Assignment 2

MIT 805

Connor McDonald
u16040725

Big Data

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science
University of Pretoria
14 November 2021

1 Overview

The dataset being used for this assignment is comprised of 6.25 million online chess games recorded over a month and published online by lichess.org. The purpose of collection was to analyse the techniques used by highly skilled players in order to train new players more efficiently. Since a chess board consists of 64 squares and 32 pieces, the number of unique games that can played is incredibly large. This can create a lot of noise when trying to identify "good" games. However, highly skilled players often play with some sort of strategy, based on well-documented openings. Therefore, we can filter out a lot of the noise by removing all players with a rating of less than 1500. The filtered dataset was 3.3Gb in size and contained 4.6 million rows. Figure 1 shows that the majority of players in the dataset fell above the rating threshold.

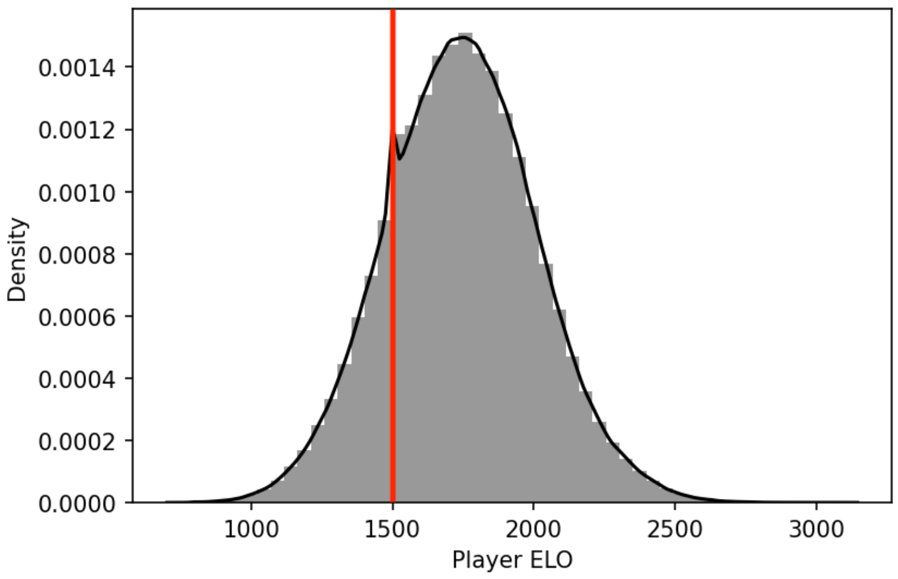


Figure 1: Distribution of player ratings

The data was analysed using MapReduce, which is a processing technique and a program model for distributed computing based on the Java programming language. However, the algorithms where coded in Python using MRJob, which is an active Framework for MapReduce programming or Hadoop Streaming jobs. The ultimate goal of these algorithms was to add business value by providing insight into the player-base of lichess.org and using these insights to train new players, improve user experience and optimise app performance. This was done through four analyses which focused on: player distribution, opening efficiency, opening similarity and player skill levels. These analysis are discussed in detail in the next section and the repository containing all the information relating to this project can be found on GitHub[1].

[1] https://github.com/u16040725/MIT-805-Project/tree/main/Assignment%202

2 Data Analysis

2.1 Player Distribution Analysis

The first analysis looked at player rating, time of day, and game type to see if any patterns could be identified. To express this problem in terms of MapReduce, a key containing multiple fields was needed. Rating was rounded to the nearest 100 and only the hour of the time stamp was collected, these two fields were combined with game type to make a single key structured as (game type, time, rating) and contained a value of 1. This was then inputted into the reducer were they keys were grouped and their values were summed to get the total number of occurrences for each key. The file containing the code for this algorithm can be found here: player_distribution.py .

This algorithm yielded the plot shown in figure 2, where the size of the circle represents how many times that rating/time/game type combination was observed in the data. It was observed that most players play between 10am and 8pm during the day, and that the most popular game mode is blitz. This information can be used to add business value by encouraging new players to play these game modes using in-app prompts. Furthermore, if the app servers start taking strain, the plot helps identify the periods of heavy user traffic and when a load balancing system should kick in.

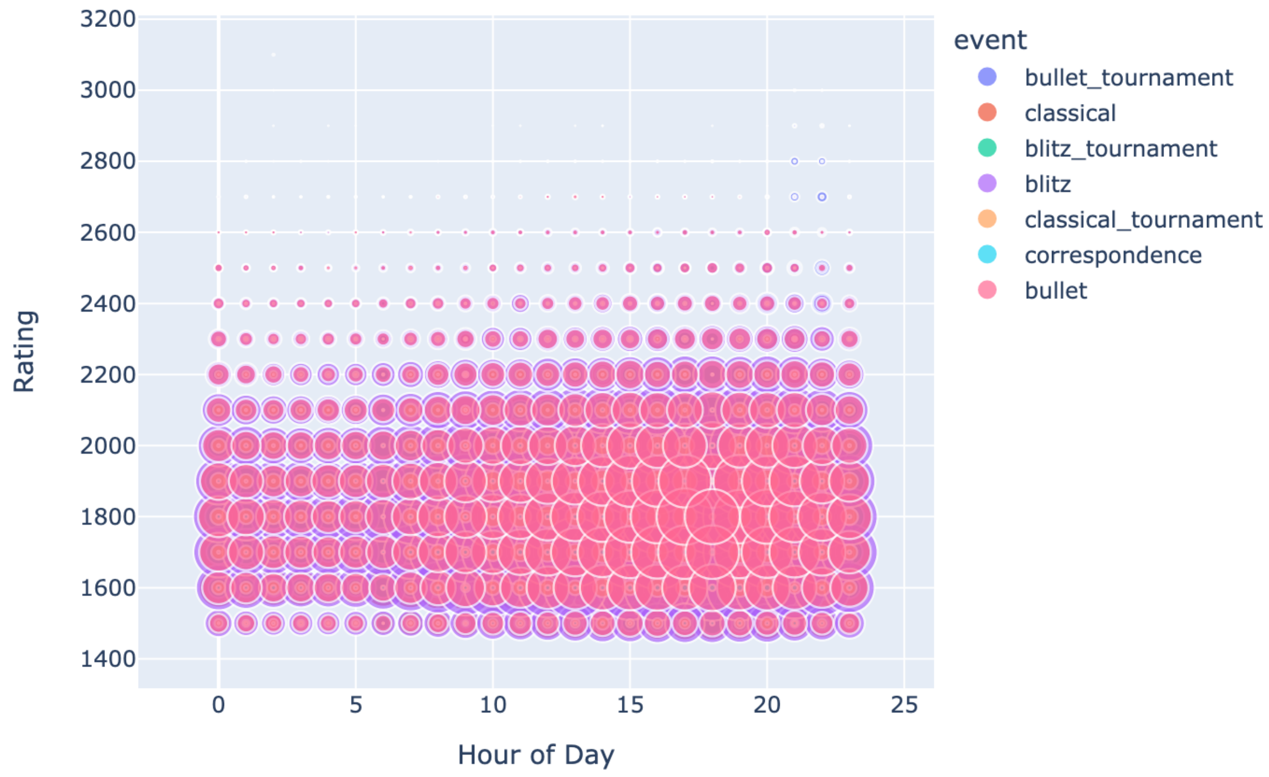


Figure 2: Rating vs. Time vs. Game Type

2.2 Opening Efficiency Analysis

The next analysis aimed to identify the best openings based on win percentage. Initially, this was done across all game types to create a bench mark, and see if the outcome changed when looking at the most popular game mode. To calculate the win rate of each opening, two algorithms were needed, the first algorithm used opening as a key and a value of 1, the value count was then summed in the reducer to get the total occurrences for each opening. This produced a dataset with "opening" and "number of occurrences" as its only two columns. The next algorithm also used opening as it's key, but the value was dynamically generated as either a 1 or 0 depending on whether the game was won or not. This yielded a dataset with the total number of wins per opening. These two datasets were then joined on the "opening" column to create a combined dataset with the opening, number of wins, and number of occurrences, and was subsequently used to calculate the win percentage per opening. All records with an occurrence count less than 1000 were filtered out so that an adequate sample size was ensured when calculating win percentage. The results of this analysis yielded figure 3 which shows that the kings pawn opening has a clear advantage with a win percentage around 93% which is significantly higher than the other openings in the top 5 based on win percentage. To see the code for these algorithms please click the following hyperlinks: Algorithm 1 and Algorithm 2.

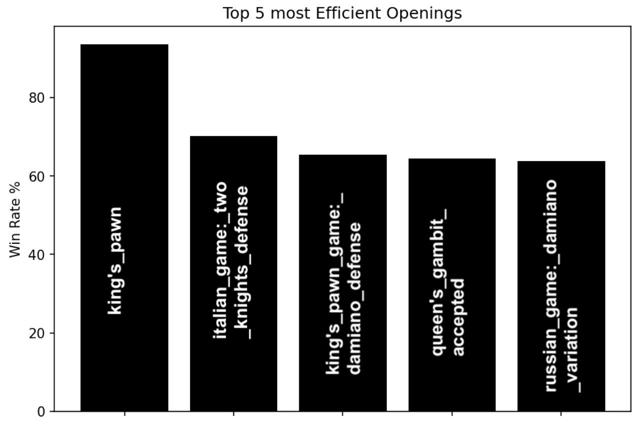


Figure 3: Top 5 Openings by Win %

To see if these results changed in the most popular game mode, a third algorithm was created to get the counts of each game mode (Algorithm 3). This was created by using the game mode as the key and a value of 1, and then summing the values grouped by key in the reducer. This yielded figure 4, which showed that the blitz game mode was most popular. A subset of the data was then created, which contained only the blitz game mode and this subset was run through Algorithm 1 and Algorithm 2 to create figure 5. Here it was observed that the kings pawn opening did not even make the top 5 for the blitz game mode and there was no opening that had an obvious advantage over the others. One thing worth noting is that the "queens gambit accepted" opening was observed in the top 5 openings ranked by win percentage for both scenarios (see figures 3, 5).

This information can be used to add business value by educating new players about openings that increase their chances of winning. Furthermore, the same opening analysis can be conducted across all game types and be used to inform players how to adjust their play styles to suit the current game type.

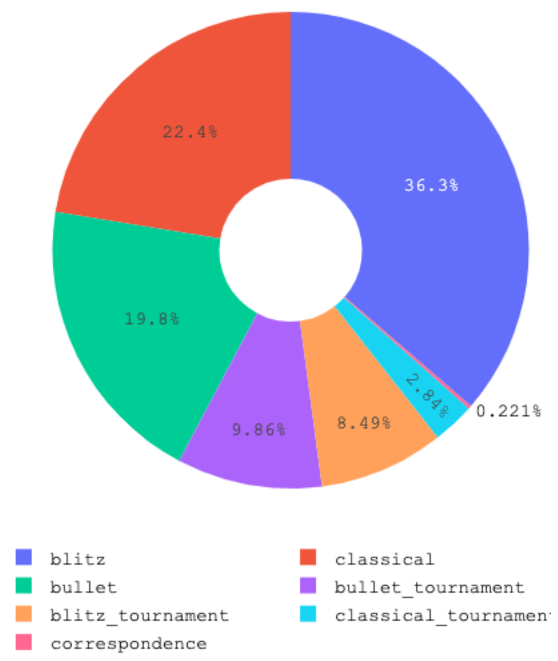


Figure 4

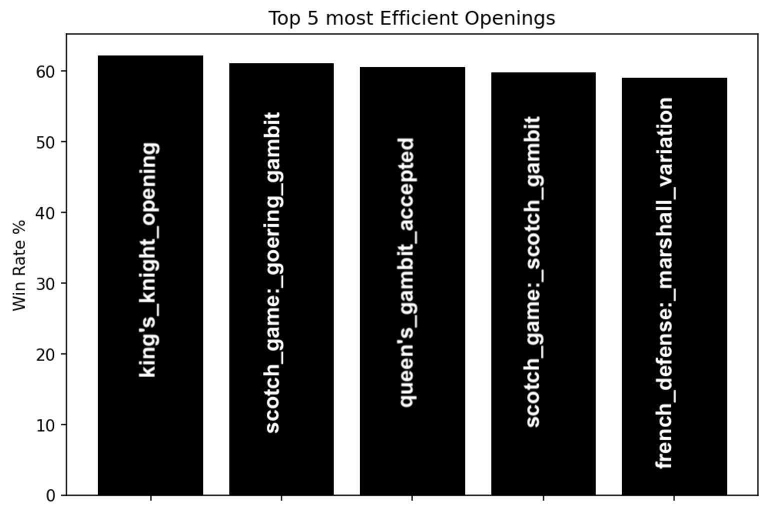


Figure 5: Highest win rate for openings in blitz game mode

2.3 Opening Similarity Analysis

The algorithm for this section is a complex multi-step similarity analysis which may be used as a recommendation engine. This is done by using "item-based collaborative filtering" where the algorithm identifies every pair of openings used by the same person, then measures the similarity in ratings across all users who used both openings by using a cosine similarity function, and finally, sorts the results by similarity strength. Figure 6 explains the inner workings of the algorithm in a visual way and the code can be found here.

Table 1 shows a simulated recommendation for a player using the "Queen's Pawn Game" opening. This could be exceptionally useful in training players as it can make them aware of different openings which may compliment their play styles, and possibly make them aware of new openings that they have not seen before. Furthermore, the same logic can be applied to creating a recommendation engine for recommending game modes, however, it would require users to provide a rating of sorts to each game mode. These recommendations can be refined by filtering for a specific number of times that both openings had been used, or a similarity threshold may be put in place. For the purpose of this example, I filtered out all opening pairs with a similarity score less than 0.95 and number of times used less than 100.

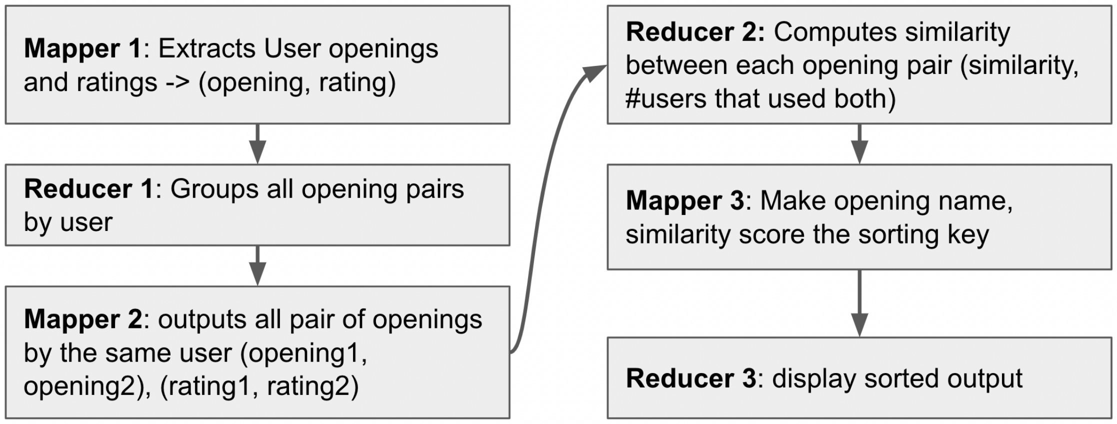

Mapper 1: Extracts User openings and ratings -> (opening, rating)
Reducer 1: Groups all opening pairs by user
Mapper 2: outputs all pair of openings by the same user (opening1, opening2), (rating1, rating2)
Reducer 2: Computes similarity between each opening pair (similarity, #users that used both)
Mapper 3: Make opening name, similarity score the sorting key
Reducer 3: display sorted output

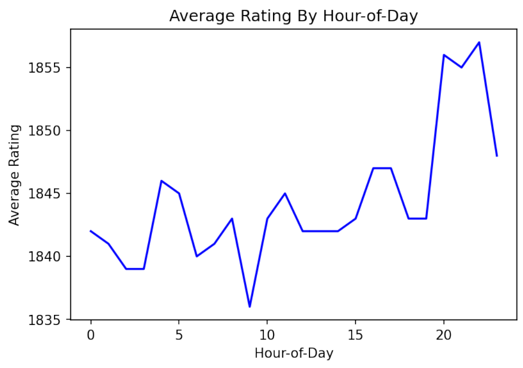
Figure 6

Table 1: Recommended Openings for Queens Pawn Game

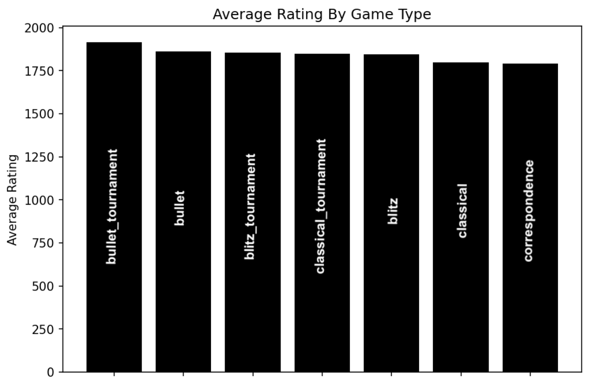
Opening Used	Recommended Opening	Similarity	# Both used
Queen's Pawn Game	Horwitz Defense	0.9996	805
Queen's Pawn Game	Indian Game	0.9996	737
Queen's Pawn Game	Modern Defense	0.9996	424
Queen's Pawn Game	Old Benoni Defense	0.9994	448
Queen's Pawn Game	Dutch Defense	0.9992	162

2.4 User Skill Level Analysis

The last analysis used two simple algorithms that analysed the average rating of players throughout the day, and the average rating of players in each game mode. Both of these algorithms used the same logic with the only difference being the key, where one used the time of day as a key and the other used game type as a key. The rating was used as the value in both instances and subsequently turned into an average rating in the reducer. The code for these algorithms can be found here. Figure 7a shows that the average rating of players increases throughout the day and figure 7b shows that the tournaments versions of each game mode typically have a higher rating than the standard version. This is expected as players may be playing more competitively in tournaments. This information can be used to improve user experience as the app can recommend times of day and game modes that are more forgiving for newer players, or more challenging for players that wish to improve.



(a)



(b)

Figure 7

3 Suggested Implementation

The data used for this assignment was comparatively small when considering "Big Data", this was so that the algorithms could be run locally and produce results without the need of a cluster. However, as algorithms get more complex they require more computing power, and the need for a Hadoop cluster becomes more apparent, this was observed in the opening similarity analysis where a subset containing only 100 000 records of the original datset had to be used, as it was not possible to run the algorithm with the full dataset locally. It is suggested that these algorithms be implemented on a cloud computing infrastructure such as Amazon Elastic MapReduce. This offers much more efficiency and scalability when compared to a single computer. Another added benefit of this is that redundancy can be added by duplicating mappers, reducers and data across multiple machines in the cluster, which protects the user from hardware failures.