasad Patil. What is exploratory data analysis? https://towardsdatascience.com.

[14] Luca Massaron John Paul Mueller. Exploring cost functions in machine learning. https://www.dummies.com/programming/big-data/data-science.

[15] Grant Sanderson. What is back propagation really doing? https://www.youtube.com/watch?v=Ilg3gGewQ5U.

[16] SciKit Learn. sklearn.model_selection.train_test_split. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html, .

[17] SciKit Learn. https://scikit-learn.org/stable/modules/cross_validation, .

[18] Rohith Gandhi. Introduction to machine learning algorithms: Linear regression. https://towardsdatascience.com/.

[19] Jason Brownlee. Logistic regression for machine learning. https://machinelearningmastery.com/logistic-regression-for-machine-learning/.

[20] Lewis Gavin. Machine learning basics with naive bayes. https://www.lewisgavin.co.uk/Machine-Learning-Basics/.

[21] SciKit Learn. Decision trees. https://scikit-learn.org/stable/modules/tree.html, .

[22] SciKit Learn. Exact pca and probabilistic interpretation. https://scikit-learn.org/stable/modules/decomposition.html#pca, .

[23] SciKit Learn. Linear and quadratic discriminant analysis. https://scikit-learn.org/stable/modules/lda_qda.html, .

[24] SciKit Learn. Neural network models (supervised). https://scikit-learn.org/stable/modules/neural_networks_supervised.html, .

[25] Rebecca Vickery. A simple guide to scikit-learn pipelines. https://medium.com/vickdata.

7 Appendix A: Machine Learning References

Machine Learning, like all fields, has its own set of vernacular that one must learn in order to know their way around the conversation. I will never forget when I used the word 'back-propagation' with a seasoned data scientist (my brother) in a way that wasn't correct. Below you will find common ML tools, models and terminology with explanations to help you learn the ropes.

7.1 ML Tools & Coding Environment

1. Anaconda and Jupyter Lab
   All coding for this project was done in Python 3.7 in a Jupyter Notebook hosted in an Anaconda environment. If these tools are unfamiliar to you, please know that they are extremely common in the data science space. If you don't wish to install them on your own machine you can use a web hosted version straight from a Google account using Colab Notebooks. You will either need Anaconda or Colab to view the source code for the projects described in this paper.[6][7]

2. Sci-kit Learn
   Sklearn is one the main ML libraries in use today. They have incredible documentation and off the shelf modeling tools that just work (with some basic understanding). Look for python import statements that start with "import sklearn.<model-type>" in the provided files.[8]

3. Numpy
   Numpy as the number one package for dealing with arrays in python. It has a lot of built in optimizations and creature comforts like applying boolean masks. You will see this used in every notebook: "import numpy as np". Again, please check out the amazing documentation for your own projects.[9]

4. Pandas
   Pandas is a library for creating and managing dataframes - basically excel csv files. It was used extensively in this proejct for reading and creating the many csvs needed during data analysis. It takes some getting used to but is also very powerful and well documented. You will see it as: "import pandas as pd"[10]

5. Beautiful Soup
   How do you automate getting data from a website like Retrosheet? You use Beautiful Soup. You will see this package used in the Retrosheets_stats_scraper notebook. This is where I automated the process of getting the team average statistics for all MLB teams over the 2012-2019 seasons. A job that would have literally taken hours of copy-pasting was reduced to a few minutes thanks to the consistent naming and display schema of Retrosheet. Baseball-reference was not as friendly, using javascript to dynamically load their data in a way that made the use of Beautiful Soup impossible.[11]

7.2 ML Terminology

1. Model
   The final product. Once created the model will make predictions on unseen data. A good model is one that generalizes well to new input, meaning that it neither under or over fits the data.[12]

2. Exploratory Data Analysis (EDA)
   One of the first steps in the ML process. Most of the time I spent on this project was collecting and analyzing the data - looking for appropriate ways to apply ML to the task. The actual model training was a small fraction of the time - perhaps only 10% of the overall work.[13]

3. Cost or loss function
   Most models when they are being trained use some sort of mathematical function to determine how well the model is performing. An example would be the Mean Squared Error (MSE).[14]

4. Back-propagation
   In a NN the input data is first 'fed forward' through mathematical operations, and then the error is sent backwards through the network to update the weights of those mathematical operations.[15]

5. Train Test Split
   This is common practice in ML to split your data set into a section that will train the model and a section that will then test the model. The purpose of this is to make sure that your model has a good balance between low bias and low variance. A model that is trained on the entirety of the data may have a very low loss function, but that could be because it has been overfit - in other words the model will not be able to generalize to new data very well. The test set is a way of making sure that your model performs well on unseen data too.[16]

6. Cross Validation
   Cross validation was the main metric I used in this paper and it involves building and testing the model on multiple train test splits and then taking the average score of each of those. It can give you a better sense of how well the model will perform compared to a single train and test set.[17]

7.3 ML Models

1. Linear Regression
   Like it sounds, this is one of the simplest ML models, it finds a single, straight best fit line for your data. The only difference between this model and what we're all used to from High School math class is that this model allows for multiple X values. It is vector based instead of scalar based. Because it is easy and fast it can be a good place to start, particularly if you believe that one or more variables may directly effect the outcome. This is a tool for regression though, which is why I did not use it on the game outcome predictions.[18]

2. Logistic Regression
   This model is similar in a lot of ways to linear regression, but it is meant for categorical classification predominantly. This is why I used it for the game outcome models. It is a good model when you have categorical outputs, but still have probabilities involved. Since we were dealing with a percentage chance that one team would win over another it seemed like a good fit, and it turned out it was one of the best performing models for that task.[19]

3. Naive Bayes
   This model is useful when you are dealing simply with probabilities. It assumes that the input features are independent of each other, but even if they're not it tends to perform well overall. It basically looks at the odds of a certain outcome occurring and uses those statistics to make its predictions. The Gaussian Naive Bayes which I used allows for non-categorical data to be used which was nice since much of the data were percentages. It will also perform well with categorical data which means that I could add things like night/day game or pitcher name to the data set and it would still work just fine.[20]

4. Decision Tree
   This model is one of the most understandable by humans. It is created by using math to identify the best criteria on which to split the data into categorical output. For example, in these datasets the tree may identify the home team winning percentage being greater than a certain value to be the best first splitting criteria and so forth. While trees can be very understandable they do suffer from the problem of overfitting. I noticed in my analysis that this was a severe problem because, with things that are odds based, like baseball, there are definitely outlier scenarios which will cause grief on decision trees. Because of this, the best accuracy scores on my trees were with no more than 3 levels of depth. Below is an example of one of these trees with a maximum depth of 2.[21]

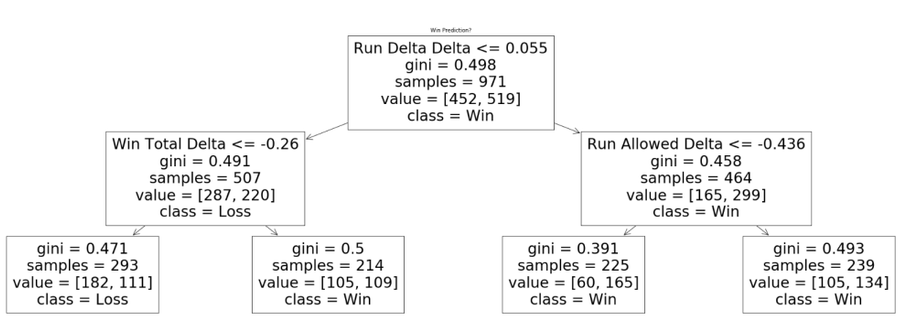

Figure 10: Example of decision tree output

5. Principal Component Analysis (PCA)
   PCA identifies the features that are responsible for the greatest variance. Using linear algebra, it transforms the input samples to produce a greater separation in classification. It is unsupervised ML.[22]

6. Linear Discriminant Analysis (LDA)
   LDA is similar to PCA in that it uses linear algebra to transform the samples input space. What makes it different is that it is supervised, so it also uses the classification to find a transformation that maximizes the separation in output classes.[23]

7. Neural Network (NN)
   The NN is a good all around performer, and that was born out in these experiments. It performed on par or better than all other models. The downside of the NN is that it behaves like a black box - there is no way to know exactly how it makes its decisions.[24]

8. ML Pipeline
   A pipeline is nothing more than chaining multiple ML tools together into one workflow object in sklearn. Pipelines are used in this project multiple times, particularly when coupling scaling and PCA with other models.[25]

7.4 SKlearn API

This is obviously not a comprehensive API explanation - see the online documentation. I include these here only to give the beginner a sense of the process of using the sklearn modelling tools. Following are the 3 most import calls and you will see them used throughout the attached Jupyter Notebooks.

1. .train()
   Inputs: X and y training values
   Return: trained model object

2. .score()
   Inputs: X and y testing values
   Return: Accuracy score

3. .predict()
   Inputs: X values Return: Output values / classes produced by the model.

8 Appendix B: Personal Notes

8.1 Lessons Learned

I would like to include a few mistakes that were made along the way and the lessons learned. While I realize this is a bit unusual, because this paper is for a term project and has use primarily for young data scientists, including myself, it makes sense to include these warnings.

1. Automate your data acquisition
   I wasted a lot of time throughout this project making my own csv files via copy-pasting from websites. While I initially started the project intent on using baseball-reference as my main resource, I was unable to webscrape it because I lacked the skills to deal with the dynamic nature of its programming. Later, I found Retrosheet which was easily webscraped and allowed me to within an hour get data from 2012-2019 - a process which would have taken me at least 10x longer to do by hand. The moral of the story here is that if you are going to make a good, accurate model, you need lots of data. Get it as quickly and effortlessly as possible.

2. Models can only be as good as the data you give them
   One of the main weaknesses of the models demonstrated in this paper are that they are perhaps too simplistic. The real world is complicated and it is unreasonable to believe that one can model with a high degree of accuracy the outcome of a sporting event without copious amounts of data.

3. Curb your expectations
   Tagging along with the previous aphorism, it is important to begin a project like this with reasonable expectations. If expectations are too high it is easy to accidentally fall into the trap where you believe results that are too good to be true (see the next point.)

4. Beware of 'barcoding' your samples
   When building and testing the season wins model with additional features I ran a test and received an accuracy score of 99.9%. I was thrilled - I had built a model that worked nearly perfectly. It turns out there was one small problem... I had accidentally included a feature in the samples that was 100% correlated to season wins - overall win percentage. This was a residual feature from the previous tests I had been conducting. I finally discovered this mistake when I noticed that one of the slopes from the linear regression was 162 - the exact length of the MLB season. This was of course, disappointing, but it taught me to be more skeptical of results that were so good. Instead of reacting to those initial results with excitement I should have responded with deep reservation.