ML Final Project Report

Team: 努估誰呦　Member: B10201051 Yu-Ling, Liao

---

1 Introduction

1.1 Background

This study evaluates the performance of 4 machine learning models with various techniques and preprocessing methods for predicting Major League Baseball (MLB) game outcomes. We aim to identify the best-performing model across two stages: Stage 1, predicting historical game results (2016–2023), and Stage 2, predicting all game results in 2024.

1.2 Model Selection

We selected four machine learning models: Logistic Regression, Random Forest, Support Vector Machine (SVM), and XGBoost. These models were chosen for their ability to handle structured data and capture various patterns in the dataset.

1.3 Methodology Overview

Data preprocessing involves feature engineering and handling missing values. To optimize model performance, we employ advanced techniques such as **cross-validation**, **train-validation split**, and optimization methods like **Optuna** and **Bayesian optimization**.

1.4 Report Structure

This report is organized as follows. Section 2 describes the data preprocessing techniques and the features built for the models. Section 3 introduces the machine learning models and the methods for training. Section 4 presents the results and compares the models' performance. Finally, Section 5 provides recommendations and concludes the study.

2 Data Preprocessing

We experimented with multiple strategies to handle missing values, categorical features, and numerical features. We developed two main directions for data preprocessing, depending on how object-type feature were treated:

2.1 Detailed Processing

There are 8 object-type features in the training dataset: `'home_team_abbr'`, `'away_team_abbr'`, `'date'`, `'is_night_game'`, `'home_pitcher'`, `'away_pitcher'`, `'home_team_season'`, and `'away_team_season'`. We dealt with them one by one as follows:

1. **Target Encoding with `'home_vs_away'` Combination**:

   We created a column `'home_vs_away'` by combining the home and away team abbreviation. For each unique combination, the home team's win rate was calculated based on

the training dataset. The resulting new feature, `'home_vs_away_win_rate'`, was added to both the training dataset and the testing dataset. If a combination in the testing dataset did not appear in the training dataset, the missing value was filled with the home team's average win rate.

2. **Dropping Irrelevant Date and Season Information**:

   Columns such as `'date'`, `'home_team_season'`, and `'away_team_season'` were dropped, leaving only the `'season'` feature. For missing `'season'` values, they were filled using the year extracted from either `'home_team_season'` or `'away_team_season'`, or `'date'`.

3. **Converting `'is_night_game'` to Boolean**:

   The `'is_night_game'` column was transformed into a boolean format. For missing values, the proportion of night games for each home team was calculated. If the proportion exceeded 0.5, missing values were filled with True; otherwise, they were filled with False.

4. **Handling Pitcher Data**:

   The names of the home and away pitchers were replaced with their respective win rates: `'home_pitcher_win_rate'` and `'away_pitcher_win_rate'`. These rates were calculated based on historical game outcomes. If a pitcher's win rate was missing, the value was filled using the corresponding team's average win rate: `'home_team_win_rate'` for home pitchers and `'away_team_win_rate'` for away pitchers.

After processing all object-type features, they were transformed into numerical features. Next, we handled the null values in the dataset. Data with more than 80 missing values were discarded as they were deemed too incomplete for meaningful analysis. For numerical features, missing values were filled using the mean or median, depending on the feature's skewness. Features with skewness > 1 were filled with the median; otherwise, they were filled with the mean.

2.2 Simplified Processing

We decided to ignore all object-type data and focused solely on numerical and boolean features. **The simplest approach** is to drop all object-type features regardless of its meaning and fill missing values with the mean.

During XGBoost training, we aim to improve the dataset processing by trying three additional approaches to handle object-type features and missing values:

1. We applied the same process on `'is_night_game'` and `'season'` as described in **Detailed Processing**. All other object-type features were dropped. Data with more than 80 missing values were discarded, and remaining missing values were filled by the mean.

2. We observed that discarding data in the training dataset could negatively affect model performance. Therefore, similar to approach 1, but we we retained all data regardless of number of the missing values in each row.

3. We hypothesized that better techniques could be used for handling null values. Similar to approach 1, we used skewness to determine whether to fill missing values with the mean or median, while retaining other preprocessing steps.

3 Methods and Experimental Setup

3.1 Models Overview

- **Logistic Regression:** A simple and interpretable linear model often used as a baseline.
- **Random Forest:** An ensemble learning method that captures non-linear relationships and handles overfitting through bagging.
- **SVM:** Offering flexibility in decision boundary creation.
- **XGBoost:** A gradient boosting framework known for its performance and scalability.
- **Blending:** A technique that combines predictions from multiple models to exploit their respective strengths and compensate for their weaknesses.

3.2 Logistic Regression Training Process

Initially, we trained a Logistic Regression model using the detailed-preprocessed dataset. The dataset was split into an 80/20 ratio for training and validation. **L1 regularization** was used via the `'liblinear'` solver, as the dataset contained a high number of features, and L1 regularization can eliminate unnecessary ones. This choice of regularization aligns with the goal of feature selection and improving model interpretability. The Logistic Regression model was defined as follows:

```
lr_model = LogisticRegression( penalty='l1', solver='liblinear',
max_iter=1000, random_state=42)
```

To evaluate the effect of data preprocessing, we trained a second Logistic Regression model using the simplified-processed dataset. However, the first model underperformed, leading us to hypothesize that overly aggressive feature elimination might degrade model performance. Consequently, we focused on hyperparameter tuning to improve the model.

A **5-fold cross-validation** approach was used for robust evaluation and optimization. Additionally, we used the `optuna` library for hyperparameter optimization. **Optuna**'s efficient search algorithms allowed us to explore a wide range of hyperparameters, identifying the best combination to enhance model performance.

We set the number of trails for optimization to `n_trials=20`, and the hyperparameter search space included:

- `C`: regularization strength, range=`[1e-3, 1e2]` (log scale).
- `penalty`: type of regularization,
- `l1` or `l2`. `solver`: optimization solver, `liblinear` or `saga`.

3.3 Random Forest Training Process

For Random Forest, we adopted a similar strategy to Logistic Regression by experimenting with different datasets. However, based on insights from Logistic Regression, we exclusively used the simplified-processed dataset for training, because models trained on the detailed-processed dataset consistently underperformed. Hyperparameter tuning search place included:

- 'n_estimators': the number of trees, range=[10, 200]
- 'max_depth': the maximum depth, range=[10, 30]
- 'min_samples_leaf': the minimum samples required to split nodes, range=[2, 30]
- 'bootstrap': whether to use replacement sampling, TRUE, or FALSE

5-fold cross-validation was also used to ensure generalizability.

3.4 SVM Training Process

SVM training utilized the best-performing setup identified in prior experiments: **the simplified dataset** combined with **Optuna** for hyperparameter tuning and **5-fold cross-validation** for validation. While SVM models achieved reasonable accuracy, their training time was significantly longer than that of Logistic Regression and Random Forest, primarily due to the computational cost of **kernel-based methods** and the large dataset size.

3.5 XGBoost Training Process

XGBoost was initially trained using the same setup as SVM. Its performance surpassed that of the other models, prompting us to refine the preprocessing pipeline further (as described in Section 2). Additionally, we implemented **Bayesian optimization** (via the 'bayes_opt' library) for hyperparameter tuning, as this method efficiently explores continuous parameter spaces. Hyperparameter tuning search place included:

- 'max_depth': range=(7, 10), 'n_estimators': range=(150, 200),
- 'subsample': range=(0.7, 0.8), 'learning_rate': range= (0.01, 0.05),
- 'alpha': range=(0, 0.002), 'gamma': range=(0, 1),
- 'min_child_weight': range=(0, 1), 'colsample_bytree': range=(6, 10),
- 'reg_alpha': range=(0, 10), 'reag_lambda': range=(0, 10),

Compared to Random Forest, XGBoost demonstrated superior performance in both stage 1 and stage 2 evaluations, likely due to its gradient boosting framework and ability to handle complex feature interactions effectively.

3.6 Blending Training Process

To further improve prediction performance, we used a blending technique that combined predictions from the Logistic Regression, Random Forest, SVM, and XGBoost models. We chose Logistic Regression as the meta-classifier for its simplicity and computational efficiency.

1. **Training Stage:**

   After training each base model on the preprocessed training dataset, we generated prediction probabilities for the train dataset. These probabilities were stacked column-wise to form a new dataset, representing the outputs of the base models. A Logistic Regression model was then trained as the meta-classifier using this stacked dataset.

2. **Testing Stage:**

   Prediction probabilities for the test dataset were generated by each base model and stacked similarly to the training stage. The trained Logistic Regression meta-classifier used these stacked probabilities as input to produce the final predictions.

4 Results and Analysis

4.1 Overview

In this section, we aim to analyze the influence of model selection and hyperparameter optimization strategies on the task of predicting. While data preprocessing plays a role in our experiment, our primary focus lies on evaluating how different models adapt to the characteristics of the task.

4.2 Analysis of Data Preprocessing and Model Selection Impact

For convenience, the processed dataset obtained from the different data preprocessing approaches are denoted as A to E. These correspond to: A - Section 2.1(Detailed Processing), B - Section 2.2 (The Simplest Approach), C - Section 2.2 (Approach 1), D - Section 2.2 (Approach 2), E - Section 2.2 (Approach 3). Each preprocessing method was paired with various machine learning models, and their performance was analyzed across both stages.

Model(Dataset)	Experimental Setup
LR(A)	feature selection + 80/20 split
LR(B)	optuna + 5-fold cv
RF(B)	optuna + 80/20 split
RF(B)	optuna + 5-fold cv
SVM(B)	optuna + 5-fold cv
XGB(A)	optuna + 5-fold cv
XGB(B)	optuna + 5-fold cv
XGB(C)	optuna + 5-fold cv
XGB(D)	bayes_opt + 5-fold cv
XGB(E)	bayes_opt(no reg_alpha) + 5-fold cv
XGB(E)	bayes_opt + 5-fold cv
Blending(E)	LR as meta-classifier

Figure 1: Methods Combinations in Stage 1

Model(Dataset)	Experimental Setup
LR(A)	feature selection + 80/20 split
LR(B)	optuna + 5-fold cv
RF(A)	feature selection + 80/20 split
RF(B)	optuna + 5-fold cv
SVM(B)	optuna + 5-fold cv
XGB(B)	optuna + 5-fold cv
XGB(D)	bayes_opt + 5-fold cv
XGB(E)	bayes_opt + 5-fold cv
Blending(E)	LR as meta-classifier

Figure 2: Methods Combinations in Stage 2

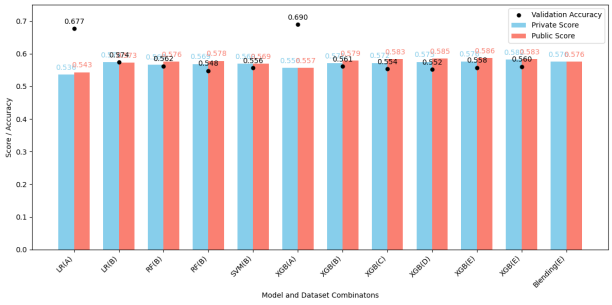


Figure 3: Results in Stage 1

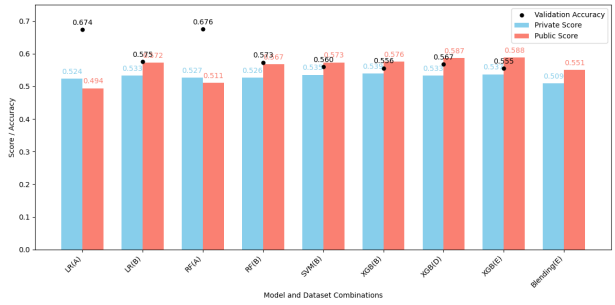


Figure 4: Results in Stage 2

The results indicate that if A data processing is used, the model validation accuracy is generally higher, but the actual prediction performance is the worst, while B and E data

processing have stable performance in both two stages, especially when using XGBoost as model training.

Model selection critically influences prediction outcomes. XGBoost consistently delivers superior performance across stages, particularly with Bayesian Optimization, which fine-tunes parameters to enhance generalization and reduce errors. Blending methods also show promise, combining strengths of base models to improve accuracy and stability. Furthermore, SVM, XGBoost and blending methods generally maintain stable performance between private and public scores, attributed to their ability to capture complex patterns and reduce overfitting through advanced tuning and ensemble learning.

4.3 Stage-wise Analysis

4.3.1 Private Scores Across Stages

This figure compares the best private scores of 4 selected models and the blending model for both Stage 1 and Stage 2. The models show weaker performance in predicting future data (Stage 2). This could be due to the similarity in data distribution and parameter settings used during training with Stage 1 data (past: 2013–2016). It indicates a lack of our consideration for features or patterns relevant to future data. On the other hand, for model stability, while XGB consistently outperforms other models, SVM demonstrates relative stability on both past and future data.

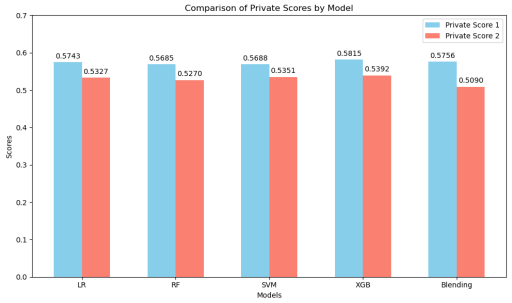
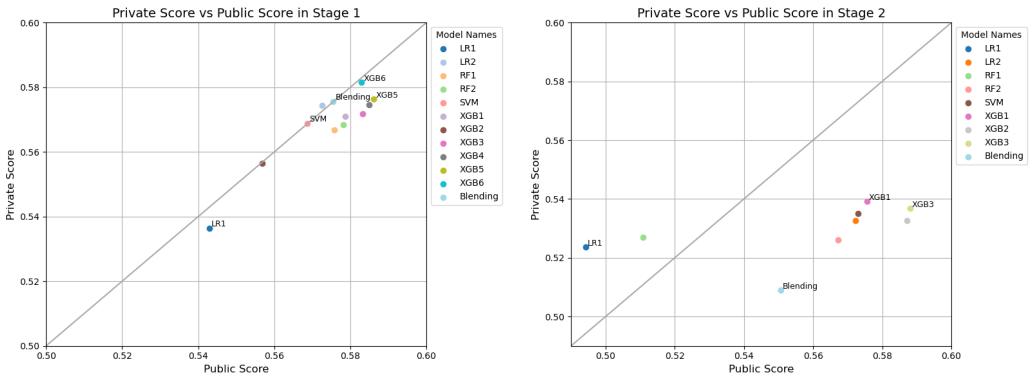


Figure 5: Private Score for Different Models

4.3.2 Public vs. Private Scores



(a) Private vs Public Score in Stage 1 (b) Private vs Public Score in Stage 2

These plot evaluates the public and private scores for each model in Stage 1 and Stage 2. Here are some findings:

- **Stage 1**: Most models exhibit overfitting on the partial test dataset, while SVM and the blending model are relatively stable, showing minimal overfitting or underfitting.
- **Stage 2**: There are significant differences between public and private scores for many models. The results highlight the need for more robust modeling techniques to address overfitting and underfitting in Stage 2.

5 Conclusion

The analysis demonstrates that different models and preprocessing techniques significantly affect our prediction accuracy. Future research could focus on enhancing model robustness to adapt to dynamic data distributions, especially for future predicting tasks.

XGBoost consistently outperforms other models in both stages, demonstrating its effectiveness across datasets by achieving a strong balance between accuracy, stability, and training efficiency. Despite its slight tendency to overfit, its strong performance makes it the most reliable option. Therefore, we recommend XGBoost as the final model choice for prediction in this study.

Pros and Cons of XGBoost Recommendation

Pros:

- **High Accuracy:** XGBoost achieves the best private and public scores across both stages.
- **Scalability:** Its efficient training process handles large datasets effectively.
- **Feature Importance:** XGBoost provides interpretable feature importance, aiding in insights extraction.
- **Robustness:** Performs consistently across different preprocessing strategies.

Cons:

- **Overfitting Risk:** Shows slight overfitting tendencies, particularly on public datasets.
- **Complexity:** Requires careful hyperparameter tuning and validation to achieve optimal results.
- **Computational Cost:** More resource-intensive compared to simpler models like Logistic Regression.

6 References

- XGBoost Documentation. https://xgboost.readthedocs.io/en/latest/.
- Optuna GitHub Repository Medium. https://github.com/optuna/optuna.
- Bayesian Optimization GitHub Repository. https://github.com/bayesian-optimization/BayesianOptimization.