Machine learning to predict baseball game outcomes: *Moneyball II*

Yixin Liu, Neelang Parghi, Yi Zhou Advisor: David Frohardt-Lane

May 18, 2018

Abstract

In this project, we used machine learning algorithms to see if we could predict the probability of whether a team would win a major league baseball game. This information would certainly be of use to gamblers, many of whom employ their own strategies for predicting winners. To give us certain thresholds that we would try to exceed, a number of simple baseline algorithms were implemented and our goal became to beat their success rate, thus giving our model an edge in terms of winning percentage.

1 Introduction

The American Gaming Association estimates that $36.5 billion was wagered on MLB games last year[1]. With such big money changing hands, it's not surprising that gamblers will look at whatever statistics, streaks, and superstitions they can think of that might give them an advantage. Our approach was to see if we could accomplish this by looking strictly at what the data was telling us.

2 The Data

Thanks to our advisor, David Frohardt-Lane, we were provided with a wealth of data from the past six baseball seasons, including pre-season and post-season data. We focused on the regular season data, each of which is comprised of 162 games per team. The data provides details on every at-bat and every pitch of every game. We know dates, locations, player names, batting outcomes, which bases have players on them at the time, and many more features were available if needed.

One of the first fundamental questions in any project of this sort is figuring out how to divide the data into training and testing sets. Since we had six full seasons of data at our disposal, we experimented with holding out the 2017 season as the testing set and the rest as training, and also holding out both the 2016 and 2017 data as testing. After some experimentation, we quickly saw that predictive performance worsens when we use training data beyond the previous two years. This was because the models have no way of "forgetting" where the older games have less impact than more recent games. As more training history

[1] `https://www.americangaming.org/newsroom/press-releasess/americans-wager-over-36-billion-major-league-baseball-season`

is introduced, recent performance becomes less relevant.

With that in mind, we decided to consider each season as a separate dataset and split them using the first 80% of the games as training data and the last 20% as testing data. Before we could begin, one initial hurdle was that the data was arranged in an inning-by-inning basis and not in terms of discrete games. In other words, there was no way to quickly look at the data and distinguishing where one game ended and the next began. We remedied this by creating a function[2] to split the data into individual games whenever the inning number was reset to 1.

3 Models

3.1 Baseline algorithms

Our next step was to define and implement our baseline algorithms. These would give us some less sophisticated results that we could compare against. Descriptions of the three baselines we implemented are below and their results are summarized in Figure 1.

1. *Presuming that each home team will win 52% of the time.*

   We randomly assign the home team a 52% probability of winning. This is a relatively simple method that was designed to give us slightly better results than randomly guessing the outcome. This is labeled as 'Fixed-Prob.'

2. *Computing the accumulated probability of each team to win considering the total number of runs made by each team before the current game.*

   Here, we predict the team that has scored the highest cumulative number of runs in previous games within the season would be more likely to win. This is labeled as 'Scored-Runs-Prob.'

3. *Looking at the ratio of runs scored over runs allowed and whichever team has the highest ratio is the predicted winner.*

   This gave us a quick metric to compare each team's performance in terms of offense and defense. It seemed natural that whichever team scored the most runs while allowing the least runs would have a higher probability of winning, thus the team with the highest ratio was the predicted winner[3] This is labeled as 'Runs-Ratio.'

[2] All coding work was done in Python 3.

[3] We were aware that this ratio may be skewed in the early portion of each season. We normalized this via Laplace smoothing, where instead of calculating the ratio as $\frac{scored}{allowed}$, we used $\frac{scored+1}{allowed+2}$. Interestingly, this ultimately degraded our results so we left the ratio as-is.

	Model	Brier	ROC-AUC	PR-AUC	Accuracy
0	Fixed-Prob	0.252248	0.500000	0.453790	0.453790
1	Scored-Runs-Prob	0.248091	0.552591	0.505833	0.547594
2	Runs-Ratio	0.245229	0.584253	0.520360	0.562825

Figure 1: Results of baseline testing.

3.2 Machine Learning algorithms

Now that we had our baselines in place, our next step was to see if we could use machine learning to improve upon their results.

3.2.1 Models

Since this is a binary classification problem, logistic regression was a natural choice since it's output is a binary variable. However we were mindful that our predictions should be in terms of probabilities rather than 0's and 1's so we made sure to use the `predict_proba` classifier provided in `sklearn`. We hand-picked a number of different C parameters, which determine the regularization strength, and also used cross-validation to find an optimized logistic regression. In addition, we used a bagging classifier which distributes random subsets of the total data to an ensemble of classifiers and aggregates their results to form a final prediction. Another ensemble method we employed was random forest, which constructs multiple decision trees and outputs the mean prediction[4] In recent weeks, we've learned about neural networks in David Rosenberg's class so we decided to include multilayer perceptron as well.

3.2.2 Features

Our first foray into using machine learning was to train a logistic regression model where the best regularization strength was chosen using K-fold cross validation. We considered the historical number of runs scored, allowed, number of games played by each team, and their respective win rates (labeled as 'Base model'). Next, we added a 'bag-of-words' of players that participated in the game to this base model ('Base model + players'). We built up the model further by adding historical statistics of each type of batting result (e.g. pop out, walk, strikeout, etc.) by each team ('Base model + players + desc').

We focused on the number of runs scored and allowed since this is what ultimately determines the winner of the game. Considering the outcomes for each at-bat listed in the 'descr' column also seemed to be a natural choice since teams showing higher amounts of outcomes such as 'single' and 'double' and 'walk' would be expected to be more successful, while teams with higher amounts of outcomes such as 'strikeout' and 'groundout' would be expected to not be as successful.

[4]If this were a binary classification problem, the output would be the mode prediction of all the trees.

Some new features we added were starting pitcher, average of the team's previous scores, and outcome of the previous game, and win/loss ratio of the previous 10 games. While the final score of each game could be seen by glancing at the score columns for the final inning row of each game, we added a 'result' column to the dataframe containing a binary 0 or 1 for whether the team won or lost. This was the outcome column necessary for making our predictions.

4 Measuring success

We compared all predictions with their actual outcomes by using the Brier score:

$$BS = \frac{1}{N}\sum_{t=1}^{N}(f_t - o_t)^2$$

where f is the prediction probability, o is the outcome in [0,1], and N is the number of games. In words, this is the sum of the square difference between the predicted probability and the actual result.

We chose the Brier score as our main metric because it accounts for the extent which each team deviates from what traditional odds would tell us. This measure is important because baseball (and betting in general!) is unpredictable. Algorithms can look at the data and make their predictions, which may be fine for a binary classification problem. But when looking at probabilities of winning, there are certain differences in algorithms that the Brier score captures.

The Brier score is important because it takes into account a prediction's reliability and resolution. *Reliability* is how well a prediction reflects the actual outcome. At the moment, the New York Yankees have won 70% of their games this season. If we had always predicted that they would win on 70% of the days they played, this prediction would have high reliability but would not be very useful. Over a longer period of time, we can expect this winning rate to change. *Resolution* is how well an algorithm differentiates different probabilities. Predictions with perfect resolution would only contain 0% or 100% probabilities of winning. While these prediction are perfectly resolved, they have low reliability. The Brier score takes each prediction, subtracts either 0 if the predicted outcome did not occur or 1 if it did, and squares this difference. The average of these squares is computed to produce the final score[5].

In addition, traditional machine learning metrics such as ROC-AUC, PR-AUC, and accuracy where also produced.

[5] `http://www.cawcr.gov.au/projects/verification/reliability_resolution.html`

	Model	Brier	ROC-AUC	PR-AUC	Accuracy
13	Voting classifier	0.242854	0.597979	0.546797	0.578401
10	Runs-Ratio	0.245229	0.584253	0.519859	0.562825
3	Base model	0.245803	0.568089	0.506188	0.548287
14	Scored-Runs-Prob	0.248091	0.552591	0.504975	0.547594
1	Fixed-Prob	0.252248	0.500000	0.726895	0.453790
2	BaggingClassifier	0.262177	0.561689	0.506227	0.560402
5	Base model + players + desc	0.262777	0.589765	0.524893	0.563863
0	Random forest	0.265164	0.541522	0.495306	0.547940
7	Optimized LogisticRegressionCV	0.267736	0.583152	0.523172	0.562825
11	Base model + players	0.279531	0.585300	0.522351	0.562825
12	Logistic regression(C=.8)	0.296283	0.588802	0.520866	0.564901
9	Logistic regression(C=1.)	0.303611	0.587161	0.519125	0.564555
8	Logistic regression(C=1.1)	0.306912	0.586456	0.518226	0.565247
6	Logistic regression(C=1.2)	0.309894	0.585816	0.517567	0.566632
4	MLPClassifier	0.314162	0.566729	0.511304	0.532364

Figure 2: Table of overall results across all seasons.

5 Results

As we see in Figure 2, our arbitrary Voting Classifier outperformed every other model. The Voting Classifier comes from Python's `sklearn` library and is built upon combinations of different algorithms from `sklearn` and their respective weights that the user chooses. We chose a combination of the three best performing algorithms from `sklearn`, which were Random Forest, Logistic Regression, and Bagging and gave each equal weighting. We decided to include this after seeing that the best performing algorithms differed season-to-season[6]

Surprisingly, the second-best performing algorithm in terms of Brier score was 'Runs-Ratio,' which is the baseline algorithm looking at the ratio of runs scored over runs allowed and predicting whichever team has the highest ratio as the winner. In fact, we see that all three of our baseline algorithms ranked in the top five when looking at the Brier score.

These results also demonstrate the importance of the Brier score under these conditions. At first, there was some skepticism about the results after seeing how the Brier score result would sometimes contradict the ROC-AUC, which we had considered to be the standard for measuring results in any machine learning problem. But as we investigated further and learned how the Brier score

[6]Individual season results available in the appendix.

is ideally suited for this type of problem, it soon made sense why this conflict might occur and why the other metrics may not be as reliable. The fundamental lesson was that different prediction problems have different yardsticks to be used for measuring success.

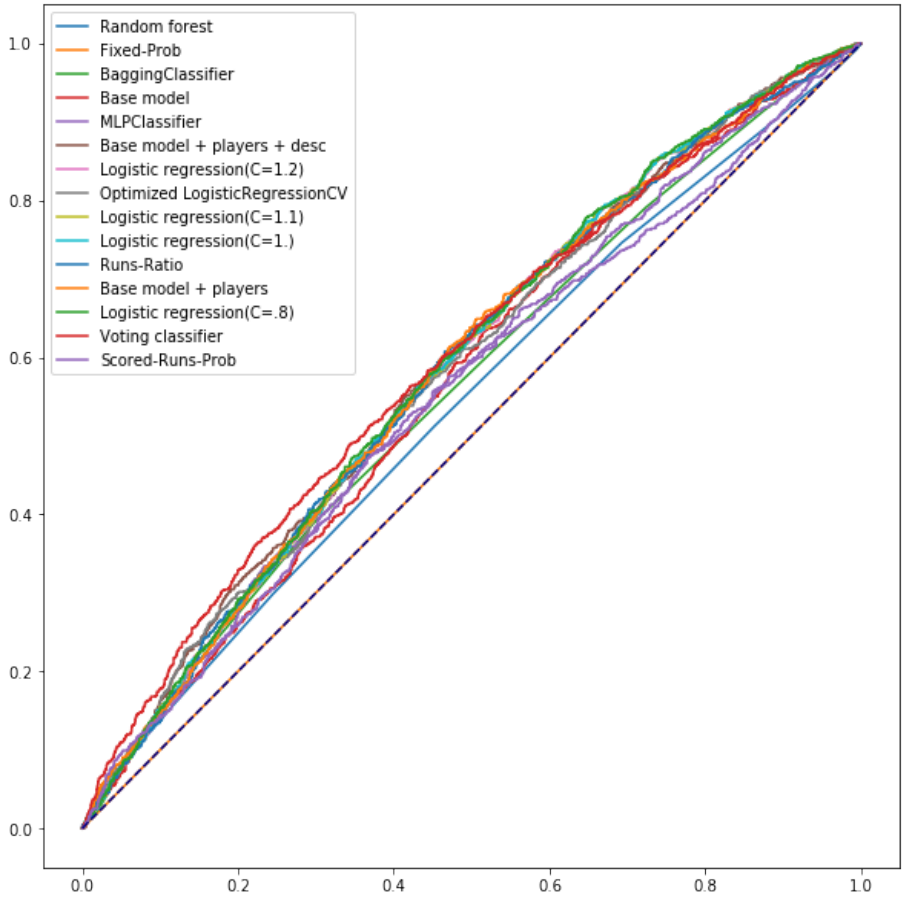


Figure 3: ROC curve plot of total results across all seasons.

6 Discussion

Our results demonstrated the need for flexibility in sports prediction. Our Voting Classifier performed the best because it took into account multiple algorithms, rather than a "one-size-fits-all" approach. Even then, the algorithms which composed the Voting Classifier may not be best choices in future seasons. They performed well enough during the 2012-2017 seasons for inclusion in the classifier, but they may not fair as well in future seasons. Thus if someone wanted to use a Voting Classifier strategy, we would recommend that they also monitor the performance of its component algorithms and adjust the classifier accordingly if any of them don't perform as well. We would also strongly sug-

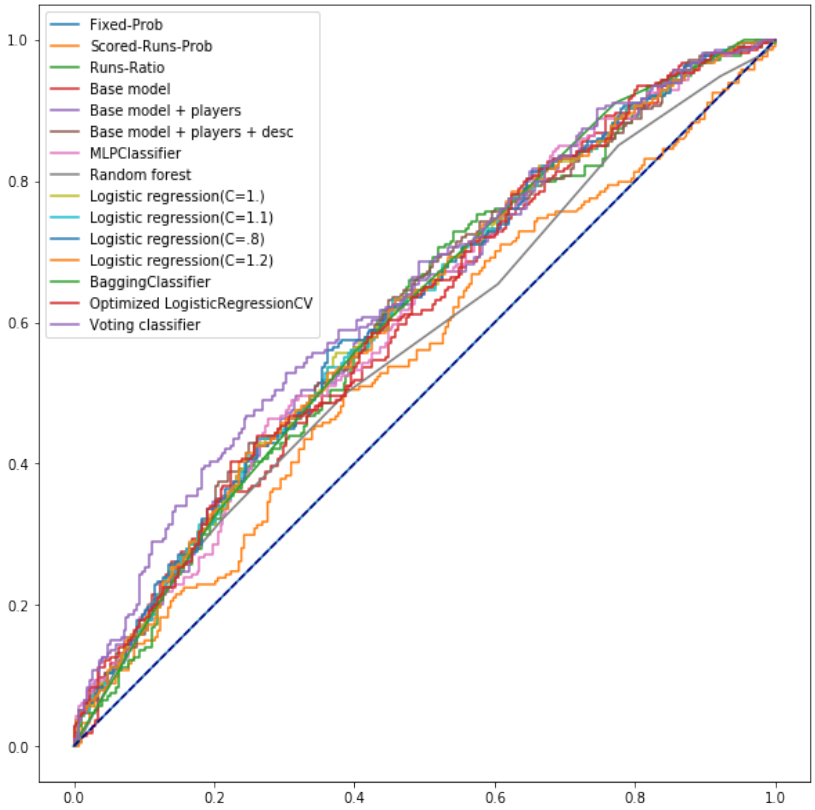
gest that they consider the risk in using the Voting Classifier. One idea we had was to use cross validation to find the perfect mix of component algorithms and their perfect respective weights. This could possibly produce great results but such a classifier would severely overfit the data and thus be useless when future data is added.

We also realize that using the first 80% of a season's data to make predictions on the final 20% is far from ideal for gambling purposes. This would amount to asking gamblers to wait until August to place any bets so that enough training data can be collected. Future work could involve looking at pre-season data to see if predictions can be made about games earlier in the regular season. While this is a much smaller sample size of data to work with, it certainly seems worthwhile in exploring to see if any insight can be drawn between pre-season and initial regular season performance. Another approach to this problem is to further explore individual player statistics. While a team that has a very successful record one year can have a much poorer record in the next year, this generally isn't the case for individual players. An all-star player with a .325 batting average can be generally expected to maintain that output as long as he remains healthy, even if his team doesn't perform too well. Thus looking at teams whose members have strong batting, pitching, and fielding statistics could be another approach to predicting results earlier in the season, moving beyond the simple 'bag-of-words' approach we employed for this project.

One more aspect that would also be of particular interest to gamblers would be to not only predict the game winner, but also to predict the final score. Predicting the winner isn't always necessary to win a bet. If the scores can be predicted reliably enough to allow the point spread to be beaten consistently, then this would also be very useful.

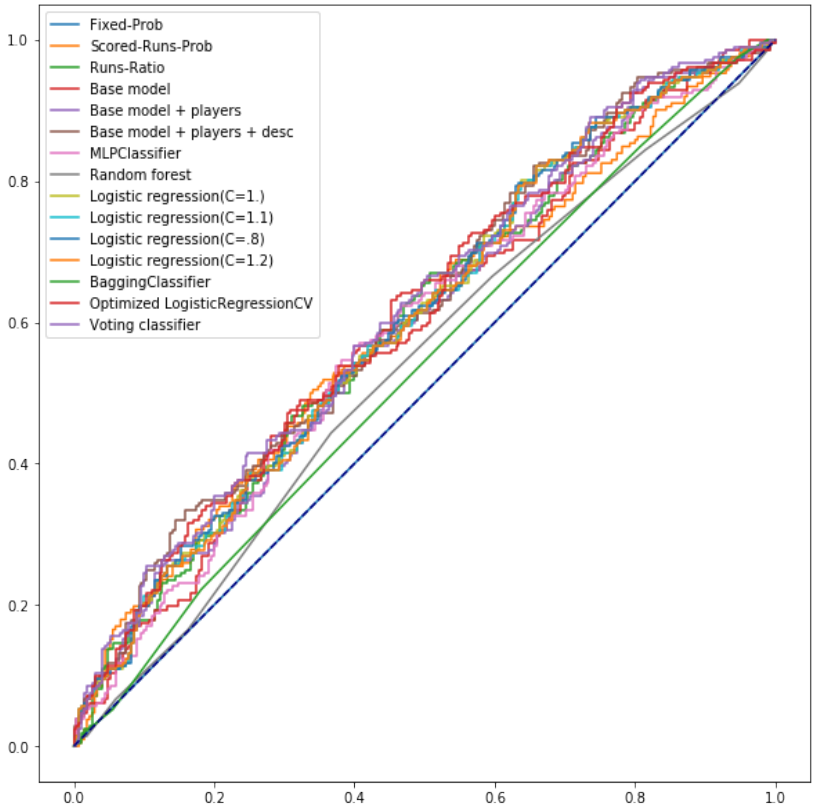
Appendix

A 2012 Results



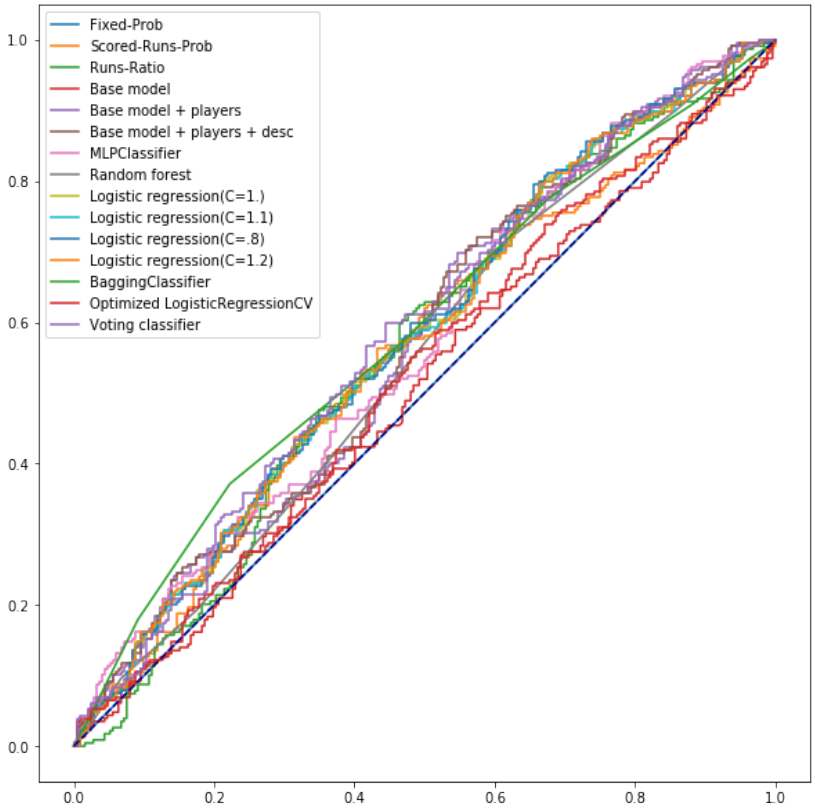
	Model	Brier	ROC-AUC	PR-AUC	Accuracy
14	Voting classifier	0.233396	0.642541	0.587815	0.605809
3	Base model	0.242385	0.600746	0.531367	0.553942
2	Runs-Ratio	0.243023	0.605890	0.530520	0.576763
1	Scored-Runs-Prob	0.247954	0.549562	0.491682	0.545643
12	BaggingClassifier	0.252365	0.613492	0.535626	0.580913
0	Fixed-Prob	0.252641	0.500000	0.721992	0.443983
5	Base model + players + desc	0.258243	0.614678	0.555288	0.580913
7	Random forest	0.265560	0.570913	0.501574	0.562241
13	Optimized LogisticRegressionCV	0.272100	0.612673	0.559504	0.572614
10	Logistic regression(C=.8)	0.280700	0.615637	0.556987	0.576763
4	Base model + players	0.281227	0.614538	0.556783	0.572614
8	Logistic regression(C=1.)	0.286902	0.614050	0.555507	0.578838
9	Logistic regression(C=1.1)	0.290070	0.613492	0.554327	0.580913
11	Logistic regression(C=1.2)	0.292629	0.612847	0.553492	0.578838
6	MLPClassifier	0.309971	0.609674	0.555066	0.512448

B 2013 Results



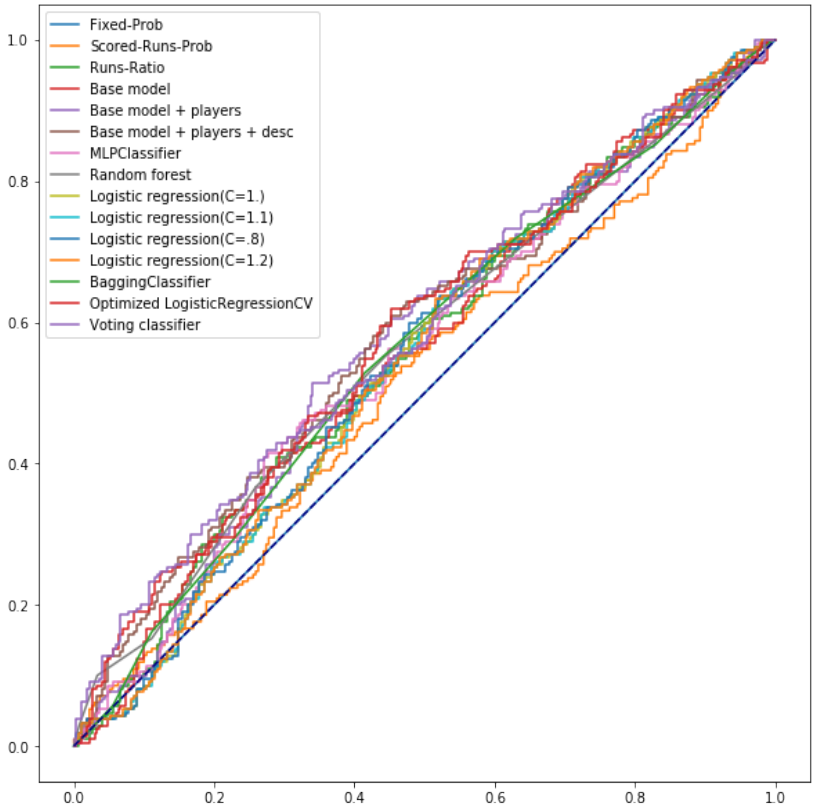
	Model	Brier	ROC-AUC	PR-AUC	Accuracy
14	Voting classifier	0.236012	0.617191	0.549082	0.593361
3	Base model	0.240091	0.593501	0.522544	0.574689
2	Runs-Ratio	0.243808	0.597449	0.521861	0.570539
1	Scored-Runs-Prob	0.245803	0.594410	0.527945	0.566390
0	Fixed-Prob	0.252807	0.500000	0.719917	0.439834
5	Base model + players + desc	0.256034	0.613889	0.561125	0.585062
12	BaggingClassifier	0.262593	0.535543	0.461481	0.556017
7	Random forest	0.265871	0.537072	0.463104	0.541494
13	Optimized LogisticRegressionCV	0.270698	0.608648	0.554256	0.578838
10	Logistic regression(C=.8)	0.286475	0.604018	0.543052	0.576763
8	Logistic regression(C=1.)	0.294319	0.602131	0.541005	0.572614
9	Logistic regression(C=1.1)	0.297863	0.601275	0.539339	0.578838
11	Logistic regression(C=1.2)	0.300878	0.600804	0.538646	0.576763
6	MLPClassifier	0.303288	0.588871	0.524006	0.576763
4	Base model + players	0.308875	0.594899	0.538517	0.574689

C 2014 Results



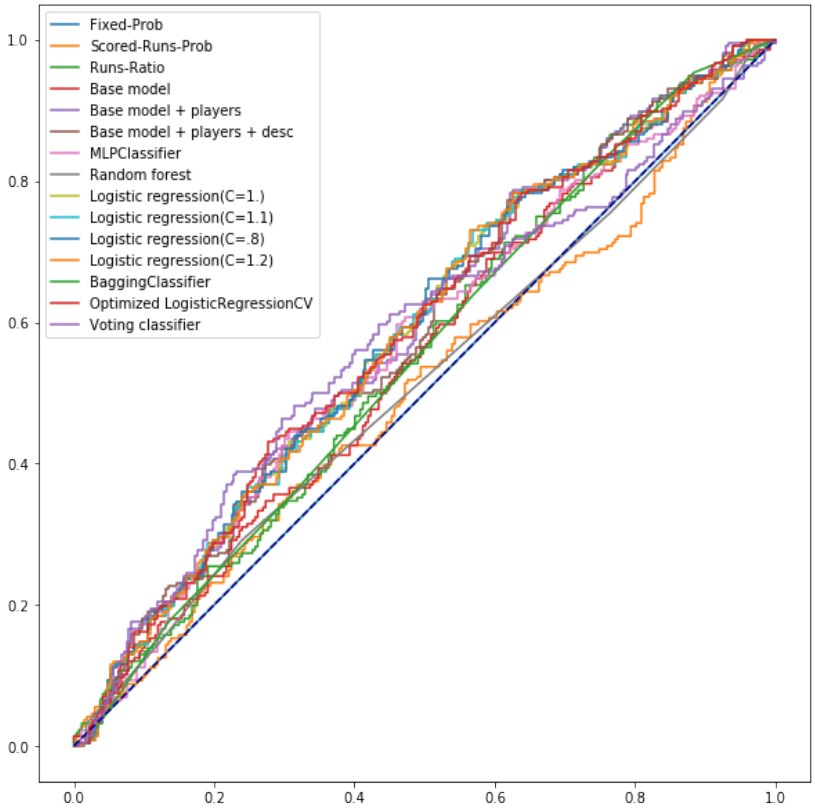
	Model	Brier	ROC-AUC	PR-AUC	Accuracy
14	Voting classifier	0.244870	0.583524	0.545175	0.565489
2	Runs-Ratio	0.248182	0.561187	0.496577	0.559252
1	Scored-Runs-Prob	0.248198	0.557505	0.523604	0.553015
5	Base model + players + desc	0.250594	0.570926	0.536493	0.530146
4	Base model + players	0.250962	0.569366	0.534299	0.534304
0	Fixed-Prob	0.251356	0.500000	0.738046	0.476091
12	BaggingClassifier	0.252682	0.586193	0.569427	0.584200
13	Optimized LogisticRegressionCV	0.254124	0.526599	0.503059	0.515593
3	Base model	0.256451	0.507035	0.479634	0.519751
7	Random forest	0.263077	0.547411	0.513550	0.523909
10	Logistic regression(C=.8)	0.299899	0.577840	0.532109	0.548857
8	Logistic regression(C=1.)	0.307430	0.575986	0.530462	0.550936
9	Logistic regression(C=1.1)	0.310856	0.574686	0.529972	0.555094
11	Logistic regression(C=1.2)	0.313620	0.574080	0.529807	0.559252
6	MLPClassifier	0.331785	0.569262	0.542025	0.513514

D 2015 Results



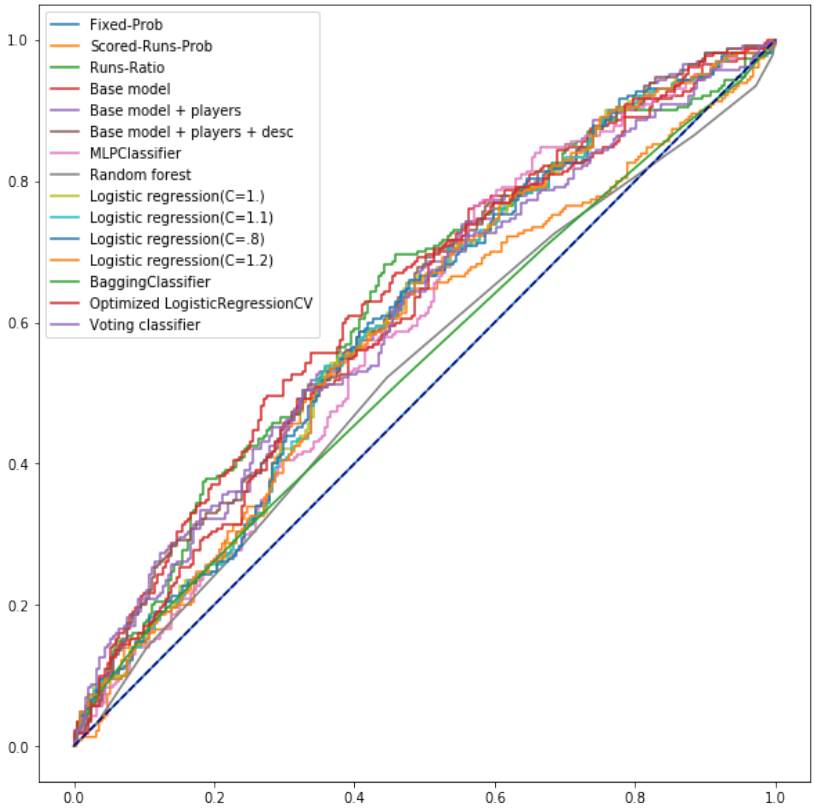
	Model	Brier	ROC-AUC	PR-AUC	Accuracy
13	Optimized LogisticRegressionCV	0.246710	0.580582	0.514109	0.568750
14	Voting classifier	0.247560	0.584303	0.537721	0.545833
2	Runs-Ratio	0.247584	0.559277	0.474540	0.539583
3	Base model	0.248306	0.560406	0.477853	0.550000
1	Scored-Runs-Prob	0.249405	0.517522	0.463716	0.533333
5	Base model + players + desc	0.249422	0.581164	0.517659	0.575000
0	Fixed-Prob	0.252900	0.500000	0.718750	0.437500
7	Random forest	0.255354	0.564938	0.519580	0.577083
12	BaggingClassifier	0.273875	0.560917	0.483658	0.560417
4	Base model + players	0.276750	0.571429	0.493342	0.570833
10	Logistic regression(C=.8)	0.324184	0.552716	0.468230	0.550000
8	Logistic regression(C=1.)	0.332639	0.550423	0.464835	0.543750
6	MLPClassifier	0.336042	0.554497	0.479542	0.489583
9	Logistic regression(C=1.1)	0.336316	0.550106	0.464441	0.535417
11	Logistic regression(C=1.2)	0.339595	0.549806	0.464388	0.537500

E 2016 Results



	Model	Brier	ROC-AUC	PR-AUC	Accuracy
3	Base model	0.246160	0.551264	0.491729	0.530021
2	Runs-Ratio	0.247613	0.552911	0.495080	0.532091
1	Scored-Runs-Prob	0.249794	0.509398	0.465168	0.513458
0	Fixed-Prob	0.252512	0.500000	0.723602	0.447205
14	Voting classifier	0.253719	0.577698	0.510269	0.577640
12	BaggingClassifier	0.260207	0.550692	0.473636	0.540373
7	Random forest	0.266667	0.516438	0.466380	0.550725
6	MLPClassifier	0.280990	0.565508	0.494873	0.565217
10	Logistic regression(C=.8)	0.304249	0.584599	0.504009	0.556936
8	Logistic regression(C=1.)	0.311405	0.583403	0.502405	0.559006
13	Optimized LogisticRegressionCV	0.314148	0.584755	0.503431	0.556936
9	Logistic regression(C=1.1)	0.314451	0.583715	0.502257	0.561077
11	Logistic regression(C=1.2)	0.317623	0.583680	0.502441	0.563147
4	Base model + players	0.320436	0.586732	0.505380	0.546584
5	Base model + players + desc	0.323365	0.581478	0.503373	0.534161

F 2017 Results



	Model	Brier	ROC-AUC	PR-AUC	Accuracy
4	Base model + players	0.238696	0.621185	0.582552	0.577963
5	Base model + players + desc	0.238749	0.620925	0.580157	0.577963
2	Runs-Ratio	0.241169	0.623818	0.593898	0.598753
3	Base model	0.241448	0.635822	0.608916	0.561331
14	Voting classifier	0.241569	0.606167	0.589668	0.582121
1	Scored-Runs-Prob	0.247395	0.571315	0.531065	0.573805
13	Optimized LogisticRegressionCV	0.248384	0.607535	0.571622	0.584200
0	Fixed-Prob	0.251273	0.500000	0.739085	0.478170
12	BaggingClassifier	0.271393	0.538524	0.537017	0.540541
7	Random forest	0.274428	0.531795	0.507173	0.532225
10	Logistic regression(C=.8)	0.282267	0.601386	0.564275	0.580042
8	Logistic regression(C=1.)	0.289055	0.600468	0.564011	0.582121
9	Logistic regression(C=1.1)	0.292002	0.599532	0.562850	0.580042
11	Logistic regression(C=1.2)	0.295101	0.598666	0.561241	0.584200
6	MLPClassifier	0.323108	0.591547	0.546130	0.536383