Predicting Chess Outcomes By Measuring Game Uniqueness

Group 33 Project Report

August Boge
anboge@ncsu.edu

Drew Hughlett
arhughle@ncsu.edu

James Leeder
jmleeder@ncsu.edu

1 INTRO AND BACKGROUND

Is it possible to predict the outcome of a chess match given how unique the individual match is relative to every other chess match played before? In the game of chess, there are a staggering amount of possible games that can be played based on the 16 pieces each player starts with and the different movement options available to each piece. In his 1950 paper "Programming a Computer for Playing Chess", American mathematician Claude Shannon calculated that within the first five turns (or 10 plies), there are already approximately 69 trillion ($10^{13.83}$) possible games that could have been played[2]. Going even further, the Shannon number provides a lower-bound estimate of 10^{120} possible games assuming a chess match takes an average of 40 turns[2].

Shannon calculated these estimations to help demonstrate that it was unrealistic to try to solve chess using brute force because of the astronomical number of games and board positions possible. But what if we could use the fact that there are a near infinite number of chess games possible to help better predict the winner of a match? Ignoring any background knowledge about chess or chess strategy, and given the Shannon number, you would expect each and every chess game that has played since the inception of the game to be totally unique. But are some games more unique than others?

Throughout the lifetime of a chess match, the game has three different stages: the opening, the middlegame, and the endgame. The opening stage of the game usually lasts an average of five turns or so and is critical for setting the tone of the game. Because of how important the opening stage of the game is, much time and effort has gone into figuring out what moves, or "openings", are worth playing and how to play against them. So in a chess game between two players who have some understanding of basic chess strategy, we would expect the two players to play an opening worth playing and not a random move.

Using the assumption that the two players have some knowledge of chess and chess strategy, we can narrow down the total set of moves possible in the first couple turns of game to a subset of moves that we would expect both players to possibly choose. More simply, we would expect that someone who knows how to play chess would prefer to play good moves over bad ones. Putting everything together, we want to know if knowing how quickly a chess player diverges from the tree of expected or "good" moves that someone who understands chess would play in the opening stage would make it easier to predict the outcome of the match.

2 METHOD

2.1 Approach

We will start by preprocessing the data. We are going to remove all games that are unrated, games that did not finish in either a mate or a resignation, games that had less than 5 turns, and games that are faster than 5 minutes. We are also going to transform some of the data in order to get more meaningful attributes. One new attribute we are creating is a rating difference column, which is the White Player Rating - Black Player Rating. Another attribute we are creating is an average rating attribute, which is the $\frac{whiteRating+blackRating}{2}$. Also, we have the existing attribute Opening ECO, that has a letter, A through E, followed by 2 digits. We are going to edit that column to remove the digits and just have the letter. These new attributes, as well as number of turns and time increment, will be used to develop a baseline model.

We are creating a secondary model that has one new attribute called, number of turns until a unique game. This attribute is achieved by preprocessing the data from the moves attribute, which is in standard chess notation, and determining how many moves it takes the players to reach the unique state. We will also have a new attribute that says which player, Black or White, reached the unique state. We will then develop a new model, using the same classification techniques as the baseline model, but with these new attributes to predict who will win the chess game. From here we will compare the two models to determine which one performed better.

To develop these models we are going to use the machine learning technique of a decision tree classifier. These decision tree classifiers will be used with a bagging ensemble approach. The models produced by the bagging ensemble methods will be compared to each other to reach a conclusion about whether or not these new attributes described above can help predict who will win a chess game. We will test the bagging ensemble by modifying two different hyperparameters, the number of decision trees used in the ensemble as well as the number of samples used in the ensemble. We will fine tune these hyperparameters to determine which ones lead to the best model. Repeated stratified 10-fold cross validation will be used to figure out the best hyperparameters for the bagging ensemble method.

2.2 Rationale

We want to preprocess the data to get rid of unrated games because we think player rating can contribute greatly to who will win a game, and these games do not include player rating. We also removed games that did not finish in a mate or a resignation because the other games could have resulted in someone running out of time or people drawing, but we want to focus on whether someone will win or lose, not if they draw, also players that ran out of time might have been disconnected from the server or they might not have fully completed the game. We are also removing games that had less than 5 turns and games that are less than 5 minutes because we think that people that finish in less than 5 turns are not actually losing, but rather are deciding to just abandon the game, so this information wouldn't be that useful. We are also assuming that people that play games that are less than 5 minutes would be more likely to stray from proper play since they do not have as much time to think about what moves to make.

We decided that transforming the rating metrics into a rating difference and rating average group would be easier to interpret in the decision tree. Since the individual ratings are not that important, but rather how they relate to each other is more important. So by having the difference attribute, we can look at how much better or worse the players are from each other, and the rating average can determine at what level of gameplay these people are playing at, since we expect people at higher ratings to play more accurately than people at lower ranks. We also edited the Opening ECO to just have the letters and not the digit since the letters A, B, C, D, and E indicate 5 different styles of openings. This makes it easier to split in the decision tree than just using the original Opening ECO since technically there are 500 different ECO codes. This groups these 500 codes into 5 categories, making it easier to interpret.

The new attribute is being processed because we think that the number of moves until a unique game state, when paired with who made that move to reach the unique game state, can provide significant information on who performed better, since players that reach the unique game state early would most likely have made an error compared to someone who reached that state later.

We decided to use a decision tree classification technique because we are curious about figuring out how the attributes actually affect the game. As of now, we have just made assumptions about which attributes we think would affect the outcome of the game, but we have no evidence on this. So, by creating a decision tree classifier, we can easily investigate which attributes are more important than other attributes. Also, because our hypothesis states that we think the number of moves until a unique game will greatly affect the outcome of the game, as well as who made it, by using a decision tree and comparing it to the baseline tree, we can actually see if this attribute makes a significant difference.

We want to use a bagging ensemble with this decision tree classifier because we think that this would be a good way to reduce the amount of variance in our model. Also, since decision trees are prone to overfitting, we thought a bagging ensemble would be better than a boosting ensemble, since boosting can lead to even more overfitting. This way, we eliminate the risk of more overfitting. We chose the hyperparameters of the number of trees and the number of samples to fine tune because for the number of trees, we can increase the number of trees until the model performs better without having to worry about overfitting. As for the number of samples, we can check if accuracy increases if the sample size increases, which could lead to a better model. We decided to use repeated stratified 10-fold cross validation because it repeats the k-fold cross validation process, which can help get a better average cross validation score than if we had just run it once. We decided that 10-fold would be sufficient since our sample size is relatively large.

3 EXPERIMENT

3.1 Dataset

The dataset that we chose to use for this project comes from Kaggle and was created by Kaggle user Mitchell Jolly. The dataset consists of 20,058 chess games from the chess website Lichess. The author gathered the data by using the Lichess API to select all the users of several Lichess teams and gathering their individual game histories. Out of the 16 different attributes, the following 9 attributes are relevant to our project:

- Whether the game was rated or not (T/F)
- Total number of turns
- Game Status (mate, resign, draw, out of time)
- Winner (black, white, draw)
- Time Increment
- White Player Rating
- Black Player Rating
- All Moves in Standard Chess Notation
- Opening Name

The source of this dataset and its attributes are particularly useful to us for several reasons:

(1) Because our project relies on our players having some base knowledge of chess and chess strategy, it is important that our dataset accurately reflects that. Lichess is a chess platform that is popular with those who play chess online and would typically not be the first place someone totally new to chess would find immediately online. Furthermore, the author of the dataset collected games from Lichess users who are members of one or more Lichess teams. A small assumption can be made that a user who takes the time to join a Lichess team is more likely to be active chess players than a user who is not on a team. Our hope is that because of how the data was collected, the subset of users whose games were collected will have games that represent basic chess strategy.

(2) The meta attributes that describe the conditions of each game such as whether the game was rated or not, the total number of turns, and the time increment that the players chose to use are all important for pre-processing the dataset, each for their own reason.

   (a) The rating attribute is important because it allowed us to filter out games that were played competitively. While not every unrated game is a “joke game” that does not always represent chess strategy, we knew we did not want to include games where players did not always have some motivation to play well and try to win. Keeping and analyzing only the rated games allowed us to get closer to our ideal dataset of games played seriously, as rated players have motivation not to lose rating.

   (b) The total number of turns attribute is useful for eliminating outliers from the dataset. In an online game setting, it is frequent that a player may resign or quit the game if they make a mistake early in the game and get frustrated or do not believe that they have a chance of winning. Although a few real games really only do last a couple of turns due to a blunder or serious mistake, we wanted to remove all games that only lasted a few turns from our analysis to make sure our dataset better represented only traditional games.

   (c) The time increment attribute is interesting because it shows how long each player has to make a move and how much time they are given after making a move. A time increment of 15+5 translates to each player starts the game with 15 minutes and receives 5 seconds after each move they make. The fastest time increment that Lichess offers is 1+0 called bullet chess where players are frantically playing to avoid running out of time. Because of how little time each player has to think about each move, common bullet strategy involves playing unorthodox moves that are sometimes suboptimal in order to throw the opposing player off. We wanted to eliminate any games from the dataset that used bullet time increments because of how player's perform and behave under these time restrictions.

3.2 Hypotheses

(1) Our first hypothesis is that having knowledge of how many turns each game took to reach a unique game relative to the rest of the games in the dataset will improve the base model's performance. We expect that for games where the number of turns to reach a unique game is significantly low, it will be more likely that the player who made the unique move to lose the game. We are led to believe this because we would expect players who deviate considerably early from the decision tree of traditional openings to have made a mistake that their opponent can capitalize upon and use to win the game. We can answer this hypothesis by comparing the resulting confusion matrices from the base model and the newly created model that has access to the new feature.

(2) Our second hypothesis is that the new "turns to unique game" attribute will not be as useful for higher ranking players as it will be for average rated players. We believe that higher rated players will have a high average "turns to unique game" attribute because of their experience playing the game and opening knowledge. In other words, we do not expect the more experienced players from the dataset to diverge from expected play until the middlegame or late opening stages of the game. But in the rare instance where a highly ranked player does reach a unique game in the opening stage, we would expect the move not to be a blunder or serious mistake because of how uncommon it is for a skilled player to fail an opening. We can answer this hypothesis by first defining what range of ratings defines a highly rated player, and then comparing the confusion matrices of the base model and the new model using only the highly rated players data.

3.3 Experimental Design

The first step of preprocessing the data was to remove unnecessary columns and duplicate row values. We imported the dataset into a Pandas dataframe and used built in methods to drop the columns created-at, last-move-at, victory-status, white-id, black-id, opening-name, and opening-ply. We decided these columns did not provide us with much meaningful information that would be useful in our analysis, and would likely add unnecessary complexity to our model. The columns created-at and last-move-at contained data about the length of the game, but the columns were rounded to too few decimal places and did not provide us with useful information.

During this step, we made transformations to several columns in order to make them easier to use in our model. For example, we transformed the two columns for white-rating and black-rating into average-rating and rating-difference. Rating-difference was calculated as white-rating - black-rating, and gives us a value corresponding to the balance in player skill between the two players. Average rating was simply the average of the two player's ratings, and is what we used to measure the level of play. Two players with a high average rating might play completely differently to two players with a lower average rating, even if the difference in player rating was the same. Using these values meant we kept all of the original information from the base data, but we transformed it into a form that would be easier to create splits on.

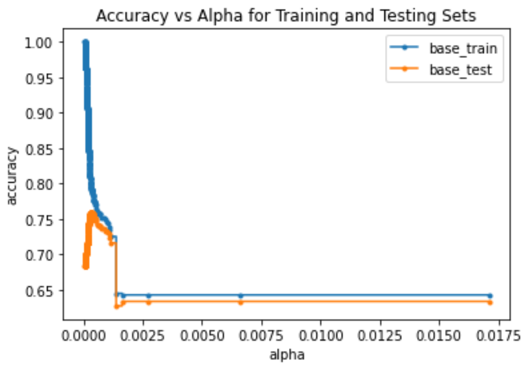
The other columns we transformed were the time-increment and opening-eco columns. Time-increment was originally formatted [minutes]+[seconds], where the minutes value is each player's total time at the start of the game, and the seconds are how many seconds are added to the clock for each turn. We classified games that had a base minute value less than 10 as a "blitz" game, between 10-30 minutes as "rapid," and greater than 30 minutes as "classical." This allowed us to transform this column from a difficult and widely varying set of time scales to three main groups. This will make the data much easier to classify and split on. The opening-eco value is a standardized code used to classify opening moves. Every opening move has a unique code associated with it, where the first letter, A through E, is a class of openings, and the following number specifies which opening in particular. To aid in classification, we decided to drop the number and only keep the letter, separating each game into five categories based on the type of opening played.

In addition to modifying existing columns, we also generated some new columns based on the moves performed during the game. We first sorted the data based on the list of moves, and created a method called "get-turns-to-unique" which looks at the moves from a game and compares it with the moves done in the surrounding games. Similar games will be next to each other in the sorted list. Then we determined how far into the game you had to go to find a unique state that was not present anywhere else in the dataset. This gave us a new metric, turns-to-unique, which we will use to try to improve our classification of games played. We also used this data to determine which player made the first "unique" move, i.e. which player made the move that first deviated from another game in the dataset.

The code written for this project can be found here at NCSU's GitHub (https://github.ncsu.edu/arhughle/csc422-chess).

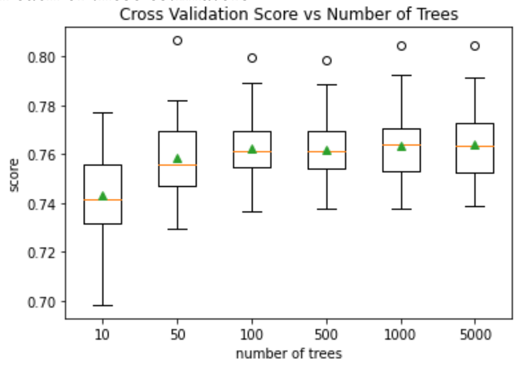
4 RESULTS

We started with creating a baseline model. We decided to use a Decision Tree Classifier that would eventually be used in a bagging ensemble method. But before we developed the bagging ensemble method, we wanted to create a Decision Tree Classifier that would have the best parameters. From the scikit-learn python library, we were able to post-prune our initial decision tree with cost complexity to determine the alpha that maximizes the accuracy of our test data. The graph below shows the Accuracy vs Alpha values from the decision tree classifier:



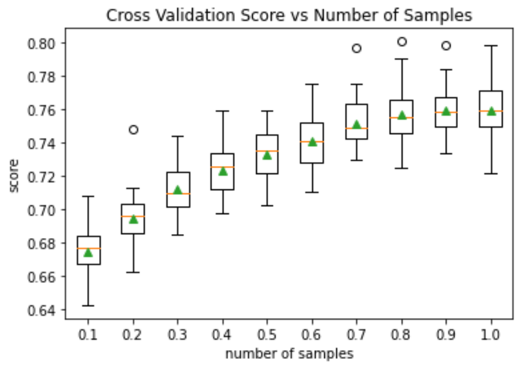
From the graph, the alpha that maximizes the accuracy of our test data is 0.000310309306649608, and it gives us an accuracy of 0.7593323216995448.

So, in our bagging ensemble, we used the Decision Tree Classifier with the cost complexity pruning alpha value of 0.000310309306649608. We wanted to fine tune the hyperparameters of our bagging ensemble, specifically the number of estimators as well as the number of samples. We decided to figure this out by using repeated stratified 10-fold cross validation. For the number of estimators we chose to try 10, 50, 100, 500, 1000, and 5000 estimators. Below is a box plot showing the cross validation scores for each bagging ensemble with each of these estimators:

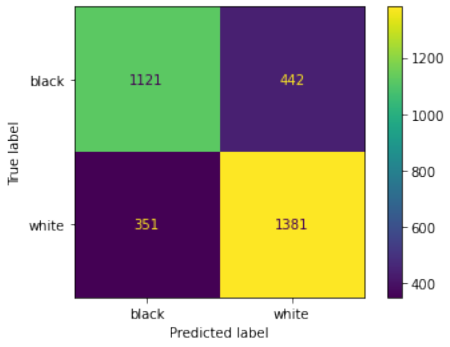


From this above graph we can see that the mean cross validation score starts to plateau starting at 100 decision trees. The score does increase slightly, but for the time and cost that it takes to perform repeated stratified 10-fold cross validation on 5000 estimators, it is not worth increasing the amount of parameters past 100. Jason Brownlee's article about how to use a bagging ensemble with decision trees was helpful with the code implementation to achieve the results from above [1].

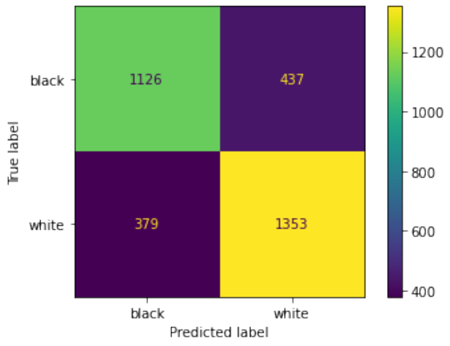
We decided to use k-repeated stratified 10-fold cross validation to get the best value for the amount of samples. From the table below, it can be seen that at 100 estimators, using the default max samples value of 100% is the best for this ensemble method.



The following confusion matrix was then produced using this baseline model and the test data.



For the construction of the new model, we used the same steps that we took to create the baseline model, with the only difference being that our new model would have access to the "turns_to_unique" and "unique_by_white" attributes. Once the Decision Tree was made and the bagging ensemble completed, we produced the following confusion matrix.



Comparing the accuracy scores of the two matrices, the baseline model (0.759) and the new model (0.752) are almost identical in correctly classifying who won the match.

5 CONCLUSION

As we can see from the results above, both models showed very similar levels of accuracy. There are a few reasons we believe this occurred. The first is that we did not have enough data to draw meaningful conclusions from. There are millions of possible chess games, and our data set only included a few thousand games. This meant that the vast majority of games reached a unique state within about two to three moves, making it difficult to tell which games were predictable games, and which were not. Most randomly generated games would have a value of around two to three because there are much fewer possibilities in the early game than there are after several moves.

We also only measured how many turns it took to get to a unique state, which doesn't give us as strong of a metric as a more complex "predictability" measure would. For example, if two games deviate from a known opening immediately, but match each other for eight turns by chance, both of those games would have a "turns to unique" value of nine, but if ten games all followed a common but short opening of only three turns and diverged immediately after, those ten games would all have a "turns to unique" value of four, despite following a more predictable pattern.

On a similar note, common chess openings can vary in length wildly, from as short as two or three moves in some cases to as many as fifteen or more. Our metric is supposed to measure if a player is unpredictable by seeing if they break from a common opening before it is completed, but a player can break from a longer opening part-way through on turn eight or nine and our metric would count that as more predictable than a player who follows a short opening until the end, then diverges from other games once the opening is finished.

In short, the number of turns it takes to get to a unique state has too many chances to be wrong or misleading as a measure of unpredictability, and we did not have enough data to overcome this randomness. It's also possible that the unpredictability of a game or player does not have a meaningful impact on the outcome of the game. If we were to run this experiment again, we could use a more reliable measure of unpredictability by looking at the number of games that had the same first move, second move, etc, and creating some sort of function that could give us a continuous measure of predictability rather than a discrete integer value.

REFERENCES

[1] Jason Brownlee. 2020. How to Develop a Bagging Ensemble with Python. (Sep 2020). https://machinelearningmastery.com/bagging-ensemble-with-python/

[2] Claude E. Shannon. 1959. Programming a Computer Playing Chess. *Philos. Mag.* Ser.7, 41, 312 (1959).