Predictive Analytics in Online Chess: Assessing Match Outcomes Based on Player Performance and Behavioral Patterns

Adam Cunningham, Ian Campbell

STA4724 Final Report

December 9, 2023

Abstract

This report explores the prediction of online chess game outcomes by integrating player performance metrics and chessboard analysis. We examine factors like time of day, match volume and accuracy, along with Elo rating trends. The study also categorizes game states by phase (opening, middle, endgame) and color advantage, enabling the clustering of players by playstyle. This includes analysis of pawn structure and comparison with engine-recommended moves. Our approach aims to provide deeper insights into chess game dynamics and improve matchmaking predictions.

Contents

1 Introduction

This introduction sets the stage for the report, providing context on the nature of chess and online matchmaking, and clearly outlining the objectives of the study in analyzing matchmaking and playstyle-based features.

1.1 Overview of Chess and Online Matchmaking

Chess, a game with a rich historical lineage, has evolved into a globally celebrated mind sport, epitomizing strategic depth and intellectual rigor. The advent of the digital age has transformed chess into an online phenomenon, with platforms like Chess.com and Lichess.org democratizing access and facilitating global competition. The diversity of possible chess games is virtually limitless, and the skill ceiling is unparalleled, making it a perpetually challenging and evolving puzzle.

In the realm of online chess, players engage not just in gameplay but also in rigorous theoretical study. This includes mastering openings, developing strategies, and understanding common tactics and errors. Mastery of chess pieces and their potential moves is fundamental, serving not only the purpose of gameplay but also offering a rich source for analytical insight.

1.2 General Mechanics of Matchmaking and Elo Rating System

Online chess platforms typically employ a ranking system, most commonly the Elo system or its derivatives like Glicko2, to match players. These systems adjust a player's rating based on their game outcomes – increasing with wins and decreasing with losses. The matchmaking process aims to pair players queuing for a game who have similar ratings, theoretically indicating comparable skill levels. The ideal matchmaking outcome is to ensure that each player wins about 50% of their games, with each match being as competitive and engaging as possible. As players' win rates deviate from this 50% mark over time, their ratings are adjusted to converge towards their theoretical skill level, thus matching them with appropriately skilled opponents.

1.3 Objectives of the Study

The primary objective of our study is to analyze patterns in players' matchmaking data to identify factors influencing game outcomes. For instance, we aim to determine if a player's performance varies based on the time of day or their sequential game number (e.g., first game of the day vs. tenth game). By exploring these matchmaking-based features, we hope to develop a predictive model that surpasses the 50% win/loss baseline in predicting game outcomes, excluding draws.

Our secondary objective delves into the more complex realm of playstyle-based features. The challenge lies in identifying and quantifying meaningful aspects of a player's style. Our approach includes creating detailed playstyle profiles for players and comparing these against opponents' profiles. This comparison aims to uncover patterns in win rates against various playstyles, with a

particular focus on the openings used by players. Should the playstyle-based approach not yield significant predictive value, we propose methodologies for refining and enhancing these analytical techniques.

2 Related Work

2.1 Chess Programming

Chess Programming, a field within computer and data science, focuses on chess games. Its modern objectives, which align with those of online chess platforms like chess.com, include but are not limited to:

- **Chess Engine Development**
  Development of algorithms that comprehend chess rules and strategize moves. Engines generate legal moves efficiently, evaluate chess positions, and employ search algorithms like Minimax and Alpha-Beta pruning. This helps in choosing the optimum move from possible alternatives. Popular chess engines include Google Deepmind's **AlphaZero** and its open-source rival **Stockfish**, the strongest chess engine publicly available.
- **User Interface Development**
  Designing user-friendly interfaces for chess software, which includes the visual representation of the chessboard, move input methods, and options to analyze games, import/export games, etc.
- **Game Analysis Tools**
  Developing tools for game analysis, including move-by-move analysis, tactical puzzles generation, and positional evaluation to aid players in improving their skills.
- **Accessibility Features**
  Making chess software accessible to all users, including those with disabilities, through features like screen reader compatibility, voice commands, and customizable visual elements.

2.2 Insightful Papers on Chess Programming

Two helpful resources that aided us in understanding some of the intricacies of chess programming are a study done by De Marzo and Servedio [1] on quantitatively analyzing chess openings, and a summary article by Antoine Champion [2, 3] explaining the high-level ideas behind the strongest open-source chess engine Stockfish.

Both of these publications help bridge the gap between raw chess data and creating meaningful interpretations by introducing mathematical and matrix notation, such as bipartite networks between players and openings, and "bitboards" representing piece state of a chess board.

3 Database Building

3.1 Data Sourcing from Chess.com

We source data from Chess.com, which holds full records of player game histories. We select a "player of interest" and query their game history, then for every opponent, query the opponent's history. The choice to focus on a single player, rather than attempting to generalize across a larger player base, enables us to model to the context of an individual player. This approach more easily enables us to grab player-dependent insights from the data, which is necessary for matchmaking-based prediction features, which in turn improves accuracy. The model described in this study can be conducted on any chosen chess player with a history of games on Chess.com, however performance will vary, depending on the amount of games available to analyze and the relevance of said game history.

This study highlights Tyler Steinkamp, known as BIG_TONKA_T on Chess.com, as our "player of interest" due to his substantial game history. Rated around 1400 (98^{th} percentile), Steinkamp has played over 1500 games between October and November 2023. This large volume enhances statistical validity and exploration of both his matchmaking features and playing style. Steinkamp's consistent performance and engagement on Chess.com make his data both comprehensive and representative of a dedicated chess player's profile.

Each monthly archive received from api.chess.com offers a time-series JSON format, containing a detailed record of every game played within that month. This dataset the time format of each game, unix timestamp, color played by each participant, rating status of the game (rated or unrated), post-match ELO, and the final result. These features make up the basis of the matchmaking based predictors.

Figure 1: Left: Example JSON for one game of data. The PGN holds every move of the game. Right: Some moves displayed after processing the PGN, courtesy of the game page on Chess.com.

The Portable Game Notation (PGN) for each match is also provided. PGN is a standard format for recording chess games, encompassing all the moves made during a game, enabling a derivation of each board state and the moves made at every juncture. This part of the data

contributes almost solely to the 'playstyle' component of a player. Figure 1 shows an example of data for a particular chess game.

3.2 Dataset Building and Filtering Criteria

The data gathered is managed using a custom Python module we developed specifically for querying data from Chess.com's API. This module extracts, processes, and organizes the data. Further details on the methodologies employed in data processing, including the structure of the code, is discussed in the Codebase section of this paper.

Once the data is obtained, it is cached locally, minimizing the dependency on live API calls. From there, we can perform repeated analyses and build hand-selected datasets in the form of CSV files.

The datasets we build are typically subject to some filtering. Throughout the paper, games are almost exclusively filtered on the basis that they are rated, rapid time format, and result in either a win or loss for the player of interest (draws are omitted).

One last note about dataset building is that we correctly use the post-game Elo for the previous game as the pre-game Elo for the current game, by simply shifting the Elo column by one in the player history data as well as the Elo column in each of the opponent game histories. The Python module mentioned above and discussed in the Codebase section of this paper handles this correction by default.

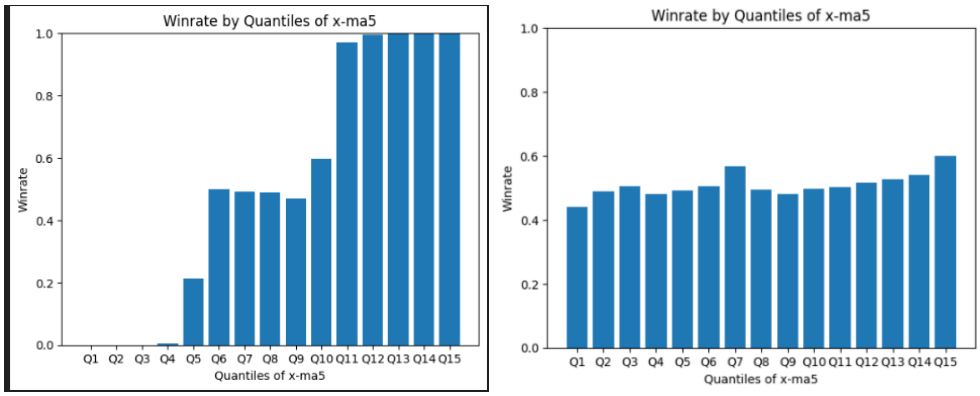


Figure 2: Left: Winrate by quantiles of the difference between current elo and the 5 game moving average, before the correction. This graph suggests that the bottom three and top three quantiles have virtually 100% accuracy in predictions. Right: Same as left, but after the correction. The data still has an identifiable pattern, but is greatly subdued.

Figure 2 shows the potentially drastic difference between analysis on post-game and pre-game elo. Clearly, we want a model that only predicts games using past and current information, so the correction is necessary and as such is reported here.

4 Match Performance Predictive Features

4.1 Elo Moving Averages

The Simple Moving Average (SMA) is given by the formula:

$$\text{SMA} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{1}$$

where N is the number of periods (games) in the average, and x_i represents the value (Elo) in each period.

Using Equation 1, we will do 2 simple moving averages for the player, one with $N = 5$ and another with $N = 20$. This will give us the MA5 and the MA20 for the player respectively. We expect the MA5 to follow closely to the current Elo and the MA20 to follow more loosely. We hypothesize that looking at differences between the current Elo and moving averages can provide predictive power to a model. Figure 3 shows a line plot of Tyler's Elo over the course of 300 games, with a MA20 line to compare.

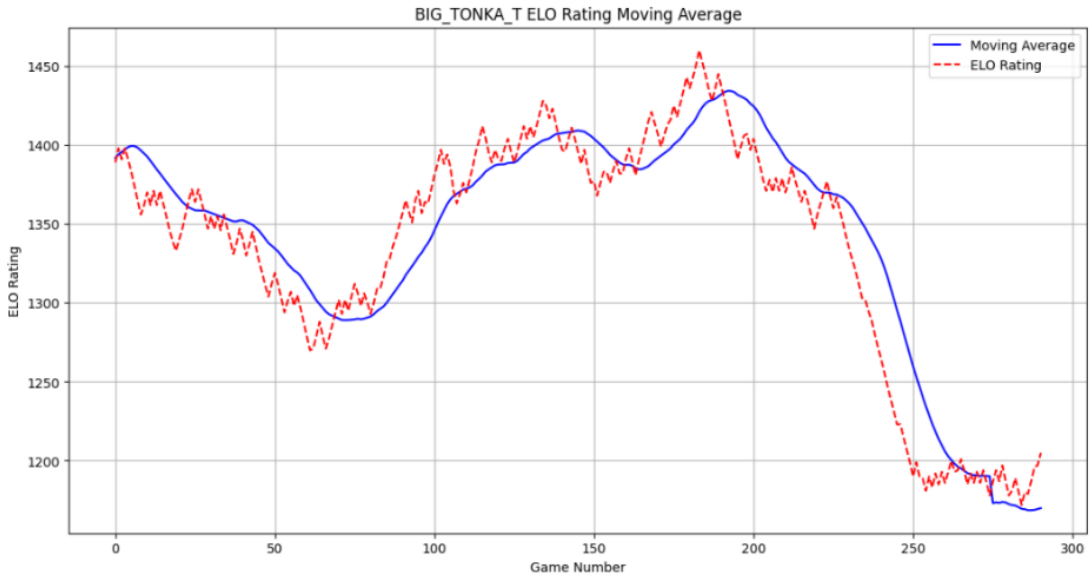


Figure 3: A line plot showing Elo on per-game basis over 300 games for Steinkamp. Red dashed line shows exact Elo. Blue line shows the MA20, or the moving average of 20 games Elo.

MA5 and MA20 can also be calculated for each opponent in the history, by querying the recent opponent history up to the Unix timestamp of the game the player of interest matched against the opponent. Figure 4 shows a sample of entries in a dataframe after the moving averages have been calculated.

Running a logistic regression model with the 4 moving average features with 5-fold cross validation nets an accuracy of 52.91% across 1561 games between October and November 2023 (some games were dropped due to an inability to calculate opponent moving averages if they did not have a sufficient number of games in the past month).

	player_name	opp_name	player_elo	opp_elo	elo_diff	color	x-ma5	x-ma20	opp_x-ma5	opp_x-ma20	won
1108	BIG_TONKA_T	richie340	1442	1383	59	False	13.0	-54.85	-9.8	0.20	0
1204	BIG_TONKA_T	BoilerAndy	1332	1301	31	False	4.8	-2.05	16.8	41.60	0
784	BIG_TONKA_T	macaphella	1413	1397	16	True	-3.8	-11.75	-13.0	-39.15	1
1411	BIG_TONKA_T	rajesh_gangam	1402	1449	-47	True	17.4	94.35	5.4	-10.45	1
902	BIG_TONKA_T	kabanjahe51	1543	1631	-88	True	10.0	110.25	7.2	71.75	0

Figure 4: A sample of 5 entries from a dataframe where MA5 and MA20 have been calculated for the player and the opponent. For regression, we are interested in the difference between the current Elo and the moving average, hence the naming X-MA(N).

To account for the non-random missingness, a Decision Tree classifier is used on all 1663 games, and reports roughly 52.5% accuracy after optimizing for the maximum number of leaf nodes, as shown in 5.

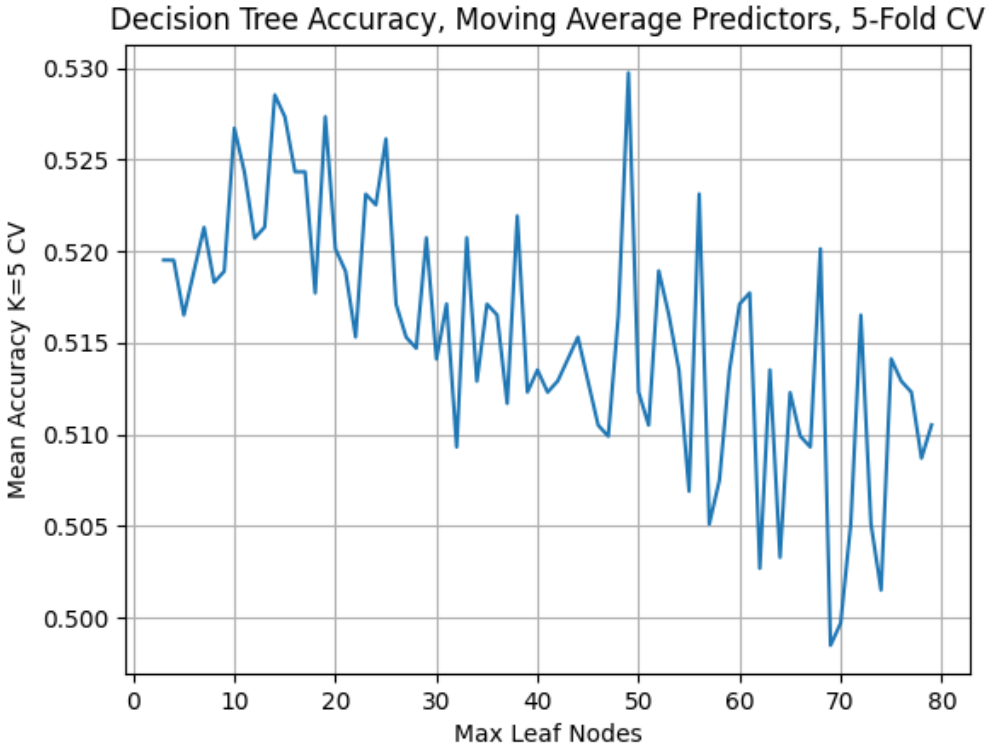


Figure 5: A Decision Tree classifier is used on the data using the 4 moving average predictors. Accuracy is plotted to view the most optimal number of maximum leaf nodes, which lies between 10 and 20, netting an accuracy of about 52.5%.

4.2 Recent Queue Activity

The time series data gives us Unix timestamps of every match played in the dataset. Before filtering for draws, we can get an idea for the amount of time in between each game, by subtracting the Unix timestamp of the current game from the preceding game. Figure 6 shows the shape of the distribution of time in between online chess games played by Steinkamp, which becomes apparent with a large enough sample size as the queue wait time and the duration of the chess

match become neglible.

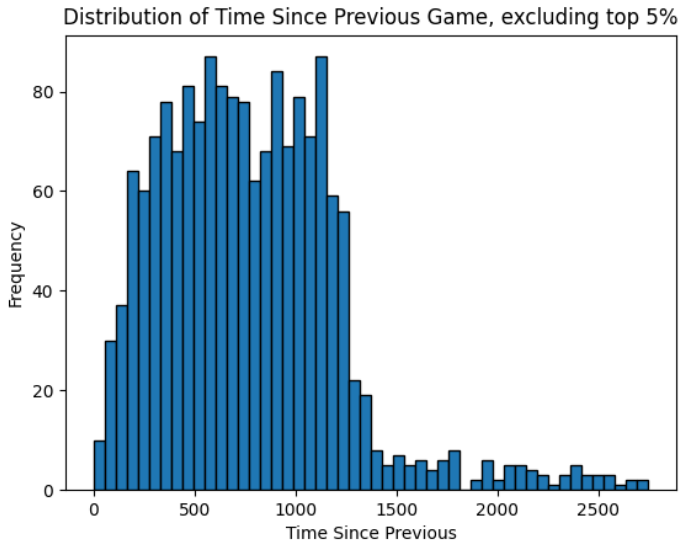


Figure 6: A histogram displaying the distribution of amount of time since the previous game. For clarity, the top 5% of times recorded have been omitted, as they tail off to the right.

On chess.com, or any online game featuring real-time matchmaking, players can 'queue' for a game, in which they will be in a state where they wait to be matched with an opponent. We will define a *reque* as an instance where a player queues up for another game very shortly after concluding the previous game. The amount of time where less than it becomes a reque and more than it is deemed not is the *reque time threshold*. We will define the *number of reques* preceding a game to be the number of games in a row where the time since previous was less than the reque time threshold. Figure 7 shows what a dataframe looks like after the time since previous and number of reques preceding each matchup is calculated. Figure 8 shows the number of reques distribution.

	unix	won	player_acc	opp_acc	acc_diff	num_reque	time_since_prev
0	1696185098	1.0	69.43	62.25	7.18	0	0.0
1	1696186352	1.0	78.20	69.40	8.80	0	1254.0
2	1696187659	1.0	92.66	80.43	12.23	0	1307.0
3	1696188843	NaN	67.25	60.95	6.30	1	1184.0
4	1696189372	1.0	83.82	72.41	11.41	2	529.0
5	1696190619	0.0	80.57	78.02	2.55	3	1247.0
6	1696191447	1.0	64.50	57.93	6.57	4	828.0
7	1696192994	0.0	51.35	64.64	-13.29	0	1547.0
8	1696193859	1.0	73.48	58.18	15.30	1	865.0

Figure 7: The dataframe with columns for features related to queue activity. *time_since_prev* refers to the amount of time in seconds between the end time unix of this game and the preceding game. *num_reque* gives us the number of games played back to back preceding this game. The threshold used was 1250 seconds.

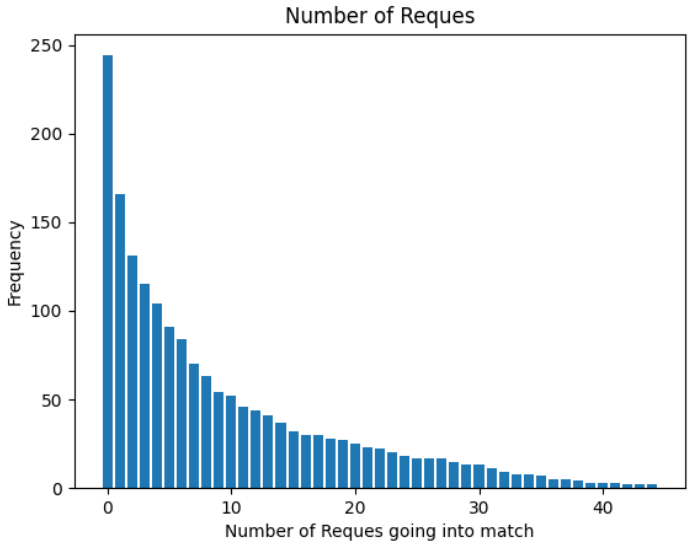


Figure 8: A histogram displaying the distribution of number of reques. Clearly, there must be a game with N reques in order for there to be one with $N-1$. Therefore the distribution is non-ascending.

Does the amount of time since the previous game affect the performance of the current game? We allow Figure 9 to help analyze.

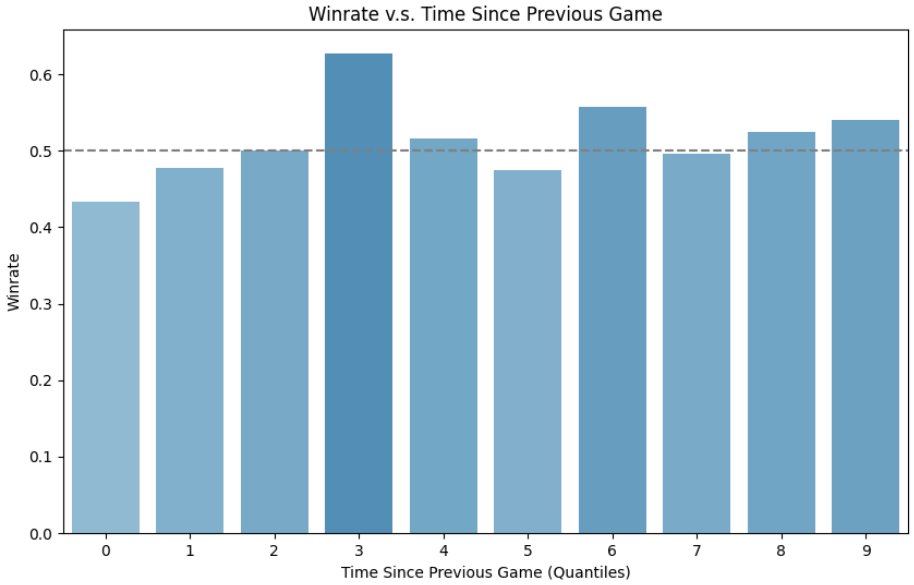


Figure 9: Winrate against the amount of time since the previous game. Horizontal line at 50% winrate for comparison.

Clearly, there is some sort of relationship between time since the previous game and winrate. In particular, the first two quantiles suggest that for the lowest values of time since previous, winrate suffers heavily, to almost 40% in the case of the lowest quantile. We also see in Figure 10

that winrate performance drops with the number of games queued in succession. From these two charts we can hypothesize that there is a link between queue activity and game performance, perhaps due to the naturally increasing difficulty in keeping a consistent level of performance over an increasing number of games in succession.

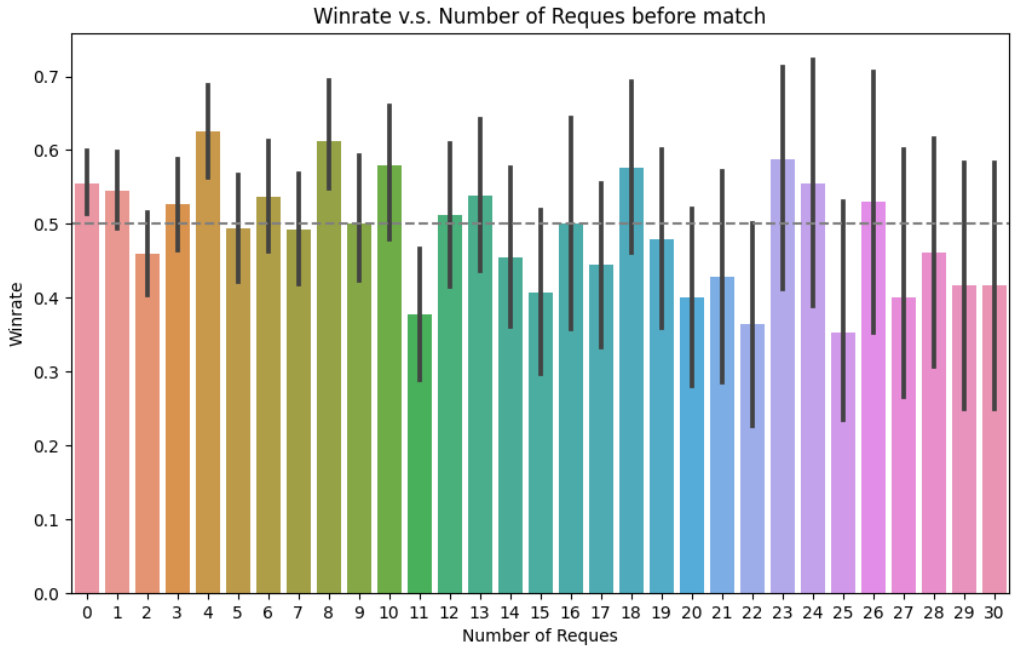


Figure 10: Winrate against the number of reques, represented as bars with 80% confidence intervals. Number of reques are listed up to 30, as past 30 there are not enough games played to make meaningful assertions.

Using the same prediction methods as in Elo Moving Averages, we use logistic regression on all games where the features can be calculated, which in this case is actually all of the games, meaning no missgness. Logistic regression with 5-fold cross validation applied to a dataset of 1656 games returns an accuracy mean of 52.11%, using just the predictor of number of reques.

The decision tree model works best with both features as predictors, using the number of reques and the time since previous game. The 5-fold cross validation accuracy plot against maximum number of leaf nodes is show in Figure 11.

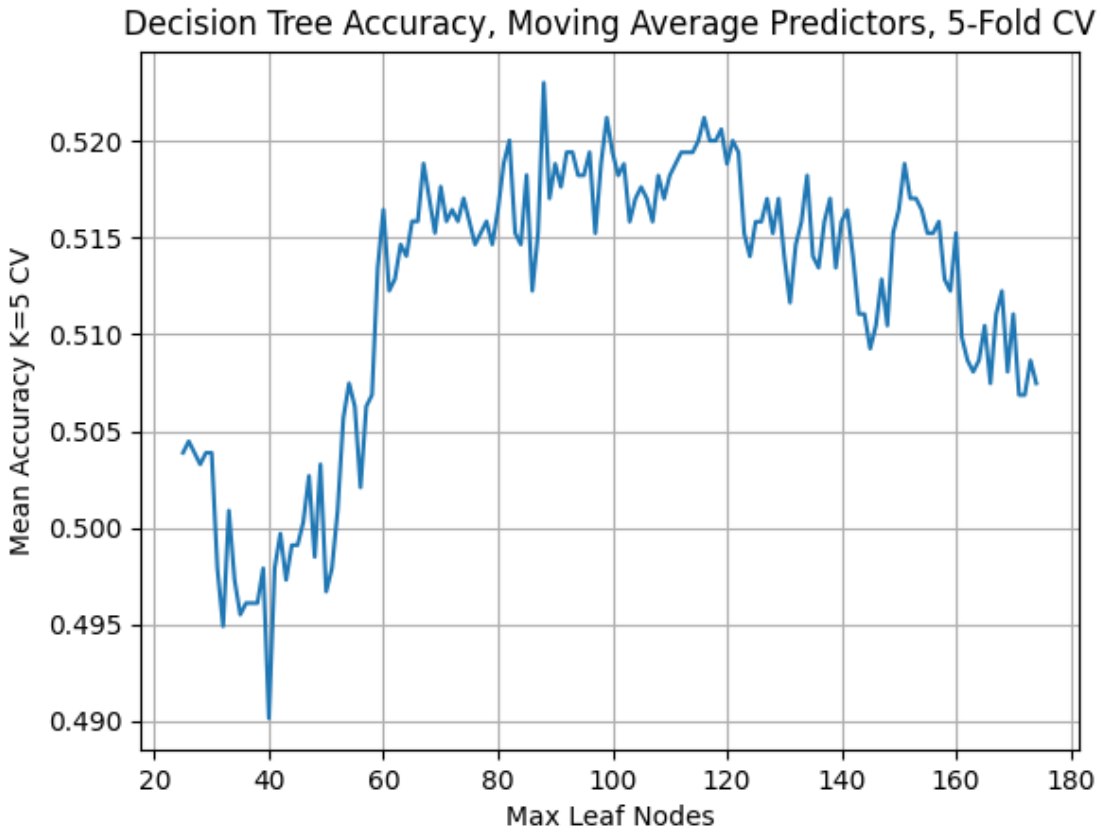


Figure 11: A Decision Tree classifier is used on the data using the number of reques and time since previous match as predictors. Accuracy is calculated using 5-fold cross validation and is plotted to view the most optimal number of maximum leaf nodes, which lies around 100, netting an accuracy of about 51.8%.

4.3 Previous Accuracy

The final match based feature we look at is previous rated accuracy difference. Simply this is the accuracy of the player, subtract the accuracy of the opponent, and shifted forward by one observation to show the previous accuracy difference.

Here, logistic regression provides 52.6% accuracy. There are some missing values due to accuracy ratings not being available for every game, however this missingness is missing completely at random.

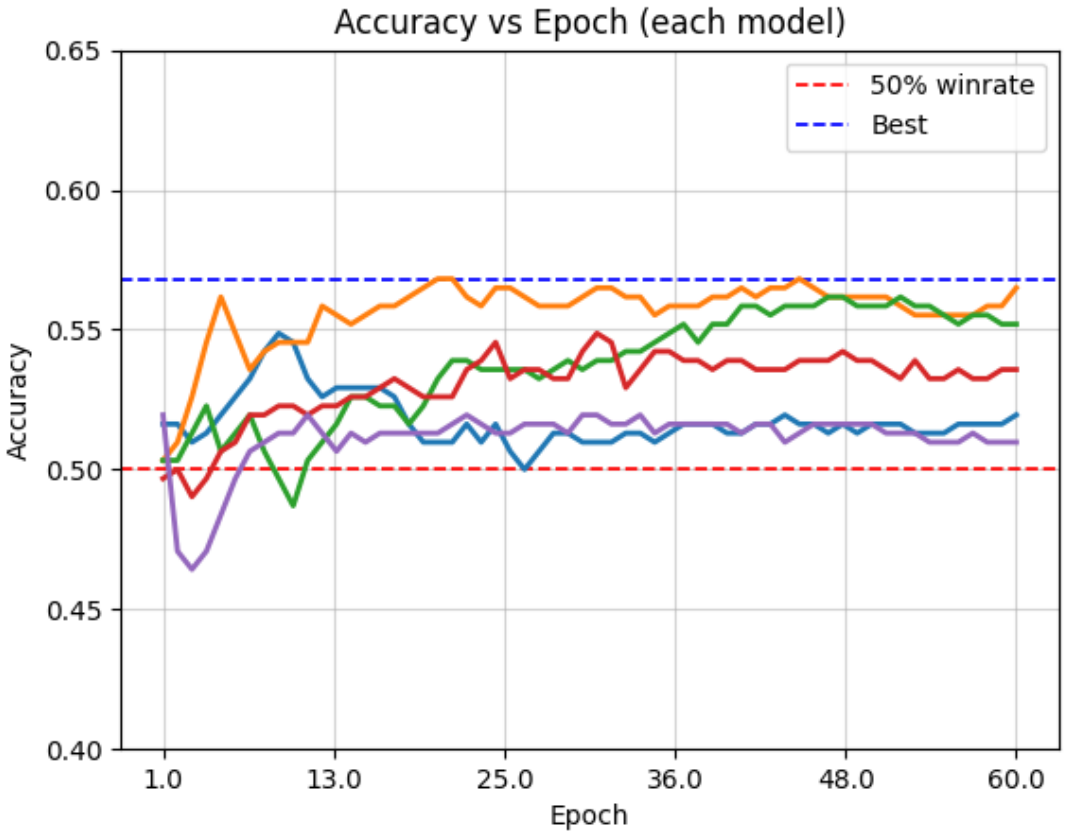
The decision tree approach for this predictor alone was not found to be statistically significant.

5 Combining Match Performance Predictive Features

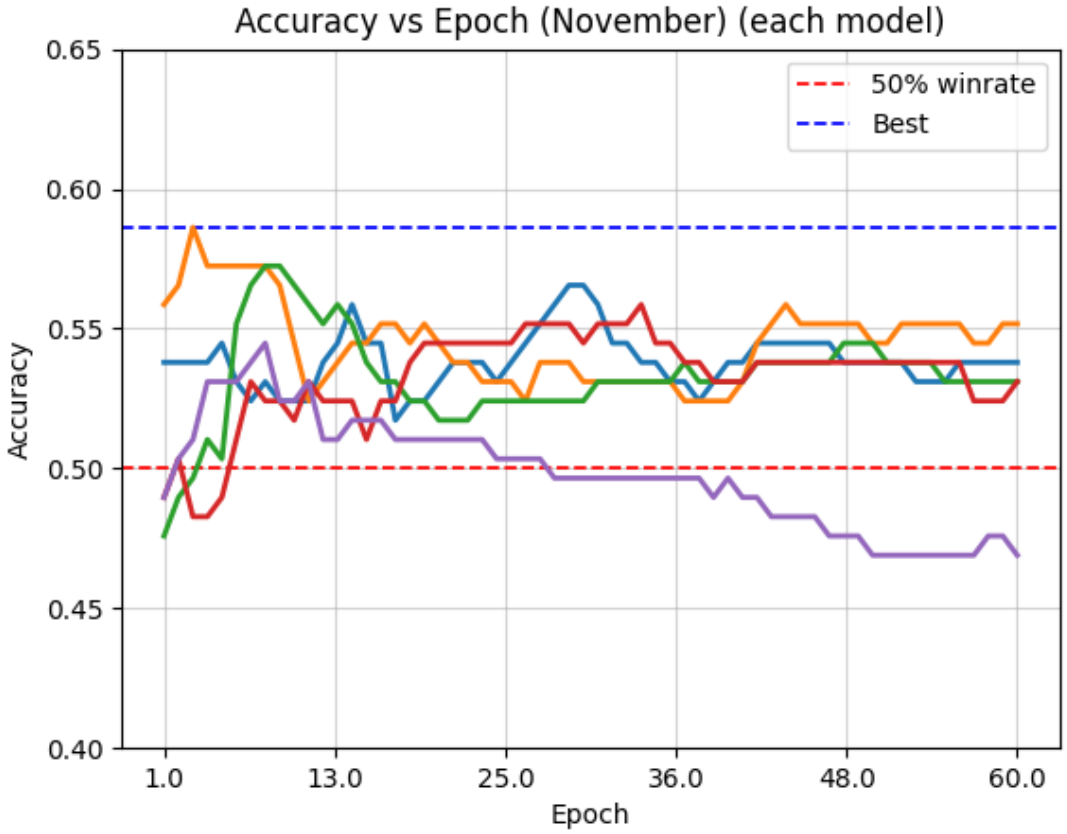
5.1 Neural Network

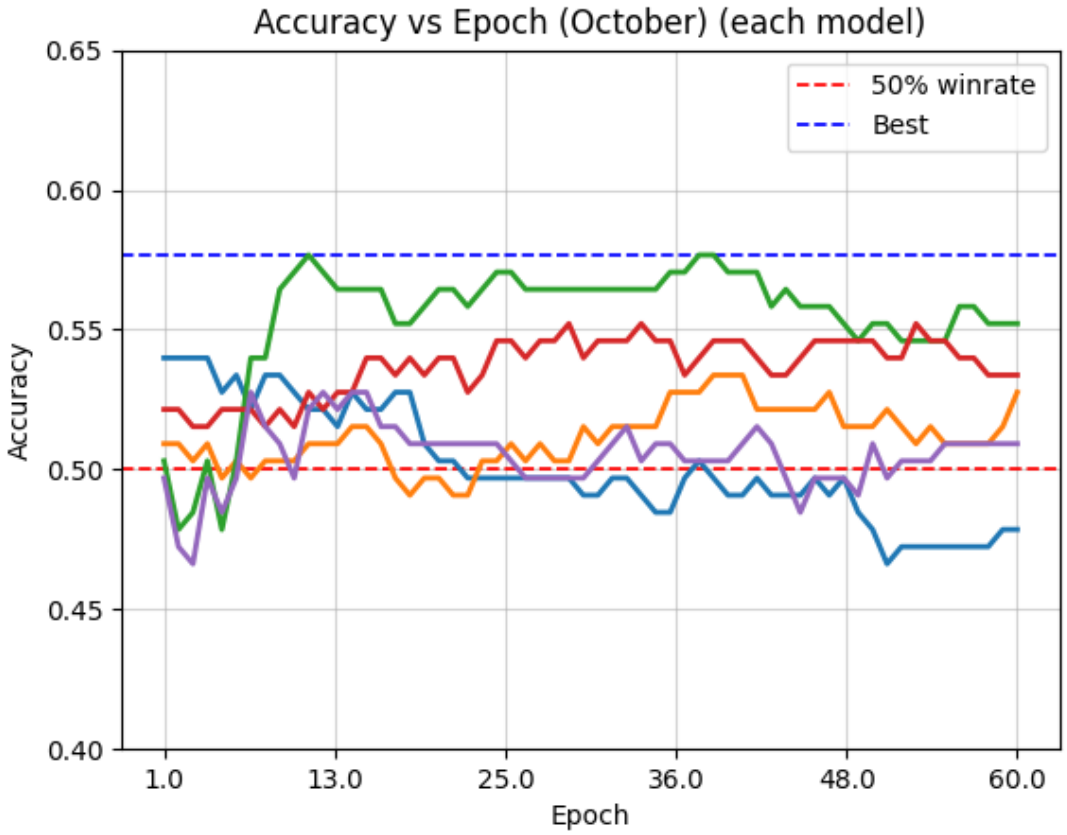
In this section we take each and every of the previous match based predictors and allow a neural network to find patterns in the data. We hypothesize that a network of linear models will generalize well and find patterns amongst the separate match performance predictors.

The accumulation of all predictors across all games in Steinkamp's game history over November and October can be use to train models that predict with over 56% accuracy. This is significant considering the balance of the dataset and the core objective of online matchmaking - to produce 50% winrates. The results across 5 trained models on validation data are show in Figure 5.1.



Similar models were trained but with a less generalized dataset, by selecting only games played in a particular month. These can potentially show slight improvements over the models trained to generalize over both months. Validation accuracy while training shown in Figure 5.1.


Accuracy vs Epoch (November) (each model)
50% winrate
Best
Accuracy
Epoch
0.65
0.60
0.55
0.50
0.45
0.40
1.0
13.0
25.0
36.0
48.0
60.0


Accuracy vs Epoch (October) (each model)
50% winrate
Best
Accuracy
Epoch
0.65
0.60
0.55
0.50
0.45
0.40
1.0
13.0
25.0
36.0
48.0
60.0

6 Codebase

This is text about the codebase

6.1 Archives Manager

This Python package interacts with Chess.com's API to fetch and analyze chess game data. It sets up directories for JSON storage and initializes a requests session with specific user-agent headers. The script can retrieve monthly archive lists, handling HTTP responses and raising an ArchiveRetrievalError for retrieval issues.

The package stores and retrieves data in JSON, reducing repeated API calls and enabling offline access. It allows filtering and analyzing games by date, Elo rating, and game result, and includes a function to correct Elo ratings. Users can extract specific information from game data, such as opponent's username, game accuracy, and outcome. Custom filters for archived games based on various criteria are supported.

Additionally, the package simplifies game data for quick overviews or integration into larger applications, focusing on key details like URL, end time, date, ratings, and results. This automates data retrieval from Chess.com and provides tools for game analysis and player performance assessment.

6.2 Board Feature Extractor

The board_feature_extractor.py module can analyze a chess board, counting each player's pawns and other pieces, identifying moved non-pawn pieces, open files without pawns, and passed pawns that have unblocked paths to promotion. It assesses pawn structure, piece activity, and promotion potential, key elements in chess strategy.

6.3 Game State Classifier

The game_state_classifier.py module classifies chess game states into opening, middle game, or endgame using the chess library [4]. It extracts features like move count, piece counts, open files, and passed pawns using extract_predictors_from_board. The module integrates K-Nearest Neighbors (KNN) and Decision Tree Classifier models, loaded from 'game_state_classifier_knn.joblib' and 'game_state_classifier_bst.joblib'. It predicts the game state using predict_knn and predict_bst, which can handle single or multiple board states in dictionary or DataFrame formats. An enumeration, GAME_STATE, defines the possible states and includes methods for equality comparison. The module checks for model file existence, loads the models, and uses them to predict the game state based on extracted features.

6.4 Pawn Structure Analysis

The pawn_structure_analysis.py module in uses the chess Python library to analyze pawn structures on a chess board. It counts the number of pawns for each color, calculates the distribution and advancement of pawns, highlighting the aggressiveness and spatial control of the structure. It identifies isolated and doubled pawns. The module also detects blockades and measures pawn tension, where pawns can capture each other, and counts passed pawns. It evaluates open, semi-open, and closed files, affecting rook activity and attack strategies. This analysis can update with each move, providing ongoing insights throughout a game.

7 Conclusion

We conclude this report, and our findings, that although online matchmaking systems are designed to give even winrates, with a little bit of match history heuristics, we can predict match outcomes statistically beyond the 50%, and do data exploratory analysis along the way to see what features contribute to winrate the most, and when.

Although we were unable to incorporate play style components into the analysis, we provide methods for building datasets from the ground up on a player of interest, their game history, and each of their opponent's game history. Our codebase also has methods to extract board features, and the Game State Classifier we wrote can classify whether a board is opening, middle, or endgame with high accuracy, which lays the path for future work. Theoretically, moves can be analyzed on a per-board basis, whether it be openings or end game evaluations, per player, and profiles/embeddings can be made to cluster by play style.

Work was split between two groupmates on this project. Both of us contributed to each component of this project and report in some way or another.

Adam Cunningham:

- Integrating Chess.com API
- Building Datasets
- Exploratory Data Analysis
- Report Write Up

Ian Campbell:

- Building and Testing Linear Models
- Building and Enhancing Visuals
- Presentation Slides and General Organization

References

[1] G. D. Marzo and V. D. P. Servedio. "Quantifying the complexity and similarity of chess openings using online chess community data". In: *Scientific Reports* (2023). URL: https://www.nature.com/articles/s41598-023-31658-w.pdf.

[2] A. Champion. *Dissecting Stockfish Part 1: In-Depth Look at a Chess Engine*. Towards Data Science. 2021. URL: https://towardsdatascience.com/dissecting-stockfish-part-1-in-depth-look-at-a-chess-engine-7fddd1d83579.

[3] A. Champion. *Dissecting Stockfish Part 2: In-Depth Look at a Chess Engine*. Towards Data Science. 2021. URL: https://medium.com/towards-data-science/dissecting-stockfish-part-2-in-depth-look-at-a-chess-engine-2643cdc35c9a.

[4] N. Fiekas. *python-chess: Chess in Python*. 2023. URL: https://pypi.org/project/chess/.