Analyzing Positional Play in Chess using Machine Learning

Sameep Bagadia
sameepb@stanford.edu

Pranav Jindal
pranavj@stanford.edu

Rohit Mundra
rohitm92@stanford.edu

December 13, 2014

Abstract

With increases in computational power and better algorithms, chess engines are able to explore game trees of greater depths and thus have excelled at calculative play. However, since the growth of the move search space is exponential, even with increase in computational power chess engines perform only marginally better year-to-year. In this paper, we explore a novel technique for assessing board positions using machine learning techniques that can be used to supplement and improve current chess engines. We model chess board positions as networks of interacting pieces and use supervised machine learning techniques to analyze positions and predict outcomes in chess games.

1 INTRODUCTION

Chess has two broad approaches to game-play, tactical and positional. Tactical play is the approach of calculating maneuvers and employing tactics that take advantage of short-term opportunities, while positional play is dominated by long-term maneuvers for advantage and requires judgement more than calculations.

Current generation chess engines predominantly employ tactical play and thus outplay top human players given their much superior computational abilities. Engines do so by searching game trees of depths typically between 20 and 30 moves and calculating a large number of variations. However, human play is often a combination of both, tactical and positional approaches, since humans have some intuition about which board positions are intrinsically better than others.

In our project, we use machine learning to identify elements of positional play that can be incorporated in chess engines. We model chess board positions as networks of interacting pieces and predict game outcomes when the engine evaluates both sides to be of comparable strength. Our findings indicate that we can make such predictions with reasonable accuracy and thus, this technique can be augmented with current chess engines to improve their performance.

2 DATASET

In this section, we describe our dataset followed by the preprocessing steps that were required to structure the data.

2.1 Source and Structure of Data

We used board positions from 6200 games from an engine vs. engine tournament in 2006 [1]. This provided us with a total of 1,075,137 board positions in Portable Game Notation (PGN) where a board position is a snapshot of the game at a point in time. We parsed the PGN data to a more convenient Forsyth–Edwards Notation (FEN) for further processing.

2.2 Data Selection Criteria

Each board position also had an associated score calculated by the engine. A board score is a measure of the relative strength of white compared to black; thus, a positive value indicates that the chess engine's evaluation believes that white is in a stronger position while a negative value indicates the converse. If a board score is near zero but the game then goes on to end decisively, we hypothesize that aspects of positional play were unaccounted for by the chess engine. Thus, we consider only those board positions where the board scores are nearly zero but the game ends decisively. It is worth noting we restrict our focus to engine vs. engine games where we know that the successive moves are going to be tactically optimal. Thus, the board positions we considered are those whose outcome can be attributed to positional aspects with high confidence. The hypothesis can be seen in Table 1.

Table 1: Cause of Outcome

Outcome	**Engine Score**		
	≈ 0	$\gg 0$	$\ll 0$
White win	Positional	Tactical	Improbable
Black win	Positional	Improbable	Tactical
Draw	Variable	Improbable	Improbable

3 NETWORKS & FEATURES

To capture interactions between various pieces and locations on the board, we modeled the board as networks with board squares and pieces as nodes, and the interactions as edges with attributes. This approach is similar to that seen in [2]. For each board position, we create the following two networks:

1. **Support Network:** This is a network where each node represents a piece and each directed edge represents that the first piece can reach the second piece. If the edge is between pieces of the same color, then it is attributed to be a defense edge (since the first piece can defend the second piece). Conversely, the edge between pieces of different color is attributed to be an attack edge. Figure 1 demonstrates attack and defense edges on a chess board. The corresponding network for this board position can be seen in Figure 2a.

2. **Mobility Network:** This is a bipartite network from chess pieces to board squares. An edge from a piece to a location represents that the piece can reach that location on its next move. Mobility edges on a chess board can also be seen in Figure 1. The corresponding network can be seen in Figure 2b.

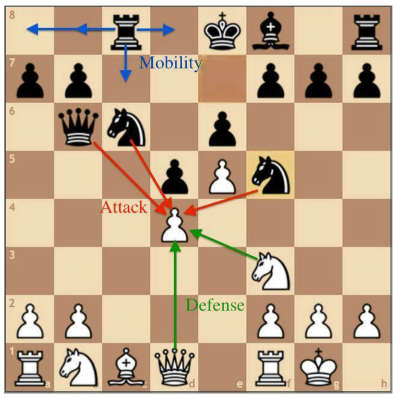
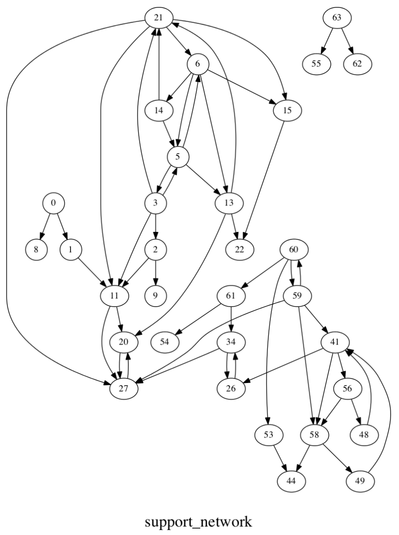


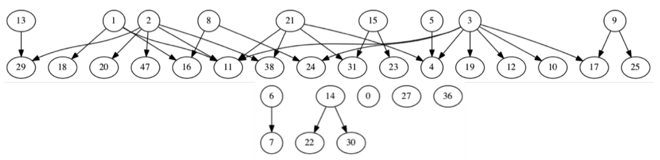
Figure 1: Attack, Defense, Mobility

After creating the aforementioned networks for each board, we extract features from these network representations. Some of the features are:

- Number of edges in the Mobility network: indicator of the mobility of various chess pieces on the board (termed as 'activity' in chess)
- Number of nodes in the Mobility network: captures the notion of a "space advantage" in chess
- Number of attack and defense edges: Each side will ideally want to have a fewer attack edges from the opponent and more support edges to its own pieces
- Number of weakly connected components: This measures how well the pieces coordinate with each other



(a) A Chess Board Support Network



(b) Mobility Network for White

Figure 2: Network Structures for Board Positions

- Mobility around the king: Captures the concept of king-safety in chess, it is desired to have low mobility of opponent pieces around your king

We also used some well recognized heuristics for positional features. Some of these can be seen in Figure 3 and are listed below:

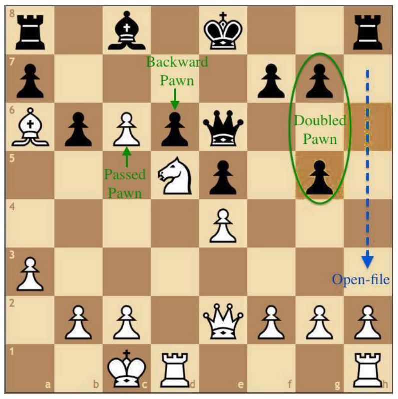


Figure 3: Pawn structures

- Isolated pawns: Pawns that cannot be supported by other pawns (indicates a weakness in the pawn structure)
- Doubled pawns: Multiple pawns in the same column, again indicative of weaknesses in the structure as they barricade movement

- Open files: A column on the chess board without any pawns of the same color. Considered an advantage in presence of rooks, since it allows for additional mobility
- Passed pawn: A pawn which cannot be attacked by any enemy pawn and has cross the 5^{th} rank on the chess board.

4 MODEL AND FEATURE SELECTION

In this section, we describe the models that we used for our classification problem and the approach taken for feature selection.

4.1 Models Used:

Considering the differences in discriminative and generative models, we used the following classifiers for our binary labelling task:

- Naive Bayes (Generative)
- Linear Discriminant Analysis (Generative)
- Logistic Regression (Discriminative)
- Support Vector Machines (Discriminative)
- Random Forests (Discriminative, Ensemble)

The performance of the different models can be seen in the Table 2 and have been discussed in greater detail in the next section.

4.2 Overview of Training, Testing and Feature Selection:

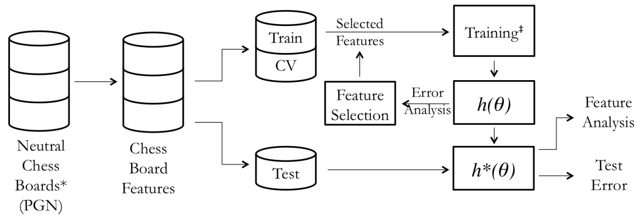


Figure 4: Overview of approach to feature and model selection

Our first step involved mapping the data in FEN to the set of features described earlier in addition to some others by constructing the chess game networks. We split this dataset into a training and cross-validation set (70%), and a test set (30%). For each of the models described above, we trained the model and optimized hyper-parameters using the training and cross-validation dataset to prevent bias-variance issues. After analyzing the feature importance using data visualization techniques, we removed poor predictors and noisy features. After this, we retrained the features and compared performance and repeated the feature selection process if necessary. At last, we used the final model to evaluate the test error.

5 RESULTS

In this section we discuss our prediction accuracy and compare it with engine score as a predictor of the outcome.

5.1 Baseline Accuracy

For each board position we know what the engine-predicted score for that position is. Since a positive score for one side indicates a stronger position for that side, we were able to measure a baseline prediction accuracy using the engine's scores as a proxy for the result of the game. In other words, if the engine had to predict the result of a game by looking at the board positions, it would simply calculate the board score, s, and predict:

$$h(s) = \begin{cases} 1 & : s \geq 0 \\ 0 & : s < 0 \end{cases}$$

This perceptron-like approach of calculating baseline accuracy allowed us to evaluate how well engine score can predict game outcomes at different board scores.

5.2 Machine Learning Accuracy

Consider the case when engine-predicted score is exactly 0 but the game ends decisively. Thus, engine predicts equal probability for both sides to win making the baseline accuracy as 50%. The prediction accuracies of our method in this case are shown in Figure 2. Random Forests performed the best with test accuracy of 63.06% which is quite good as compared to baseline accuracy 50%.

Table 2: Machine Learning Accuracy Results

	Classifier	**Train**	**Test**
1	Random Forest	62.91%	63.06%
2	Logistic Regression	56.99%	55.45%
3	SVM (RBF)	56.19%	55.59%
4	LDA	57.97%	57.26%
5	Naive Bayes	54.88%	53.88%

Chess positions are often very complex to analyze using this feature-based approach. Many features might indicate a weakness depending on a particular configuration for the other features but a strength for another configuration. For example, Open files may be a strong advantage when a side has rooks on these files but can be a weakness if the side does not possess rooks but the opposition has rooks with the same set of open files. It is very hard for most discriminative/generative models to capture this complexity on a case by case basis, but decision trees are intrinsically designed to handle such kind of feature interactions. Thus, random forests is able to outperform other classifiers.

5.3 Analysis of Results

We considered board positions where the absolute value of the engine score was less than some value x close to 0. We varied the value of x from 0 to 1 and compared the machine learning accuracy with the baseline accuracy for that set of board positions. The results can be seen in figure 5. When x is very close to 0, machine learning approach using positional features performs very well and outperforms the baseline accuracy with a good margin. When x is greater than 0.35, the baseline accuracy is higher. This is because having positions with different scores makes it harder for the features to capture the variety of detail in the board positions. On the other hand, by training only on a small range of board scores we are able to achieve much higher accuracies.

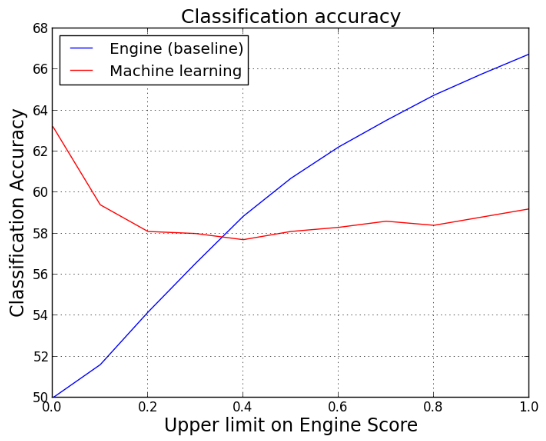


Figure 5: Machine Learning vs Baseline

We also identified features that are most important in predicting the board outcomes. The top five features with their relative importance are shown in figure 6. We notice that mobility network features play an important role in positional features.

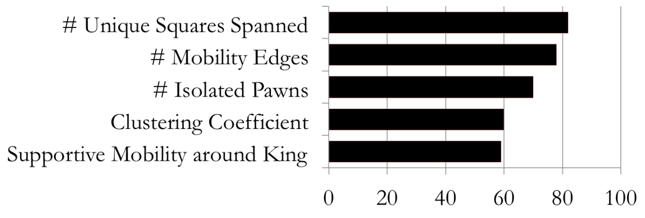


Figure 6: Relative importance of top features

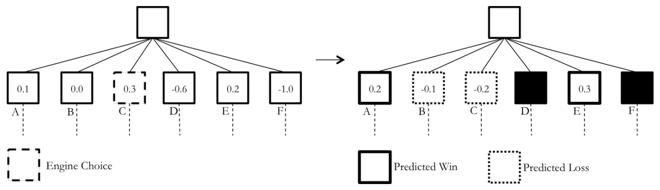


Figure 7: Augmenting Machine Learning with Current Chess Engines

6 CONCLUSION & FUTURE WORK

Using machine learning we were able to predict the game outcome with approximately 63% accuracy for positions where the engines were unable to assign an advantage to either side (i.e. prediction accuracy of 50% if we use the engine score as a predictor). These results demonstrate that engines lack important positional insight into the game that our features and models capture. Given this, possible future directions of our work are:

- **Enhanced features**: A case-by-case analysis of existing features with the possibility of additional features would provide additional insight to incorporating positional play in chess engines.

- **Augmenting chess engines with positional play**: The results from machine learning can be added to the board evaluation function that chess engines use and can be tested against state-of-the-art engines. An example of how our predictions can be used to recalculate board scores and improve decision-making can be seen in Figure 7. In this example, positions predicted to win and lose are awarded and penalized 0.1 points respectively. **Note**: Since feature computation can be expensive, it should be included only at the top few levels in the game-tree search rather than at the leaf-level. The source code for many state-of-the-art engines is available to the public [3].

References

[1] H. van Kempen. (2006) Nunn tournaments / cegt 40/120. [Online]. Available: http://www.husvankempen.de/nunn/downloads/40_120_new/downloads.htm

[2] D. Farren, D. Templeton, and M. Wang, “Analysis of networks in chess,” Stanford University, Tech. Rep., 2013.

[3] [Online]. Available: https://github.com/mcostalba/Stockfish