13 December 2024

Modeling the Win Probability of Chess Games Using Chess Engines

All associated code can be found at the following link:
https://github.com/piercingecho/chess-win-probability-modeling/

Motivation

Win probability models are valued heavily in sports, where play by plays are the standard way of accumulating data on the game's state. With the growth of esports, there is substantial data that can be extracted during games using in-game variables, storing individual player inputs, and so forth. As the game progresses, we expect the win probability to oscillate until the game eventually reaches a conclusive state. Using a live win probability model, this research hopes to answer "At what point will a game go from uncertain to decided?"

To examine the possibility of extending win probability to esports, the following research builds a win probability model using chess games. There is substantial existing data in chess that is readily accessible, and because chess is turn-based rather than played in continuous time, we have a simpler method of splitting our time steps for our model. Finally, substantial progress has been made in analyzing the strength of a chess board, utilizing chess engines. For these reasons, chess seems the most viable starting point for this research.

Exploratory Data Analysis

We are using a Kaggle dataset with Portable Game Notation (PGN) of various chess games, a standard way of representing chess moves. Each move is represented by a 2-5 character string, separated by a space. We also have access to the elo of both players and stockfish ratings of each chess move, which will be expanded upon later.

We begin by describing our data and the variables that are available to us. We are predicting the end result of a chess game. Each game results in white winning, black winning, or a draw, which we represent with a multinomial categorical variable.

For this paper, we will refer to a "turn" as a sequence where white moves, then black moves. We will refer to a "move" as a moment where either white or black move. These both indicate how far the game has progressed. Figure 1 shows the distribution of the number of turns in a game, which is right-skewed with a mean of 41 (and an associated mean of 82 moves).

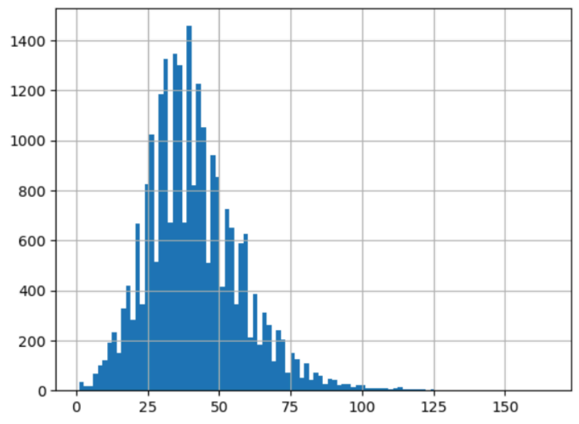

Figure 1: Distribution of Number of Turns

Elo is an extremely common way to rank players, and was designed with chess in mind. Players gain elo for winning matches and lose elo for losing matches, so after numerous games, it becomes a good estimate for a player's skill. One note on the future scope of this project is that elo is designed for individual actors in a 1v1 setting. It does not consider teams substituting members, or the elo of each player in a team, and so in team games such as Overwatch one may want to use a different metric.

Furthermore, for each game we have the moves made from both white and black in PGN. We utilize the scores given from the Stockfish chess engine as a feature extraction tool. In other words, we ask Stockfish to evaluate the strength of the board at every point in the game. Stockfish (and many other chess engines) rate a board's position in centipawns, where 100 points is equivalent to one pawn's value. This board rating takes both players' material (number of pieces) and arrangement of pieces (how much of the center do they control?) into account.

Stockfish takes time to calculate the score for each board state, and in moments where this calculation would take too much computational time [more than one second on one CPU core], the move's score is NA. We clean these values by replacing NA with the stockfish score of the previous move.

We take three main aggregate values from the stockfish scores. We first take the stockfish score at the move we have most recent access to, which tells us by how many centipawns white is winning at that point in the game. If black is winning, this value will be negative. We use two other aggregate values in our search, which aim to average the relative strength of the moves white and black play. From our stockfish scores, we build a new feature using the 'stockfish delta,' which takes the current board's rating and subtracts the board rating from one move prior. The stockfish delta allows us to see how much stronger the board is by one player's move. We take the average of all these deltas to build an "average delta" for white and black. This is to show by how much the board state is influenced by white's good moves, versus black's bad moves..

Building a Model for Any-Move Win Probability

We now expand our model to handle a chess game at any point in the match. Our current dataset includes one row for each entire game. For the remainder of this paper, we will create and use a new dataset that will contain one row for each non-checkmating move, per game (e.g. if there was a single game that ended on move 30, that would be converted into 29 rows. The fifteenth row will know the game's data from move 1-15). The full dataset from 25,000 games contains 1,316,297 rows.

For each of these rows in the new dataset, we use the aggregate stockfish values, the elo of black and white, the difference in their elo, and the current move number.

Before going further into model building, we construct a correlation heatmap with our new dataset to check if there is multicollinearity in our data (Figure 2). We see heavy correlation between black's elo and white's elo, which suggests that most games are occurring with players of similar elo. This makes sense in our data, because online matchmaking tends to place players with similar elo against each other for closer matches. Further, both black's elo and white's elo are correlated with the difference in their elo, which is consistent with the fact that the elo difference is a linear combination of the two. For our model building, we will drop the row for black's elo, as it seems to be the least useful for us. We also note the correlation between white and black's average delta, but we will leave both variables in because of the relative lack of features. All other variables have relatively low correlation.

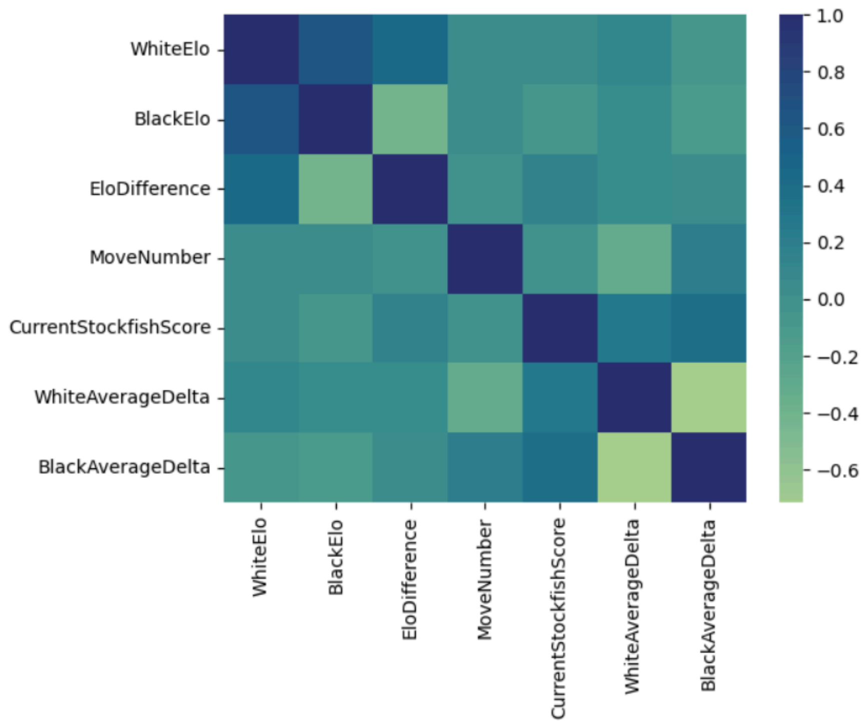

Figure 2: Correlation Heatmap of Data Features

We perform a train-test-validation split on the chess games before creating this new dataset. This is so that we won't have a single game's data in both the training and test data (a form of data leakage). We also stratify our train and test sets on the length of each game (using 3 bins) and game's outcome. The final split has 80% of the games in our training set, and 10% of our data in each of the remaining datasets. We use 80% of the games in our training set in order to ensure we have sufficient data when working with non-parametric models, in particular K-Nearest Neighbors (K-NN).

We will compare Logistic Regression, K-NN, and XGBoost models on multiple metrics. The first is **accuracy**: we consider a model accurate for a specific row if its observed outcome matches our model's most probable outcome. The second is **OvR ROC AUC (AUC)**. The ROC curve, or Receiving Operating Characteristic, is a graphical representation of the tradeoff between precision and recall for a binary classification model. In order to apply this to multinomial logistic regression, we first create three One Versus the Rest (OvR) ROC curves, for white VS not white, black VS not black, and draw VS not draw. The **Area Under the Curve** is the area under a single ROC curve, which ranges from 0.5 to 1. We can take the ROC of all three OvR ROC curves and average them for a final AUC score. This aggregates all of the different visual representations of model strength into a single quantitative value, which simplifies finding our best model.

We also consider variable interactions. In particular, we experiment with the move number interacting with the stockfish score, white average delta, and black average delta. We believed this might be influential because the later we observe a game, the more likely it has a definitive outcome.

Model Results

Our best model was a **Logistic Regression** model with lbfgs solver and 100 max iterations. The interaction terms . It has an accuracy of 66% and an AUC score of 0.84.

Metric	Logistic Regression	K-NN (19-NN)	XGBoost
Accuracy	**0.667**	0.626	0.659
ROC-OvR AUC	**0.84**	0.80	0.83

Table 4: Comparison of models. The best model for each criteria is in bold.

XGBoost and K-NN perform slightly worse, as shown in Table 4. Because XGBoost has a history of performing well in other WP models, this was surprising. It might be less effective in this chess dataset because of the lack of complex linear relationships among hundreds of variables: our feature extraction method of the chess engine gives a single value for each move. This limitation in our data likely affects the strength of the logistic regression and K-NN models, as well.

Moving forward with the logistic regression model, its accuracy of 66% is significantly stronger than randomly guessing (~33%), or always guessing that white will win (~40%). Figure 5 below shows the importance of each feature of our model. Observe that the **stockfish score** and interaction between **move number and white average delta** are the most important, followed by the **difference in elo** between white and black. This ordering makes sense intuitively because we would expect a board with a significant advantage for one player to supercede any difference in ranking before the match began. It's understandable that the move number interactions are also significant contributors, as it allows the scores to increasingly affect WP as we progress in a match (and likewise, for difference in elo to affect the WP more in earlier moves).

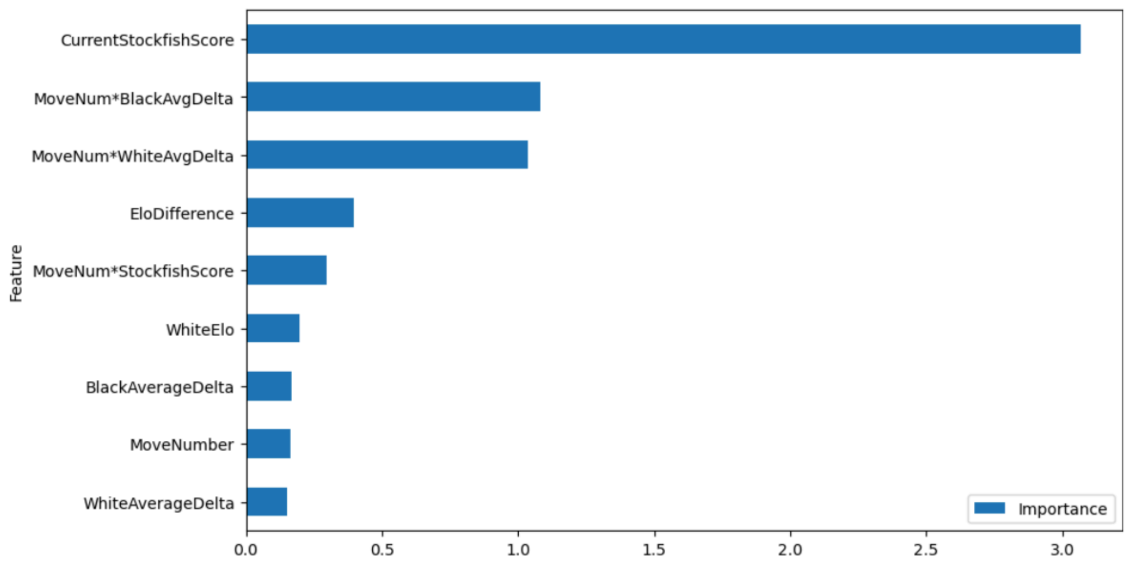

Figure 5: Feature Importance of Logistic Model.

The inaccuracies with the model could be explained through the nature of our data. First, because white and black elo are fairly correlated, a significant part of our data will have similar elos for them, and hence be less useful for our prediction. Second, win probability models intuitively work with incomplete data in the early game. Unless one side gains an overwhelming initial board state, there is a decent chance for their advantage to be reversed over the course of a game.

Finally, we must consider draws as a pain point of this research. Unlike many other esports, chess has an in-game method to create a draw, where neither side performs a capture in 50 moves, a stalemate occurs, or the same board state occurs thrice with the same player to move. Draws can also occur prematurely in a game, if both teams decide to draw. It is impossible to tell from our data whether a draw occurred legitimately or if, for example, the player with an advantage had to leave a match prematurely and agreed to a draw. Because we cannot clean this part of the data, we should not expect our model to be perfectly accurate.

Figure 6 addresses the concern of accuracy in the early game. The figure displays the accuracy of our best model over all games at a specific move number; for instance, over all games that got to move 80, our model is 77% accurate at predicting the outcome from our knowledge at move 80. It is clear from this figure that the **model accuracy improves as a game progresses**. We calculate an error for this point estimate by bootstrapping our data with replacement. This error increases at higher move counts because of the incrementally less data we have for longer games. Above 150 moves, our error becomes exorbitantly high, which would not be useful for analysis.

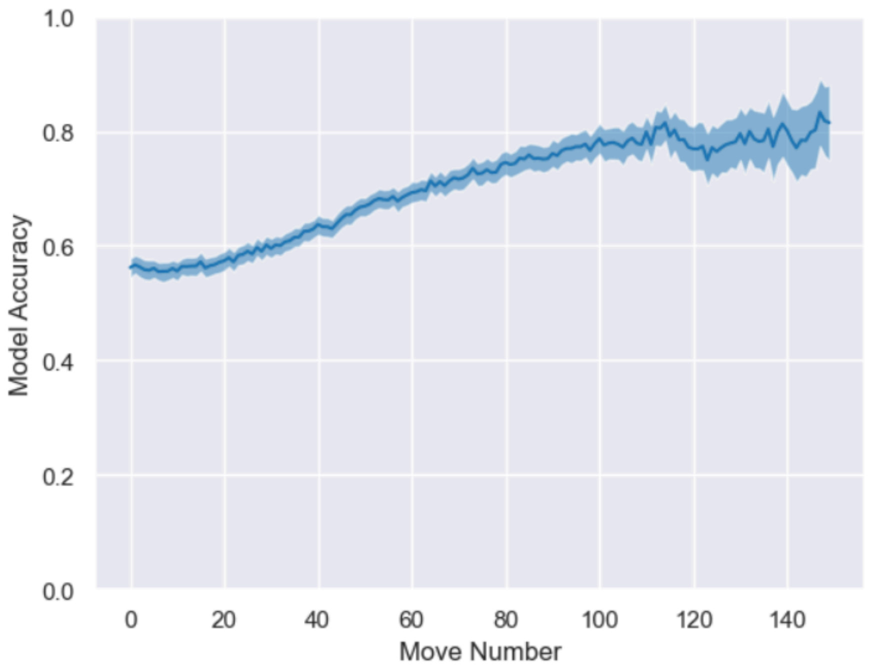

Figure 6: Accuracy of model at different possible points in the game.

Model Applications

With this model in mind, there are ways we can use this model to gain similar insights as what other sports have done. First, we can further scrutinize the results of a single game while or after it has happened, through a visualization such as Figure 7. Here, we have a game between white and black, with Elo rankings of 2556 and 2329, respectively. Because white has a significant advantage, we see this reflected in their initial win probability being around 55%. This win probability spikes at move 55, due to a move that guaranteed checkmate.

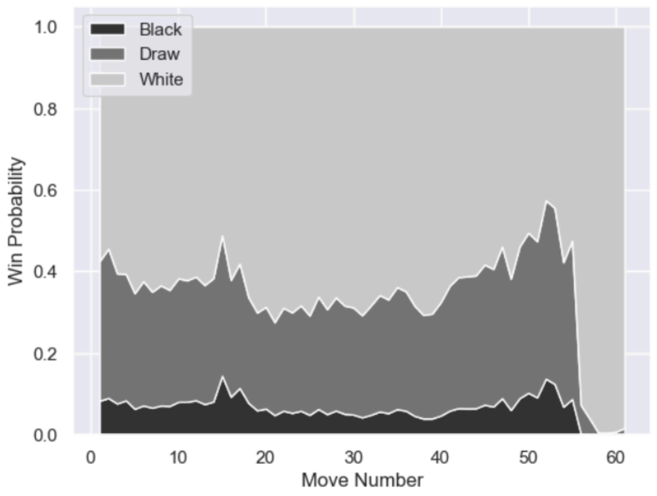

Figure 7: Win Probability Changing for a Single Game

Tools currently exist to judge chess games and individual moves (similarly using chess engines), but **our WP model grants additional insight by considering player elo**. When recommending a player to change their playstyle, it is often beneficial to meet them where they are as a player and their current strengths and weaknesses, rather than prescribing an optimal game plan that takes significant adjustment.

For example, our model could improve recommendations by recognizing how a player's opponents tend to punish mistakes. If lower elo players are more optimized at punishing mistakes at late game rather than near the beginning, it would make sense for a player to improve their endgame first, even if a chess engine deems their early game to be weaker than their late game. By considering how players actually play the game, instead of how they might play against a perfect opponent, our model can help support an iterative improvement regiment, rather than seeing their moves as a set of imperfections.

Future Steps

For this model in particular, one of the main drawbacks is that the chess engine we use is specialized to win chess games, rather than predict the actions of two independent agents. So, it is likely that another chess engine that specializes in WP and predicting two players' decisions would be more effective than Stockfish. In general, a future step would be to compare multiple chess engine scores, to find if one happens to be more effective than the others. In that same light, our data is severely limited by having the chess board extracted to a single numerical feature. We could glean more insights (or potentially have a stronger XGBoost model) given more features which might have useful interactions.

This WP model proves the usefulness of integrating this type of model into esports generally. It is possible in highly popular esports such as League of Legends and Overwatch to use computer vision or game modifications to directly extract game variables (such as each player's position, the state of objective, and what resources are available to each team). This level of precision allows the WP model to perhaps extract game data in real-time to usefully improve commentary and analysis. We hope this lays the groundwork for gleaning insights into those esports with much less-defined optimizations in their play compared to chess.