Predict Baseball WAR with Classic Stats

HyunJun Kim

Abstract

Wins Above Replacement (WAR) is a widely used metric in baseball. However, it is too complex to calculate. Therefore, we built a position player's WAR predicting model only using 12 easily accessible classic stats. The final model is an ensemble stacking model, which consists of two best-performing models, random forest and XGBoost. The two base model was chosen among 8 different algorithms, by hyperparameters random searching. The final model has a strong performance not only in predicting WAR of the season, but also another seasons. By using the model and machine learning methodology we used, there will be an innovative change in baseball.

Introduction

Wins Above Replacement (WAR) is one of the most widely used, but the most complex baseball sabermetrics. It describes the number of additional wins the player's team has achieved by using the player instead of a replacement-level player [1]. (A replacement-level player does not mean the substitutes of the same position, it means the minimum-level player, who can be added to the team for minimal cost and effort [2].)

WAR has various useful properties. First, it is intuitive because the value means the number of wins itself. For example, if a team appoints a player of WAR 3 instead of 0, the team will get 3 more wins. Second, WAR is useful to compare multiple players in a different position. One of the important motivations of devising WAR is to line up players in a line without considering their positions [1]. Therefore, WAR of a pitcher and an outfielder have the same meaning.

There is no standardized formula to calculate WAR. Each baseball stats website uses its own method, and each method is slightly different from others. The following is a basic formula of WAR [3].

$$WAR = \frac{\begin{array}{c}(Batting\ Runs + Base\ Running\ Runs + Fielding\ Runs \\ + Positional\ Adjustment + League\ Adjustment + Replacement\ Runs)\end{array}}{(Runs\ Per\ Win)}$$

Batting Runs means the contribution of batting. It is calculated by using stats like wOBA (weighted On-Base Average), AVG (Average). It also considers the park factor, the environment of each stadium. Base Running Runs is about the contribution of base running such as success of steals. Fielding Runs is about defense contribution. Positional adjustment is determined by the position of players, providing an advantage to hard positions such as a catcher. Replacement Runs is an additional score based on a replacement-level player [3].

As shown above, calculating WAR is complicating because it contains 6 components, and each component is determined by multiples of factors. Additionally, some factors are not classic stats. (Classic stats are used for a long time because of easy calculation, such as AVG (Average), OBP (On Base Percentage), SLG (Slugging Average).) Therefore, additional complex calculation should be involved to get a factor.

Considering these facts, WAR is a double-edged sword, the most powerful but hard to get and understand. If there is a tool that can predict WAR with classic stats, it will be like an innovation.

There were a few studies that applied machine learning to baseball. A study focuses on predicting the pitcher's next pitch by using Support Vector Machine (SVM) and k-nearest neighbor (k-NN) [4,5] Another study used machine learning and deep learning to predict the result of Major League Baseball (MLB) matches (or playoffs) [6,7]. There was also a study utilized machine learning to identify undervalued players [8]. However, WAR was barely covered, and most of studies are focused on MLB dataset.

In our study, we built a WAR predicting model (for position players) by using 12 classic features, which are AB (At Bat), R (Runs), HR (Home Run), TB (Total Bases), RBI (Runs Batted In), BB (Based on Balls), IB (Intentional Based on Balls), SF (Sacrifice Fly), AVG (Average), OBP (On Base Percentage), SLG (Slugging Average), and OPS (On Base Plus Slugging). We used all batters' stats from 2024 season Korea Baseball Organization (KBO) League. In this study, we tested 8 different algorithms, including a neural network, and hyperparameters of each algorithm is searched for 20 iterations. After going through the process, we finally found the best hyperparameter combinations two models, random forest and XGBoost, and we used them as base estimators of a stacking model.

Results

1. Dataset analysis

First, we looked at a dataset. We used 13 features including WAR, and their correlation to WAR is shown in **Figure 1** and **Table 1**. We found out that R is the most related feature to WAR.

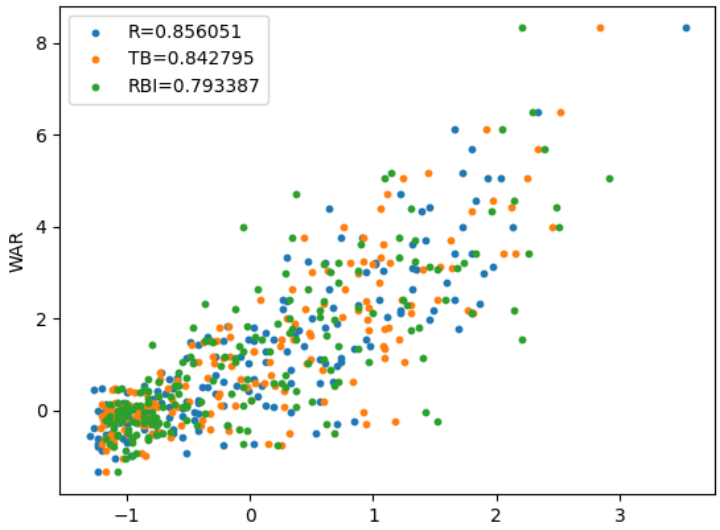

Figure 1. Scatter graph of WAR with respect to three major features. The legend shows the correlation coefficient. R, TB, and RBI values are all standardized.

Table 1. Coefficient of features with respect to WAR.

Rank	Features	Coefficient
1	WAR	1
2	R	0.856051
3	TB	0.842795
4	RBI	0.793387
5	BB	0.792048
6	AB	0.772725
7	OPS	0.747921
8	SLG	0.693797
9	OBP	0.692001
10	HR	0.681543
11	AVG	0.668566
12	IB	0.65256
13	SF	0.633494

Also, we checked the distribution of each feature (See **Figure 2**). As the distribution of R, HR, TB, RBI, BB, IB, and SF are concentrated to the left (have a long tail), we used log transformation to make the distribution similar to normal distribution. During the process, we made values less than 0 into 1, which was necessary to apply log function. Similarly, we applied a log transformer to WAR, by adding 1.351 (the absolute value of smallest WAR) first. Therefore, we always applied an exponential function and subtract 1.351 to the prediction of models.

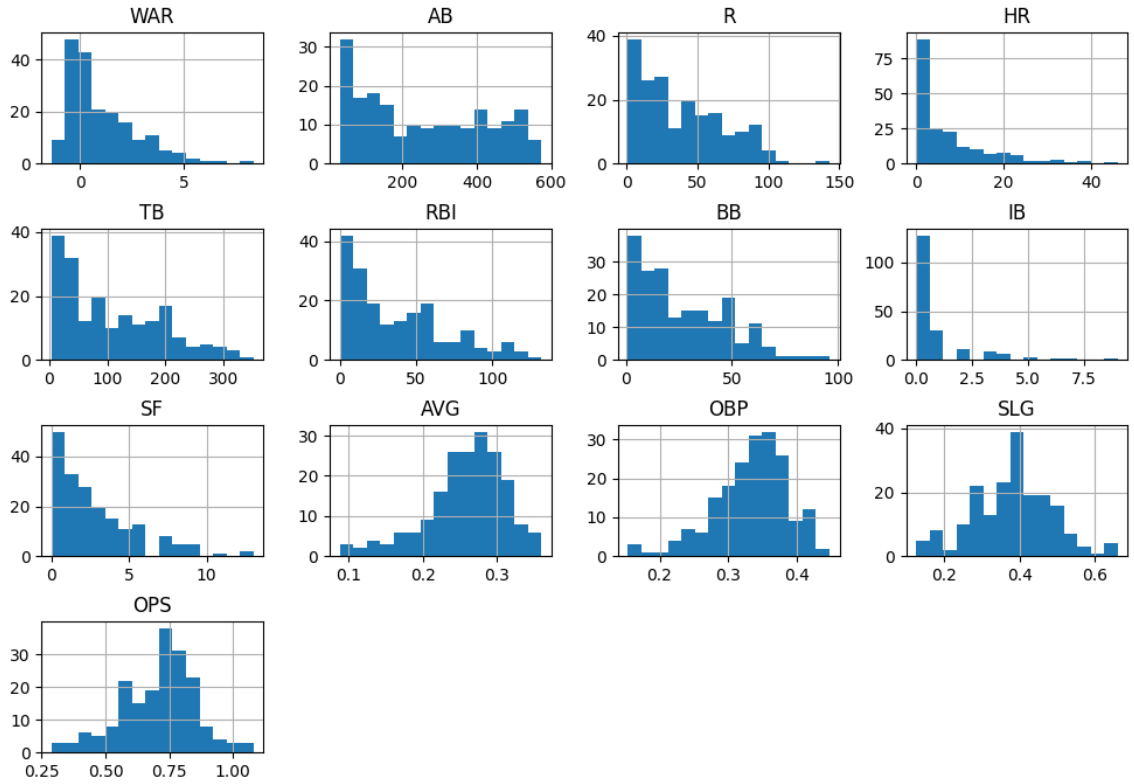

Figure 2. Histograms of features. WAR, R, HR, TB, RBI, BB, IB, and SF have left-concentrated distributions.

2. Hyperparameter Searching

To select appropriate models, we randomly searched hyperparameters. Twenty iterations were conducted for each model.

Measuring RMSE was important, however, whether predictions are in an appropriate range of WAR is also important. Therefore, we used WAR of 5 players from 2023 KBO seasons to check predicted WAR value is useful.

2-1. Nonlinear SVM

Support Vector Machine (SVM) is a powerful tool for small size dataset [9]. To reflect a variety of data, we used nonlinear SVM. First, we conducted hyperparameter tuning with respect to degree, C, and epsilon values. C and epsilon are regularization hyperparameters, which are inversely proportional to regularization [9].

We ran 20 iterations of randomized search. Degree values are randomly chosen between 2 and 11, C and epsilon are random between 0.01 and 0.1. The result is shown in **Table 2** (and **Additional Table 1**).

Table 2. Top 3 hyperparameter searching result of Nonlinear SVM. Mean and std are the results of 3 cross validations. For all 20 iterations, see **Additional Table 1**.

Rank	epsilon	degree	C	mean	std
1	0.02	3	0.09	-0.6017	0.2428
2	0.08	3	0.10	-0.6103	0.2561
3	0.02	7	0.03	-0.6830	0.1695

However, for the best hyperparameters (degree=3, C=0.09, epsilon=0.02), the train and test set RMSE was not satisfactory. Train RMSE was 0.9467 and test RMSE was 1.2723. The prediction result for 2023 players were worse, see **Table 3**.

Table 3. 2023 Players prediction results of all algorithms.

	Chang-Ki Hong	Austin Dean	Jae-Gyun Hwang	Shin-Soo Choo	Eun-Won Jung
Real WAR	6.67	4.97	3.14	1.72	0.74
Nonlinear SVM	-1.351	43.899	-1.351	-1.35	-1.351
Random Forest	5.797	4.63	3.272	3.689	1.437
Adaboost	3.54	3.531	3.292	3.308	1.849
Gradient Boosting	3.067	2.083	1.685	1.929	1.714
XGBoost	8.252	5.427	3.182	3.346	1.473
LightGBM	4.444	4.487	2.728	3.633	0.87
Catboost	3.697	4.631	2.697	3.807	1.1195
Neural Network	14.14	38.266	6.059	7.388	5.786

2-2. Random Forest

Random forest is an ensemble of decision trees, a powerful tool for complicated datasets [10]. To prevent overfitting, hyperparameters setting is needed.

In random forests, we set 'the number of estimators' and 'maximum depth of trees' as hyperparameters to search. The number of estimators is randomly selected between 10 and 100, and the maximum depth is randomly selected between 4 and 20.

Table 4 shows the results of hyperparameters searching.

Table 4. Top 3 hyperparameter searching result of all algorithms (except SVM). The name of algorithms are the concatenation of name and their rank. All values are the results of cross validation. For all results of 20 iterations, go to **Additional Materials**.

Algorithm and its Rank	n_estimators	Max_depth	Learning_rate	mean	std
Random Forest_1	98	10	-	-0.5441	0.3047
Random Forest_2	46	7	-	-0.5457	0.3024
Random Forest_3	90	7	-	-0.5457	0.3037
Adaboost_1	50	-	0.000302	-0.5405	0.3172
Adaboost_2	65	-	0.007576	-0.5426	0.3129
Adaboost_3	60	-	0.001918	-0.5433	0.3160
Gradient Boosting_1	91	14	0.009394	-0.6287	0.3147
Gradient Boosting_2	86	5	0.007778	-0.6581	0.3062
Gradient Boosting_3	75	11	0.008384	-0.6645	0.3090
XGBoost_1	90	7	-	-0.5283	0.3115
XGBoost_2	46	7	-	-0.5283	0.3115

XGBoost_3	50	14	-	-0.5348	0.3100
LightGBM_1	35	5	-	-0.5504	0.3250
LightGBM_2	18	10	-	-0.5525	0.3396
LightGBM_3	39	17	-	-0.5531	0.3226
Catboost_1	90	7	-	-0.5192	0.3212
Catboost_2	88	5	-	-0.5240	0.3242
Catboost_3	46	7	-	-0.5255	0.3195

The best number of estimators was 98, and the best maximum depth was 10. By using the values, train RMSE was 0.2768, and test RMSE was 0.8474. Prediction results of 2023 players were encouraging (see **Table 3**).

2-3. AdaBoost

Boosting is a method to connect multiple week learners to make a strong learner [11]. AdaBoost is to increase the weights of samples which previous predictors underfitted [12].

For AdaBoost, we tuned the number of estimators and learning rate. The number of estimators is randomly selected between 10 and 100, and learning rate is random between 0.0001 and 0.01.

The best combination was 'n_estimators=50' and 'learning rate=0.000302'. By using the values, train RMSE was 0.7801, and test RMSE was 0.9476. Prediction results of 2023 players were not bad, but the performance to catch high WAR was not enough.

2-4. Gradient Booosting

Gradient boosting is a kind of boosting. It uses residual error of a prior predictor to train the next predictor [13,14].

For gradient boosting model, we tuned the number of estimators, maximum depth of trees, and learning rate. The number of estimators is randomly selected between 10 and 100, the maximum depth is random between 4 and 20, and learning rate is random between 0.0001 and 0.01.

The best hyperparameters were 'n_estimators=91', 'max_depth=14', and 'learning_rate= 0.009393877551020408'. Train RMSE was 0.9743, and test RMSE was 1.072. Prediction results of 2023 players are in **Table 3.**

2-5. XGBoost

XGBoost is an enhanced gradient boosting. The difference of XGBoost to Gradient Boosting is the existence of regularization method and parallel processing [15]. XGBoost is one of the most widely used algorithms in data science competitions [16].

For XGBoost, we tuned the number of estimators and maximum depth. The best combination was 'n_estimators=90', 'max_depth=7'. The train RMSE was 0.0026, and test RMSE was 0.9209. Prediction results of 2023 players were fine.

2-6. LightGBM

LightGBM is based on gradient boosting, but it has a few differences. LightGBM uses a leaf-wise tree growth, which can make a stronger tree with a risk of overfitting [17]. LightGBM is also a widely used algorithm in competitions.

We decided to tune the same two hyperparameters of XGBoost. The best number of estimators was 35, and the best maximum depth was 5.

The RMSE of the best combination is train 0.5968, test 0.7813. RMSE score seems fine, however, prediction results of 2023 players are not satisfactory.

2-7. Catboost

Catboost is a variation of gradient boosting. It is designed to handle categorical variables efficiently [18]. It also uses ordered boosting, using train and dev set for each step to prevent overfitting [19].
We decided to tune the same two hyperparameters as XGBoost and LightGBM. The best number of estimators was 90, and the best maximum depth was 7.
The RMSE of the best combination is train 0.1748, test 0.7647. RMSE score seems fine, however, prediction results of 2023 players were not satisfactory.

2-8. Neural Network

We also check whether neural networks were effective. The hyperparameters, and their range is shown in **Table 5**. Hyperparameter searching was done for 50 iterations.

Table 5. Hyperparameters and their range in neural networks.

Hyperparameters	Range
the number of hidden layers	4-12
The number of neurons in each layer	32-128
Learning rate	0.0001-0.01
Optimizer	Adam, NAG, RMSprop

The best score validation RMSE was 0.8984, with 8 hidden layers, 36 neurons, learning rate 0.005218086982564764 by RMSprop. However, the test RMSE was 1.11847, and the results of 2023 players are also bad.

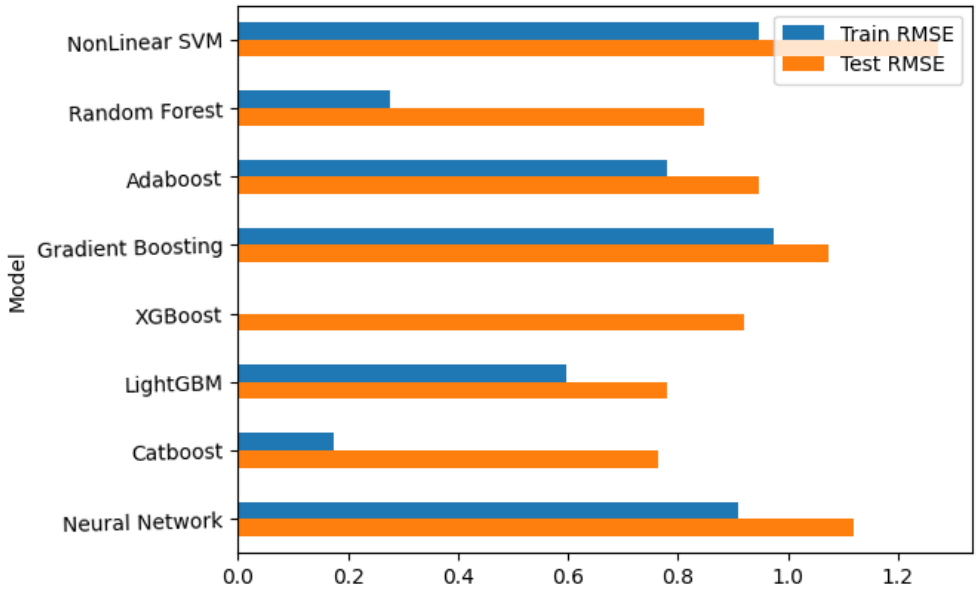

Figure 3. Comparison of RMSE across models. Train RMSE of XGBoost doesn't show up, as the value is too small.

3. Final Model

To take advantage of multiple models, we decided to use a stacking model. Several rules were used to select the component estimators.

- Decent RMSE score for test set.
- Good prediction result for 2023 players WAR.

The second rule was important, as our goal was to predict WAR. Too low value (like negative) or too large value (e.g. 38) is not appropriate to use it as a prediction of WAR.

According to the rules, we used Random Forest model, and XGBoost model as base estimators of a stacking model. The stacking model also contained a standardization (StandardScaler), and the final estimator was a linear regression model. The structure is in **Figure 4**.

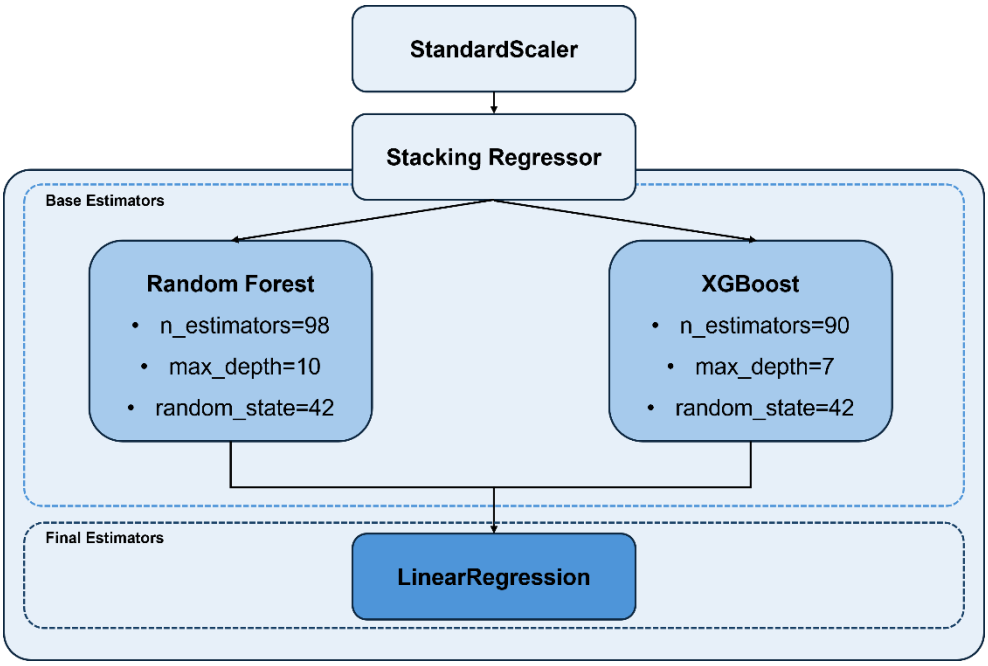

Figure 4. The structure of final model, using a stacking regressor.

Train RMSE of the final model was 0.2133, and test RMSE was 0.9747. The prediction results of 2023 players are in **Table 6**.

Table 6. 2023 Players Prediction results of the final stacking model

Name	Real WAR	Predicted WAR
Chang-Ki Hong	6.67	8.727
Austin Dean	4.97	5.928
Jae-Gyun Hwang	3.14	3.558
Shin-Soo Choo	1.72	3.847
Eun-Won Jung	0.74	1.522

The final model has a good RMSE score, and also works well on the data of the prior season.

Discussion

We built an ensemble stacking model that can predict WAR to a fairly accurate level. Among 8 machine learning algorithms, random forest and XGBoost were the most powerful. Therefore, they were used as base estimators of a stacking model. The final model had fine RMSE, and it also had overwhelming results of predicting WAR of another season. Although WAR of Shin-Soo Choo had a noticeable difference, it was a result of stat itself, not a result of wrong model. This can be cross verified from other various results of models above.

The model we built could be used widely in the real field of baseball. It can work as a simple tool to get a player's WAR in a short time or a metric to compare players on the recruiting list. Additionally, our approach, using machine learning algorithms in baseball, still has a lot of possibilities. From predicting a small component of a ball game, such as pitch type of next pitch or the match result, to rookie draft, trades success, and even marketing. It is hard to imagine all probabilities one by one.

Currently, machine learning and Artificial Intelligence (AI) are becoming a new standard of global industries. Baseball should follow the changing paradigm, and this study will be the trigger for the start.

Methods

Dataset

We used all KBO first team players' data from 2024 season, which is crawled from STATIZ(https://statiz.sporki.com), one of the biggest baseball (KBO) statistics websites. It provides multiple stats for batters, oWAR (offence WAR), dWAR (defense WAR), G (Games), PA (Plate Appearance), ePA (effective Plate Appearance), AB (At Bat), R (Runs), H (Hits), 2B (Double), 3B (Triple), HR (Home Run), TB (Total Bases), RBI (Runs Batted In), SB (Stolen Bases), CS (Caught Stealing), BB (Based on Balls), HP(Hit by Pitch), IB (Intentional Based on Balls), SO (Strike Out), GDP (Grounded into Double Play), SH (Sacrifice Hit), SF (Sacrifice Fly), AVG (Average), OBP (On Base Percentage), SLG (Slugging Average), OPS (On Base Plus Slugging), R/ePA (Runs per effective Plate Appearance), wRC+ (weighted Runs and Created Plus), and WAR. In this research, we only used 12 features (already mentioned in **Introduction**) that are closely related to WAR (by correlation coefficient).

Preprocessing

After crawling data and selecting features, we picked data which is only useful. We considered that players' data who appeared too little are not helpful to build a model. Therefore, we used players' data whose AB is more than 30.
The next preprocessing step (such as a log transformer) for model building is mentioned in **Results**.

Scoring Metrics

We used RMSE (Root Mean Squared Error). However, for Scikit-learn models, there was an error that all scores are shown 'nan' because of too small RMSE. Therefore, we used MSE (Mean Squared Error) only for hyperparameters searching in Scikit-learn models.
As log transformers were applied to target value of train set we first applied exponential function to the output and then calculated RMSE.

Models

1. Scikit-learn

We used 4 models of Scikit-learn, including SVR(Support Vector Machine Regression), RandomForestRegressor, AdaboostRegressor, and GradientBoostingRegressor.
Ramdomized hyperparameter searching (RandomizedSearchCV), and other multiple steps (such as standardization, train and test set splitting, and defining error) also used Scikit-learn's frameworks.

2. TensorFlow

TensorFlow is used to build a neural network. We built a sequential model that batch normalizations and dense layers were repeated in hidden layers. At the output layer, one batch normalization layer and a dense layer with one unit were used.

3. Others

XGBoost, LightGBM, and Catboost regressors were all installed in advance and used.

Environments

All works were implemented in Google Colab (Python 3 Google Compute Engine), with free plan.

Notebooks

All notebooks are uploaded in Github (https://github.com/Hyun3246/Predict-WAR-with-Classic-Stats).

Reference

1 Slowinski, P. *What is WAR?*, <https://library.fangraphs.com/misc/war/> (2010).

2 Slowinski, P. *Replacement Level*, <https://library.fangraphs.com/misc/war/replacement-level/> (2010).

3 Slowinski, P. *WAR for Position Players*, <https://library.fangraphs.com/war/war-position-players/> (2010).

4 Hamilton, M. *et al.* in *Proceedings of the 3rd International Conference on Pattern Recognition Applications and Methods* 520-527 (2014).

5 Koseler, K. & Stephan, M. Machine Learning Applications in Baseball: A Systematic Literature Review. *Applied Artificial Intelligence* **31**, 745-763 (2018). https://doi.org/10.1080/08839514.2018.1442991

6 Huang, M.-L. & Li, Y.-Z. Use of Machine Learning and Deep Learning to Predict the Outcomes of Major League Baseball Matches. *Applied Sciences* **11** (2021). https://doi.org/10.3390/app11104499

7 Yaseen, A. S., Marhoon, A. F. & Saleem, S. A. Multimodal Machine Learning for Major League Baseball Playoff Prediction. *Informatica* **46** (2022). https://doi.org/10.31449/inf.v46i6.3864

8 Ishii, T. Using Machine Learning Algorithms to Identify Undervalued Baseball Players. (2016).

9 developers, s.-l. *SVR*, <https://scikit-learn.org/1.5/modules/generated/sklearn.svm.SVR.html> (2025).

10 Louppe, G. & Geurts, P. in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2012, Bristol, UK, September 24-28, 2012. Proceedings, Part I 23.* 346-361 (Springer).

11 Géron, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition*. (O'Reilly Media, Inc, 2022).

12 Freund, Y. & Schapire, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences* **55**, 119-139 (1997).

13 Breiman, L. Arcing the edge. (Citeseer, 1997).

14 Friedman, J. H. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189-1232 (2001).

15 Chen, T. & Guestrin, C. in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (2016).

16 Wade, C. *Hands-On Gradient Boosting with XGBoost and scikit-learn*. (Packt Publishing, 2020).

17 Ke, G. *et al.* Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* **30** (2017).

18 Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V. & Gulin, A. CatBoost: unbiased boosting with categorical features. *Advances in neural information processing systems* **31** (2018).

19 Dorogush, A. V., Ershov, V. & Gulin, A. CatBoost: gradient boosting with categorical features support. *arXiv preprint arXiv* **1810** (2018).

Additional Materials

Additional Table 1. Hyperparameter searching results of Nonlinear SVM. Split_0, 1, 2 are the result of 3 cross-validations.

Rank	epsilon	degree	C	split0	split1	split2	mean	std
1	0.02	3	0.09	-0.4288	-0.9451	-0.4312	-0.6017	0.2428
2	0.08	3	0.10	-0.4233	-0.9724	-0.4353	-0.6103	0.2561
3	0.02	7	0.03	-0.5619	-0.9228	-0.5643	-0.6830	0.1695
4	0.01	5	0.02	-0.6749	-0.9912	-0.5119	-0.7260	0.1990
5	0.10	5	0.01	-0.6994	-0.9988	-0.5530	-0.7504	0.1855
6	0.01	4	0.02	-0.6640	-1.1612	-0.6193	-0.8149	0.2456
7	0.10	4	0.04	-0.7101	-1.1432	-0.5974	-0.8169	0.2353
8	0.01	7	0.07	-0.6854	-1.2364	-0.6214	-0.8477	0.2761
9	0.09	6	0.07	-0.9933	-1.1674	-0.5446	-0.9018	0.2624
10	0.10	6	0.02	-1.0319	-1.1901	-0.6154	-0.9458	0.2424
11	0.03	9	0.01	-0.6766	-1.7288	-0.5892	-0.9982	0.5178
12	0.01	9	0.01	-0.6707	-1.7629	-0.5873	-1.0070	0.5356
13	0.08	8	0.01	-0.7765	-1.9908	-0.5777	-1.1150	0.6246
14	0.09	9	0.02	-1.1040	-2.6196	-0.7768	-1.5001	0.8028
15	0.03	8	0.05	-3.7344	-2.9214	-0.8035	-2.4864	1.2355
16	0.07	10	0.01	-6.8612	-7.9545	-0.7784	-5.1980	3.1569
17	0.08	10	0.05	-12.6180	-12.9775	-1.6212	-9.0722	5.2707
18	0.04	10	0.06	-14.4759	-14.6568	-1.7323	-10.2883	6.0505
19	0.02	10	0.08	-16.6371	-18.4185	-1.9339	-12.3298	7.3869
20	0.02	10	0.09	-16.8254	-20.6099	-2.0125	-13.1493	8.0250

Additional Table 2. Hyperparameter searching results of random forest. Split_0, 1, 2 are the result of 3 cross-validations.

Rank	n_estimators	Max_depth	split0	split1	split2	mean	std
1	98	10	-0.3324	-0.9750	-0.3249	-0.5441	0.3047
2	46	7	-0.3425	-0.9731	-0.3215	-0.5457	0.3024
3	90	7	-0.3418	-0.9751	-0.3202	-0.5457	0.3037
4	59	10	-0.3467	-0.9688	-0.3286	-0.5480	0.2977
5	88	5	-0.3505	-0.9773	-0.3190	-0.5489	0.3032
6	75	11	-0.3534	-0.9717	-0.3235	-0.5495	0.2988
7	75	10	-0.3524	-0.9720	-0.3249	-0.5498	0.2988
8	46	12	-0.3466	-0.9723	-0.3318	-0.5503	0.2985
9	50	19	-0.3500	-0.9712	-0.3301	-0.5504	0.2976
10	50	14	-0.3503	-0.9712	-0.3301	-0.5505	0.2976
11	61	5	-0.3598	-0.9709	-0.3215	-0.5507	0.2975
12	63	17	-0.3571	-0.9686	-0.3269	-0.5509	0.2956
13	82	19	-0.3573	-0.9724	-0.3241	-0.5513	0.2981
14	74	14	-0.3564	-0.9716	-0.3258	-0.5513	0.2975
15	74	6	-0.3575	-0.9719	-0.3254	-0.5516	0.2975
16	55	18	-0.3568	-0.9710	-0.3291	-0.5523	0.2963
17	75	4	-0.3616	-0.9758	-0.3275	-0.5549	0.2979
18	35	5	-0.3723	-0.9729	-0.3294	-0.5582	0.2937
19	39	17	-0.3687	-0.9719	-0.3360	-0.5589	0.2924

20	18	10	-0.4062	-0.9724	-0.3348	-0.5711	0.2852

Additional Table 3. Hyperparameter searching results of Adaboost. Split_0, 1, 2 are the result of 3 cross-validations.

Rank	n_estimators	learning_rate	split0	split1	split2	mean	std
1	50	0.000302	-0.2981	-0.9886	-0.3347	-0.5405	0.3172
2	65	0.007576	-0.3058	-0.9848	-0.3372	-0.5426	0.3129
3	60	0.001918	-0.3034	-0.9898	-0.3369	-0.5433	0.3160
4	92	0.008384	-0.3137	-0.9844	-0.3370	-0.5450	0.3108
5	77	0.009192	-0.3110	-0.9840	-0.3405	-0.5451	0.3105
6	34	0.007778	-0.3130	-0.9839	-0.3397	-0.5455	0.3102
7	59	0.001716	-0.3056	-0.9870	-0.3450	-0.5459	0.3123
8	42	0.006969	-0.3101	-0.9844	-0.3435	-0.5460	0.3103
9	26	0.01	-0.3110	-0.9825	-0.3459	-0.5464	0.3087
10	34	0.00111	-0.3104	-0.9872	-0.3424	-0.5467	0.3118
11	26	0.00111	-0.3106	-0.9882	-0.3413	-0.5467	0.3124
12	48	0.005757	-0.3084	-0.9840	-0.3493	-0.5472	0.3093
13	61	0.005353	-0.3091	-0.9842	-0.3501	-0.5478	0.3090
14	75	0.003737	-0.3112	-0.9856	-0.3493	-0.5487	0.3093
15	49	0.006565	-0.3051	-0.9867	-0.3555	-0.5491	0.3101
16	34	0.006565	-0.3176	-0.9842	-0.3457	-0.5491	0.3078
17	31	0.007171	-0.3151	-0.9858	-0.3466	-0.5492	0.3090
18	12	0.002524	-0.3108	-0.9771	-0.3611	-0.5497	0.3029
19	24	0.002727	-0.3138	-0.9824	-0.3529	-0.5497	0.3064
20	13	0.005555	-0.3233	-0.9790	-0.3595	-0.5539	0.3009

Additional Table 4. Hyperparameter searching results of Gradient Boosting. Split_0, 1, 2 are the result of 3 cross-validations.

Rank	n_estimators	Max_depth	learning_rate	split0	split1	split2	mean	std
1	91	14	0.009394	-0.3761	-1.0723	-0.4376	-0.6287	0.3147
2	86	5	0.007778	-0.4103	-1.0896	-0.4746	-0.6581	0.3062
3	75	11	0.008384	-0.4123	-1.0997	-0.4814	-0.6645	0.3090
4	77	8	0.007576	-0.4227	-1.1067	-0.4925	-0.6739	0.3073
5	63	17	0.008384	-0.4431	-1.1184	-0.5075	-0.6897	0.3043
6	55	15	0.008788	-0.4570	-1.1270	-0.5188	-0.7009	0.3023
7	45	5	0.009192	-0.4860	-1.1384	-0.5424	-0.7223	0.2952
8	60	12	0.005757	-0.5073	-1.1559	-0.5619	-0.7417	0.2937
9	83	18	0.003939	-0.5143	-1.1609	-0.5685	-0.7479	0.2929
10	32	19	0.01	-0.5182	-1.1618	-0.5695	-0.7498	0.2921
11	29	7	0.009596	-0.5347	-1.1706	-0.5860	-0.7638	0.2885
12	30	4	0.009192	-0.5472	-1.1730	-0.5957	-0.7720	0.2843
13	41	14	0.006161	-0.5485	-1.1790	-0.5952	-0.7742	0.2869
14	34	17	0.005151	-0.5879	-1.2024	-0.6268	-0.8057	0.2810
15	55	19	0.00212	-0.6208	-1.2211	-0.6530	-0.8316	0.2757
16	65	9	0.000908	-0.6556	-1.2411	-0.6797	-0.8588	0.2705

17	13	6	0.002322	-0.6750	-1.2516	-0.6947	-0.8738	0.2673
18	11	15	0.000706	-0.6893	-1.2606	-0.7062	-0.8854	0.2654
19	60	13	0.0001	-0.6905	-1.2613	-0.7072	-0.8863	0.2652
20	59	12	0.0001	-0.6905	-1.2613	-0.7072	-0.8864	0.2652

Additional Table 5. Hyperparameter searching results of XGBoost. Split_0, 1, 2 are the result of 3 cross-validations..

Rank	n_estimators	Max_depth	split0	split1	split2	mean	std
1	90	7	-0.2827	-0.9678	-0.3345	-0.5283	0.3115
2	46	7	-0.2827	-0.9678	-0.3345	-0.5283	0.3115
3	50	14	-0.2993	-0.9728	-0.3323	-0.5348	0.3100
4	74	14	-0.2994	-0.9728	-0.3323	-0.5348	0.3100
5	39	17	-0.2995	-0.9728	-0.3324	-0.5349	0.3100
6	50	19	-0.2995	-0.9728	-0.3323	-0.5349	0.3099
7	55	18	-0.2995	-0.9728	-0.3323	-0.5349	0.3099
8	63	17	-0.2995	-0.9728	-0.3323	-0.5349	0.3099
9	82	19	-0.2995	-0.9728	-0.3323	-0.5349	0.3099
10	46	12	-0.2993	-0.9728	-0.3332	-0.5351	0.3098
11	59	10	-0.3027	-0.9728	-0.3338	-0.5365	0.3088
12	75	10	-0.3027	-0.9728	-0.3338	-0.5365	0.3088
13	98	10	-0.3027	-0.9728	-0.3338	-0.5365	0.3088
14	18	10	-0.3007	-0.9744	-0.3345	-0.5366	0.3099
15	75	11	-0.3047	-0.9728	-0.3332	-0.5369	0.3085
16	75	4	-0.2706	-0.9899	-0.3531	-0.5379	0.3214
17	35	5	-0.2892	-0.9855	-0.3416	-0.5388	0.3166
18	88	5	-0.2901	-0.9858	-0.3417	-0.5392	0.3165
19	61	5	-0.2901	-0.9858	-0.3417	-0.5392	0.3165
20	74	6	-0.2831	-0.9882	-0.3571	-0.5428	0.3164

Additional Table 6. Hyperparameter searching results of LightGBM. Split_0, 1, 2 are the result of 3 cross-validations.

Rank	n_estimators	Max_depth	split0	split1	split2	mean	std
1	35	5	-0.3059	-1.0097	-0.3356	-0.5504	0.3250
2	18	10	-0.2801	-1.0313	-0.3459	-0.5525	0.3396
3	39	17	-0.3121	-1.0091	-0.3382	-0.5531	0.3226
4	46	12	-0.3178	-1.0088	-0.3375	-0.5547	0.3212
5	46	7	-0.3178	-1.0088	-0.3375	-0.5547	0.3212
6	50	14	-0.3206	-1.0092	-0.3373	-0.5557	0.3207
7	50	19	-0.3206	-1.0092	-0.3373	-0.5557	0.3207
8	55	18	-0.3226	-1.0093	-0.3380	-0.5567	0.3201
9	63	17	-0.3242	-1.0098	-0.3379	-0.5573	0.3200
10	59	10	-0.3232	-1.0094	-0.3393	-0.5573	0.3198
11	61	5	-0.3245	-1.0097	-0.3380	-0.5574	0.3199
12	75	11	-0.3251	-1.0114	-0.3382	-0.5582	0.3205
13	75	4	-0.3251	-1.0114	-0.3382	-0.5582	0.3205
14	75	10	-0.3251	-1.0114	-0.3382	-0.5582	0.3205
15	74	14	-0.3263	-1.0116	-0.3374	-0.5584	0.3205

16	74	6	-0.3263	-1.0116	-0.3374	-0.5584	0.3205
17	82	19	-0.3272	-1.0123	-0.3414	-0.5603	0.3197
18	88	5	-0.3274	-1.0128	-0.3415	-0.5606	0.3198
19	90	7	-0.3278	-1.0128	-0.3425	-0.5610	0.3195
20	98	10	-0.3293	-1.0137	-0.3433	-0.5621	0.3194

Additional Table 7. Hyperparameter searching results of Catboost. Split_0, 1, 2 are the result of 3 cross-validations.

Rank	n_estimators	Max_depth	split0	split1	split2	mean	std
1	90	7	-0.2710	-0.9728	-0.3138	-0.5192	0.3212
2	88	5	-0.2665	-0.9812	-0.3242	-0.5240	0.3242
3	46	7	-0.2666	-0.9756	-0.3341	-0.5255	0.3195
4	75	4	-0.2770	-0.9914	-0.3173	-0.5286	0.3277
5	35	5	-0.3030	-0.9673	-0.3216	-0.5306	0.3089
6	61	5	-0.2713	-0.9760	-0.3520	-0.5331	0.3149
7	75	11	-0.2869	-0.9799	-0.3426	-0.5364	0.3144
8	74	6	-0.2840	-0.9779	-0.3516	-0.5379	0.3124
9	98	10	-0.2845	-0.9920	-0.3431	-0.5399	0.3206
10	75	10	-0.2900	-0.9879	-0.3451	-0.5410	0.3168
11	59	10	-0.2911	-0.9917	-0.3471	-0.5433	0.3179
12	18	10	-0.3201	-0.9900	-0.3273	-0.5458	0.3141
13	74	14	-0.3079	-0.9838	-0.3469	-0.5462	0.3099
14	46	12	-0.3310	-0.9680	-0.3463	-0.5485	0.2967
15	50	14	-0.3359	-0.9722	-0.3403	-0.5495	0.2989
16	82	19	NaN	NaN	NaN	NaN	NaN
17	63	17	NaN	NaN	NaN	NaN	NaN
18	50	19	NaN	NaN	NaN	NaN	NaN
19	55	18	NaN	NaN	NaN	NaN	NaN
20	39	17	NaN	NaN	NaN	NaN	NaN