Predictive Model for Chess Blunder

Raymond Wang
UC San Diego
San Diego, California, USA
rywang@ucsd.edu

Huize Mao
UC San Diego
San Diego, California, USA
h2mao@ucsd.edu

1 Introduction

Chess is a game of deep strategy and calculation, where players must constantly evaluate positions and make optimal moves. However, even skilled players are prone to making blunders—critical mistakes that drastically change the evaluation of a position. Detecting and understanding the nature of blunders is essential in chess training, as it allows players to improve decision-making and avoid common pitfalls. Traditionally, blunder detection has been performed post-move using chess engines such as Stockfish, which evaluate positions and assess move quality retrospectively. While effective, this approach does not provide insights into the likelihood of a blunder occurring before a move is made.

We propose a novel framework for pre-move blunder prediction by transforming raw chess data into a rich set of features and employing multiple machine learning models. We develop an extensive feature engineering pipeline that converts board states into high-dimensional representations—including spatial details, castling rights, and turn information—augmented by one-hot encoded game types, average player ratings, and time-series lagged features computed on a per-game basis. We evaluate a range of models, from traditional Logistic Regression and Naive Bayes to ensemble methods like Random Forest and XGBoost, as well as convolutional neural networks that extract intermediary features from board convolutions. Through rigorous hyperparameter tuning and loss function adjustments to prioritize blunder detection, our results demonstrate that our integrated approach can modestly improve predictive performance, ultimately offering chess players a more cautious and informative decision support tool.

2 Code Availability

We have made our implementation and experiments publicly available at the following GitHub repository:

https://github.com/HuizeMao/Chess-Blunder-Detection

This repository contains all the source code, data processing scripts.

3 Dataset

The dataset used for this project is the Games.csv dataset from Kaggle, specifically sourced from lichess.org. It contains a collection of 20,000 chess games played on the platform. Each row in the dataset corresponds to an individual game, with details such as the moves made, the winner, and the players' usernames.

3.1 Data Cleaning

The dataset was thoroughly cleaned to ensure it was in an appropriate format for analysis and modeling:

Missing Values: After inspecting the dataset, we found that there are no missing values (NaN) in any of the columns, ensuring completeness of the data.

Anonymization: To protect the privacy of players, the player_id and id columns were removed, ensuring that the dataset does not reveal personal information.

Irrelevant Columns: The columns Opening_eco, Opening_name, and Opening_ply were removed since the focus of this analysis is not on opening theory but rather on the game moves and predictions related to blunders.

Data Transformation: Initially, each game occupied one row in the dataset. However, to facilitate a move-based analysis, we expanded the dataset so that each move in the game is now represented by a separate row. The board state at each move is captured as an 8x8x17 numpy array, where: The first 8x8 grid represents the chessboard. The 12 layers in this grid correspond to the different types of chess pieces for both players. 4 layers are used to indicate the castling rights of both players. An additional 1 layer indicates whose turn it is (Player 1 or Player 2).

3.2 Exploratory Data Analysis (EDA)

In the exploratory data analysis (EDA) phase, several key characteristics of the dataset were investigated:

Player Ratings To understand the distribution of player ratings in the dataset, a histogram was plotted. This visualization shows the range of ratings and helps identify the concentration of players within certain rating brackets. The histogram reveals that most

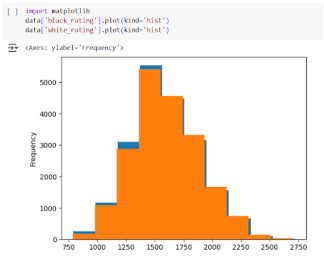

Figure 1: player rating: blue is black and orange is white

players have ratings between 1500 and 1700. We decided to filter the dataset to include only games played by players within this rating range as players of similar strength are more likely to make comparable blunders

Game Length (Turns) The number of turns played in each game provides insight into how long the games typically last. A histogram of the game lengths shows the distribution of the number of turns per game.

Increment Code (Time Control): The Increment Code column indicates the time control used in each game. A table was created to show the distribution of different time controls (increment codes) across the dataset. This table reveals that the most common time

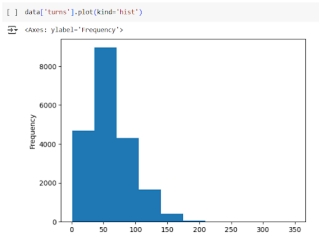

Figure 2: a histogram of the number a turn a game lasts

increment_code	
10+0	7721
15+0	1311
15+15	850
5+5	738
5+8	697
8+0	588
10+5	579
15+10	461
20+0	448
30+0	375
10+10	355
15+5	314
7+2	265
10+2	187
30+30	169
5+10	155
25+0	152
10+3	149
10+8	123
20+10	103
10+15	103

Figure 3: Time control and the number of game played in that time control

control by far is 10+0 (10 minutes with no increment). Since 10+0 is considered rapid time control, the dataset is filtered to only games in the rapid category (10+0, 10+5, 15+10) to make sure players are under similar time pressure during the game.

4 Predictive Task

The predictive task is to predict whether a given move in a chess game is a blunder. Our dataset contains features such as board representations, move numbers, and game types (one-hot encoded). The goal is to build a classifier that outputs a binary decision—"blunder" or "no blunder"—for each move.

4.1 Evaluation Metric

F1 Score: Since the dataset is imbalanced (with far fewer blunders than non-blunders), the F1 score provides a balanced measure that considers both precision and recall. Confusion Matrices: These will help us visualize the types of errors (false positives and false negatives) each model makes.

4.2 Baseline Models

Naive Bayes: Naive Bayes classifiers are a family of probabilistic models based on Bayes' theorem with the assumption that features are conditionally independent given the class label. Despite this strong assumption, Naive Bayes models are valued as baselines because they are extremely fast to train and are simple to implement, which makes them a good starting point for classification tasks. In our application, we experimented with different Naive Bayes variants such as Gaussian, Bernoulli, and Multinomial Naive Bayes.

Baseline Results: F1 scores of approximately 0.1903 on the training data and 0.1849 on the test data. The confusion matrix for one of the Naive Bayes variants on the test set was [[14698, 15607], [892, 1871]]. The low performance suggests that the independence assumption made by Naive Bayes may not hold well in the context of our chess dataset, where the features—such as board embeddings and move numbers—are likely to exhibit complex interdependencies. This underperformance emphasizes the need for more sophisticated models that can capture these interactions more effectively.

5 Model

5.1 Feature Engineering

5.1.1 Board Representation. **Board Representation:** The chess board state is encoded using a one-hot scheme across 1088 dimensions (i.e., `board_dim_0` to `board_dim_1087`). This encoding captures not only the occupancy of pieces on the board but also includes additional game-specific details such as **castling rights** and **turn information**, providing a comprehensive representation of the game state.

5.1.2 One-hot Encoding of Game Types . **One-hot Encoding of Game Types:** Categorical features such as game types (e.g., 10+0, 15+5) are one-hot encoded to preserve the distinct characteristics of different time controls.

5.1.3 Time-series Lagged Features. **Time-series Lagged Features:** Given the sequential nature of chess, we generate lagged features by grouping moves by game and using the `move_number` column to infer a **game_id** (which resets to 1 for each new game). For each game, the board representation is shifted by 1, 2, and 3 moves (e.g., `board_dim_0_t-1`, `board_dim_0_t-2`, `board_dim_0_t-3`) to capture the temporal dynamics. The train-test split is performed on a per-game basis to maintain the temporal continuity within each game.

5.1.4 Efficient Data Representation. **Efficient Data Representation:** Due to the high dimensionality of the board features, memory efficiency is paramount. We convert the board representation from `int64` to `int8` since the values are binary (0 or 1) and store these features in a sparse format. This significantly reduces the memory footprint while retaining the critical spatial information.

5.1.5 Convolutional Feature Extraction. **Convolutional Feature Extraction:** To better capture the spatial hierarchies in the board representation, we applied Convolutional Neural Networks (CNNs). A CNN trained solely on the board data achieved an F1 score of approximately 0.19. However, to leverage the strengths of both approaches, we extract features from an intermediary convolutional

layer and integrate these **CNN-derived features** with the other engineered features (including those used by Naive Bayes but not by the CNN). This fusion of heterogeneous features is expected to improve the overall predictive performance by capturing both spatial and contextual information more effectively.

In summary, our feature engineering pipeline combines the comprehensive board representation (including around 1088 dimensions that encode board configuration, castling rights, and turn information), one-hot encoding of game types, the average player rating, and time-series lagged features. These steps yield a final dataset with approximately 5000 features, which robustly captures the spatial, categorical, and temporal complexities of chess games for downstream predictive modeling.

5.2 Choice of Model

We explored three primary models for the blunder prediction task: **Random Forests**, **XGBoost**, and **Logistic Regression**.

- **Logistic Regression** is a classic linear model.
- **Random Forests** are an ensemble of decision trees that reduce overfitting by averaging multiple trees. They are well-suited for structured data, can capture non-linear relationships, and handle a large number of features robustly.
- **XGBoost** (Extreme Gradient Boosting) is a gradient boosting framework known for its efficiency and accuracy on tabular data. It incrementally adds weak learners (decision trees) to minimize a differentiable loss function, often achieving state-of-the-art performance in structured data competitions.

5.3 Training Procedure and 70–30 Split

To preserve the sequential nature of games, we split the dataset at the game level rather than at individual moves. Specifically, **70% of the games** were used for training and **30%** for testing. This ensures that no single game is partially split between the train and test sets, thus avoiding data leakage and maintaining the temporal consistency within each game.

5.4 Hyperparameters and Penalty for Blunders

For each model, we performed hyperparameter tuning to optimize the F1 score. Key hyperparameters included:

- **Logistic Regression**: `max_iter` (e.g., 1000), and `class_weight='balanced'` to address class imbalance.
- **Random Forests**: `n_estimators` (number of trees), `max_depth`, and `min_samples_split`, among others.
- **XGBoost**: `max_depth`, `learning_rate`, `n_estimators`, `subsample`, `colsample_bytree`, and `scale_pos_weight` (which penalizes misclassifications of the minority class more heavily).

To further penalize wrong predictions of blunders, we introduced an additional **penalty factor** in the loss function (e.g., using `class_weight='balanced'` in Logistic Regression, increasing `scale_pos_weight` in XGBoost, or applying a custom penalty term). Across all three models, this adjustment improved the F1 score by approximately 0.01, reflecting the importance of focusing on the minority (blunder) class.

Overall, the combination of careful hyperparameter tuning, game-based splitting, and increased penalties for misclassifying blunders led to a measurable improvement in performance, demonstrating the effectiveness of these strategies in addressing the challenges of an imbalanced chess dataset.

6 Literature

In our project, the novelty lies in predicting the **probability of making a blunder given a particular position**, rather than evaluating a blunder after it has already occurred. Traditional methods rely on detecting blunders after a move is made, using engines to evaluate the move and then identifying mistakes. However, our approach predicts the likelihood of a blunder happening in advance, based on the current game situation, thus offering a proactive way to analyze potential errors.

The **Maia Chess project** offers valuable insights for our approach, particularly in how they use neural networks to model human decision-making and predict moves. Maia's dataset consists of games played by humans at various skill levels, and it is used to train models that simulate human play and predict decision-making. Maia does not specifically focus on blunders but incorporates human error by modeling player tendencies, such as moves leading to mistakes. They use neural networks to understand these patterns and forecast the next move, effectively predicting human decisions.

While our task of predicting blunders is different in its specific focus, it aligns with Maia's use of neural networks to predict human behavior in chess. Maia's approach is grounded in modeling human decision processes and predicting moves that may be suboptimal, and this inspired the methodology for our blunder prediction task. Others in the field typically approach blunder detection post-move, relying on engines to evaluate a move's quality after it's played. However, Maia's method of modeling the decision-making process via neural networks is more aligned with the idea of anticipating potential blunders before they happen, a concept we have applied specifically to error prediction.

State-of-the-art methods in this area focus on move prediction, either using chess engines like **Stockfish** or deep learning models to suggest optimal plays. While post-move evaluation remains standard for blunder detection, Maia's use of neural networks to anticipate moves and human decisions represents a shift towards more predictive, forward-looking analysis, which influenced our work on blunder prediction.

7 Results

In this section, we present and analyze the performance of three models—Logistic Regression, XGBoost, and Random Forest—on the chess blunder detection task. We provide their hyperparameters, F1 scores, and confusion matrices, with particular attention to the minority class (blunders).

Precision measures the proportion of predicted positives that are truly positive, i.e., among all moves classified as blunders, how many are actual blunders. **Recall** (also known as sensitivity) measures the proportion of actual positives that are correctly predicted, i.e., among all real blunders, how many the model identifies correctly. Balancing these two metrics is crucial in an imbalanced

classification task such as blunder detection, which is why we also track the F1 score as a harmonic mean of precision and recall.

7.1 Logistic Regression

Hyperparameters: `max_iter=1000`, `class_weight='balanced'`.

Performance:

- **F1 Score (Training Data):** 0.2384
- **F1 Score (Testing Data):** 0.2315

Confusion Matrix (Testing Data):

$$\begin{bmatrix} 19297 & 11008 \\ 960 & 1803 \end{bmatrix}$$

For the minority class (blunders), the **precision** is

$$\frac{1803}{1803 + 11008} \approx 0.14 \quad (14\%)$$

and the **recall** is

$$\frac{1803}{1803 + 960} \approx 0.65 \quad (65\%).$$

The relatively high recall indicates that Logistic Regression identifies a substantial fraction of actual blunders, although the precision is low due to a high number of false positives.

7.2 XGBoost

Hyperparameters:

- `n_estimators=1000`, `max_depth=3`, `min_child_weight=2`
- `learning_rate=0.05`, `subsample=0.9`, `colsample_bytree=0.8`
- `scale_pos_weight=10`, `alpha=5`, and a `custom_objective` that penalizes false negatives.

Performance:

- **F1 Score (Training Data):** 0.3718
- **F1 Score (Testing Data):** 0.2414

Confusion Matrix (Testing Data):

$$\begin{bmatrix} 22877 & 7428 \\ 1364 & 1399 \end{bmatrix}$$

For XGBoost, the **precision** for the blunder class is

$$\frac{1399}{1399 + 7428} \approx 0.16 \quad (16\%)$$

and the **recall** is

$$\frac{1399}{1399 + 1364} \approx 0.51 \quad (51\%).$$

Thus, while XGBoost slightly improves the overall F1 score compared to Logistic Regression, it achieves this by trading some recall for a marginally better precision.

7.3 Random Forest

Hyperparameters:

- `n_estimators=300`, `max_depth=5`, `min_samples_split=2`
- `min_samples_leaf=1`, `max_features=0.9`, `class_weight='balanced'`.

Performance:

- **F1 Score (Training Data):** 0.2393
- **F1 Score (Testing Data):** 0.2011

Confusion Matrix (Testing Data):

$$\begin{bmatrix} 14462 & 15843 \\ 683 & 2080 \end{bmatrix}$$

For Random Forest, the **precision** for the blunder class is

$$\frac{2080}{2080 + 15843} \approx 0.12 \quad (12\%)$$

and the **recall** is

$$\frac{2080}{2080 + 683} \approx 0.75 \quad (75\%).$$

Random Forest achieves the highest recall, meaning it identifies most actual blunders. However, its precision is very low due to an excessive number of false positives, leading to a lower overall F1 score.

7.4 Analysis of Blunder Detection

Overall, these results show varying trade-offs between precision and recall for the minority class. **Logistic Regression** and **XGBoost** maintain more balanced precision-recall trade-offs, while **Random Forest** excels at capturing more true positives (higher recall) but at the cost of many false positives (leading to a lower overall F1). The custom penalty for blunders (e.g., `scale_pos_weight` in XGBoost or `class_weight='balanced'` in Random Forest/Logistic Regression) helped increase the detection rate for the minority class. Our modeling approach, which includes Logistic Regression, XGBoost, and Random Forest, shows that predicting blunders is challenging due to the imbalanced nature of the data. We observed that even though some models, such as Random Forest, achieve high recall, their low precision results in an overall low F1 score. Importantly, our strategy deliberately errs on the side of caution by predicting more potential blunders. In the context of chess, this approach is preferable: even if a blunder prediction turns out to be a false alarm, it serves as a warning for the player to exercise extra caution. This is a less severe outcome compared to missing a true blunder, which could lead to a critical error in gameplay. Therefore, by incorporating a higher penalty for misclassifying the blunder class and adjusting the loss functions accordingly, we achieved an improvement of approximately 0.01 in F1 score. Overall, our system encourages a more cautious play style, which is beneficial in high-stakes competitive settings.

8 Discussion

During the training of our models, we encountered several challenges that impacted performance and scalability.

8.1 Computing Power

One of the major obstacles was the limitation of available computing resources, particularly RAM. Due to memory constraints, we were unable to utilize large datasets efficiently, which restricted the amount of data we could process at once. This forced us to downsample our dataset or use batch processing techniques, ultimately affecting model performance.

8.2 Data Size

The relatively small dataset size posed another challenge. We suspect that more complex models, such as deep learning approaches,

were unable to reach their full potential due to insufficient training data. A larger dataset would allow models to generalize better and learn more meaningful patterns, reducing overfitting.

8.3 Class Imbalance

The dataset exhibited a significant class imbalance, with a much higher number of "no blunder" cases compared to "blunders." This imbalance led to biased models that favored the majority class. To mitigate this, we employed several strategies:

Bootstrapping: We generated a more balanced dataset by resampling data points from the minority class. Class Weight Adjustment: We adjusted the class weights in our models to ensure equal importance for both classes, reducing bias. Alternative Sampling Methods: Other approaches, such as oversampling the minority class or undersampling the majority class, were also explored. In this work, we presented a novel framework for pre-move blunder prediction in chess by leveraging a comprehensive feature engineering pipeline and a variety of machine learning models. Our approach integrates high-dimensional board representations—including spatial configurations, castling rights, and turn information—with one-hot encoded game types, average player ratings, and time-series lagged features computed on a per-game basis. Additionally, we explored convolutional neural networks to extract intermediary spatial features and combined these with traditional models such as Logistic Regression, Random Forests, and XGBoost. Through rigorous hyperparameter tuning and the introduction of loss penalties that emphasize the detection of blunders, we achieved modest improvements in F1 score. Importantly, our strategy prioritizes recall—ensuring that potential blunders are rarely missed—even at the expense of precision, which is acceptable in this context since a false alarm merely serves as a cautionary signal for players.

8.4 Conclusion and Takeaways

- **Comprehensive Feature Engineering:** Our pipeline transforms raw chess data into an informative feature set of approximately 5000 dimensions. The board representation not only encodes the piece positions but also integrates castling rights and turn information, providing a detailed snapshot of the game state. In addition, one-hot encoding of game types and the calculation of average player ratings supply essential contextual data.
- **Temporal Dynamics:** By constructing time-series lagged features on a per-game basis, we capture the evolution of board positions over time. This approach ensures that the sequential nature of chess is preserved, and models can learn temporal dependencies that are critical for predicting blunders.
- **Model Trade-offs:** Our experiments with Logistic Regression, XGBoost, and Random Forest highlight the inherent trade-offs in imbalanced classification. While Random Forests achieved high recall for blunder detection, they suffered from low precision. In contrast, Logistic Regression and XGBoost offered a more balanced performance, though overall F1 scores remained low due to the high number of false positives.
- **Emphasis on Caution:** We adjusted the loss functions (via parameters like `class_weight` and `scale_pos_weight`) to penalize misclassifications of blunders more heavily. This approach, which improved F1 scores by approximately 0.01, aligns with the practical consideration that in a chess context, it is preferable to warn players about a potential blunder—even if it results in false alarms—rather than miss a critical mistake.
- **Integrated Approach for Enhanced Performance:** Combining traditional machine learning models with convolutional feature extraction enables the system to capture both spatial and temporal patterns in the data. While each individual model has its own limitations, their integration through an ensemble strategy holds promise for further improvements in predictive performance.

References

[1] **McIlroy-Young**, et al. (2020). *[Aligning Superhuman AI with Human Behavior: Chess as a Model System]*. arXiv preprint, arXiv:2006.01855. https://arxiv.org/pdf/2006.01855

[2] CSSLab. *Maia Chess: Blunder Prediction*. GitHub Repository. Available: https://github.com/CSSLab/maia-chess/tree/master/blunder_prediction