Chess Position Evaluator

Ryan Whitney

***Abstract*— Chess is a complex game that has yet to be solved. Although there are a finite number of possible positions, the processing power of current computers is insufficient to attempt a brute force solution to determine advantageous chess positions. This paper aims to utilize machine learning to evaluate whether a given position will likely lead to a win or loss. Chess games played by humans on online websites are used for training and testing. Several classifiers are implemented such as KNN and SVM.**

I. INTRODUCTION

The estimated number of subatomic particles in the universe is on the order of 10^{80} whereas the number of possible chess games, Shannons number, is conservatively estimated to be on the order of 10^{120} [1]. Considering the space complexity of chess, brute force algorithms are entirely reliant on available computing power and time to process information. In 1997, then-world champion Garry Kasparov faced off against the Deep Blue IBM chess computer and marked the first time in history that a computer was able to defeat a reigning chess champion. Deep Blue was calculating up to one billion chess positions per second to achieve a performance only marginally better than Kasparov [2]. Nowadays, virtually all the top chess engines utilize neural networks to change the parameter weightings rather than using hardcoded rules to evaluate the given position.

Chess has some terminology that will be used throughout this paper. A “ply” refers to the movement of a piece from one player. The “material” value refers purely to the value of the pieces and does not consider the coordination of the position, the pawn structure, checkmating opportunities, etc. A pawn is considered to have a value of 1, a knight and bishop have a value of 3, a rook 5, and a queen 9 [3]. Chess positions are evaluated on a numerical scale where 0 represents a draw. A position that has an evaluation of +2 would represent that white is winning by the equivalent of 2 pawns up, although it could be purely a positional advantage and not necessarily a material advantage. On the contrary, a position with a negative value, such as -3 would indicate that black has a winning advantage. It is generally considered that if one side has an advantage of 3, the fate of the game is relatively certain among master players. Typically, chess engines will use the Glicko-2 rating system which is very similar to the Elo rating system to compare the relative strength of a player to the other players in the rating pool [4]. This chess position evaluator will predict either a win or loss for the given chess position rather than a numerical evaluation or rating prediction.

The scope of this paper does not extend to deep learning or reinforcement learning techniques which are the current standard for chess engines [5]. This paper will also not develop any algorithmic based approaches although they will be discussed in the literature review given their importance to current chess engines. Several classifiers will be used such as KNN and SVM. The success of this chess position evaluator will be whether it can produce a meaningful output (better than a random guess).

II. LITERATURE REVIEW

The only task a chess player has is to evaluate the position accurately, including both their current position and future possible positions after certain moves are played. Humans generally only need to evaluate the new positions after a few specific pieces are moved (they understand that there is no value investigating most other moves). Computers do not have the same intuition but there are some algorithms which can help narrow the min-max search space such as alpha-beta pruning [6]. The min-max algorithm in the context of evaluating a chess position would return a numerical value, typically from -5 to +5 where a higher/lower number would indicate an almost certain victory. The min-max algorithm simply branches out and calculates every possible move, and the associated position evaluation after the move is played [7]. The branching factor for min-max searches at the start of a chess game is 20 which represents all the possible moves. As the game progresses, the branching factor increases to an average of about 35 [8] during the "middlegame" and decreases to around 10 during the "endgame". Alpha-beta pruning [9] is used to decrease the min-max search space. It works by keeping track of the best and worst moves and uses that information to skip branches which are guaranteed to have worse evaluations [10]. In chess, there are oftentimes examples of temporary material loss which results in an unstoppable checkmate a few moves later. If the search ends at the node with the temporary material loss, the algorithm will likely evaluate the position poorly. One solution to minimize this problem is to prioritize the search and dive deeper into nodes where there are piece captures, which is called a quiescence search [11]. Another common search is the Monte Carlo Tree Search [12] [13] which runs simulated games between players making random moves and assigns a value to the win rate of the games, instead of considering every move.

The current best chess engines such as Stockfish, AlphaZero, and Leela Chess Zero, utilize a combination of search algorithms and neural networks. The neural network is used to adjust the search trees' leaf node evaluations [14]. Recently, efficiently updatable neural networks (NNUE) were added to Stockfish and improved its rating [15]. NNUE implements neural networks for position evaluation instead of the previous hardcoded evaluation [15].

III. METHODOLOGY

A. *Dataset*

The dataset that is used contains chess games from the open-source chess website Lichess.org. This website is one of two major online chess websites, and the chosen dataset is the games from all players during January 2013. This includes

121,332 games and this dataset was chosen since it was the smallest and most manageable to download from the database.lichess.org. Games from every month are available since January 2013 and the size of the more recent datasets are considerably larger where November 2022 has 89,319,297 games played. Since the dataset will be pre-processed to include every position of the dataset, only a fraction of the January 2013 dataset will be needed. There are a few known issues with the datasets which are listed database.lichess.org but the overwhelming majority of the data has no issues, and the few issues present will not meaningfully affect the outcome of the machine learning models.

The dataset is in PGN (Portable Game Notation) format which is a plain text file that includes the move orders of the chess games as well as some metadata such as the player ratings, date, time, result, rating difference, etc. A sample game in PGN format can be seen in Figure 1.

```
[Event "Rated Bullet game"]
[Site "https://lichess.org/0zdpz2lh"]
[White "Lain"]
[Black "k074"]
[Result "1-0"]
[UTCDate "2013.01.08"]
[UTCTime "21:07:47"]
[WhiteElo "1574"]
[BlackElo "1594"]
[WhiteRatingDiff "+12"]
[BlackRatingDiff "-12"]
[ECO "C00"]
[Opening "French Defense: La Bourdonnais Variation"]
[TimeControl "0+1"]
[Termination "Time forfeit"]

1. e4 e6 2. f4 d5 3. exd5 exd5 4. c4 dxc4 5. Bxc4 Bd6
6. Nc3 Nf6 7. d4 O-O 8. Nf3 b6 9. O-O Bb7 10. Bd2 Bxf3
11. Qxf3 a6 12. Qxa8 b5 13. Bb3 Re8 14. Nd5 Nxd5
15. Qxd5 1-0
```

Fig. 1. Example game from dataset in PGN format.

B. *Pre-Processing*

Since the goal of this paper is to evaluate chess positions, all the positions that occur throughout the games need to be extracted from the larger dataset. At the start of the chess game, both players have identical positions where the only difference is the colour of the player. Since player colour alone is insufficient in determining the winner of otherwise identical positions, the first 10 positions arising from the chess games will be omitted. Therefore, any chess games that have 10 or fewer moves will also be omitted. The positions that arise during games that result in draws will also be omitted.

When the program is run to use 1000 games from the dataset, approximately 39,000 – 40,000 positions are saved for use by the model. This is inline with the "average length of a chess game" that was found to be 40 [16]. This was deemed to be a reasonable number of games.

C. *Feature Selection*

There are numerous features that can be extracted from the dataset being used. All features that require metadata will be excluded such as player ratings, date the game was played, opening played, etc. The reasoning for this is that the model being developed should be able to take any game as an input and not rely on metadata features that may not always be available.

The fit_transform function from the sklearn Python library is used to compute the mean and standard deviation for data scaling. The numerical values of the features are normalized between 0 and 1 using min-max normalization of the data. This preserves the relationship among the original data values and reduces the effect of outliers. This normalization technique can be used since the data follows a gaussian distribution or a binary 1 or 0 value which remains unchanged. The data was normalized since the models being used are dependent on the magnitude of the data. The numerical values for the material difference feature include a positive or negative sign. A comparison between non-normalized data and normalized data will be examined in the results section.

The following features were extracted from the dataset using the python-chess library for PGN parsing:

- The material difference
- The total white material
- The total black material
- The player colour
- The ply number (half move)
- If the white king is "in check"
- If the black king is "in check"
- If the white queen exists
- If the black queen exists
- The number of squares that white attacks
- The number of squares that black attacks

The features listed above were selected and deemed relevant from empirical knowledge of the author.

The material difference represents the difference in the total material value of the white pieces and black pieces. A negative number indicates that black has more material, whereas a positive number indicates that white has more material. Based on empirical knowledge, this feature is the most important from the list of features above.

The total white material and total black material have a similar level of relevance as the material difference feature. These features provide information about the total amount of material each colour has.

The player colour is also relevant as it is known that statistically speaking, white has a slight advantage over black since they get the first move.

The ply number which represents the half moves (one side plays a piece) is likely not very relevant as a standalone feature but may have high correlation in some scenarios. For example, towards the end of the game, features such as the material difference become more important. This is because a difference of 1 point of material is more noticeable when, for example, each player only has 10 points of material total, rather than when they both have 30 points of material total. For reference, each player starts with 39 points of material at the start of the game. The average chess game was found to be approximately 39 moves (78 ply) in the dataset which aligns with the average length of a chess game [16]. There are several instances of chess games reaching well over 100 moves (200 ply) which can

skew the data. These outliers reduce the separability of the average games.

The white king “in check” and the black king “in check” represent whether the current board position has the given king “in check”, denoted by a 1 for yes and a 0 for no. Being “in check” is not necessarily a bad thing to happen during a chess game but may be indicative of a weak king, or looming checkmate.

The existence of the white queen and black queen are denoted by a 1 for exists, and a 0 for does not exist. For human games, especially average players, it is easier to play with a queen rather than the equivalent material value in other pieces (e.g., 2 bishops and a knight for a queen). This is not as true for chess engines, but since the dataset includes only human games the relevance of having the queens on the board will be explored in the models.

The number of squares that white attacks and the number of squares black attacks is generally seen as an important metric when determining which player has the advantage. The number of squares under attack indicate the “board control” one player has where more enemy squares under attack may indicate an advantageous position.

The label is the result of the game where a 1 indicates that white wins and a 0 indicate black wins. A snapshot of the raw data for one chess game can be viewed below in Figure 2.

1	result,materialDifference,totalWhite…
2	0,2,37,35,1,21,0,0,1,1,39,34
3	0,2,37,35,0,22,0,0,1,1,39,35
4	0,2,37,35,1,23,0,0,1,1,41,32
5	0,2,37,35,0,24,0,0,1,1,41,33
6	0,2,37,35,1,25,0,0,1,1,39,36
7	0,2,37,35,0,26,0,0,1,1,39,36
8	0,2,37,35,1,27,0,0,1,1,39,36
9	0,2,37,35,0,28,0,0,1,1,39,42
10	0,5,37,32,1,29,0,0,1,1,38,41
11	0,0,32,32,0,30,1,0,1,1,38,40
12	0,9,32,23,1,31,0,0,1,0,37,33
13	0,6,29,23,0,32,0,0,1,0,34,32
14	0,7,29,22,1,33,0,1,1,0,35,32

Fig. 2. Raw data before normalization of 1 chess game

D. *Classification*

Two different classifiers are used to attempt at solving the 2-class problem presented in this paper. A KNN model is used with varying values of K. A SVM is also used where several different kernels are used such as RBF, linear, polynomial, and sigmoid. The model_selection.train_test_split function from the sklearn Python library is used to split the data into training and testing partitions. All the models use 20% of the data for testing and the remaining 80% for training.

IV. RESULTS

The 2-class problem presented in this paper has a random classification accuracy of 50%. It is expected that the models presented will increase the accuracy above 50%. The results of each model can be viewed in their respective sections below.

A. *Classifier Results: KNN*

The accuracy of the KNN model with varying values of K from 1 to 15 and normalized data can be seen below in Figure 3. The best K value is K = 1 where the accuracy was 74.4%.

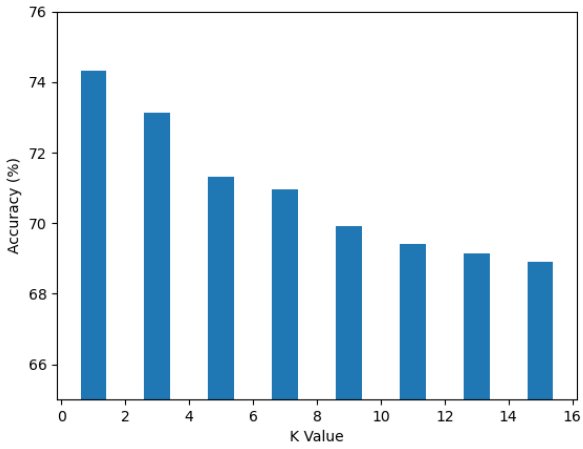

Fig. 3. KNN results with varying K values from 1 to 17 with normalized data

The accuracy of the KNN model with varying values of K from 1 to 15 and without normalized data can be seen below in Figure 4. The best K value is K = 1 where the accuracy was 74.2%.

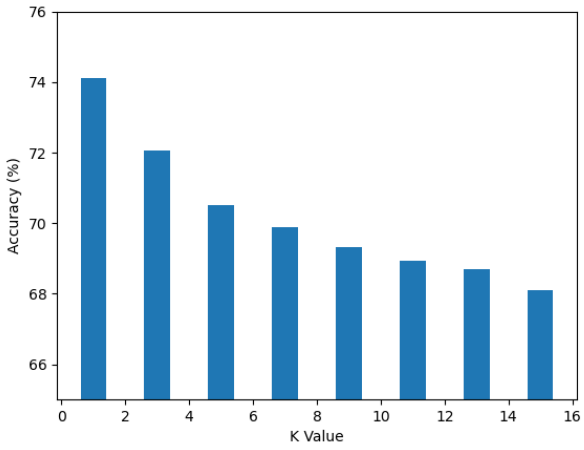

Fig. 4. KNN results with varying K values from 1 to 17 without normalized data

B. *Classifier Results: SVM*

The accuracy of the SVM with different kernels including RBF, linear, polynomial, and sigmoid with normalized data can be seen below in Figure 5. The accuracies of these models hovered around 62% except for the sigmoid kernel type, where the accuracy was approximately 50% (random chance).

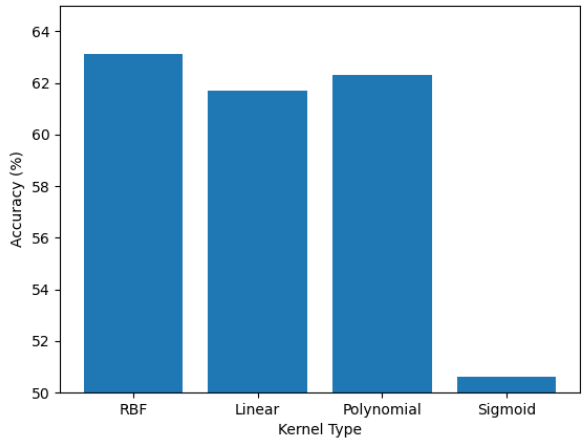

Fig. 5. SVM with different kernel types with normalized data

The accuracy of the SVM with different kernels including RBF, linear, polynomial, and sigmoid with normalized data can be seen below in Figure 6. The accuracies of these models hovered around 62% except for the sigmoid kernel type, where the accuracy was approximately 50% (random chance).

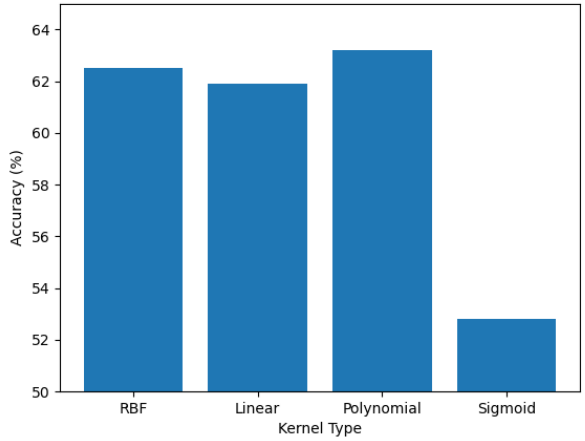

Fig. 6. SVM with different kernel types without normalized data

C. *Overall Results Comparison*

A comparison between the accuracies of the KNN model and SVM model can be seen below in Figure 7. The KNN model has an accuracy of 74.4% whereas the SVM model has an accuracy of 62.6%.

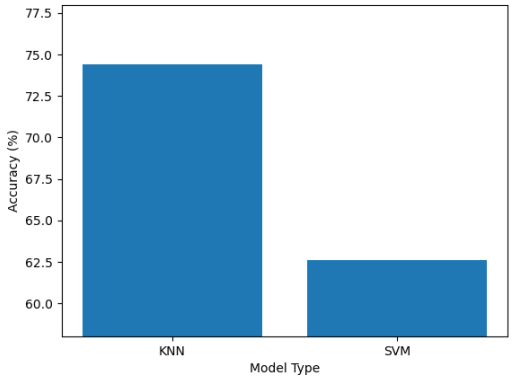

Fig. 7. Best performing KNN and SVM models

V. DISCUSSION

The KNN model performed significantly better than the SVM model. The normalization of the data had virtually no impact on the output of the models. This is likely due to the fact that the majority of the features were binary values (1 or 0) and thus the normalization had no effect. The majority of the remaining non-binary features, such as the total material, were generally empirically known to be the most important features so it was acceptable that they were more heavily weighted (the use of Euclidean distance favours the larger numerical values).

Given that the numerical values associated with the majority of features was a binary 1 or 0, the fact that the KNN model performed best when K = 1 does not indicate overfitting.

The SVM model performed relatively consistently across three of the kernel types, RBF, Linear, and Polynomial. The model performed slightly above random chance with the Sigmoid kernel. The poor performance with the Sigmoid kernel is due to the fact that this kernel type is essentially an activation function in the shape of the tanh() function which does not draw an accurate decision boundary for the given data.

Since there is currently no literature exploring the use of any machine learning models for the evaluation of chess positions (excluding deep learning), this paper outlines the first public results in the field. Although the results of the best performing model was found to be 74% accurate at determining which player will win, this is not necessarily applicable to the development of chess engines. The reasoning is that chess engines need to evaluate positions using a continuous output rather than a binary "win" or "loss" to search through the best possible moves. Future work could potentially see the combination of the KNN or SVM models with the currently used deep learning methods.

ACKNOWLEDGEMENT

The author would like to thank Dr. E. Scheme for his encouraging words and insightful lessons in the ECE4553 Introduction to Pattern Recognition course at UNB.

REFERENCES

[1] D. Rasskin-Gutman, "Chess Metaphors: Artificial Intelligence and the Human Mind," The MIT Press, 2009.

[2] F.-H. Hsu, “IBM's Deep Blue Chess grandmaster chips,” IEEE, 1999.

[3] "Chess Piece Value," *Chess Terms.* [Online]. Available: https://www.chess.com/terms/chess-piece-value#:~:text=Chess%20Piece%20Values,-As%20mentioned%2C%20each&text=A%20pawn%20is%20worth%20one,t%20have%20a%20point

[4] "Glicko-2 Rating System vs. ELO Rating," *Lichess Forum*. [Online]. Available: https://lichess.org/forum/general-chess-discussion/glicko-2-rating-system-vs-elo-rating.

[5] "Neural network topology." [Online]. Available: https://lczero.org/dev/backend/mm/.

[6] S. P. Singhal and M. Sridevi, “Comparative study of performance of parallel alpha Beta Pruning for different architectures,” IEEE, 2019.

[7] P. Sadeghian and A. Olmsted, “Assessment of fuzzy min-max neural networks for classification tasks,” IEEE, 2017.

[8] G. Badhrinathan, A. Agarwal and R. A. Kumar, “Implementation of distributed chess engine using PaaS,” IEEE, 2012.

[9] N. Sato and K. Ikeda, “Three types of forward pruning techniques to apply the alpha beta algorithm to turn-based strategy games,” IEEE, 2016.

[10] Z.-C. and C.-J. , “Improved Alpha-Beta Pruning of Heuristic Search in Game-Playing Tree,” IEEE, 2009.

[11] M. Winands and M. Schadd, “Quiescence Search for Stratego,” in *Belgian/Netherlands Artificial Intelligence Conference.*, 2009.

[12] J. Lu, X. Wang, D. Wang and Y. Wang, “Parallel Monte Carlo tree search in perfect information game with chance,” IEEE, 2016.

[13] Z. Li, H. Liu, Y. Wang and J. Zuo, “Application of Monte Carlo Tree Optimization Algorithm on Hex Chess,” IEEE, 2020.

[14] H. Baier and M. Winands, “Monte-Carlo Tree Search and Minimax Hybrids”.

[15] D. Klein, “Neural Networks for Chess The magic of deep and reinforcement,” 2022.

[16] "Total Number of reasonable moves per turn." *Chess.com Forums.* [Online]. Available: https://www.chess.com/forum/view/fun-with-chess/total-number-of-reasonable-moves