Predicting Chess Winners with Machine Learning

Caleb Stasiuk

T00585211

COMP 4980 Machine Learning

April 16th, 2022

Data Description

I decided to change my dataset due to my feedback received in the exercise. The dataset that I am now using is on chess game data. I have always been fascinated by chess and enjoy playing it so I thought it would be interesting to investigate the game while employing the use of Machine Learning. The dataset tracks whether a game is rated, start and end time, total number of turns, the game result, the winning colour, time increment, white and black player ID and rating, all moves in the game in standard chess notation, opening eco (standardised code for any given opening, opening name, the number of moves in the opening phase and assigns each game an ID. The data is categorical as each game is divided into multiple categories. The dataset tracks 20,000 chess games, each game with 15 features if we exclude the game ID and is 7.3 MB. For the variety of data, the ID is a string of unique upper and lowercase characters and numbers. Rated is a Boolean value that says whether the game is rated. The time created and last move is a value in Unix time format. Turns is an integer value that shows how many turns took place in the game before it ended. Victory status shows the outcome of the game, like a mate, draw, resign, etc. Winner can be 1 of 3 possible values, white, black, or draw. Increment is a specified amount of time added to the players clock each move, the first value is how many minutes are in the game, the value after the + is how many seconds are added after each move so in a 5+10 increment (or Fischer delay) each player starts with 5 minutes and adds 10 seconds to their time each move. White and black player ID shows what each player's username is in the game. Similarly, the rating shows what each player's ratings are with the lowest possible value being 0 and the top-rated chess players in the world are around 2800. Moves shows every move that takes place in the game, this is shown based by the name of the piece signified by the first character, and the square that the move too. The opening eco code is a standardized code for the opening, begins with a letter followed by two digits. The opening is simply the name of the opening, like Queen's Pawn, Sicilian Defense, English Opening, etc. Finally, the opening ply feature shows how many moves took place in the opening phase of the game. The format of the data is in a .csv file called games.csv. The data seems to be usable and wouldn't take a lot of work to use in python. Additionally, there doesn't appear to be any values missing or errors. It seems like there hasn't been any updates to the dataset for a few years but considering that chess is a game that has been played and studied for thousands of years I don't think having the most recent up to date data is a big problem. I would consider the value of this data to be subjective, as it really comes down to how a person feels about the game. To some who enjoy the game and studying it deeply, this dataset is extremely valuable, but on the other hand, someone who has no interest in the game would find no use in examining it further.

Data Exploration

For my data exploration I ended up dropping the game ID, player IDs, opening eco, moves, and rated columns as I decided I would not be using them in this project. After performing a PCA on that data, it appeared that one principal component (0.99339...) makes up 96% of the variance in the data. This means that we could reduce the dataset to just one feature if we wanted to improve performance, but in this case, I don't feel like it's necessary. For the decision tree, I ended up using a classifier decision tree with a max depth of 4. My F1_score produced the best accuracy when using macro average of around 72%. When we look at Fig. 1(see notebook), an interesting relationship is occurring towards the bottom of the tree. The nodes are splitting off from a white node into two white nodes, I believe that this is happening because the further down it goes in the tree the more confident it is getting in its decision. The tree is not palatable to read if the max depth is larger than 4 or at the very most 5 but it would be interesting to see what a tree with a max depth of 10 or even 20 comes up with but for now these results will have to suffice. Something else that we can observe is the effect each players rating has on the game. The left most node white_rating <= 181.4, we can see that if this condition is true, it's likely that black will have an advantage but if its false, white could have an advantage. The same example can be looked at in the rightmost node black_rating <= 936.5, if true then white has an advantage, if false then black. The number of samples also increases as we go up in ratings, comparing the two nodes with black's ratings of 627.5 and 936.5 the number of samples is 2293 and 10097 respectively. This just mean that there are more players in this dataset around the average chess rating.

Experimental Method

Due to the complex nature of the game, I believe that it will be difficult to achieve very accurate results. Through the implementation of various Machine Learning models, it should be feasible to predict the winner of a game of chess with an accuracy of at least 50%. To test his hypothesis, some of the models I will be using include Multi-layer Perception (MLP), Random Forest Classifier, GradientBoost Classifier, and AdaBoost Classifier. I predict that MLP will be the most successful model and the results from my analysis can be found below Fig 1.

Based off the MLP accuracy, my initial prediction seems like it might have been wrong, but I will have to see how the rest of the models end up performing. For MLP, we have an accuracy of 61.2%, with hidden layer size of (10,5), 1000 max iterations, logistic activation function, and an alpha 0.0.

Looking at the Random Forest Classifier now confirms my initial hypothesis of what model I would perform the best was wrong. Our accuracy for Random Forest Classifier is 66.7%, with a max depth of 5, and 300 n estimators.

Truthfully, I was not expecting GradientBoost to perform as well as it did, without changing anything from the notebook examples. It achieved an accuracy score of 76.3% with 100 n estimators.

AdaBoost has been the performed the worst of all the models tested but is still holds true to my original hypothesis of being at least 50%. For AdaBoost, our accuracy score is 56.8% with 100 n estimators.

To refine my results, I will be running a GridSearch to tune my hyperparameters for MLP and GradientBoost. My parameters for MLP follow: {'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],

'activation': ['logistic', 'tanh', 'relu'],

'solver': ['sgd', 'adam'],

'alpha': [0.0, 0.0001, 0.05],

'learning_rate': ['constant', 'adaptive']}.

The best parameters found were: {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'solver': 'sgd'}.

After plugging in the best parameters, our accuracy improves marginally to 62.1% from 61.2%, so there is virtually no difference.

Hopefully better results will result from doing a GridSearch on GradientBoost. I reduced the number of parameters used in this GridSearch, as the first one took over half an hour. My parameters for GradientBoost follow: {'n_estimators' : [10, 50, 100, 200, 300, 400]}. The best parameter found was 400 n estimators, which produces a result of 89.1%. This is quite a dramatic increase from 76.3%, but the execution time is significantly longer at around 42 seconds. I also wanted to see what the result would be when using 200 estimators, so I tried with that as well and managed to achieve an accuracy of 85.4%. Taking half the number of estimators also was half the time being 21 seconds and only perform 3.7% worse. I tried again with 150 estimators, and this took 16 seconds with an accuracy of 82.1%, so for accuracy, 400 performed the best but for performance time, 200 performed much better with only marginally worse results.

Results and Analysis

The model that produced the best results as shown in the above section is GradientBoost with the highest accuracy being 89.1%. A classification report of this algorithm outcome follows (I made a table so it's a bit more palatable):

	Precision	Recall	F1-Score	Support
Black	0.89	0.88	0.88	1850
Draw	0.99	0.95	0.97	202
White	0.88	0.90	0.89	1960
Accuracy			0.89	4012
Macro Avg	0.92	0.91	0.91	4012
Weighted Avg	0.89	0.89	0.89	4012

Looking at the table, we can see that distribution of values between black and white to be pretty much even, being in one to two percentages of each other, while the accuracy of draw is extremely accurate. We can also look at the cross-validation score, we see the following: [0.88958126 0.88085743 0.89431705], from this we can see that the model is accurate as there is little variation between the three accuracy scores. Overall, I am pleased with how this model performed, and it exceeded my expectations.

I think that these results showcase how powerful Machine Learning can be when applied to games such as chess. With my still rather limited knowledge of Machine Learning and being able to produce predictions with a decent accuracy it showcases how someone with much more knowledge can apply Machine Learning, to make things like complex chess bots that can accurately respond to players moves and even predict the coming moves before the player might even know what they will be doing. Some of the relationships that I observed is that the accuracy, is close for the values of predicting black and white, but draw is predicted extremely accurately. This is not surprising though as the outcome of a draw has much less conditions to occur then a checkmate, as a draw typically only happens when a player makes a move that does not check the king but removes all legal moves the king can make, if the game is reduced to blacks king versus whites king, or if a player runs out of time and the other player does not have enough material to mate.

The practical applications of this models could be used to take a deep dive into understanding some of the many factors that contribute to winning a game of chess. If further refinement was put into the model, it could be used to create a chess bot. There are still however limitations when it comes to chess bots, there are still many problems that prove to be hard to deal with. Some tactics that can be used against them is blockades, and positional draws are a few of the tactics that a programmer would have to deal with when creating a bot [3]. These challenges, however, have been beaten as the software AlphaGo, an AI made to play the Chinese game of Go, has never lost a single game of chess against the world's top players [4]. Could my models be used in an AI like AlphaGo? Probably not, but with enough dedication and a deeper understanding, a very powerful bot may be able to be produced. Ultimately, these results do not provide any real-world applications that are beneficial to the world compared to something like medical applications, but to people like me who are in love with the game, it is relevant.

References

1. Dataset: https://www.kaggle.com/datasets/datasnaek/chess?resource=download
2. *Analyzing chess data*. (Kunauska, M). Kaggle.com. Retrieved April 17, 2022, from https://www.kaggle.com/code/mariuskunauskas/analyzing-chess-data
3. *When Computers Go Wrong - Chess Lessons*. (Williams, S). Chess.com. Retrieved April 17, 2022, from https://www.chess.com/lessons/when-computers-go-wrong
4. *Can A Human Beat Alphazero?* Williams, K. (2021, November 27). Morethansport.com. https://www.morethansport.com/can-a-human-beat-alphazero