2016 Baseball Prediction Project Report

Kristin Fasiang, Ricardo German, Sid Mehrota, Melchor Ronquillo, Rob Sisto

Introduction

For the final project, we decided to use the 2016 MLB Season dataset from Kaggle (link: www.kaggle.com/datasets/cyaris/2016-mlb-season). This dataset has statistics from each game played in the 2016 Major League Baseball season, with features such as attendance, game time, date and day of the week, weather, home and away teams, as well as the number of hits, errors, and runs scored by each team in the game.

In building our models, we decided to predict whether a team would win a game given only pre-game factors. This meant not looking at the hits, errors, and runs scored by each team, or any other in-game statistics. Instead, we wanted to be able to guess, before a game was played, which team would win. This is a notoriously difficult task, as explored in an article on whether it is possible to accurately predict the outcome of Major League games with only home and away records to go off of [3], and although this team had a much greater volume of data, they were only able to achieve a little under a 55% accuracy. Thus, for our classification, we were hoping to approach 55% accuracy.

After testing various models and creating various ways of preprocessing data, the best result found was an accuracy of 57% on the training set using SVM.

Dataset Description

The original 2016 MLB Season dataset found on Kaggle had 26 columns in it, and we eliminated 12 of them because they contained information that would not be known before a game started. These eliminated features held statistics on runs, errors, and hits scored by each team, as well as game length and the outcome of the game for each team. The features remaining were: attendance, away team, date, field type, game type (day or night), home team, start time, day of week, temperature, wind speed, wind direction, sky, and season (regular or post-season).

We found that using only this information only yielded accuracy of 52%, so we decided to add more information to the dataset. This information included the win percentage to date for the home and away teams, the number of hits and runs by and allowed by each team in the season so far. In later iterations, we also included home and away split winning percentages for the home and away teams, a statistic called the ballpark factor, run differentials for the home and away team, and the pythagorean statistic for each team.

Baseline Approach Description

The baseline approach for our team was to each split up and conduct preprocessing of the data in our own way. Then, we would bring our approaches together and each test different classifiers on our own to see who could get the highest accuracy in their predictions on the testing set.

Our main goal was to be able to predict the outcome of a major league game using only information that would be known before the game began. By splitting the work between us, we hoped the competition would allow us to think more diversely and get a higher accuracy than we might have if we were all working on the same approach.

Preprocessing

In order to have the data in a state where we all could use it, Rob calculated the winning percentage of each team to the point of the current game by keeping track of the record and then calculating the winning percentage. He then kept track of the total number of hits and runs up to the current game. He did not do the same for errors because errors are a flawed statistic which do not necessarily represent how good a team is at fielding. Rob then kept track of home and away records and found the winning percentage of each. After this, he found the ballpark factor of each ballpark and added that to the dataset. The ballpark factor is a measure of how much a player's stats are affected by the ballpark they are playing in. Since not all ballparks are the same size or are built in the same location, a player's stats may be skewed if they play at a given ballpark that favors hitters or pitchers. A prime example of this is Coor's Field in Denver, Colorado. As Denver is high up in the mountains, the air is thinner which allows the baseball to travel farther when hit with similar force, as opposed to other ballparks that are closer to sea level and have thicker air. According to baseball's Statcast, the average hitter would expect to hit 16% better in Coor's Field than at the average ballpark, thereby skewing his stats. [1]. Adding this variable into our dataset would hopefully help our models to better account for these differences and to improve their prediction accuracy. The last variable that was added to the dataset was each team's Pythagorean winning percentage. The Pythagorean winning percentage computes each team's record by using their runs scored and allowed. [4]. This statistic is supposed to be a better indicator of a team's true performance. In keeping with our theme of wanting to predict the outcome of a ballgame before it starts, all of these stats can be accessed before a game starts.

Method Description

Each of us took a slightly different approach to creating classifiers for the dataset, and our approaches are outlined below.

Ricardo used the dataset prepared by Rob to run a Logistic Regression model. In his preprocessing, he included features such as environmental factors, percentages (hits, runs, wins), and an overall ballpark factor. He also excluded any post-season games, therefore, this dataset only trained on the season games. Features with any nulls were filled in by the average number of that feature. He converted categorical variables 'sky','field_type', 'game_type', 'wind_direction', 'day_of_week' into 1s and 0s; The rest of the features were normalized. Also, his dataset is shuffled for every simulation, and he used the holdout method to train/test. After implementing a grid-search, he discovered that the best parameters to use were an 'l2' penalty, 'saga' solver, and a 0.01 regularization strength. He achieved a testing accuracy of 53%, testing accuracy of 54%, and an F1-score of 0.69.

Sid used the dataset prepared by Rob. He dropped all the variables that one would know after a game has finished such as hits, errors, runs scored in the game, etc. He also dropped a few features he thought were not important such as "venue". He then did some normalizing of the data via sklearn preprocessing. Sid then used a function of pandas, .get_dummies to convert the categorical variables into dummy/indicator variables which created a lot more features to use. With so many features, Sid then ran some L1 regularization and then dropped the features the L1 regularization determined to be insignificant. He then randomly split the data into 60% training, 20% development, and 20% testing. For Sid's classifiers, he implemented Adaline and Perceptron. He ran a grid search for both perceptron and adaline in which he had some different options for learning rates and kept the epochs at 50. In one specific random state the perceptron had a 56.6% accuracy, while adaline had a 56.8% accuracy.

Melchor used the dataset prepared by Rob. He began by dropping all the variables that he believed were insignificant to predict 'home_team_wins' , as well as variables that represented in-game statistics. Examples of these variables included: 'date', 'start_time', 'game_hours', 'away/home_team_errors', 'away/home_team_hits', 'away/home_team_runs', etc. Now having a dataframe containing columns of interest, he went in every column with categorical variables and changed the values to categorical data types, replacing the strings with numbers to represent each specific unique value respectively. Examples of these columns include: 'away/home_team', 'field_type', 'game_type', 'wind_direction', etc. The next focus was to scale the columns with continuous values such as 'attendance', 'temperature', 'wind_speed', 'home_runs_for', 'ballpark_factor', etc. The data was then randomly sampled into a training and validation set with a 80 / 20% split. To get a baseline comparison for predicting "home_team_wins" with this data, a Dummy Classifier was used and the highest accuracy yielded from this was 52%. Moving on to fitting the data with an SVM using the default parameters, this model predicted the outcome variables of the test set with around 53% accuracy. To find a better model, he used a 10-fold cross validation grid search on SVM to determine which parameters could yield a better accuracy. This grid search returned the parameters C = 1 and gamma = 0.01 being the best parameters, having the highest score of 59%. Using these parameters, the training set was fit and the model predicted the test data with 56% accuracy. One interesting thing that he discovered was that despite the grid search returning these parameters as the best ones for the model, there is actually a different set of parameters that returned a higher test prediction accuracy. In a previous trial, he split the data into a 90 / 10 train test ratio split and after fitting the grid search on this differently sized data, the best parameters that were returned were C = 100 and gamma = 0.001, which yielded an accuracy of 61% accuracy. When the test data was predicted on this model, the accuracy achieved a 63% accuracy. Curiously, Melchor wanted to see if the parameters from this grid search would have an effect on the model with the proper 80 / 20 % split, and after doing so he improved the accuracy to 57%. To compare if other classification methods could beat SVM, he fit the data to a logistic regression classifier with default parameters, and the test predictions returned 54% accuracy. A 10 fold cross validation grid search was also utilized for logistic regression, and parameters C = 1, penalty = ‘l2’, and solver = ‘liblinear’ returned the same accuracy as the model with default parameters. Overall, the SVM returned the highest accuracy.

Kristin used the dataset prepared by Rob. The only change made to the preprocessing was to find the difference between the home and away team on the metrics win percentage to date, runs by and against, and hits by and against, and eliminating the original columns. She also assigned indices to the teams by order of their records in the 2015 season. First in the code, she built a rudimentary Neural Network using Keras with 4 densely connected layers. The parameters used for the NN were mostly taken from the slides on Keras, using relu activation on each layer except the final, which used sigmoid. The model was compiled with rmsprop as the optimizer, binary crossentropy as the loss, and accuracy as the metric. With 10 epochs and a batch size of 512, the model performed with a range of accuracy from 0.53-0.55 accuracy, depending on the random seed. She also performed a grid search over SVM and Logistic Regression, comparing accuracies on the training set using 5-fold cross validation, finding that SVM with parameters C = 0.1, kernel = 'linear', and tolerance = 1 yielded the best success, with training accuracy of 0.569 and testing accuracy of 0.553.

Evaluation

The best model overall was an SVM with parameters C = 100, gamma = 0.001, and kernel = ‘rbf’, which yielded 57% accuracy. Some factors that may have contributed to the success of this model over the others may have been the elimination of features that were not anticipated to be predictive for the model, the use of 10-fold cross-validation as opposed to 5-fold validation, and increasing the test/train ratio to 90/10. By paring down the number of features analyzed, the model may have been able to better adjust weights that were more directly related to win/loss outcomes. Further, increasing the number of folds used in cross-validation in the grid search would have allowed the model to train on larger portions of the training set at a time, allowing it to pull its predictions from a greater base of data. Similarly, the increase in the ratio of test/train data may have made the model more successful by giving it a larger amount of prior data to pull from in creating its predictions on the testing set.

Discussion

In all, we approached a very difficult problem through our classification task. One consideration that might have given us more accuracy would be increasing the amount of data trained and tested upon. As it was, our dataset only contained information on 2416 games. If we had information from previous seasons, we may have been able to make better predictions because we would have had a stronger body of knowledge to work from.

Another thing that might have been interesting to consider is also the statistics of different players starting the game on each side. Knowing the ERA to date for the starting pitchers and the batting averages of different players may have given us more meaningful features to weigh teams against each other with.

If we had more time, it may have also been worthwhile to explore more of the models that could be generated with Neural Networks. Especially if we were to add all the information previously discussed, the information may have been better interpreted by a more complex model. CNNs may have had better luck with detecting patterns such as win-streaks for a given team. However, since we weren't able to experiment with these models in much depth, it's hard to know if something like a CNN would work well on this sort of data.

A final consideration could have been to hold our testing set to data past a certain threshold in the season. It is probable that models will be better able to predict outcomes of games that occur later in the season because the stats that we have about a team's overall performance thus far would be more accurate. It may be interesting to see if our models had any more success on data solely from the last 20% of games in the season, or just the postseason, for example.

Conclusion

Predicting the outcome of a baseball game with only pre-game factors is a highly difficult task for machine learning models to tackle. This may be partly because it is a very hard problem for even the most skilled humans, using the most skilled data modeling and forecasting, to do. This can be seen by the way that the best accuracy we were able to find over the course of this project was just barely over 55%.

However, because we were able to perform better and more consistently than the dummy classifiers, it seems that the task is not impossible. There are countless things that can happen during a baseball game that cannot be predicted before the game starts, so even the best model would have a lot of limits in terms of what it could predict before a game started. That's why we have to play the game, and not let a model determine who should win— but maybe one day the models will be a little better at guessing.

Appendix (contribution of each team member)

Rob: Preprocessing

Ricardo: Logistic model tuning

Sid: Adaline, Perceptron and Logistic Regression

Melchor: Baseline model, SVM default / Grid Search, Logistic default / Grid Search

Kristin: Neural Network, SVM and Logistic Regression Grid Search

Sources:
[1]
https://baseballsavant.mlb.com/leaderboard/statcast-park-factors?type=year&year=2015&batSide=&stat=index_wOBA&condition=All&rolling=
[2]
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8871522/
[3]
https://towardsdatascience.com/can-you-accurately-predict-mlb-games-based-on-home-and-away-records-8a9a919bad29
[4]
https://www.baseball-reference.com/bullpen/Pythagorean_Theorem_of_Baseball