Analyzing Chess Games for Predictive Strategies

Ravinder Rai

Feb 24, 2023

Introduction

This project was about analyzing a dataset of about 20000 chess games. The data came from Kaggle, and the goal was to perform some kind of predictive analysis to suggest possible changes to a chess player's strategy, which would hopefully in turn improve their chances of winning. The process mainly involved exploring the data, after which was pre-processed appropriately. Then finding the best predictive model was the next step, where predicting the winner was the goal. Finally, the best predictive model was used on some random players in the test set to analyze their chess games and suggest changes to their game strategies.

Problem Statement

Although chess may be considered a solved game, since artificial intelligence can beat the best human now, it remains one of the most popular games. To that end, in many cases, chess tournaments with monetary prizes are held. Thus, using data to find optimal strategies to win in these kinds of settings has great value. More specifically, improving a player's chance of victory and receiving monetary rewards, as well as overall player satisfaction when playing the game, are desired outcomes.

If we can find such a relationship that could suggest changes to a player's strategy, that would be of great benefit. More specifically, a relationship between aspects of a game, and the winner of a game, would help the most. These aspects could be the moves made during a game, the opening names of a game, or the number of turns of a game. Naturally, the aim would then be to predict who would win, white or black, where players who play as white are the players who go first. It should be noted that players cannot choose which color they wish to play as, so strategy change suggestions would have to take this into account.

Data Wrangling

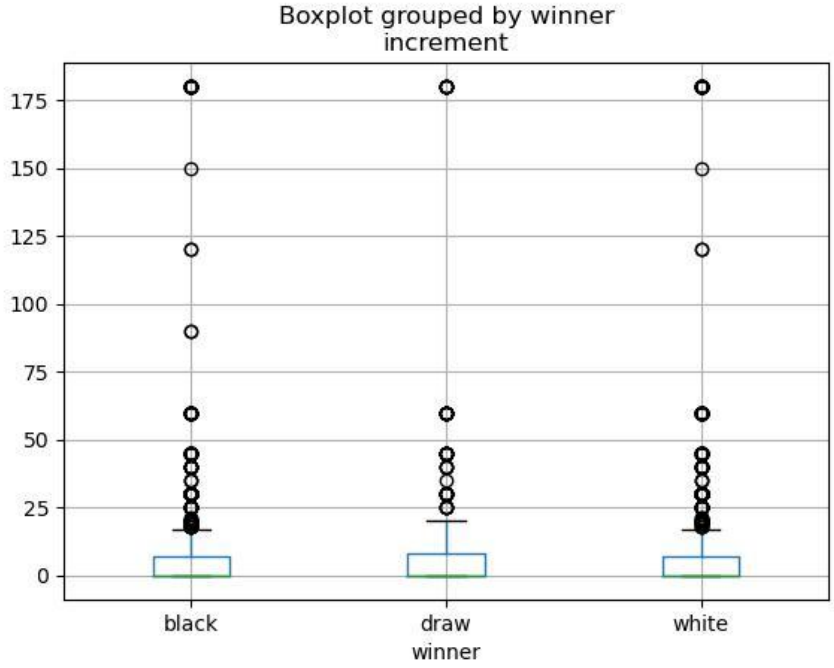
This chess dataset came from Kaggle, a wonderful website with lots of datasets for data analysts and data scientists. Unfortunately, this does not mean the data was clean and ready to go. Luckily, there were no null values, and in fact, there were only really two features that were faulty. These were features that described the start and end times of the games. More specifically, the end-time feature had the same value as the start-time feature for almost half of the data. This meant that some games were reported as games that took no time to complete, despite having lots of turns completed. In the end, these features were deemed unusable and were dropped.

Aside from that, there were a few features with notable outliers, like turn counts, or time allotted for games. There were a lot of these outliers though, and in any case, chess is a game that

can go on for longer than normal, so outliers are not really out of the norm and were deemed worth keeping.

Exploratory Data Analysis and Pre-Processing

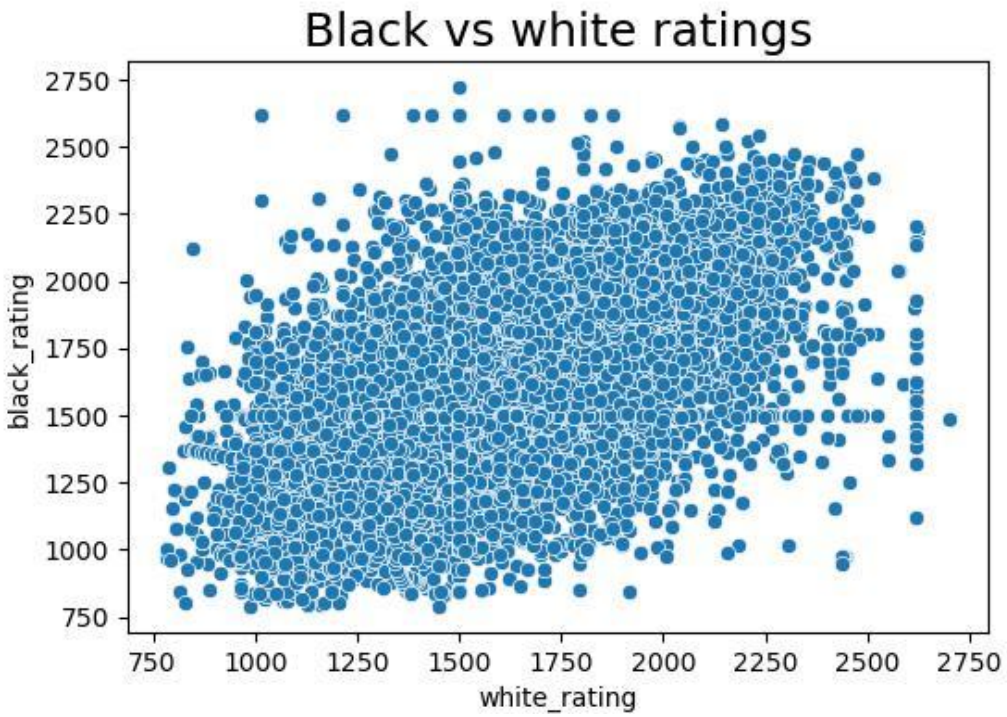
When exploring the data, one of the first things that were done was to explore the outliers mentioned in the previous section more. The features that had such outliers were the turns, increment, time allotted, opening play, and increment features. In fact, after doing box plots for all of these features, it turns out they all looked quite similar, so showing just one will give the idea of what they looked like. The below figure is a box plot of the increment feature, which describes the amount of time (in seconds) each player gets added to their clock when they start a new turn.



So, you can see in the above figure, that there were too many outliers to warrant simply removing them all. Regardless, as mentioned before, chess games with non-typical time increments, starting time allotted, and other features, are not uncommon, so it made sense to keep them. But it is also good to note that this is occurring in multiple features.

Something else important to mention is that a new feature describing the total time of each game was made from the starting and ending time faulty features mentioned above. This actually gave a similar plot to the one above but was also dragged down by so many zeroes that this was where the decision was made to remove it. Moreover, there are still a couple of other noteworthy things to cover here: the first being that there were two features regarding players'

ratings. One was for the white player's ratings and the other black. Plotting these against each other suggested that they might be redundant, as you can see below.



Turns out, both of these features followed a normal distribution, so a hypothesis test was formed. The null hypothesis test here was that there was no strong relationship between these two features. In the end, however, p values were calculated, and some other tests were done to check this as well, and it was concluded that the null hypothesis could not be rejected.

The other noteworthy thing to mention was that some basic modeling was done, using logistic regression, on just one feature at a time, trying to predict the winner, and quite often these basic models would favor predicting white as the winner. This did not seem to be reflected in the final models though, and it seems like this was the case because some of the features in this dataset were text-based, which seems to have been quite important in the modeling section.

Moreover, when it comes to pre-processing, these text features were the most important features to pre-process, since they described the game's opening move-set and all of the game's moves as well. These features were pre-processed by using python's count vectorizer function, making a vector for each unique chess move in the move-set. Similarly, this was also done for the features that described the opening moves. This whole process brought the dataset to having over 5000 columns. The rest of the pre-processing was more simple, where categorical variables were either one-hot encoded or scaled via standardization.

Modeling

When modeling this dataset, three models were tested, namely logistic regression, random forest, and XGBoost. The model metrics are as below, along with the best model parameters.

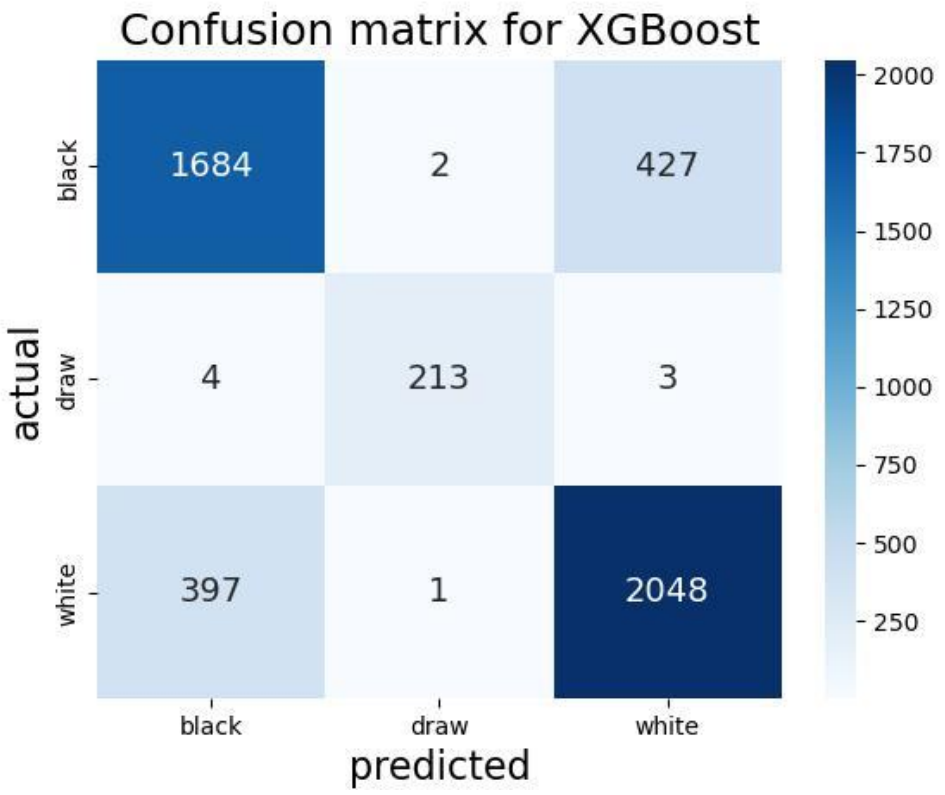
	best parameters	kappa training scores	kappa testing scores
Logistic Regression	{'C': 0.1, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_scaling': 1, 'l1_ratio': None, 'max_iter': 25, 'multi_class': 'auto', 'n_jobs': None, 'penalty': 'l2', 'random_state': 42, 'solver': 'newton-cg', 'tol': 0.0001, 'verbose': 0, 'warm_start': False}	0.666293	0.557663
Random Forest	{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 12, 'max_features': 'sqrt', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 5, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 150, 'n_jobs': -1, 'oob_score': False, 'random_state': 42, 'verbose': 0, 'warm_start': False}	0.649482	0.386385
XGBoost	{'objective': 'multi:softmax', 'use_label_encoder': None, 'base_score': None, 'booster': None, 'callbacks': None, 'colsample_bylevel': None, 'colsample_bynode': None, 'colsample_bytree': None, 'early_stopping_rounds': None, 'enable_categorical': False, 'eval_metric': None, 'feature_types': None, 'gamma': None, 'gpu_id': None, 'grow_policy': None, 'importance_type': None, 'interaction_constraints': None, 'learning_rate': 0.4, 'max_bin': None, 'max_cat_threshold': None, 'max_cat_to_onehot': None, 'max_delta_step': None, 'max_depth': 5, 'max_leaves': None, 'min_child_weight': 7, 'missing': nan, 'monotone_constraints': None, 'n_estimators': 150, 'n_jobs': -1, 'num_parallel_tree': None, 'predictor': None, 'random_state': 42, 'reg_alpha': None, 'reg_lambda': None, 'sampling_method': None, 'scale_pos_weight': None, 'subsample': None, 'tree_method': 'gpu_hist', 'validate_parameters': None, 'verbosity': None, 'num_class': 3}	0.906376	0.676605

At this point, it is important to note what these scores are. The training and testing part simply indicates the scores on the training and testing sets. The Cohen Kappa score is a special score that follows the following equation:

$$\frac{p_0 - p_e}{1 - p_e}.$$

p_0 here is simply the accuracy, and the p_e you can think of as how much the predictions and actual values agree. The Cohen Kappa score is then supposed to give an overall better estimate of how well the actual and predicted values agree. This score was used as the main score in grid search and other functions, and this was because this score is good for unbalanced multi-classification problems. Finally, note that Cohen Kappa scores from 0.61-0.8 are substantial, and anything above 0.8 is considered to be an almost perfect agreement.

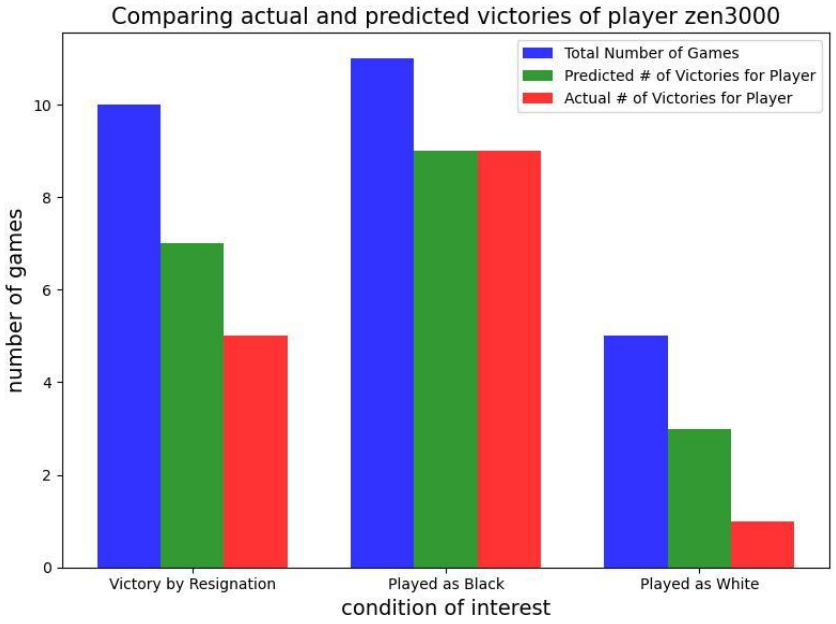
Now, logistic regression ended up being the second best, behind XGBoost, and ahead of random forest. Random forest was actually the fastest algorithm here, but as you can see above it performed the poorest. And of course, XGBoost performed the best. It is worth noting that XGBoost is known for doing well in lots of problems, so it is no shock it did so here. The next figure is the confusion matrix for XGBoost.



So you can see just how well the best model did. In fact, you can clearly see that the model is excellent at predicting draws. This is probably due to the fact that the victory status feature in the dataset has a categorical value that indicates that the game ended in a draw. This means that there is an obvious direct relationship between one of the variables and one of the target labels in this problem. Still, it is good to see that XGBoost picked up on this, because random forest did not at all, and logistic regression did a bit but not as much.

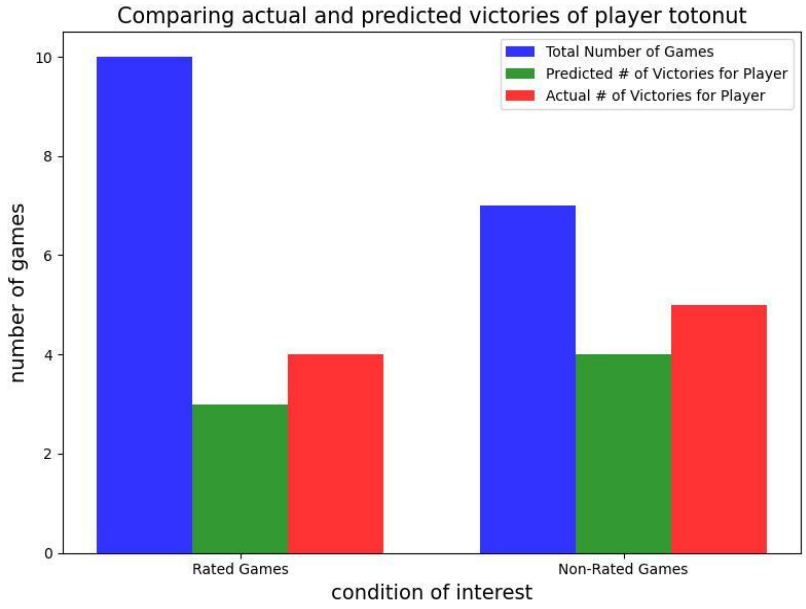
Modeling Analysis

Now that we have the model, we can use it to make suggestions to alter a player's strategy. So, imagine a chess player came to you with only 10 or so games. This is not really enough to draw any conclusions about his skills yet, but with the predictive model that was trained on over 20000 games, we can still make some suggestions (or at least get a second opinion). Using a random player from the test set, with player ID: zen3000, we can gather all chess games that this player took part in, and perform some data analysis.

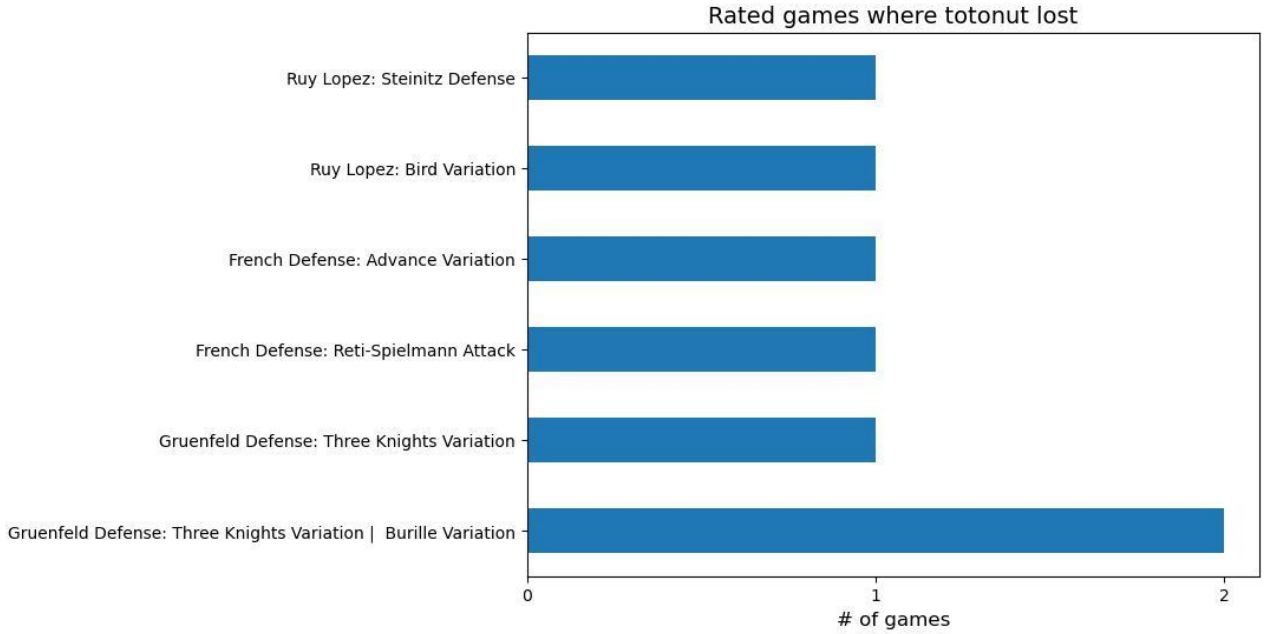


Looking at the figure above, you can see that the model agrees with the data that zen3000 is more likely to win when playing as black. That being said, when playing as white the model seems to suggest that zen3000 has a decent chance at winning, whereas the data itself would suggest otherwise. Thus, a strategy change that you could suggest to zen3000 would be to keep playing as black to optimize the chances of winning. As for playing as white, from zen3000's actual victories, you might think playing as white should be avoided, but from the model predictions perhaps a little more practice is all that's needed.

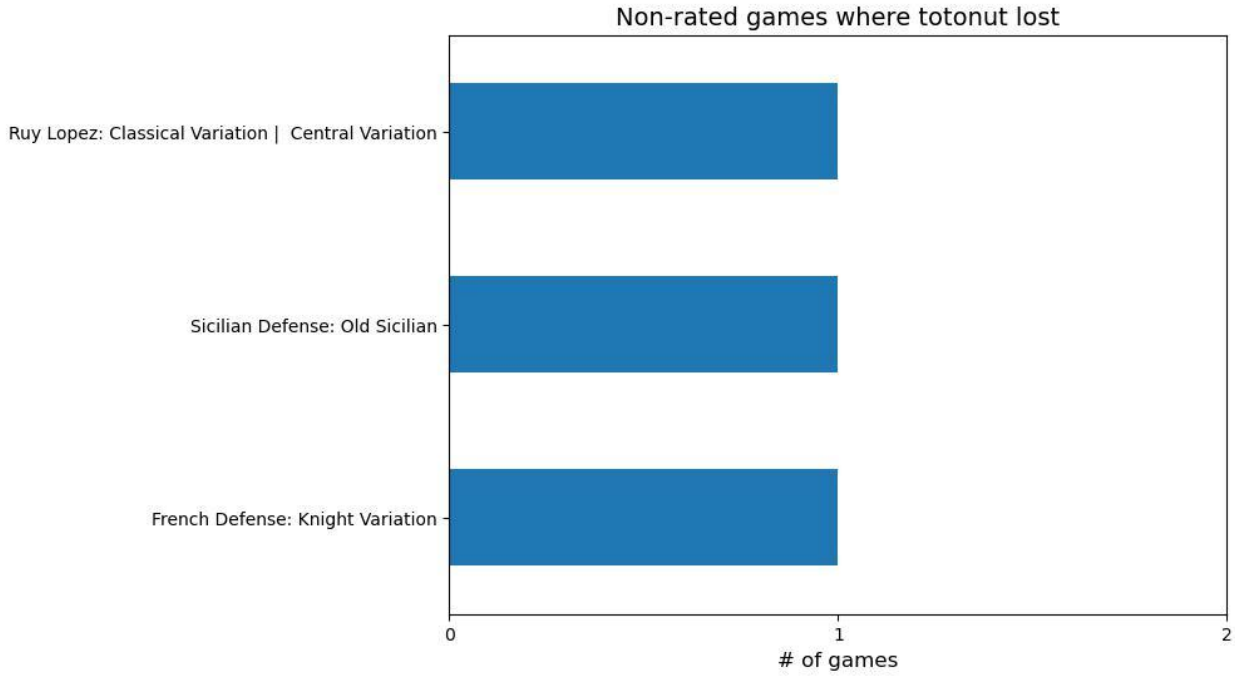
We can look at another player now, with player ID: totonut. Totonut has a mix of rated and unrated games, so it was more interesting to look at these games separately.



So you can see that in both actual and predicted wins, totonut has a tendency to lose rated games, while in non-rated games totonut does much better. We can explore the rated games further, by looking at opening moves.



In the figure above the Gruenfeld Defense games show up a lot, suggesting that perhaps totonut should abandon this opening move strategy. We can also look at the losses in non-rated games next.



So here, French Defense and Ruy Lopez opening moves show up here again. This makes the total number of losses for games with these opening moves 3 - the same number of losses for Gruenfeld Defense opening move games. While not shown here for legibility, out of all games that totonut did win, these opening move strategies did not show up a significant amount, thus indicating that totonut should abandon all three of these opening move strategies.

Conclusion and Future Work

Chess strategy improvements are obviously vital to winning, so even if all this model does is give a second opinion on what a player should do, it could still influence a player to win an important game. That being said, there are a few other things you could do with this model:

1. Look at patterns for games that have a large disparity in white and black ratings.
2. Look at all rated games and find most common opening moves when white or black was the winner.
3. Filter for all draws and find patterns there, like opening moves or black-and-white rating differences.

These are just a few more things you could do with this model. There are other tasks in chess analytics that you could use this dataset for, like using the board state to predict the winner. This would require changing the move set to fen notation, which is a way of representing the current game state in chess. The other more intuitive task would be to make predictions on what the best next move would be. This would probably require a neural network, though. In any case, there are lots more interesting tasks you could do, with this chess dataset or others.