2024 Fall HTML Final Project Report
MLB Prediction

Yu-Cheng Lin
B11901152
Department of Electrical engineering

Liang-Jia Kao
B11901130
Department of Electrical engineering

Yu-Jui Wang
B11611009
Department of Biomechatronics Engineering

Hong-Kai Yang
B11611047
Department of Business Administration

***Abstract*—In this competition, our objective was to predict the winning team for each MLB game across two stages. Stage 1 utilized first-half season data to forecast second-half outcomes, while Stage 2 required predictions for the entire 2024 season. We employed data augmentation and engineering techniques to enhance feature sets and built models using logistic regression, decision trees, gradient-boosted trees, and neural networks. Enhancements included blending, polynomial transformations, AdaBoost for logistic regression, and Random Forests for decision trees. During model development, we discovered potential information leakage between the training and testing datasets, enabling us to achieve high accuracy using only a few features. XGBoost achieved the highest accuracy in Stage 1 (0.59167), while Neural Networks excelled in Stage 2 (0.58554). Overall, XGBoost demonstrated the best balance of accuracy, scalability, and efficiency across both stages.**

1. Data Preprocessing and Validation Methods

Proper data preprocessing and validation are essential for ensuring reliable and generalizable machine learning models. A key challenge in this study was handling missing data, addressed using techniques like mean, median, mode, and zero imputation.

For validation, different approaches were used in the two stages. In Stage 1, standard methods such as splitting the training set into 80% training and 20% validation data, suitable for initial model development. In Stage 2, to predict 2024 outcomes, 2023 data was reserved for validation, and 2016–2022 data was used for training. This temporal validation better simulated real-world conditions and provided a robust assessment for future predictions.All validation results discussed in this report are based on Stage 1's validation method unless explicitly stated otherwise.

2. Logistic Regression

The instructor emphasized machine learning should start with simple models, so we began with logistic regression to analyze basic characteristics. This section explores its application to this problem and methods to improve the model.

2.1. Baseline Logistic Regression Experiments

In this experiment, we implemented logistic regression using the `LogisticRegression` function from the `sklearn.linear_model` library. The training data was split into 80% for training and 20% for validation. The model achieved a validation accuracy of 0.5655. On the stage 1 test set, it attained a public score of 0.56689 and a private score of 0.56617.

2.1.1. Feature Importance Analysis. To analyze the contributions of individual features, we conducted a feature importance analysis using the coefficients from the logistic regression model. As shown in Figure 1, the most important features are closely tied to winning percentages or scoring efficiency, highlighting the model's focus on metrics that strongly influence game outcomes.

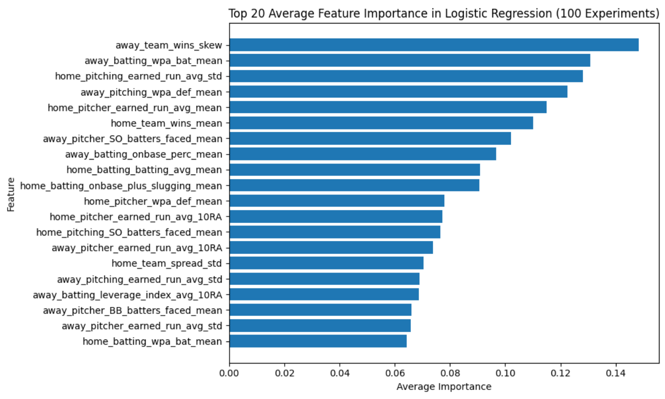

Figure 1: Feature importance analysis from the baseline logistic regression model.

2.1.2. Data Imputation Methods. Handling missing data is a key challenge in this dataset. We evaluated several imputation methods, including mean, median, mode, zero imputation, and dropping rows with missing values. Among these approaches, mode imputation performed best in terms of validation accuracy, while dropping rows proved ineffective due to the significant reduction in dataset size, which hindered the model's generalization.

Imputation Method	Validation Accuracy
Mean Imputation	0.560976
Median Imputation	0.562782
Mode Imputation	0.565492
Zero Imputation	0.560318

TABLE 1: Validation accuracy using different imputation methods.

2.2. Enhancing the Model

The basic logistic regression model, while effective as a starting point, did not deliver satisfactory performance for this problem. To address this, we will explore advanced techniques such as feature transformations and blending methods.

2.2.1. Polynomial Transformations. To capture non-linear relationships, we applied polynomial features with logistic regression, which improved fitting but increased overfitting and computational time. SVM with polynomial kernels provided better efficiency, but overfitting persisted. As seen in figure 2, the best results, achieved with L1 regularization, $C = 0.01$, and a polynomial degree of 2, yielded a validation accuracy of 0.56679.

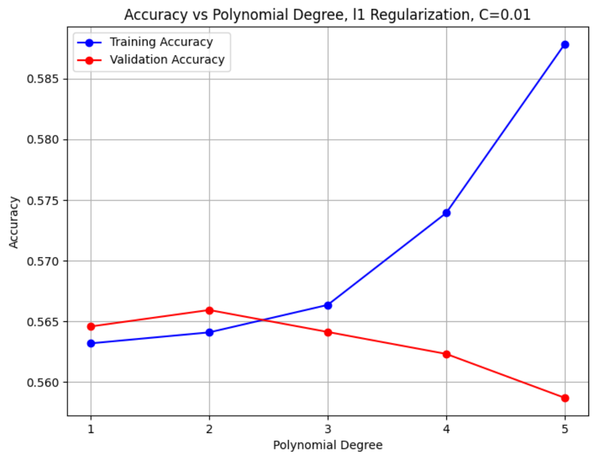

Figure 2: Training and validation accuracy vs. polynomial degree.

2.2.2. Blending Approach. Next, We tested three blending approaches to enhance model performance. The number of models was gradually increased, and validation accuracy was recorded at each step to determine the optimal configuration. The best setup was then used for test dataset predictions.

Uniform Blending with Boostrapping. We trained multiple weak models on bootstrapped subsets of the training data with size equal to 80% of the training data, averaging their predictions for the final output. As shown in Figure 3, training accuracy increased with the ensemble size, stabilizing at 0.582. Validation accuracy peaked at 0.555 during early iterations and plateaued, with the highest accuracy of 0.55074 achieved within 200 iterations.

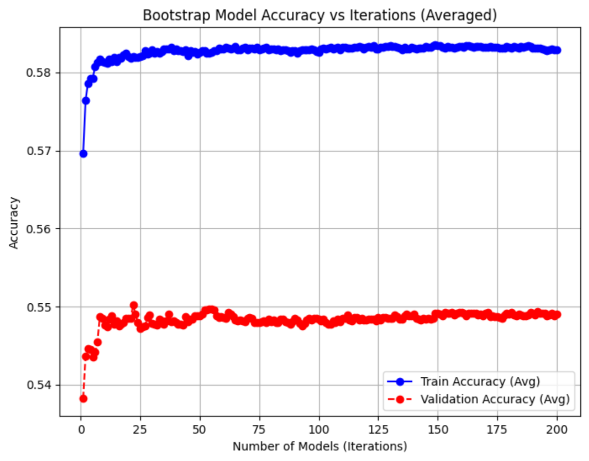

Figure 3: Bootstrap Accuracy vs Number of Models.

Uniform Blending with Features limiting. In this approach, weak models were trained using m randomly selected features. As shown in Figure 4, the best result was achieved with $m = 20$. Training accuracy stabilized at around 0.567, while validation accuracy peaked at 0.56012 after approximately 50 models. Although adding more models initially improved validation accuracy, the benefits diminished as the ensemble size grew.

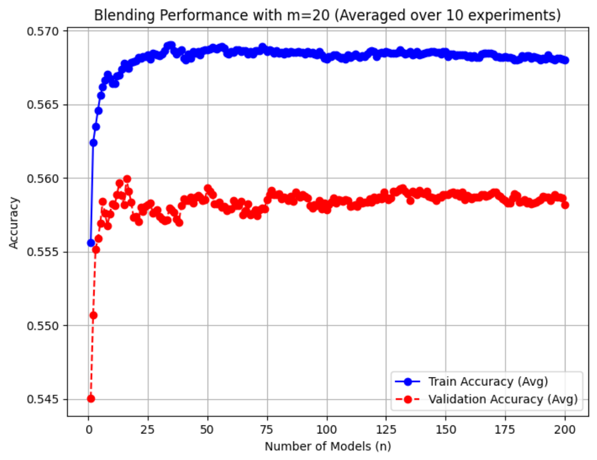

Figure 4: Features limiting blending accuracy over iterations.

Adaboost. We implemented AdaBoost with logistic regression as the weak learner, as introduced in class. As shown in Figure 5, training accuracy increased steadily, reaching nearly 1.0, while validation accuracy stabilized around 0.56, indicating overfitting. The best validation accuracy achieved was 0.56932.

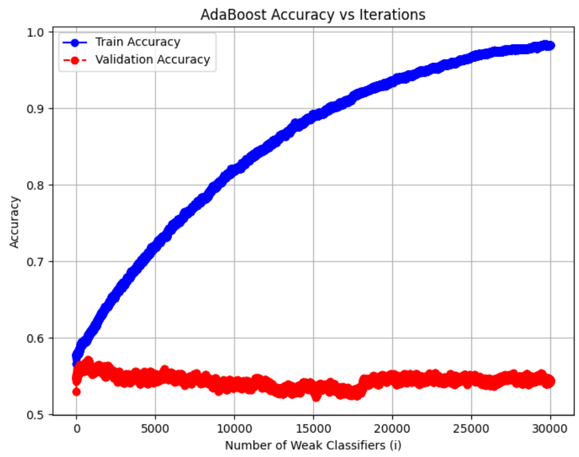

Figure 5: Adaboost accuracy over iterations.

2.3. Summary of Logistic Regression Models

The exploration of linear models highlighted their strengths and limitations. Logistic regression demonstrated moderate accuracy and stability but struggled with complex relationships. Polynomial transformations offered slight improvements but caused overfitting and higher costs. AdaBoost, the best blending method, was still insufficient. These limitations highlight the need for advanced models beyond logistic regression to handle non-linear patterns effectively.

Models	stage 1	stage 2
Logistic Regression	0.56689	0.56127
Polynomial Regression (2nd Order)	0.56715	0.55413
Blending (Adaboost)	0.57032	0.56573

TABLE 2: Public Test Accuracy of Different Models.

3. Decision Tree

3.1. Decision Tree

The most common method of classification is the decision tree. And we start with the basic model to do the estimation. Hence, we used the decision tree first and tuned for the best parameter.

We study and explore the effectiveness of decision trees in predicting MLB game outcomes, focusing on key aspects:

- **Splitting Criteria:** Comparing Gini impurity and entropy to identify the better predictor.
- **Tree Height:** Evaluating tree depth to balance underfitting and overfitting.
- **Feature Importance:** Highlighting top features critical for predicting MLB winners.

Additionally, by tuning hyperparameters, such as tree depth, decision trees can balance predictive accuracy with robustness, making them well suited for datasets with varying levels of noise and complexity.

3.1.1. Splitting Criteria. Decision trees use specific criteria to split data at each node:

- **Gini Impurity:** Measures the probability of incorrect classification and is computationally efficient.
- **Entropy:** Measures information gain but requires logarithmic computations, which makes it slightly slower.

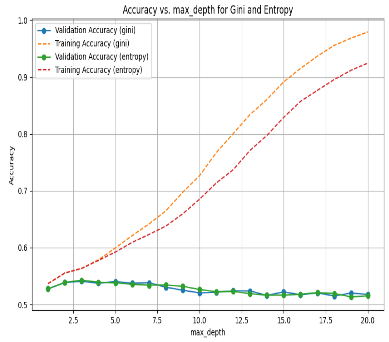

Figure 6: Validation and training accuracy among Gini index and Entropy.

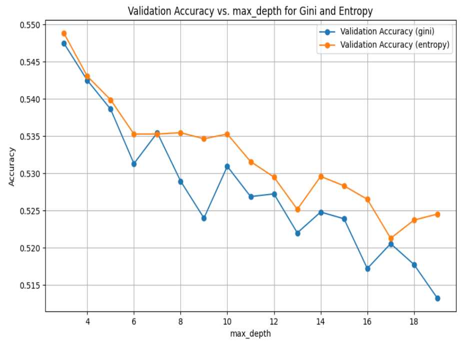

Figure 7: Validation accuracy comparison between Gini index and Entropy.

Observations.

- Validation accuracy for both Gini and Entropy decreases as tree depth increases, indicating overfitting at deeper depths.
- Entropy achieves slightly higher validation accuracy at all depths, capturing nuanced splits better.

Conclusion. Both Gini and Entropy are effective splitting criteria.

- **Gini** is preferred for its efficiency in large-scale applications.
- **Entropy** may be better for smaller datasets requiring higher accuracy or handling class imbalances.

Thus in the following algorithm, we use Entropy to evaluate.

3.1.2. Tree Depth.

Observations from the Graph. Figure7 illustrates different values of `max_depth` and achieve maximum accuracy when **max_depth** = 4 . When the depth become larger, the accuracy become lower and this is consistent with what we taught in class that large depth usually cause overfitting

In the following experiment, we use $depth = 4$ as default.

3.1.3. Features. The following analysis focus on how many estimators we used will achieve the best validation accuracy. The **max_features** $= None$ means using all the features.

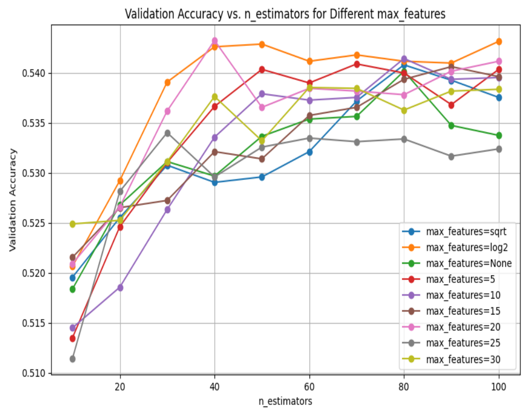

Figure 8: Validation accuracy comparison between different number of features been used.

Performance of Different `max_features`:.

- **max_features=log2** achieve higher validation accuracy compared to other number of estimators.
- We can see the more of the features that we use may decrease the validation accuracy.

Trend Across Estimators:.

- For most configurations, validation accuracy improves as `n_estimators` increases up to 50–80, after which it stabilizes.
- The **max_features** is achieve by $\log_2(features)$ which is about 7 and decrease after large number of features.

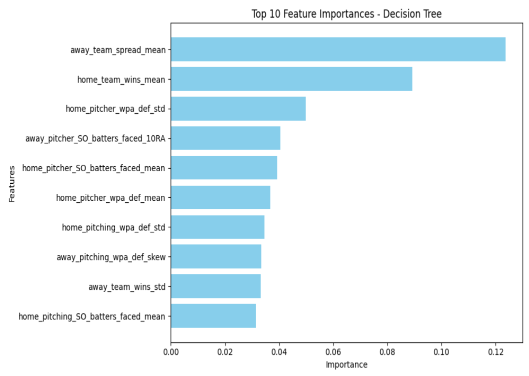

Figure 9: 10 most important features in decision tree.

3.1.4. Insights from Top 10 Feature Importance. The 10 most important features identified by the decision tree are comprehensive, with home_team_wins initially appearing as the most logical choice we think. However, game performance metrics proved to be more significant. This is likely because game performance is critical in predicting outcomes, and the home_team_wins_mean reflects how well the home team adapts to their court, making it more closely related to the likelihood of winning.

3.1.5. Conclusion. To achieve the best decision tree. I used **depth** $= 5$, **n_estimator**$= 50$, **max_features**$= \log_2$, **criterion**=Entropy and achieve 0.58069 on Kaggle public for case 1, 0.57392 for case 2.

3.2. Random Forest

3.2.1. max_depth. The plot illustrates how validation accuracy changes as the maximum depth (**max_depth**) of the trees in the Random Forest model increases.

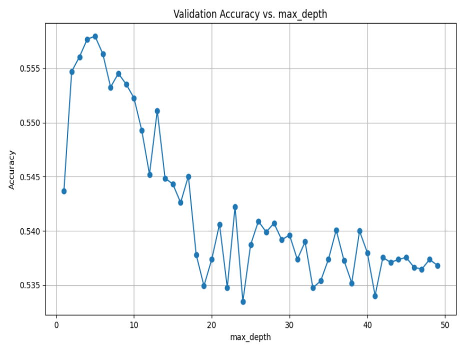

Figure 10: Validation Accuracy vs. `max_depth`.

Observations.

- Validation accuracy rises as **max_depth** increases from 1 to approximately 5.

- The highest validation accuracy is observed around **max_depth = 4–5**, showing the optimal balance between underfitting and overfitting.
- Beyond **max_depth = 7**, validation accuracy steadily declines as the model starts to overfit.

3.2.2. Conclusion. Setting an optimal **max_depth** (5–7) allows Random Forest models to achieve better accuracy and robustness compared to standalone decision trees. Using Gini as the base algorithm with 80 **estimators** and **depth** of 5, the model achieved a public accuracy of 0.58876 in Stage 1 and 0.58222 in Stage 2.

4. XGBoost

4.1. Introduction to XGBoost

XGBoost (eXtreme Gradient Boosting) is an enhanced version of Gradient Boosted Decision Trees (GBDT) designed for speed and ease of use. While GBDT builds decision trees sequentially, learning from the errors of the previous tree, XGBoost improves this process with features like built-in tools, parallel processing, and more, making it ideal for large or complex datasets. Additionally, XGBoost includes advanced features such as tree pruning to remove unnecessary parts and optimized handling of large datasets, ensuring faster and more accurate performance compared to traditional GBDT.

For this project, XGBoost was chosen because it's not only powerful but also handles data effectively, giving us better results with less effort.

4.2. Tuning of Parameters

4.2.1. Optuna. When using XGBoost, managing multiple hyperparameters is difficult, making **Optuna** an excellent choice. `Optuna` is an advanced hyperparameter optimization framework that intelligently searches for the best parameter combinations. Unlike grid search or random search, it uses techniques like Bayesian optimization and pruning to focus on promising regions of the parameter space, avoiding unnecessary computations.

In this project, `Optuna` effectively tuned XGBoost hyperparameters, striking a balance between improving accuracy and minimizing computational costs.

4.2.2. Logloss. In this program, **Logloss** is used as the objective function to guide the optimization of the **XGBoost** model. Logloss is calculated using the formula:

$$\text{Logloss} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(p_i) + (1-y_i)\log(1-p_i)\right]$$

where N is the number of samples, y_i is the actual label, and p_i is the predicted probability for a win.

Logloss handles imbalanced datasets well by addressing both majority and minority classes. As XGBoost's default loss function for binary classification, it ensures efficient learning and reliable predictions.

4.2.3. Hyperparameters. Below are hyperparameters used in the model:

- **N estimators** defines the number of trees in the model. More trees improve performance but increase training time.
- **Max depth** sets the maximum depth of each tree. Deeper trees capture complex patterns but risk overfitting.
- **Learning rate** controls the step size in boosting. Lower values lead to slower but more stable learning.
- **Subsample** determines the fraction of samples used for each tree. Lower values prevent overfitting.
- **Colsample by tree** limits the features considered for splitting, reducing overfitting.
- **Gamma** sets the minimum loss reduction required to split a node. Higher values make the model more conservative.
- **Minimum child weight** specifies the minimum sum of instance weights in a leaf. Higher values prevent overly specific patterns.
- **Reg_alpha** (L1 regularization) helps prevent overfitting by shrinking less important weights to zero.
- **Reg_lambda** (L2 regularization) reduces large coefficients, controlling model complexity.

Figure 11 is a figure of the importance of the above hyperparameters by using the function in `Optuna`. The figure shows that `learning rate` is the most critical parameter, likely because smaller, more effective updates improve convergence and reduce reliance on a large number of trees.

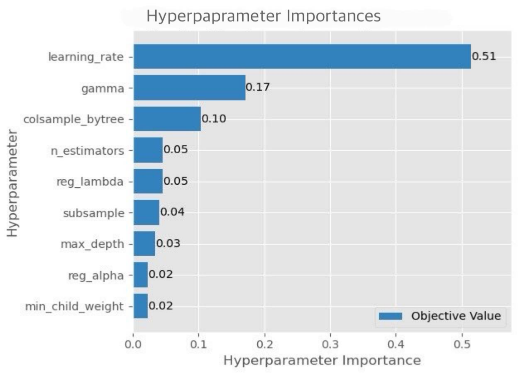

Figure 11: Hyperparameter importances

4.3. Model Training Process Analysis

After tuning, the XGBoost model was configured with **n_estimators** = 60, **max_depth** = 3, **learning_rate** = 0.078, **subsample** = 0.976, **colsample_bytree** = 0.954, **gamma** = 0.308, **min_child_weight** = 8.83, **reg_alpha** = 0.875, **reg_lambda** = 5.76.

4.3.1. Important Features. From Figure 12, we can observe that the key factor influencing a baseball game lies in overall team performance, including **away_team_spread_mean**, **home_team_spread_mean**, and more. Additionally, the pitching ability is also crucial, as seen from **away_pitching_SO_batters_faced_10RA**, **away_pitching_SO_batters_faced_mean**, and more. These metrics demonstrate how a pitcher's ability to suppress opposing batters significantly impacts the outcome of the game.

Interestingly, home-related statistics rank lower than away-related ones, despite the problem is to predict home team wins. The model suggests that the away team's challenges have a greater impact on the probability of a home victory.

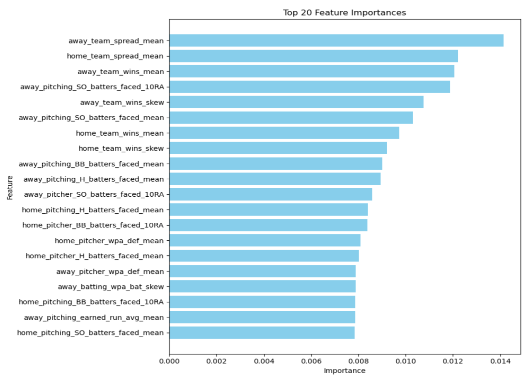

Figure 12: Top 20 features importance analysis of XGBoost

4.3.2. Randomness of the training process. The XGBoost model exhibits slight variability in results during training, even with fixed parameters. To investigate this, we ran XGBoost 1000 times with tuned parameters. As shown in Figure 13, the results followed a normal distribution, with performance varying predictably and extreme accuracy being rare. The final validation accuracy was 0.5890, leading to a public score of 0.5917 and a private score of 0.5876 in the competition.

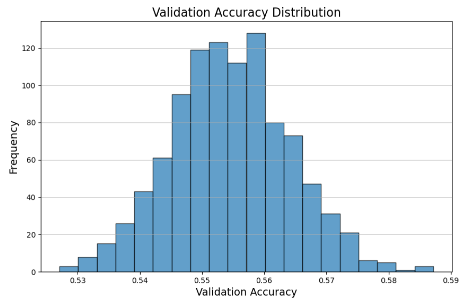

Figure 13: Validation accuracy distribution over 1000 random XGBoost test

5. Neural Network

Neural networks, composed of interconnected layers, use nonlinear activations to process inputs and learn complex patterns from large datasets. The application of predicting MLB game outcomes where numerous variables and complex relationships may be suitable.

5.1. Method

In this approach, we added a "Home-Away Average Difference" feature to enhance the Neural Network's predictive capability. For the second phase, we used a Time-Weighted loss, where sample weights decayed exponentially based on their temporal distance from the present.

5.1.1. Hyperparameter Optimization. Since Neural Networks involve numerous hyperparameters, we used `Optuna` to adopted minimization of validation loss over 100 epochs, and get the best set of hyperparameters automatically. Specifically, we varied:

- **Number of layers** from 1 to 5, controlling the network depth.
- **Hidden sizes** for each layer, ranging from 16 to 256 in increments of 16, which determines the width of each layer.
- **Dropout rates** between 0.0 and 0.5 for each layer to mitigate overfitting.
- **Initial Learning rate** on a log scale between $1 \times e^{-5}$ and $1 \times e^{-1}$, impacting gradient update magnitudes.
- **Initial Weight decay** on a log scale between $1 \times e^{-6}$ and $1 \times e^{-2}$, providing L2 regularization.
- **Activation function** among `ReLU`, `LeakyReLU`, and `ELU`, influencing how the network handles nonlinearities.
- **Batch size** selected from $\{16, 32, 64, 128, 256, 512, 1024\}$, affecting the trade-off between training stability and speed.

These parameters, selected by `Optuna` to minimize validation loss, helped identify the optimal architecture and training setup for our Neural Network.

5.1.2. Activation Functions and Optimizers. For the activation function, we provided `ReLU`, `tanh`, and `ELU` options in `Optuna` to find the optimal activation corresponding to different optimizers. Compared to the `tanh` and `ReLU` approaches mentioned in the class, `ELU` applies exponential decay when the input is below zero.

Throughout our testing, we selected three optimizers:

- **AdamW** (Adam with Weight Decay) provides more precise weight updates and helps curb overfitting
- **RAdam** (Rectified Adam) adds a warm-up mechanism for learning rate adjustments, yielding more stable convergence in the early phase
- **AdaBound** bounds the learning rate within an upper and lower limit, aiming to achieve faster convergence initially and maintain stability at later stages.

5.2. Parameter Selection

Completing the Optuna search for all models revealed that the best configurations favored the ELU activator. Unlike ReLU, ELU applies exponential decay for inputs below zero, helping to prevent the "Dying ReLU" problem while maintaining nonlinearity.

The optimized hyperparameters commonly included a batch size of 64 and four or more layers. However, hidden-layer sizes, dropout rates, and learning rates varied significantly, indicating the neural network's sensitivity to hyper-parameter changes and limited scalability.

After determining the optimal hyperparameters, we trained the model on the full dataset with early stopping based on training loss. Training was halted if the loss did not decrease for 20 consecutive epochs.

5.3. Model Performance Analysis

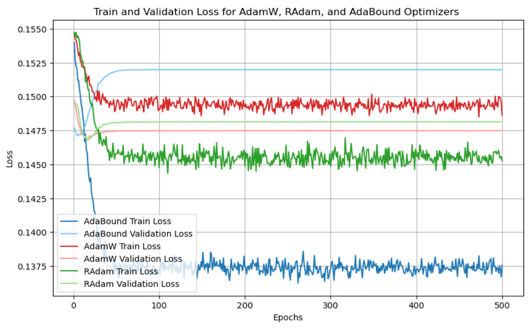

Figure 14: Train and Validation Loss

optimizer	Validation Accuracy (stage 1)	Test Accuracy (stage 1)	Validation Accuracy (stage 2)	Test Accuracy (stage 2)
AdamW	0.5641	0.5703	0.5763	0.5593
RAdam	0.5754	0.5796	0.5772	0.5552
Adabound	0.5768	0.5503	0.5827	0.5854

TABLE 3: Validation accuracy using different imputation methods.

By analyzing Table 3 and Figure 14, we find that AdaBound quickly reduces both training and validation losses early and maintains low levels, achieving final validation and test accuracies of 0.5768 and 0.5503. RAdam uses a warm-up mechanism, resulting in a smooth initial loss decrease and stable accuracies. AdamW exhibits an unusual trend where training loss is consistently higher than validation loss, with final accuracies of 0.5641 (validation) and 0.5703 (test), performing moderately but lagging behind RAdam and AdaBound in certain stages.

Figure 14 shows that the model reaches a low validation loss before 50 epochs but begins to overfit as validation loss increases afterward. Although the neural network achieved a promising public Kaggle score (0.58554), the private score (0.53349) highlighted a significant gap, likely due to the model's sensitivity to noise and irrelevant features, leading to overfitting and performance drops.

Compared to other models, neural networks are computationally intensive due to their complex structure and numerous parameters, especially with features like RAdam's warm-up and ELU's computations. Additionally, Optuna's parameter tuning indicates that optimal configurations vary, highlighting the limited scalability and efficiency of neural network models.

6. Final Analysis and Insights

This study explored different models to predict MLB game outcomes, showing varied results across two stages. In Stage 1, **XGBoost** achieved the highest public test accuracy of **0.59167**, while in Stage 2, the **Neural Network** performed best with **0.58554**. These results emphasize the effectiveness of different methods under varying conditions.

In addition to individual model evaluations, we also experimented with a uniform blending approach that combined predictions from all the models we developed. This method achieved a public test accuracy of **0.58553** in Stage 1, demonstrating its ability to integrate multiple models effectively.

Among all machine learning methods, we recommend **XGBoost** for baseball game prediction, as it performed well in both Stage 1 and Stage 2, achieving private test scores of **0.58762** for Stage 1 and **0.54738** for Stage 2.

XGBoost outperformed **Logistic Regression**, **Decision Trees**, and **Neural Networks**. It has a great ability to handle the nonlinear interactions and feature complexities in structured baseball data. Unlike Logistic Regression, which struggles with nonlinear relationships, and Decision Trees, which are prone to overfitting, **XGBoost** uses regularization and subsampling to balance complexity and generalization. Neural Networks, while powerful for high-dimensional and unstructured data, often require significantly larger datasets to perform well. With only 11,000 samples, the dataset is relatively small for Neural Networks, especially when the case here has many high-dimensional features, leading it to a higher risk of overfitting and inefficiency compared to **XGBoost**. This is why we think **XGBoost** performs better than the other methods.

A unique aspect of the dataset was the inclusion of team winning percentages during the season, which allowed us to use depth-first search (DFS) to deduce game outcomes. While the technique is unrelated to machine learning and not detailed in this report, this method proved highly effective, achieving an accuracy of **0.78758** in Stage 2, highlighting the value of domain-specific knowledge.

Workload Balance

B11901152 Yu-Cheng Lin: Decision tree, Random forest
B11901130 Liang-Jia Kao: Logistic regression, Adaboost
B11611009 Yu-Jui Wang: XGBoost
B11611047 Hong-Kai Yang: Neural network