Matthew Kingsbury, Kaushik Chintam, Will Ndudirim - Functional Programming Group Coursework

What is the Application?

The application uses an API called LichessAPI which allows queries to retrieve data from the lichess.org platform database. This database stores all the matches played on the lichess online platform. It allows us to retrieve relevant player and game information, which we will store in the database. The user can add, retrieve and delete the data, as well as pass the data through a machine learning model to perform chess match predictions. An example output of the 'getPlayerInfo' command looks like the following:

```
Player Information:
Username: birdlarry
URL: www.lichess.org/@/birdlarry
Total Games Played: 7770
Total Wins: 3650
Total Draws: 223
Total Losses: 3897
Ratings...
bullet: 1936, blitz: 1775, rapid: 1388, classical: 1600
```

We retrieve the url, all the games played, wins, draws and losses for the given username. We also use a join to display the user's ratings for the four most common chess time controls. The data is retrieved through specific API calls, and the results are put through parsed functions which find specific features of the ndjson/json responses and extract the relevant data accordingly. The program is separated into 'Main.hs' in app, and 'Database.hs', 'Fetch.hs', 'Parse.hs', 'Types.hs' and 'Model.hs' in src. Separating by the features of the program reduces clutter and makes it easier to locate specific functionality.

The application makes use of a database which contains three tables: players, gametypes, and games. Players and gametypes are connected with a primary/foreign key relationship as users may have a different number of ratings due to not playing a certain game mode. It made it easier to calculate averages and differentials between ratings versus storing all the data in one table. The games data stores relevant data for games extracted from the API which are used for training the classification model.

How to Run the Application

The application is executed with the 'stack run' command. The user passes in any desired commands and arguments in order to execute specific actions. As the program draws from the arguments given, each execution will start with 'stack run', contain some command (and potentially one or two arguments) and then some action will be performed. Incorrect commands or arguments are handled with exceptions.

In the application, the following commands are available for the user:

1. "getPlayerInfo [p]" → retrieve a player's lichess information from the database
2. "addPlayer [p]" → add a new player to the database
3. "deletePlayer [p]" → delete a player from the database
4. "initialiseModel" → initialise the machine learning model
5. "predict [p1, p2]" → predict a winner between two players in the database
6. "export" → writes the database to file in a json format"

Extra challenging feature
The extra feature in the application is the two commands:

"initialiseModel" → initialise the machine learning model
"predict [p1, p2]" → predict a winner between two players in the database

Above initialises and trains the machine learning model, on which the user can compare two players who have been stored inside the players table in the database. We applied feature engineering to extract the specific aspect of the data we wanted to train on (the difference in rating between player 1 and player 2) and then split the data 80/20 into training and testing datasets. We were looking to model the correlation between the difference in ratings and the probability of winning - our hypothesis was that if player 1 has a much stronger rating than player 2, they would be more likely to win, and vice versa.

The model is trained on 5 players * max 200 games ~ 1000 datapoints

Each datapoint is parsed, feature engineered, passed through the logistic regression function, classified, and compared to its ground truth value. The final weights of training are written to a text file “weights.txt”, then read back in when we are performing the predictions. This is a long and extensive process that required a lot of difficult, complex code, and proved very challenging to implement - Haskell does not have the same extensive libraries for performing machine learning compared to a language like Python, and we implemented the entirety of the model functionally instead of utilising instances of classes.

We believe that implementing a simple regression-based binary classifier was a good choice for an extra challenging feature as the data we receive from the API contains many numerical values that are very much open for extensive data analysis, and this was a fantastic opportunity to implement a simple proof-of-concept within a functional programming language. We all had prior experience developing machine learning models, however doing it within Haskell was a new challenge that allowed us to apply our skills in a new and interesting way while drawing from our prior knowledge.