Predictive Model for Chess Game Results

Nicholas Swetlin
University of California, San Diego
San Diego, USA
nswetlin@ucsd.edu

Gallant (Chia-Lun) Tsao
University of California, San Diego
San Diego, USA
ctsao@ucsd.edu

Figure 1: A typical chess board

1 Introduction

1.1 Background

Chess is a game where two players, playing as the white and black side respectively (with the same chess pieces), try and win the game by forcing the opponent to forfeit the game, or by capturing the opponent's king piece. The game of chess can be traced back 1500 years to India, where the prototype was called chaturanga.

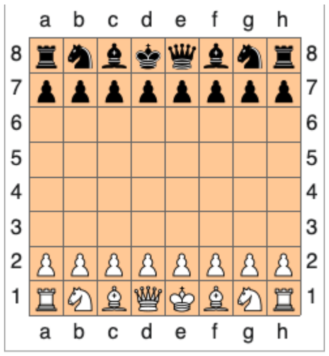

Figure 2: Chaturanga

Throughout its evolution, countless famous people have experienced this strategic game, and many strategies have evolved throughout time, since each chess piece has its own moving pattern. In the past decade, the emergence of artificial intelligence dramatically changed the format of chess as its computational speed is much faster than that of a human being, but until now people still enjoy playing chess with others. Moreover, the development of computers, and later the internet, have made playing chess much more accessible than before: Simply open chess.com or lichess.org and find someone to compete.

1.2 Motivation

From amateur players to professional chess grandmasters, there are often many factors that can influence whether the player is able to win: From the opening strategy, mid-game tactics, endgame strategies, and even to opponents' mistakes. Chess is a game of deep strategy and skill, yet predicting the outcome of a match based on several factors, including players' characteristics, past performance, and in-game decisions remains a complex challenge. With the rise of machine learning, we now have the ability to analyze vast amounts of chess game data to uncover patterns that influence a player's likelihood of winning. This project aims to leverage machine learning techniques to predict the outcome of a chess game, providing valuable insights for players looking to improve their strategies, coaches analyzing game trends, and even chess engines enhancing their decision-making processes. By exploring factors such as ELO ratings, opening moves, time control, and positional advantages, we seek to develop a model that accurately forecasts game results, offering a data-driven approach to understanding competitive chess dynamics.

1.3 Previous Literature

There have been numerous efforts in predictive tasks for chess games, including those which try to analyze whether a player would win/lose the game based on their move, such as open-source engines including Stockfish and RubiChess. There is also some work on data-driven predictive models for chess games:

Chess Game Result Prediction System by Zheyuan Fan, Yuming Kuang, Xiaolin Lin Fan et al.'s work focuses on predicting chess game result based on Hidden Markov Models, which are models which have a hidden Markov Process behind them. Their result assumed the underlying Markov Model to be a Bernoulli, as they considered a multinomial model to be too complex as it requires extra hyperparameters. Moreover, they modeled the noise factor as the difference in player performance between games. Their HMM model produced an accuracy of 55.64% accuracy, as their model did not perform too well on games that were on the border of draws (or games that lasted longer rounds in general).

2 Exploratory Data Analysis

2.1 Dataset Description

Our dataset (stored in chess.csv) comes from Kaggle, consisting of 6.3 million chess games encompassing approximately the last decade from the official Lichess website. The data consists of some information about the games that people have played on Lichess, including their side (white/black), result, opening, chess moves, type of tournament, sequence of chess moves written in algebraic notation (AN), etc.

The data generating process involves players with registered accounts competing on Lichess, in which case the website will record all available information that it can get, including the players' moves, their ID, the time of play, etc. All of the data that Lichess records will be stored via Portable Game Notation (PGN) format. Then the data is converted into csv format and stored on Kaggle later.

For our analysis, we only selected a random 1,000,000 rows (with a set seed) of chess games from the raw dataframe, as we consider it to be a sufficient representation of the original dataset, while boosting the training efficiency of our algorithms by a lot.

2.2 Data Cleaning Process

When cleaning the data, the column containing the algebraic notation for the chess moves the player makes contains two different types of notation: The first one contains steps of the format similar to **"1. e4 d6 2. e6 d4, etc."**, which is the standard way that data is recorded in Lichess. The second one contains steps of the format similar to **"1. [%eval -0.19] 1... c5 [%eval 0.2] etc."**, which contain the algebraic notation from a chess engine such as Stockfish, for evaluation on performance. Therefore we have to unify the algebraic notation by removing the bracelets, the white spaces, and the extra periods from the ANs from a chess engine to that from Lichess. To extract the actual steps that players make during the game, we implemented a function using regex to extract the useful information within each step that a player takes.

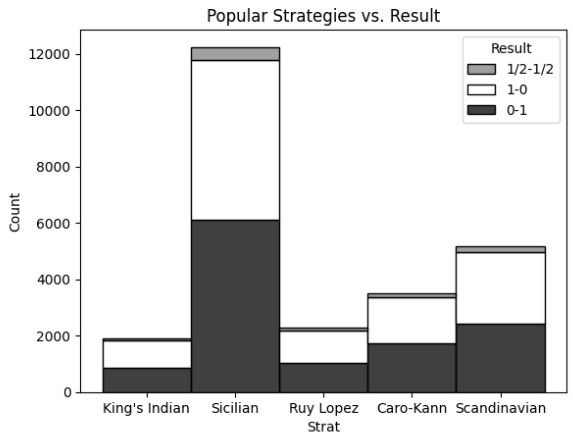

Figure 3: Distribution of Wins/Losses/Draws for Popular Strategies

The 'UTCYear' column contains strings encoded in the format "YYYY.MM.DD", therefore we converted them to datetime objects for us to do further analysis that are related to time series. The same follows with the 'UTCTime' column except that it is written in the format 'hh:mm:ss'.

For the event column, we decided that we would treat tournaments in the same way as those events which are not tournaments. For example, in our cleaned data, "Blitz Tournament" and "Blitz" mode fall under then same category that will be used in our model later. Moreover, there are both leading and trailing white spaces in the **event** column, which stripped to get rid of the spaces before and after the word.

2.3 Findings for EDA

When exploring the dataset, here are several findings that our group thought it was interesting to report.

Firstly, after some basic analysis, we found out that the games in our dataset consists of games that are from mid to high ELO, that means the players were somewhat conscious of the idea of strategies within chess. We were curious in whether the results are influenced by the opening strategies. That is, whether there is some 'optimal' strategy that players would use for their games. Since there a lot of variations in strategy for chess (in fact about 2700 for our dataset including variations of the same strategy), we encoded different variations of the same strategy into the same category, and only included the 5 most popular strategies in chess from our dataset, which are the Sicilian, Scandinavian, Caro-Kann, Ruy Lopez, and the King's Indian. Based on the diagram above, it seems like the win rates for white/black via different strategies do not vary significantly, which makes sense as in high-ELO match-ups, prioritizing on opponents' mistakes is often more important than focusing on which opening that the player is using. For instance,

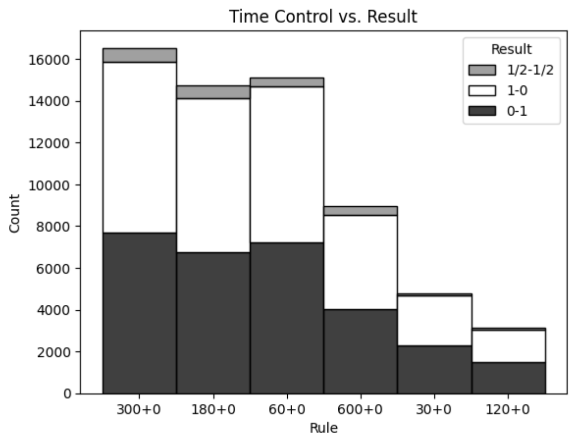

Figure 4: Distribution of Wins/Losses for Various Time Controls

Magnus Carlsen, one of the best players in chess currently, prefers the Ruiz Lopez opening much more than others, but he still has a very high win rate on either side.

Another aspect that we are interested was whether the time constraints would affect whether the player would win/lose the game.

From Figure 4, it seems like again, the results of the chess game are not affected by time constraints, even though some of the games do not provide additional time when players make a step in the game.

From the work we did in the EDA analysis above, it seems like forming a model via the features provided will be far too simple, therefore more feature engineering will be needed to increase the complexity of the predicted model, which will be shown below.

3 Methods

In the section below we include some of the models that we incorporated in predicting the chess game outcome. Before we begin our analysis, we first discuss the relevant feature engineering process to generate the features in our preceding models.

From the cleaned AN column, we would like to extract each step that the white/black sided player takes during the game. Therefore we created a function which, given a number, extracts the number of rounds for all the games within the dataset, and padded zeros for the games that ended early, where each feature represents one AN notation (i.e. an action that a player takes during the game). One problem that we encountered during the engineering process was the excess amounts of features: AN representations not only capture then locations that the chess pieces that are going to, but also capture the relevant actions within chess. For instance, pawn capturing, pawn promotion, king-side castle, and queen-side castle are all represented in different ways.

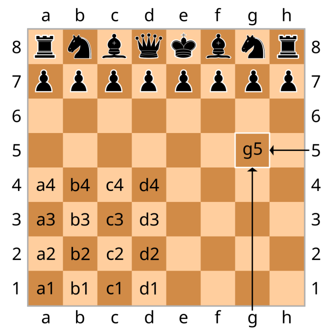

Figure 5: Algebraic Notation

We thought that the vast amounts of actions that should supposedly be encoded as categories will lead to overfitting in our model, therefore we take a much more simple approach for our feature engineering process for the AN column:

For each move that a player takes in the game (as a feature), we encode the move into a 3-tuple: The first entry contains the square that the chess piece is moved to, the second entry contains the chess piece that was moved to the square, and the third entry contains whether a capture happened on the chess square. The way that the squares locations are encoded is via algebraic notation, shown below. For castling (king or queen sided), since the king is switching its square with the rook on the king/queen side, we only encode the square of the king, as we considered it to be more important in determining the later dynamics of the game (especially affecting checkmate turns).

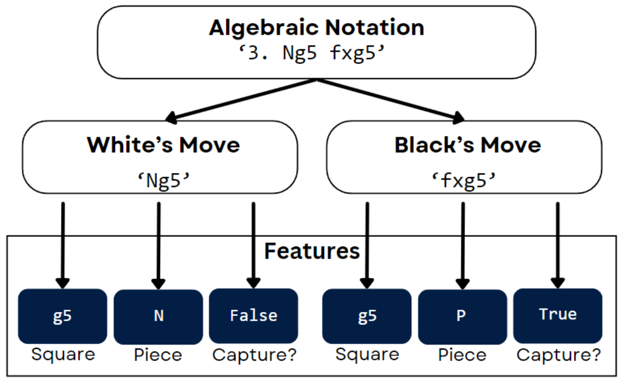

Figure 6: Feature Pipeline

Moreover, we have created a custom hyperparameter called **sight**, which represents the number of rounds in the game that we are using to predict the result. Note that this hyperparameter is crucial in the analysis of our models, because our model only takes in games that have length (i.e. number of rounds) that are AT LEAST the length of sight.
We decided to engineer our features in this manner, as we thought gaining control of the center of the chess board was an important aspect of winning the entire game, as it expands the options of

the chess player by a lot more and limits the number of moves the opponent can conduct.

3.1 Method 1: Soft-Support Vector Classifier

To start with, we began with a simpler model, the Soft-SVM, on training the data. Below is a table of the pros and cons of this method for our dataset:

Pros	Cons
Simple model to implement as a first consideration.	The decision boundary is linear in the feature space; may not be able to capture the nonlinearities that are present in our data.
Can prevent overfitting on the dataset by setting the regularization parameter C, and also allows some margin of error in the case where the data is not linearly seperable.	SVCs are difficult to train on large sets, which can present in our data as in consists of data on the millions scale.

Table 1: Soft SVC – Pros and Cons

We considered the soft SVC, as it contained measures that could reduce the effect of overfitting, considering that we are dealing with high-dimensional data. Moreover, it is acting as a simpler model that assumes that the underlying data can be classified by a linear decision boundary (at least somewhat). The optimal parameters that we got for the soft SVC were C = 10, kernel = 'rbf'. However, we did experience the fact that SVC struggles to train on large datasets, and therefore when training the model, we narrowed the training set down to only 100,000 points. Shown below are the confusion matrix and the evaluation metrics for different sights for the data:

3.2 Method 2: Histogram Gradient Boosting Classification

We considered using histogram gradient-boosting classification as a complex model that deals with large amounts of data quickly. Below is a table of pros and cons for using this method for our specific use case:

Pros	Cons
Efficient processing for large number of Chess Games.	Method known to be prone to overfitting if not tuned properly.
Can use categorical rather than numerical splits for types of pieces used each turn (P, N, B, R, Q, K).	Histogram estimators are subject to 'Curse of Dimensionality' for high-dimensional data.
Decision tree process ascribes order/chronology to features, which models how a game of chess unfolds.	

Table 2: HistGradBoost – Pros and Cons

We liked the idea of HistGradBoost because its use of decision trees could imply a **chronological order** about our features. This is important because performing chess moves are chronologically dependent. For example, if on white's move they captured a piece with a pawn (perhaps notated as exf5), the state of the game in algebraic notation is entirely dependent on what piece black had placed on the f5 square previously; the outcome of the game is entirely different if black's pawn were taken that turn versus if black's queen were taken that turn.

In the ideal case, HistGradBoost would form splits based on this chronological information, starting with a root question about the first move, and then the second move, and so on. This parallels human thinking of the game of chess, where the current board state is the ultimate driver of what the result of the game is – it just so happens that we must recreate this current board state using algebraic notation.

For parameterization, we ended up setting a small learning rate of 0.1, maximum number of iterations = 25, max_depth = 4, and max_leaf_nodes = 20, all to prevent overfitting. From there, the model was free to train.

4 Results

Below are results for SVM for sight values of 10, 20, 30, 40:

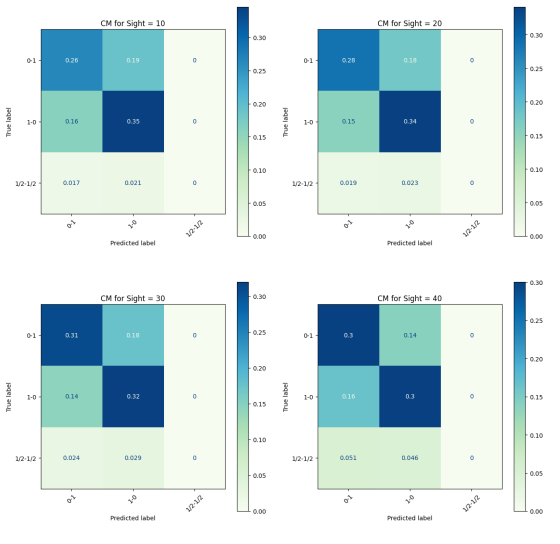

Figure 7: SVC Confusion Matrices for Different Values of Sight

	10	20	30	40
ACCURACY	0.6101	0.6253	0.6277	0.5990
PRECISION	0.4059	0.4166	0.4191	0.3997
RECALL	0.4212	0.4343	0.4424	0.4428
F1_SCORE	0.4130	0.4250	0.4299	0.4198

Table 3: Evaluation Metrics for SVC

Below are results for HistGradBoost for sight values of 10, 20, 30, 40:

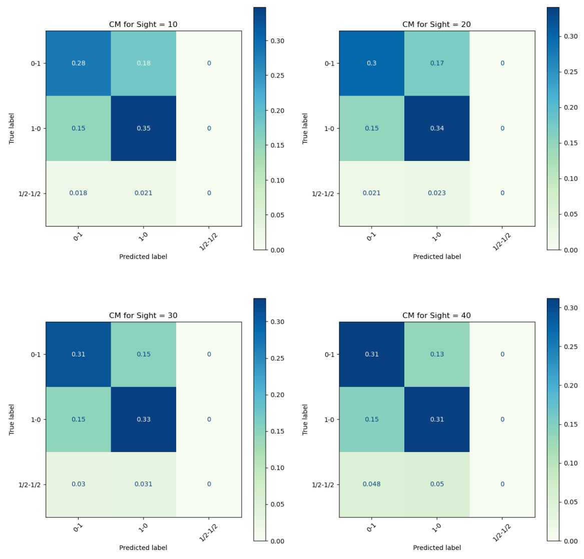

Figure 8: HistGradBoost Confusion Matrices For Different Values of Sight

	10	20	30	40
ACCURACY	0.6306	0.6385	0.6420	0.6221
PRECISION	0.4203	0.4256	0.4279	0.4147
RECALL	0.4364	0.4448	0.4556	0.4599
F1_SCORE	0.4277	0.4348	0.4413	0.4361

Table 4: Evaluation Metrics for HistGradBoost Classifier

We see that for sight = 10 and sight = 20, there is slight bias in predicting white to win. The most balanced models received were for greater values of sight, i.e. sight = 30 and sight = 40, with a sacrifice in predictive accuracy.

5 Discussion

5.1 General Findings

We saw that as sight increased, predictions were made with lower accuracy, yet, with less bias. Despite both of our proposed models being only marginally better than chance, the change in model performance across sight provides us with great insights for how predictable the game of chess is at different stages of the game.

The Opening

For example, bias to predict white introduced with sight = 10 and sight = 20 reflects a common imbalance in chess: white has the first move, gaining an innate slight advantage in the opening. As such, any reasonable predictor, human or machine, given just information about the first 10 or 20 rounds, will predict white to win more often; this is validated by the behavior of evaluation machines in top-level chess tournaments, which typically ascribe a slight positive advantage to white in the opening (+0.1 to +1.0). Openings are also quite rehearsed by players; it is not uncommon to see players memorize anywhere from five to twenty rounds of "best play" before getting into the middlegame, where more improvisational and strategic play manifests. Considering that the opening is typically contained within the first 25 rounds, the similar behavior for sight = 10 and sight = 20 is well-explained. It makes sense that if a model sees only the first 10 or 20 rounds of a game, the predicted outcome of the game will not have changed because the opening is still occurring.

The Middlegame

As players reach the middlegame, typically around round 30, play becomes more dynamic, and players each have the chance to inject their own character into the game, leaving the more "scripted" opening behind. With this new turbulence in the game, it is much more difficult for white to maintain their advantage. Therefore, the middlegame is typically more balanced in evaluation than the opening, and predicted outcome of the game is much more balanced as a result, as shown by the confusion matrix for sight = 30. Considering that the "newer" moves of rounds 20-30 provide important information to human players about the unique circumstances of the game and the personality/play-style of each player, rounds 20-30 should similarly grant our model more information about the unique outcome of the game, resulting in a more even predictive surface with less bias.

The Endgame

As the number of pieces on the board grow scarce, we enter the endgame, one of the most turbulent phases for predicting outcome of the game. Any mistake that a player makes in this phase could entirely upset the fate of the game, as the fewer pieces on the board results in fewer resources to correct mistakes. It is also possible for longer openings (30+ rounds) to lead directly to endgames. Our model reflects this with sight = 40 – which still has the less biased predictions of sight = 30, with worse predictive quality due to game turbulence. Additionally, since training data are filtered to only include games with length greater than our sight parameter, we have fewer observations to base our predictions off of for sight = 40 than, say, sight = 10; this reduced amount of data could also describe the general decrease in performance witnessed with sight = 40.

5.2 Limitations

As always findings should be considered with a layer of skepticism. Our work was limited by a number of factors:

- **Online Chess Games:** Our 6 million data comes from LiChess, an online chess website. Online chess games are different than over the board chess games – there is less player pressure and there is a tendency to play shorter rather than longer games. Only chess players who play chess online have their game behavior accounted for by our model. All of these data limitations are further extended by our subsetting of data to 100K - 1M rows for faster model training.

- **Curse of Dimensionality:** our feature extraction from algebraic notation created many features. Specifically our number of features scale at a rate of $O(10 * \text{sight})$ for Soft-SVM and $O(6*\text{sight})$ for HistGradBoost. With a greater number of features, the efficacy of histograms as estimators decays significantly, as many histogram bins are empty. Additionally, all other models have trouble parsing significance of many features when the number of features is very large. These could significantly affect our model predictions depending on hyperparameters.

- **Data Filtering:** As sight increased, we excluded chess games with length less than sight, to simulate the true use case of this model: predict outcomes of *unfinished* games. This lack of data for greater values of sight could lead to a decrease in predictive quality with the introduction of bias or variance.

- **Lack of Replication:** Due to model size, we neglected training models a large number of times to create confidence intervals. As such, confidence in our model predictions comes with asterisks about the quality of our methods, and further replication is needed to verify the efficacy of our results.

6 Conclusion

One of our greatest struggles in this project was trying to find a suitable machine learning framework for dealing with chess data. Since we are conducting multi-class classification with both numerical and categorical features, this already eliminates some of the options that we were considering (such as decision trees and random forests). Moreover, understanding the results took a lot of time for us as the Hidden Markov Models given in the paper from Fan et al. were not trivial.

Comparing the models that we explained within our literature section, for both SVC and HistGradientBoostClassifier, our models hold higher accuracy than the model proposed by Fan et al. However, there is still a lot of room for improvement.

One improvement that we could make which will require further work is how the model tackles with draws. Draws are games that last much longer, and can be much more chaotic than a regular game when one player forces the other player to forfeit the game. For example, it could be that the two players enter an infinite loop of moving the same chess pieces, resulting in a forced draw, or it could be one player deciding to draw for some reason, while the other player also agrees.

Acknowledgments

Our group wants to thank professor Jingbo Shang for the excellent and interesting lectures that he gave for inspiration in some of the models we implemented.

References

[1] https://scikit-learn.org/stable/index.html
[2] Fang, Zheyuan, et al. “Chess Game Result Prediction System.” Chess Game Result Prediction System, Dec. 2013, cs229.stanford.edu/proj2013/FanKuangLin-ChessGameResultPredictionSystem.pdf.