Sabermetrics: Predicting Games Won in a Season

Nicholas Alonzo, Ehsan Arasteh, Ari Bolton, Joel Fischbein, Chris Fong, Liandy Hardikoesoemo, David Huang, Ryan Kausin, Cecil Lam, Nathan McCall, Daniel Schlesinger, Tommer Schwarz, Alex Soong, Kevin Wang, Kathleen Zhen

ECS-171 Fall 2016, University of California, Davis

Abstract. Since professional baseball was established over 100 years ago, there have been extensive efforts to predict the success of players and teams based on their statistics and performance. We set out to find which statistics are most influential in determining a team's winning percentage through the process of feature selection, and to build a performance/win predictor for baseball teams using neural networks and regression analysis. We found the Run Difference of a team to be the most statistically significant in determining the number of wins in a season for a team and built predictors for number of wins with up to 90% accuracy.

1 Introduction

Baseball is unique from other professional sports in that the statistics of a team is more important than having one or two star players. In today's baseball world, every professional team employs data scientists to find the best combination of players on the field for the smallest price.

Classification methods such as Logistic Regression and SVM have been used to predict the winner of a game and the next pitch type that a pitcher will do. Linear Regression methods were widely used to predict the number of runs a team will score and the contract salary of a player. The features that were most commonly used to build these prediction methods were the number of wins/losses and the better statistics. [1] [2] [3] However, while it is important to find trends in individual player value, no single player can carry the load of a team over an entire MLB season like in basketball or football. Furthermore, while random factors can have a significant impact on an individuals performance, such as injury in the season, these effects are much less substantial in looking at a team's overall performance. Thus, we wanted to see, given the statistics of a team, how many games they will win.

In this paper, we will show how we created a win predictor for a given team using that team's past stats and averages, instead of their individual players.

Specifically, we will find and examine the optimal statistics that affect the number of wins an individual team produces over the course of a season. By identifying these statistics, we can construct a model to predict the number of wins using data from previous seasons. This study will give teams a better understanding of which statistics positively correlate with total season wins in order to build a team with maximum success.

2 Methods

2.1 Feature Selection

We used three datasets in the project: The original, PCA transformed, and reduced variables datasets. The original was taken from SeanLahman.com and transformed and modified into the two described below. The two were both used in each model to determine the one with the least error. Judging by the correlations between wins and each individual feature (Figure 1), expected significant variables are runs scored (R), runs allowed (RA), earned runs allowed (ER), and earned run average (ERA), each with a correlation of more than 0.5 and less than -0.5. These higher correlated features are then compared with what the models deemed to be significant.

Fig. 1. Correlation values of the features in the original dataset.

```
> sort(corrs)
        RA         ERA          ER          HA         BBA         HRA
-0.59892215 -0.59796343 -0.58381972 -0.49742543 -0.41134711 -0.38705832
         E          DP          SO         X3B          CS         PPF
-0.32781761 -0.20257241 -0.13470556 -0.09645575 -0.09016258 -0.08471405
       BPF          SB          CG          AB         X2B          FP
 0.04749800  0.05908314  0.15827497  0.17567091  0.24506347  0.27474964
         H         SOA          HR          BB         SHO      IPouts
 0.34431664  0.35772711  0.39658771  0.41128143  0.45717427  0.48525243
         R           W
 0.50743284  1.00000000
```

Principal Component Analysis Principal Component Analysis, PCA, is a procedure that transforms data into a dataset with principal components. The process attempts to capture as much variance as possible, with each principal component being orthogonal to the others. Each principal component is a different linear combination of the original set of features. Transforming the data in this way allows for potentially better predictive models, although others did not have much better success when using PCA transformed data (Jia, 5). The PCA dataset was generated using the caret package in R, with the preProcess() function. The function generated nineteen principal components (PC) and all were kept to be used for testing with both regression and neural networks.

Reduced Variables We diagnosed the original dataset to detect which of the original 24 features were unimportant for our analysis. We first checked the distribution of the wins to find that it had a normal distribution and did not require transformation. Next, we checked the multicollinearity between variables and found that there are seven variables that are strongly correlated. In order to reduce multicollinearity as much as possible, we created a new variable runs difference, RdiffRA = (R - RA), and dropped five variables: R, RA, ER, ERA, AB. After that, we performed the t-test and dropped four variables under 0.01 significance level to get our reduced variables.

2.2 Neural Network

In the Neural Network (NN) analysis of the data, we decided to use two Forward-Feeding Neural Networks (FFNN), one with a sigmoid activation function and the other a linear activation function. The FFNN that we used is the Deep Learning Toolbox, created by Rasmus Berg Palm on the Mathworks site. [4] We had to limit our data set to the results of teams between the years of 1996 and 2015, since pre-1996 teams played only 121 games per season, whereas post 1996, they play 162 per season.

Classification NN For the past 20 seasons, every team's win-loss ratio landed between 20% and 75% of the games they played. Taking this into consideration, we set these as the lower and upper bounds to our classification problem, and divided that interval into 11 equal 5% sub-intervals.

For the Reduced dataset, we began by testing several different configurations on our neural network to find the optimal setup. Our training consisted of a nested for loop, testing between one and three hidden layers, and 5-12 hidden nodes per layer. For each configuration of the hidden nodes and layers, 200 epochs of the data were run. The learning rate for our network was .085, and our optimal configuration was determined to be 16 inputs, with 2 hidden layers at 11 nodes each, and 11 outputs.

For the PCA dataset, we did much of the same setup as the reduced dataset, as the procedure was the same. The learning rate for the back propagation algorithm was 1, mostly due to the much larger numbers that we were working with. We determined that the optimal configuration were 19 input nodes, 2 hidden layers with 12 hidden nodes per layer, and 11 output nodes.

Linear NN For the FFNN with linear activation function, we had to modify the toolbox to support linear activation functions. This change was simple, adding the output type 'linear' to the neural network configuration and in the nnff.m file, where the activation functions are calculated. These changes allowed our FFNN to predict the win percentage of a team by treating it as a regression problem. We then used the same strategy as was used for the sigmoid FFNN: a nested loop, testing one to three hidden layers, and 5-12 nodes per hidden layer, with 200 epochs each. The optimal configuration for our linear FFNN is 22 input

nodes, 2 hidden layers with 11 nodes each, and 1 output node, with a learning rate of 0.1.

We then finally trained both networks with the entirety of the datasets we were using, and compared its predictions to the actual 2016 data for 3 teams in order to determine the accuracy of our predictions.

2.3 Regression

Ridge Regression We performed ridge regression with a 10 fold cross validation to find the optimal lambda. With the optimal lambda, we used a bootstrap over 10,000 samples to find the 95% confidence interval. For each bootstrap, we randomized the data and split it - 500 samples/96 samples. From the 500 samples, we randomly collected an additional 500 samples with replacement to use as our training set. The testing set were comprised of the other 96 samples. For our reduced models dataset, we found the optimal lambda to be 1.90512. We are 95% confident that the mean square error is between 7.44 and 12.31. For our PCA data, we found the optimal lambda to be 0.1515376. After running 10,000 bootstraps with that lambda, we also got the confidence interval. We are 95% confident that our mean squared error is between 6.31 and 10.56. Figure 2 at right illustrates this result.

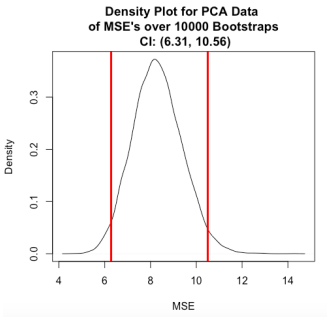


Fig. 2. Distribution of Mean Squared Error over 10,000 bootstraps

Polynomial Regression We also utilized polynomial regression with cross validation and bootstrapping. For cross validation, we decided to do 10 folds. However, because we have 596 samples, each training set in every fold cannot have the same number of samples. As a result, 4 folds of the training set contained 59 samples while the other 6 contained 60. We performed polynomial regression on the testing set up to 3rd order and regressed with each of the features to identify any strong correlations. Anything greater than 3rd order appeared unnecessary because it would overfit the data. From the results of the cross validation, we calculated the mean squared error of the models to determine how well this technique performed. We again used a bootstrap over 10,000 samples to find the 95% confidence interval. We are 95% confident that the mean square error for the 1st order is between 6.93 and 10.8, the 2nd order is between 6.69 and 10.49 and the 3rd order is between 6.26 and 10.07.

3 Results

3.1 Data Selection

Out of the two datasets used, the reduced variables data resulted in the least error. This was largely due to removing many features with low correlation and combining features to reduce multicollinearity. The newly added column, runs difference, turned out to be significant in both neural networks and linear regression as it combined two significant variables into one.

3.2 Neural Networks

For bucket classification with the reduced data set, after training and testing at 200 epochs over a series of 10 times and averaging the results, we came up with a MSE range of .3301-.4205, with an average MSE of .3754, or about 21/59 misclassifications. The classification using the PCA dataset yielded a MSE range of .3051-.4407 with an average of .3746. For both of these, repeating the tests using the full dataset for training and the stats of three 2016 teams for the year yielded an error of 66%.

For the linear output neural network, we obtained an average absolute error of around 5-11 games, meaning an error of about .10, when we used the Reduced variables data set. Using the PCA dataset, we reached an average error of around 9-14 games, or .13. Repeating the tests using the full datasets for training and the stats of the three 2016 teams for the year yielded an average error of 9 games off, creating an error of about .095.

3.3 Linear Regression

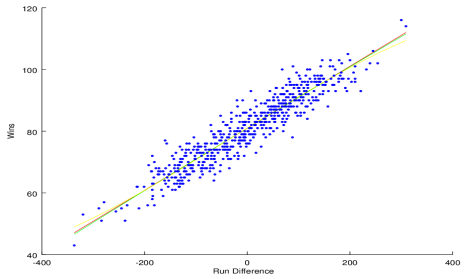
In linear regression, ridge regression provided a mean squared error between 6.3 and 10.6 while polynomial regression provided a mean squared error between 6.3 and 10.8. Both were lower than other regression techniques which had at best a mean squared error of 15. If we were to compare the MSE to Neural Networks, we would divide the polynomial regression MSE by the number of games which is 162. By doing this, ridge regression would have a MSE between .039 and .065, and polynomial regression would have a MSE between .039 and .067. Based on the graphics from the polynomial regression models the statistic that contributes most towards predicting wins is run difference, which for the season is calculated as:

$$Run\ Difference = Runs\ Scored - Runs\ Allowed \tag{1}$$

Figure 3 below shows the strong correlation between run difference and total wins. A few features such as Earned Runs, Earned Runs Average, and Saves had some correlation with number of wins. The 3rd order polynomial MSE for those were 85.4, 83.2, and 72.929 respectively. Even though they are higher than the Run Difference MSE, there is still significant correlation. Figure 4 on the following page shows the relationships the other features in the dataset had with total season wins. As can be seen, most of the other features appeared to not have any significant relationship with our output variable.

The prediction for 2016 performed very well for the 3 teams. 3rd order polynomial had the most accurate predictions. It predicted Chicago would have 105.02 wins, San Francisco with 88.125 wins, and Oakland with 68.268 wins. These results were off by an average of 1.29 wins.

Fig. 3. Illustrates the relationship between run difference and games won in a season. Red represents 1st order polynomial while green and yellow represent 2nd and 3rd order respectively.

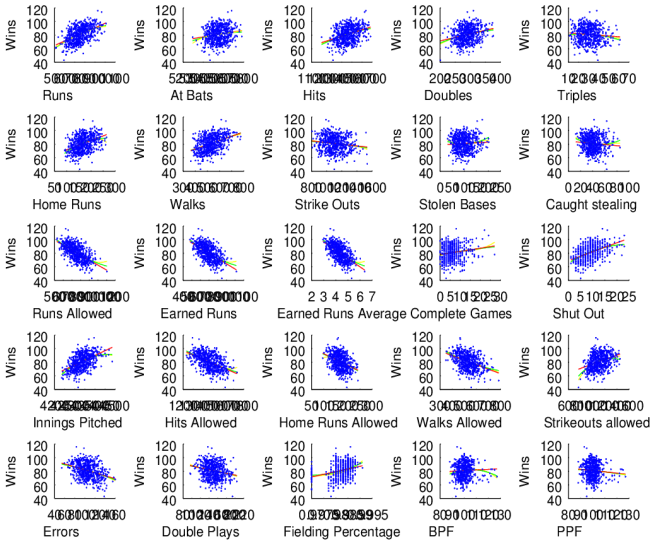


4 Discussion

4.1 Linear Regression

The linear regression models allowed us to predict team win totals with high accuracy. Both ridge and polynomial regression proved to be reliable methods to make our predictions. The strong correlation between Run Difference and number of wins agrees with the formula suggested by the Society of American Baseball Research. [5] As mentioned, other variables such as Earned Runs and Saves had a small statistical significance in relation to wins, but the rest of our data had seemingly no real significance in predicting wins. We suspect that more advanced models primarily use run difference as well in order to make their predictions.

Fig. 4. A matrix plot showing the relationship between each feature in the original data set and season wins.



4.2 Neural Networks

The sigmoid NN produced by far the highest MSE of all our prediction methods, while the linear NN produced an MSE almost on par with the ridge regression. A probable reason for the sigmoid's bad performance is due to the fact that using one-hot vectors for classifiers does not allow for dynamic error values, instead using a fixed 10 distances from the other arrays. This means that it becomes hard for the network to fine tune itself when it can only use these static values, and if the weights needed are in-between the two oscillating weights, back-propagation might never reach it in this case.

Based on this assumption, it is no surprise that the linear NN did so well, especially considering its use of both weights and linear regression to calculate its output. By using both methods, we were able to make use of the weak correlations to seed the weights in the network to the hidden nodes, and then regression to make use of the strong covariance between Runs Difference and wins.

4.3 General

In both neural networks and regression models, the reduced dataset fared better in all variations. This makes sense intuitively because it means using baseball knowledge to modify and remove columns is more effective than using a flat algorithm to transform the data. In addition, removing and combining different

variables resulted in a cleaner and more compact dataset.

The best models were the neural network with a linear activation function and the ridge regression. By adjusting the MSE's to reflect the same metric, percentage of wins, instead of number of wins, we find that ridge regression is only slightly better than the linear activation neural network, 0.0925 versus 0.0950. Therefore, the best method would be to combine the two predictors by averaging the two results.

By examining the various features in the datasets, we found that run difference had the highest correlation with total wins, which was reflected in the models. Thinking back to how runs difference was generated, this also makes intuitive sense. Runs difference gives a difference of runs scored and runs allowed. Teams that scored more while allowing fewer points by the opposing team had higher wins in the season than teams that had lower values. Not surprisingly, this indicates that teams should focus on building a team with a high-powered offense that can also prevent an opposing team's runs to achieve more success throughout the season.

5 Conclusion

When we started the project on this topic, we felt that it had been mostly overlooked because there was more use to predicting based on a team of players, and that predicting based on teams was just underdeveloped. However, as our analysis took shape, we came to realize that the reason this approach is mostly overlooked is that it is mostly already fully solved. We found significant statistical correlation based on Runs Difference and slight statistical correlation on things relating to it, like Earned Runs Average and Saves, and the rest of the data was mostly insignificant. Based on the overwhelming correlation, we found that it was easy to build an accurate predictor that relied mostly on Runs Difference, which was consistent with our findings in the real world. However, we believe that this paper has gone at least a small way to illuminating other metrics by which we can predict performance. If a further team finds a way or algorithm to magnify or combine these slightly correlated statistics, then we could come up with a different competing metric by which to predict by, and could generate more accurate predictions by either combining or averaging the new metric and Runs Difference.

References

1. R. Jia, C. Wong, and D. Zeng, "Predicting the major league baseball season," 2013.
2. S. L. Barnes and M. V. Bjarnadottir., "Great expectations: An analysis of major league baseball free agent performance.," June 2016.
3. G. Ganeshapillai and J. Guttag, "Predicting the next pitch," 2012.

4. R. B. Palm, "Prediction as a candidate for learning deep hierarchical models of data," 2012.
5. S. Rothman, "A new formula to predict a team's winning percentage," 2014.
6. P. Beneventano, P. D. Berger, and B. D. Weinberg, "Predicting run production and run prevention in baseball: the impact of sabermetrics," *Int J Bus Humanit Technol*, vol. 2, no. 4, pp. 67–75, 2012.
7. J. Albert, "An introduction to sabermetrics," *Bowling Green State University (http://www-math.bgsu.edu/albert/papers/saber.html)*, 1997.
8. T. Y. Yang and T. Swartz, "A two-stage bayesian model for predicting winners in major league baseball.," *Journal of Data Science*, no. 2, pp. 61–73, 2004.
9. N. Paine, "How to predict mlb records from early results," Apr 2014.
10. J. Boice, "How our 2016 mlb predictions work," Apr 2016.
11. O. S. Schumaker, Robert P. and H. Chen, "Predictive modeling for sports and gaming," 2010.
12. G. Donaker, "Applying machine learning to mlb prediction & analysis," Dec 2005.

6 Author contributions

Nicholas Alonzo: Wrote the code for Cross Validation with Ridge Regression to find the optimal lambda and then predicted number of wins for new data.

Ehsan Arasteh: Worked on literature review and citations, reviewed Neural network code and proofreaded their report

Ari Bolton: Helped set up both the NN classifier and linear predictor, and decide which features to use. Finished the linear NN and prepared error output.

Joel Fischbein: Worked with the neural network models to get data and information for the report and presentation. Helped write the neural network section of report and interpret results.

Chris Fong: Wrote the polynomial regression code with cross validation. Helped write the polynomial regression in methods and results section.

Liandy Hardikoesoemo: Wrote the literature review, proofread and edited Neural Network writeup.

David Huang: Created the Reduced variables dataset during the feature seletion process, so as to drop unimportant and unrelated variables. Wrote the Reduced variabls section of the report.

Ryan Kausin: Helped with polynomial regression bootstrapping. Helped write introduction and linear regression discussion

Cecil Lam: Discussed earlier feature selection, proofread report, contributed to references

Nathan McCall: Helped with ridge and polynomial regression, as well as determining how to compare results between models. Used models to collect figures to illustrate results.

Daniel Schlesinger: Set up the neural network code, did configuration and testing for the neural network, and recorded results of testing. Also did resource gathering, proofreading on the report, and general crisis management and organization.

Tommer Schwarz: Helped with initial feature selection process, part of the neural network team. Helped write abstract and introduction.

Alexander Soong: Tested the neural network configurations. Wrote the neural network report and presentation. Helped with formulating the problem and proposing the approach.

Kevin Wang: Wrote script for PCA data generation, helped prepare datasets for teams to use, wrote sections on feature selection, PCA dataset, original dataset, parts of results section, and conclusion.

Kathleen Zhen: Wrote the bootstrap ridge regression code and report, gathered 2016 teams and predicted their respective wins