Machine Learning Final Project Survey Report

TEAM NAME: wonderbanana666

1st Yu-Tang Chang
dept. CSIE
National Taiwan University
Taipei, Taiwan
r13922116@csie.ntu.edu.tw

2nd Ching-Hsin Liao
dept. CSIE
National Taiwan University
line 4: Taipei, Taiwan
r13922123@csie.ntu.edu.tw

3rd Bill Hsu
dept. CSIE
National Taiwan University
Taipei, Taiwan
b07902140@ntu.edu.tw

4th Hung-Tso Shiao
dept. CSIE
National Taiwan University
Taipei, Taiwan
r13922163@csie.ntu.edu.tw

***Abstract*— This paper presents our work on Machine Learning course competition-HTML2024FALL Final Project. We'll first introduce various data preprocessing approaches we tried and various models we used, then compare all frameworks based on accuracy, stability across the two stages, and scalability. Finally, we'll provide the best framework among all with its pros and cons. Our team- wonderbanana666 achieved rank 74/142 with a score of 0.59070 and rank 43/142 with a score of 0.58568 in stage one public and private score, respectively. Rank 63/131 with a score of 0.58471 and rank 35/131 with a score of 0.55065 in stage two public and private score, respectively.**

I. Introduction

First, data preprocessing is crucial for the model, so we test three different data preprocessing approaches. Then, we evaluate four machine learning models, along with an ensemble model. Finally, we compare the performances of combinations of data preprocessing approaches and machine learning models, and recommend that the best combination is LightGBM with data preprocessing approach one.

II. Data Preprocessing

A. Approach one

(i) Box-plot outlier removal

We aim to detect and remove abnormally large or small values in each ratio feature of the training and test data at both stages, using a box-plot method. For each ratio feature, we calculate its 25th quantile (Q1) and 75th quantile (Q3), and consider values lower than Q1 - 1.5*(Q3 - Q1) or higher than Q3 + 1.5*(Q3 - Q1) as outliers for that feature. We then replace the outlier values with NaN.

(ii) KNN missing value imputation

Impute NaN of ratio features by k-nearest neighbor algorithm with k=5.

B. Approach two

(i) Linear regression missing value imputation

In the training and test data of both stages, we found high correlations between features within the same subcategories, e.g., home_batting_batting_avg_10RA and home_batting_onbase_perc_10RA (Fig. 1). Therefore, for a pair of features with sufficiently high covariance, we calculate the regression line between them and impute NaN values of one feature using the other feature and the corresponding regression line, provided that the other feature is not NaN.

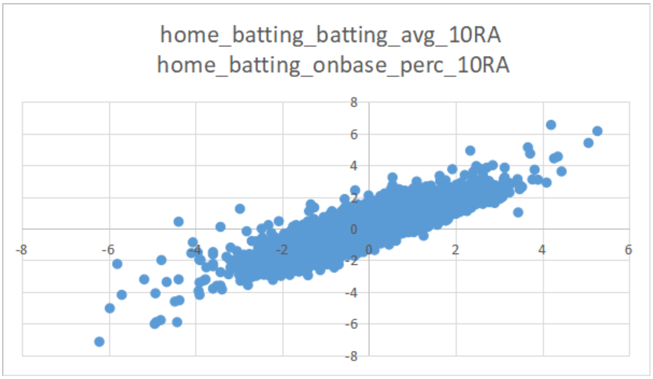

Fig 1

In the training data, there is even a clear linear relationship between _avg_10RA features and _avg_mean features, e.g., home_batting_batting_avg_10RA and home_batting_batting_avg_mean (Fig. 2). Therefore, we intend to impute in the same manner as described above. However, instead of calculating the regression line, we determine the equation of the line that is distinctly visible on the graph and then impute using the same method as below.

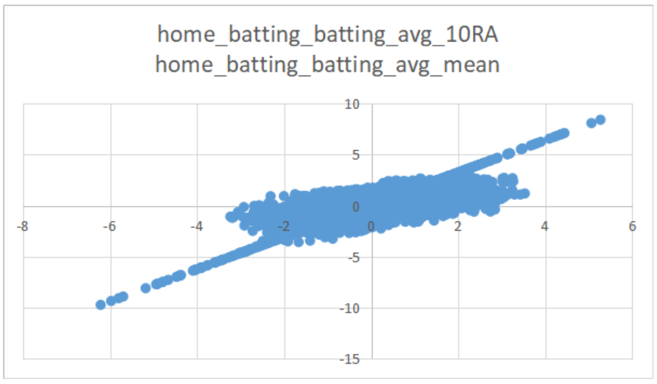

Fig 2

Then, we follow the same steps as in data preprocessing approach one to remove outliers and fully impute the ratio features. Below (Fig 3, Fig 4) are the resulting graphs showing the relationships between highly correlated features after linear regression imputation and outlier removal.

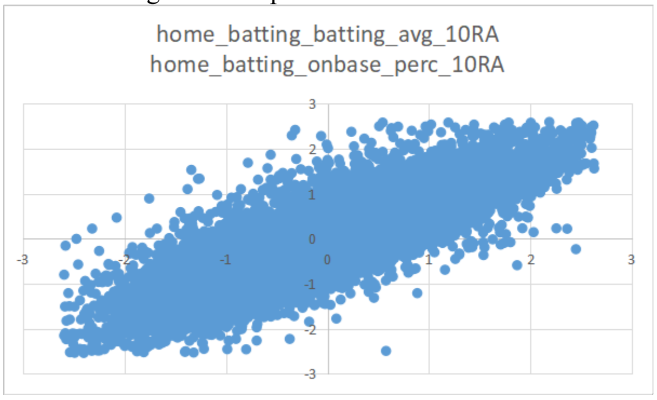

Fig 3

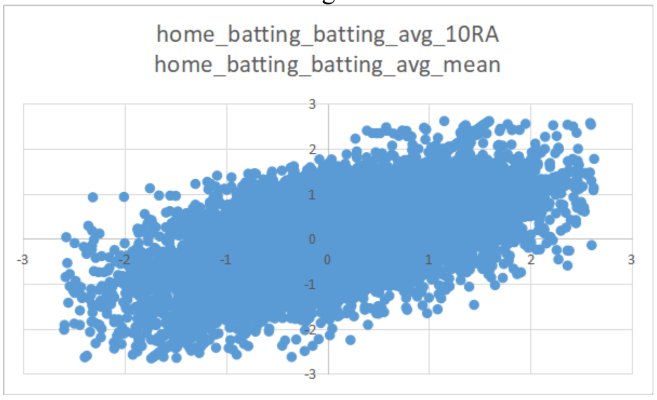

Fig4

C. *Approach three*

(i) Feature Generation

To enhance the dataset, we created several domain-specific features that provided additional context about the problem. These included performance metrics such as batting averages and earned run averages (ERA) for both home and away teams. Advanced metrics like WHIP (Walks + Hits per Inning Pitched) and K/BB ratio were introduced to evaluate pitcher effectiveness and control. Momentum indicators were calculated by incorporating recent performance trends in batting, pitching, and leverage situations, capturing the readiness and form of both teams. Interaction features, such as home vs. away matchup advantages, pitcher advantage, and batting efficiency, were added to explicitly model the interplay between competing teams. Additionally, external factors like rest days, home-field advantage, and park effects were incorporated to account for contextual influences on game outcomes.

(ii) Missing Value Imputation

Missing data was handled systematically to ensure the dataset was complete and usable. For categorical variables, missing values were replaced with the placeholder "UNKNOWN," enabling smooth encoding without losing information. For numerical features, missing values were imputed using the mean or median, depending on the data distribution. Infinite values, which could distort the model, were treated as missing and replaced with appropriate statistics. These steps minimized information loss and ensured the dataset's integrity.

(iii) Managing Outliers

Outliers were carefully managed to reduce noise and prevent distortion in the model. For numerical features, values beyond three standard deviations from the mean were clipped to the nearest boundary. This approach maintained the integrity of the data while removing extreme values that could skew the model's learning process. Domain knowledge was used to set reasonable thresholds for critical metrics, such as ERA and strikeout rates, further enhancing the robustness of the dataset.

(iv) Feature Selection

Feature selection played a pivotal role in reducing dimensionality and improving model performance. Initially, a LightGBM model was used to rank features based on their importance. From this ranking, the top 100 features were selected. To address multicollinearity, features with a correlation coefficient greater than 0.8 were removed, ensuring that the final feature set was compact and non-redundant. This reduced computational complexity and improved interpretability while retaining the predictive power of the dataset.

(v) Standardization

Finally, numerical features were standardized to ensure consistent scaling across the dataset. Using a standard scaler, values were normalized to have a mean of zero and a standard deviation of one. This step was crucial for stabilizing gradient-based algorithms and ensuring consistent performance across all models.

III. Model Description

A. TabNet [1]

TabNet is designed to bridge the gap between traditional machine learning and deep learning for tabular data. Its key advantages include:

(a) Native Support for Tabular Data: Unlike many deep learning models, TabNet is optimized for structured datasets, handling diverse data types (numerical, categorical, and ordinal) with ease.

(b) Built-in Interpretability: TabNet provides feature importance and decision-making insights at each step of prediction, fostering transparency and trust, especially in critical applications like healthcare and finance.
(c) Efficient Learning: The sparse attention mechanism selectively focuses on relevant features, reducing computational overhead and mitigating overfitting.

TabNet processes data through a sequence of decision steps using a sparse attention mechanism to dynamically select relevant features at each step. A feature transformer module captures complex relationships, while outputs from decision steps are aggregated to make final predictions. This design mimics decision trees while leveraging deep learning's flexibility, ensuring both accuracy and interpretability.

B. CatBoost [2]

CatBoost is a high-performance gradient boosting framework tailored for structured data. Key advantages include:

(a) Native Categorical Feature Handling: Efficiently encodes categorical variables using target-based statistics, requiring minimal preprocessing.
(b) State-of-the-Art Accuracy: Delivers robust performance across datasets with optimized boosting techniques.
(c) Reduced Overfitting: Employs ordered boosting to prevent data leakage and overfitting.
(d) Ease of Use: Works effectively with minimal parameter tuning, making it user-friendly.

CatBoost sequentially builds decision trees using ordered boosting to reduce data leakage and overfitting. It encodes categorical features with target-based statistics and iteratively optimizes weak learners to minimize loss, resulting in a fast, accurate, and versatile model for structured data.

C. XGBoost [3]

XGBoost is a widely used gradient boosting framework known for its speed and performance. Key advantages include:

(a) High Accuracy: It consistently delivers strong predictive performance across a wide range of applications.
(b) Scalability: XGBoost is optimized for parallel and distributed computing, making it highly efficient on large datasets.
(c) Regularization for Overfitting Control: Includes built-in regularization techniques to improve generalization.
(d) Flexibility: Supports custom loss functions and evaluation metrics, catering to diverse modeling needs.

XGBoost builds an ensemble of decision trees in an additive manner, optimizing a regularized loss function to balance model complexity and accuracy. It uses techniques like gradient-based optimization, advanced tree-pruning, and parallel processing to improve training speed and efficiency. These features make XGBoost a powerful tool for solving both classification and regression problems.

D. LightGBM [4]

LightGBM is a gradient boosting framework designed for speed and efficiency on large datasets. Key advantages include:

(a) High Efficiency: Uses histogram-based algorithms for faster training and reduced memory usage.
(b) Scalability: Handles large datasets with ease and supports distributed training.
(c) Accurate and Robust: Achieves high accuracy with features like leaf-wise tree growth and regularization.
(d) Flexibility: Supports categorical features natively, eliminating the need for one-hot encoding.

LightGBM builds decision trees iteratively, using a leaf-wise growth strategy to minimize loss more effectively than level-wise approaches. It employs histogram-based feature binning to accelerate computation and reduce memory usage. These optimizations, along with built-in support for categorical features and regularization techniques, make LightGBM a fast and accurate choice for both classification and regression tasks.

E. Ensemble

The ensemble model combines the strengths of multiple machine learning algorithms—CatBoost, XGBoost, and LightGBM—to deliver robust performance on structured datasets. Its key advantages include:

(a) Leveraging Model Diversity

The ensemble integrates three gradient boosting models, each excelling in specific areas:

(i) CatBoost: Efficient handling of categorical data and reduced need for preprocessing.
(ii) XGBoost: Exceptional scalability and optimization, particularly suited for large datasets.
(iii) LightGBM: Faster training and better performance on high-dimensional data.

By leveraging their complementary strengths, the ensemble mitigates individual model biases and enhances generalization to unseen data.

(b) Built-in Feature Importance

All three models provide intrinsic mechanisms to evaluate feature importance, allowing for deeper insights into how features contribute to predictions. This aligns well with tasks requiring interpretability alongside performance.

(c) Improved Stability and Accuracy

Combining the outputs of multiple models via ensemble methods—soft voting or stacking—smooths individual model predictions, reducing variance and improving prediction stability across different data splits.

The ensemble model aggregates predictions from CatBoost, XGBoost, and LightGBM using soft voting. We have also explored stacking:

(i) Soft Voting: A weighted average of probabilities from the three models assigns greater influence to stronger-performing models.

(ii) Stacking: A logistic regression meta-learner analyzes the strengths and weaknesses of base models to make final predictions.

This design combines the interpretability and computational efficiency of gradient boosting with the flexibility of ensemble learning.

IV. Performance

We now compare various combinations of data preprocessing approaches and models based on three metrics: accuracy, stability across the two stages, and scalability. Finally, we will present the best combination among all, along with its pros and cons.

We tested a total of seven combinations of data preprocessing approaches and models (see Table 1 for details). For each combination, we trained the model on 20%, 40%, 60%, 80%, and 100% of the training data using random sampling, and then tested these five trained models on the testing data from both stages.

The accuracy of a specific combination under specific stages and public-private settings refers to the highest testing accuracy among the five trained models with different training sample sizes.

The stability across two stages of a specific combination under specific public private setting are defined as follows: Let $s_i^{(j)}$ be the testing accuracy of the model trained on the j^{th} training data size($20 \times j\%$ of random sampled training data), on stage i , $i = 1,2$, $j = 1,2,3,4,5$. Lower stability value indicates higher stability.

$$S_1 = \left(s_1^{(1)}, s_1^{(2)}, s_1^{(3)}, s_1^{(4)}, s_1^{(5)}\right)$$
$$S_2 = \left(s_2^{(1)}, s_2^{(2)}, s_2^{(3)}, s_2^{(4)}, s_2^{(5)}\right)$$
$$Stability = \frac{1}{5}\sum_{j=1}^{5}\left|s_1^{(j)} - s_2^{(j)}\right|$$

We indicate scalability of each combination under specific stage and public private setting via Fig 5, Fig 6, Fig 7, Fig 8, these figures are graphs of testing accuracies under different training data size.

Among all 7 combinations of framework, LightGBM with data preprocessing approach1 achieves the best performance on most metric except stage2 private accuracy and private stability (see table 1 for detail). Hence, we recommend LightGBM with data preprocessing approach1 as our best framework.

We observe good performance in terms of both accuracy and stability. This may be attributed to LightGBM's effective feature selection and regularization mechanisms, which help it capture meaningful patterns in the data while reducing overfitting. Additionally, its ability to handle complex interactions between features contributes to consistent and reliable predictions.

Although LightGBM is widely recognized for its scalability, we observe relatively poor performance compared to other models. We suspect this phenomenon may be due to the dataset not being large enough for the model to generalize effectively, as LightGBM is known to perform suboptimally on smaller datasets.

Conclusion

This paper explored data preprocessing approaches, such as KNN imputation, linear regression imputation, box-plot outlier removal, feature generation to create domain-specific metrics and feature importance analysis. Combined with machine learning models, including TabNet, CatBoost, XGBoost, LightGBM, and an ensemble framework, to identify the optimal solution for the HTML2024FALL Final Project competition. LightGBM with data preprocessing approach1 achieved the best overall performance, demonstrating strong accuracy and stability.

References

[1] S. O. Arik and T. Pfister, "TabNet: Attentive Interpretable Tabular Learning," published in August 2019.

[2] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "CatBoost: Unbiased Boosting with Categorical Features," published in June 2017.

[3] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," published in 2016.

[4] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," published in 2017.

Contribution

- Yu-Tang Chang: Data Preprocessing two. Implement XGBoost model. Paper writing.
- Ching-Hsin Liao: Data Preprocessing one. Data Preprocessing two. implement LightGBM model. Paper writing
- Bill Hsu: Data preprocessing three. Implement ensemble model. Paper writing.
- Hung-Tso Shiao: Implement TabNet model. Graph generation. Paper writing.

Table 1

	TabNet approach1	TabNet approach2	XGBoost approach1	XGBoost approach2	LightGBM approach1	LightGBM approach2	Ensemble approach3
Stage1 public Accuracy	0.56197	0.55907	0.58327	0.58005	**0.59231**	0.59070	0.58198
Stage2 public Accuracy	0.54235	0.54734	0.56893	0.56146	**0.58222**	0.56976	0.55065
Stage1 Private Accuracy	0.56268	0.54778	0.58017	0.57855	**0.59151**	0.57940	0.57596
Stage2 Private Accuracy	0.53758	0.51715	0.54983	0.53921	0.54166	0.55473	**0.57724**
Public Stability	0.01502	0.01283	0.01286	0.01002	**0.00934**	0.02004	0.01045
Private Stability	**0.01909**	0.03349	0.03685	0.03940	0.04312	0.03395	0.03133

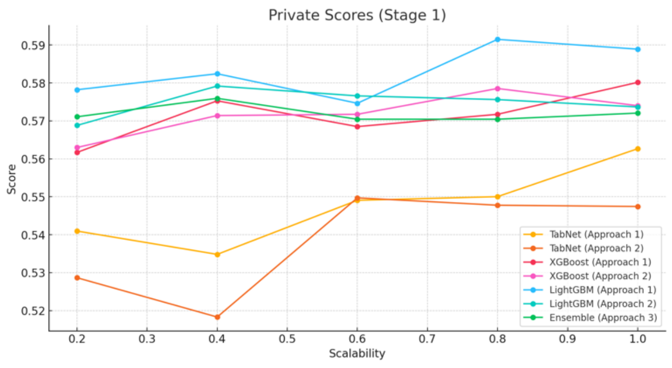

Fig 5

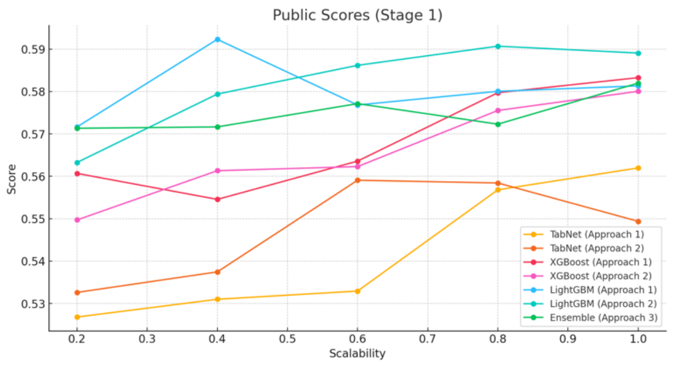

Fig 6

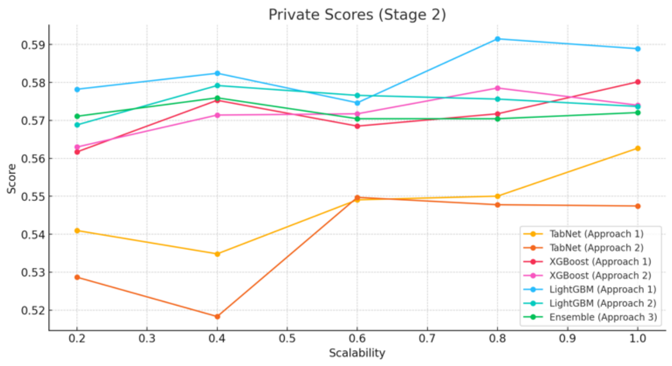

Fig 7

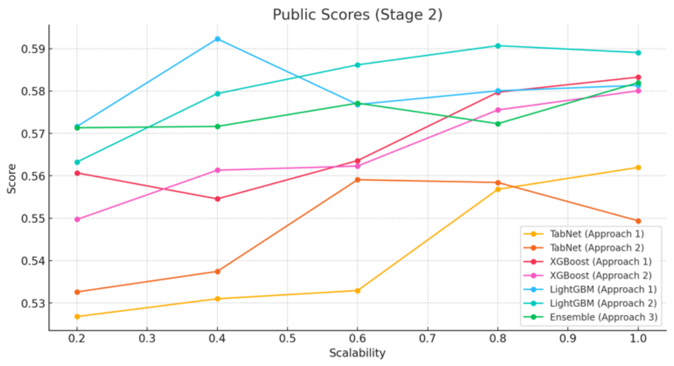

Fig 8

EXPERIMENT SETTINGS

```
#XGBoost parameters:
params = {
        'objective': 'binary:logistic',
'eval_metric': 'logloss', # Evaluation metric
'eta': 0.1, # Learning rate
'max_depth': 2, # Tree depth
'seed': 42, # Random seed
'tree_method': 'gpu_hist' # Enables GPU training
}

#LightGBM parameters:
parameter = {
        'objective': 'binary',
        'metric': 'auc', #or binary_logloss
        'boosting': 'gbdt',
        'num_leaves':16,
        'feature_fraction': 0.2,
        'bagging_fraction': 0.2,
        'bagging_freq': 30,
        'learning_rate': 0.01,
        'verbose': 0
}num_round = 200

# TabNet parameters:
        optimizer_fn: optim.AdamW
        optimizer_params: lr=0.01), # Learning rate
        scheduler_params={"step_size": 3, "gamma": 0.9}, # LR scheduler
        mask_type="sparsemax", seed=42 # For reproducibility
        max_epochs=50, # Number of epochs to train
        patience=10, # Early stopping patience
        batch_size=128, # Batch size
        virtual_batch_size=64, # Virtual batch size for larger datasets
        num_workers=0, # Number of workers
        drop_last=False

# Ensemble parameters:
catboost_params = {
   'iterations': 200,  # Number of boosting iterations
   'learning_rate': 0.01,  # Learning rate
   'depth': 8,  # Maximum depth of trees
   'l2_leaf_reg': 3,  # L2 regularization term
   'subsample': 0.8,  # Fraction of samples for training each tree
   'random_state': 42,  # Random seed for reproducibility
   'task_type': 'CPU',  # Use CPU for training
   'verbose': 100  # Verbosity level
}
XGBoost_params = {
   'objective': 'binary:logistic',  # Use for binary classification
   'eval_metric': 'logloss',  # Evaluation metric
   'eta': 0.01,  # Learning rate
   'max_depth': 4,  # Tree depth
'n_estimators': 300,  # Number of boosting rounds
   'subsample': 0.8,  # Subsample ratio of training instances
   'colsample_bytree': 0.8,  # Subsample ratio of columns
   'random_state': 42,  # Random seed
   'tree_method': 'hist'  # Use histogram-based optimization
}
LightGBM_params = {
   'objective': 'binary',  # Binary classification objective
   'metric': 'binary_logloss',  # Metric for evaluation
   'learning_rate': 0.01,  # Learning rate
   'max_depth': 4,  # Maximum depth of trees
   'num_leaves': 31,  # Number of leaves in one tree
   'n_estimators': 300,  # Number of boosting iterations
   'subsample': 0.8,  # Subsample ratio of training instances
   'colsample_bytree': 0.8,  # Subsample ratio of columns
   'random_state': 42,  # Random seed
   'verbose': -1  # Silence output
}
ensemble_weights = [0.3, 0.3, 0.4]  # Weights for CatBoost, XGBoost, and
LightGBM
```