CSE 519 - Data Science Fundamentals
PROJECT FINAL REPORT
How Good is a Chess Player?

1 Objective

To build a model that will predict the Elo score of a chess player. Additionally, we are motivated to guess the type of game depending on the time limit of the game and predicting the result of the game, given a board position as an input.

2 Introduction

Chess is a two-player strategy board game played on a checkerboard with 64 squares arranged in an 8×8 grid. Each player begins with 16 pieces: one king, one queen, two rooks, two knights, two bishops, and eight pawns. Each piece type moves differently, with the most powerful being the queen and the least powerful the pawn. The objective is to checkmate the opponent's king by placing it under an inescapable threat of capture. To this end, a player's pieces are used to attack and capture the opponent's pieces, while supporting each other. During the game, play typically involves exchanging pieces for the opponent's similar pieces, and finding and engineering opportunities to trade advantageously or to get a better position. In addition to checkmate, a player wins the game if the opponent resigns, or (in a timed game) runs out of time. There are also several ways that a game can end in a draw [1].

A chess rating system is a system used in chess to calculate an estimate of the strength of the player, based on his or her performance versus other players. They are used by organizations such as FIDE, the US Chess Federation (USCF or US Chess), International Correspondence Chess Federation, and the English Chess Federation. Most of the systems are used to recalculate ratings after a tournament or match but some are used to recalculate ratings after individual games. In almost all systems a higher number indicates a stronger player. In general, players' ratings go up if they perform better than expected and down if they perform worse than expected. The magnitude of the change depends on the rating of their opponents. The Elo rating system is currently the most widely used.

The ELO rating system is a method for calculating the relative skill levels of players in zero-sum games such as chess. It was developed by the Hungarian physicist Arpad Elo in the 1950's and adopted by the world chess federation (FIDE) in 1970. Since its development, the system has been adopted with various modifications by many national chess federations. Today it is impossible to imagine tournament chess without a rating system [2].

The Elo rating system calculates for every player a numerical rating based on performances in competitive chess. A rating is a number normally between 0 and 3000 that changes over time depending on the outcomes of tournament games. When two players compete, the rating system predicts that one with the higher rating is expected to win more often. The more marked the difference in ratings, greater the probability that the higher rated player will win [3].

In this project we are interested in anticipating how good a chess player is. Given a record of the match, we will be predicting the ELO rating of both the player. In addition to predicting the Elo score we also predict the type of the game by looking at the time taken for each move.

3 Data Pre-Processing

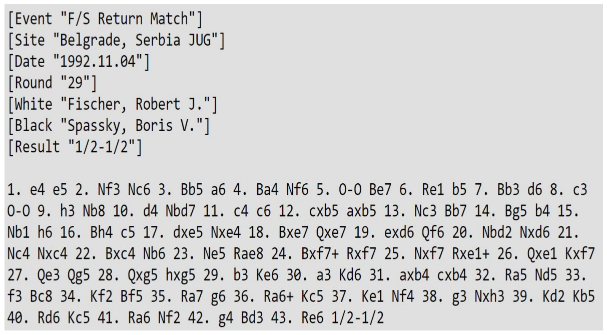
3.1 Lichess Dataset

Lichess is an open-source internet chess server. The data of all games since 2013 is available in LiChess Database. There are over 800,000,000 games in the database, each tagged with ratings of both players and the speed [4].

Following data fields are available in the Lichess database for every game:

1. Event	11. BlackRatingDiff
2. Site	12. ECO
3. Date	13. Opening
4. Round	14. Termination
5. White	15. TimeControl
6. Black	16. UTCDate
7. WhiteTitle	17. UTCTime
8. BlackTitle	18. WhiteElo
9. Result	19. WhiteRatingDiff
10. BlackElo	20. Moves

The format of the data in LiChess Database is Portable Game Notation (PGN). PGN is a plain text in computer-processable format for recording chess games (both the moves and related data), supported by many chess programs. PGN is structured "for easy reading and writing by human users and for easy parsing and generation by computer programs." The chess moves themselves are given in algebraic chess notation. Each game contains a header file and a sequence of moves made by each player [5].



```
[Event "F/S Return Match"]
[Site "Belgrade, Serbia JUG"]
[Date "1992.11.04"]
[Round "29"]
[White "Fischer, Robert J."]
[Black "Spassky, Boris V."]
[Result "1/2-1/2"]

1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 4. Ba4 Nf6 5. O-O Be7 6. Re1 b5 7. Bb3 d6 8. c3
O-O 9. h3 Nb8 10. d4 Nbd7 11. c4 c6 12. cxb5 axb5 13. Nc3 Bb7 14. Bg5 b4 15.
Nb1 h6 16. Bh4 c5 17. dxe5 Nxe4 18. Bxe7 Qxe7 19. exd6 Qf6 20. Nbd2 Nxd6 21.
Nc4 Nxc4 22. Bxc4 Nb6 23. Ne5 Rae8 24. Bxf7+ Rxf7 25. Nxf7 Rxe1+ 26. Qxe1 Kxf7
27. Qe3 Qg5 28. Qxg5 hxg5 29. b3 Ke6 30. a3 Kd6 31. axb4 cxb4 32. Ra5 Nd5 33.
f3 Bc8 34. Kf2 Bf5 35. Ra7 g6 36. Ra6+ Kc5 37. Ke1 Nf4 38. g3 Nxh3 39. Kd2 Kb5
40. Rd6 Kc5 41. Ra6 Nf2 42. g4 Bd3 43. Re6 1/2-1/2
```

Sample PGN Game Data with Minimum Headers

One of the challenges was to extract this data and convert it into a readable format for further use. The month-wise data files available in the Lichess database is in bz2 format. We first had to convert the file from bz2 to pgn format using the bz2 python library to get the header fields and moves as shown above. Parsing of the pgn data was done using the chess.pgn python library for further processing.

3.2 Feature Extraction

Each file of PGN contains multiple games and each game contains data about the game, players and the list of moves. We iterate through each of the PGN files to get the required features. We perform this activity in batches using the python-chess API. Further, for each game we iterate through the moves made and every iteration gives us a board that will tell us the position of the pieces at the time of that move.

Following features are extracted using the moves field in the game:

i. **moves**: Every move made in a game. From this following the features were extracted:
- **total_moves**: Total number of moves made in a game.
- **white_moves**, **black_moves**: Total number of moves made by white and black player respectively for each game.
- **first_move**: It gives the opening move of white player and the first move of the black player. Every next move after the opening move stores all the previous moves made before that.
- **second_move**: It gives the moves till the second move of each white and black player respectively since the second move is highly dependent on the first move.

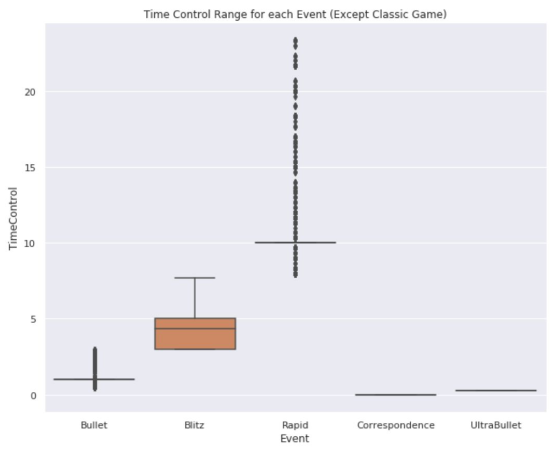
ii. **clock**: Time taken for each move, clock state at the beginning of the move. Following features were extracted from Clock:
- **play_time**: Amount of time utilized by each player for the entire game.
- **time_remaining**: Amount of time remaining after the entire game for each player.

iii. **SAN** (Standard Algebraic Notation): It contains piece information, end position, capture, check and checkmate information, quality of the move under a few categories, and special moves such as pawn_promotions and castling.
- **captures**: The number of pieces captured by both white and black players.
- **checks**: A Boolean value indicating if checkmate was done during the game.
- **king_castle**: Number of king castle moves made by the player.

- **queen_castle**: Number of queen castle moves made by the player.
- **pawn_promotes**: Number of pawn promotions to special pieces X.
- **end_board_position**: This is split into 64 columns, where each cell gives the piece present or NaN if no piece is present. The Pieces are named as R,N,B,K,Q,P,r,n,b,k,q,p, where the uppercase alphabet stands for white pieces and lowercase stands for black pieces.

iv. **NAG** (Numeric Annotation Glyph): NAG gives us the information about the type of moves made. The features extracted are:
GOOD_MOVE, **MISTAKE**, **BRILLIANT_MOVE**, **BLUNDER**, **SPECULATIVE_MOVE**, **DUBIOUS_MOVE**

v. **TimeControl**: The time control value. This was split into 3 columns:
- **FixedTime**: The base time alloted for a game in seconds.
- **Increment**: Extra time per move.
- **TotalTime**: Total time of the game
  = FixedTime + (No. of Moves * Increment)



Time Control Range for each Game Type

After each move made by the player, we evaluate the board using Stockfish engine. This engine gives us the score by evaluating the positions of the pieces at that time of the game. We extract the following features using stockfish engine:

i. **eval**: This gives us the average score of moves for both white and black players.
ii. **early_game_eval**: It gives the average score of the first 25% of the game.
iii. **middle_game_eval**: It gives the average score of 25% - 75% of the game.
iv. **end_game_eval**: It gives the average score of the last 25% (75%-100%) of the game.

3.3 Pre-processing

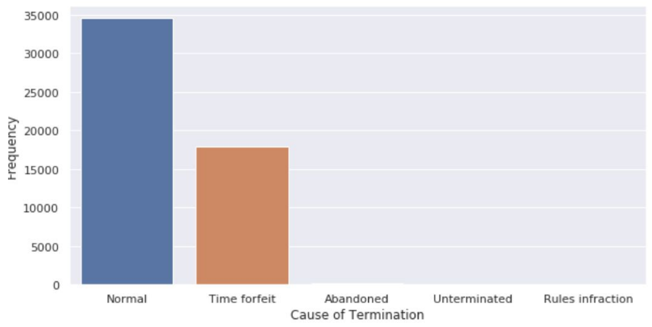
The original raw data obtained was further processed using the following steps:

3.3.1 Drop Unwanted Columns

We dropped columns like Site, Round and Date since it doesn't contribute to the analysis. The columns 'WhiteTitle' and 'BlackTitle' contain a significantly large number of NaN values and hence we dropped them as well.

3.3.2 Replace NaN values

Few columns had NaN values in some rows and few columns had values as '-' or '*' instead of NaN. Since such rows were significantly less as compared to the size of the dataset, we dropped these rows. The rows with Termination = 'Abandoned' was dropped as this data did not add any value to our analysis and very few games were abandoned.



Frequency of Cause of Termination

3.3.3 Label Encoding

In order to run Machine Learning algorithms, we used Label Encoding to convert the columns with categorical values to numeric values.

3.3.4 Add Dummy Board Position Rows

When we label encode the entire dataset for further modelling, it is possible that in each of the 64 columns generated to hold piece present in the board square, every board piece does not exist. If this occurs, a label won't be present for the missing pieces and during prediction, and this will cause inaccuracies in prediction.

In order to improve this, we add 12 dummy rows with exclusive ID. Each of these rows contains one unique piece in all the 64 columns. There are 12 unique piece representations: R,N,B,K,Q,P,r,n,b,k,q,p. After the process of label encoding, we delete these 12 dummy rows. The dataset returns to the original dataset whilst having all possible piece labels in the board position.

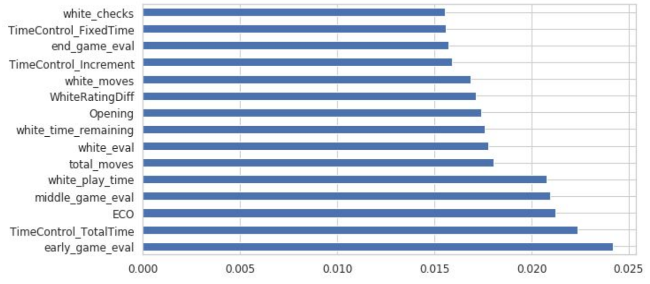
3.3.5 Split dataset into white and black datasets

Few features are common to both white and black players. But, there are many features like EloRating eval, etc. which are characteristics for each player. Hence, we create two new dataframes for white and black separately which consists of the features common to both as well as individual characteristic features.

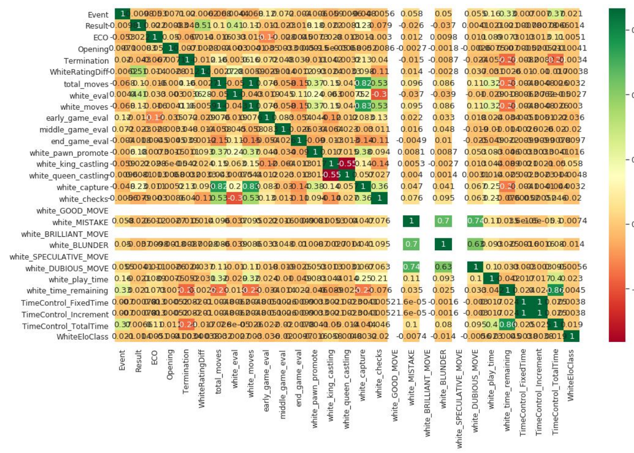
4 Implementation

4.1 Predict Elo Rating of Player from Transcript

To select the best features for building the baseline model, we apply feature importance technique. The top 15 features which are highly correlated to the features of the white player with respect to WhiteElo are as shown in the figures below.



Best Features with respect to White Elo

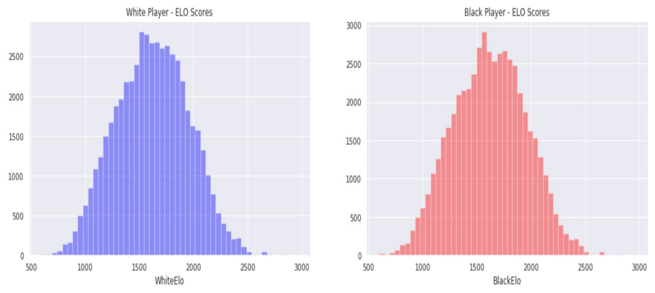


Correlation matrix for WhiteElo

Similar features and correlation matrix are obtained with respect to Elo Rating of Black player.

4.1.1 Classification Model

Elo Score is used to calculate the relative skill levels of players. From the analysis given below, we can see that the Elo Scores are distributed in the range of 500-3000 and majority of the players have Elo Scores between 1500-2000.



Elo Scores of all Players

This feature is used for tournament sectioning, prize eligibility and pairing purposes in tournaments. It avoids pairing candidates who are most likely to win a tournament during the earlier rounds of the tournament.

Therefore, we group the Elo Score into three buckets:

- Low Score: Elo Score < 1500
- Mid Score: 1500 <= Elo Score <= 2000
- High Score: Elo Score > 2000

We have trained the WhiteElo score and BlackElo score on the same training tuples. The features considered for each Model the features characteristic to the White / Black player respectively and few common features like Termination, Result, etc. The results for the models are given below:

Model	White Elo Class	Black Elo Class
KNN	61.55%	62.11%
Decision Tree	63.29%	63.87%
Logistic Regression	64.55%	63.26%
Random Forest	69.15%	68.31%
XG Boost	73.24%	75.17%
Light GBM	78.45%	79.14%

Accuracy of Model

4.1.2 Regression Model

We have also built a regression model to predict the actual Elo Score of a player. We used the same data as the classification model. The results for the models are given below:

Model	WhiteElo	BlackElo
KNN	301.35	299.76
ElasticNet	258.13	260.47
Lasso Regression	257.56	259.38
Linear Regression	256.84	259.05
Ridge Regression	256.68	259.83
Random Forest	214.25	214.57
Light GBM	189.75	194.19

RMSE of Models

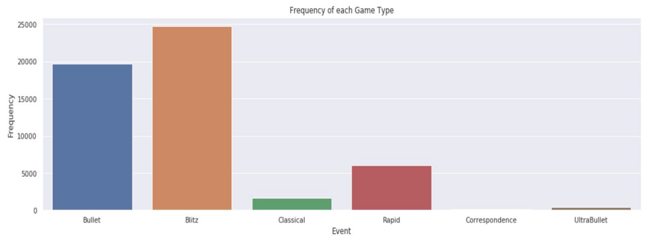
4.2 Predict Game Type from Transcript

Types of Games are UltraBullet, HyperBullet, Bullet, Blitz, Rapid and Classic.

Rapid: Each player is given less time to consider their moves than a classic tournament time controls allow.

Blitz: Players have three to five minutes to make all of their moves.

Bullet: Players have less than three minutes to make their moves. Ultra and Hyper Bullet are shorter variants of Bullet chess.



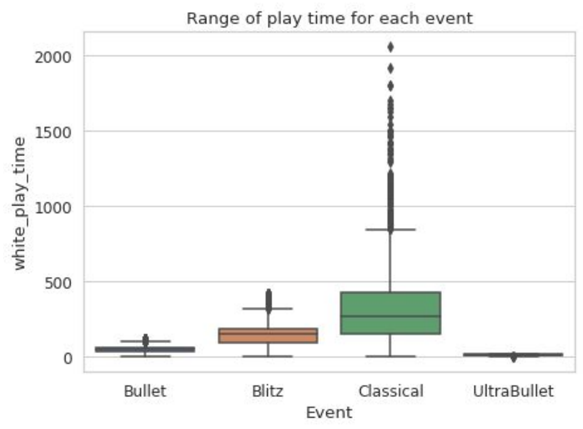
Count of each Game Type

From the game transcript the most important feature needed for this task is the Clock. The total time allowed is different for each game type.

Game Type	Time
UltraBullet	< 15 seconds
HyperBullet	< 30 seconds
Bullet	< 3 mins
Blitz	3 - 14 minutes
Rapid	15 - 25 minutes
Classic	25 minutes - 4 hours

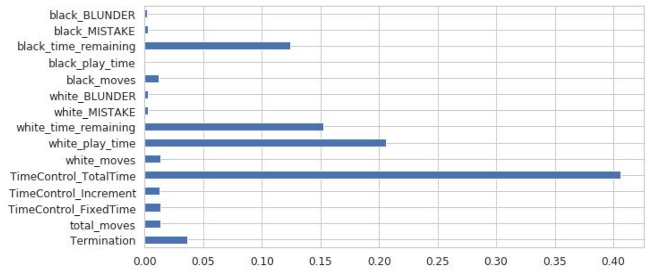
Time for each game type

Also, each player has a different time limit to perform each move in these different games. We have the time taken by each player to perform each move extracted from the 'Moves' column in terms of the 'Clock' feature. Also, if the time limit for a game is less, the number of mistakes and blunders occuring increases.



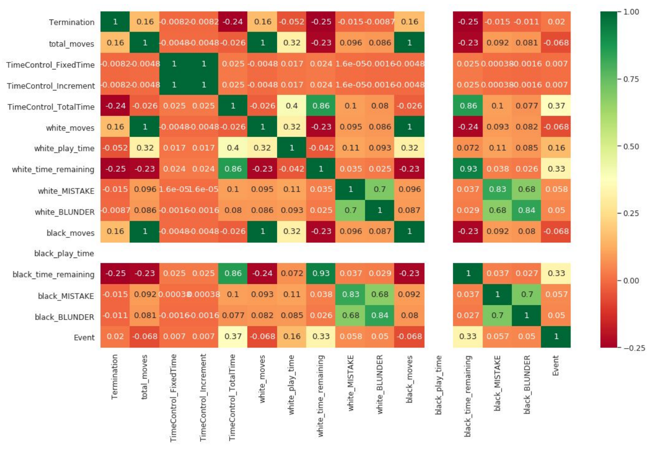
Range of play time based of Game Type

We confirm this by using feature importance technique. The correlation of the features with GameType is given below:



Feature Importance with respect to GameType

This confirms that TimeControl directly gives the GameType. Hence, we do not use this feature. We time_remaining, play_time, total_moves and termination to build this classification model.



Correlation Matrix with respect to GameType

We use classification to predict the game type based on these features. The results obtained are as follows:

Model	Accuracy
Logistic Regression	91.54%
KNN	92.33%
Decision Tree	95.80%
Random Forest	96.13%
XG Boost	96.85%
LGBM	97.57%

Accuracy of Model (in %)

4.3 Predict Result given Chess position

After every move, the probability of winning or losing the game changes drastically depending on the move made. Every board position is a very evident predictor of the result. Using this as the base, we build a model to predict the result of the game, given the board position after any move in any game.

For this problem statement, we need to create a separate database of all possible board states in all games. We need to have the result of the game corresponding to that board state along with which player made the move leading to the result. For this, we first extract data from PGN to CSV format wherein we discard unnecessary features [6]. We then process this data to get the final required features:

- 64 columns - Each representing a square of the board and which piece is present in it. 0 indicates that the square is empty.
- Whose move it is.
- The position in FEN format.
- Result of the Game

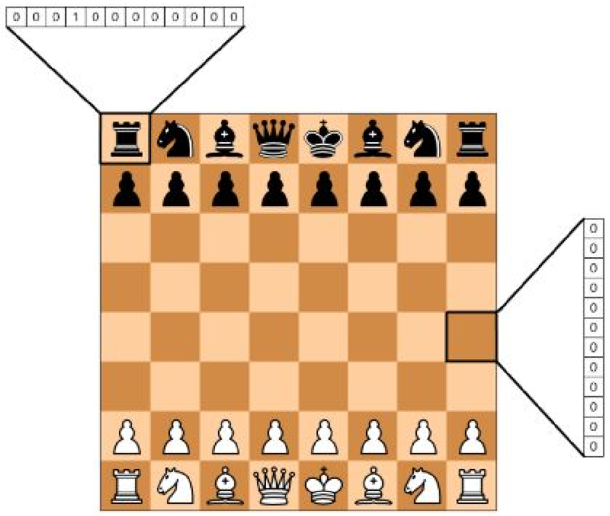
In this we used a Tensorflow Deep Neural Network (DNN) Classifier to predict the Result of the game. The target consists of three classes: 1-0 (White wins), 0-1 (Black wins), 1-1 (Draw) [7]. The results obtained from this DNN Classifier model are as follows:

Hidden Layers	Batch Size	Steps	Accuracy
[10, 20, 30]	2000	10000	51.56%
[50, 50, 50]	2000	10000	56.79%
[50, 50, 50]	4000	20000	63.72%
[50, 50, 50]	8000	40000	65.81%

DNN Classifier - Game Result Prediction

4.4 Next move prediction

In this we predict the next move given a Chess board. We first convert the chess board into a numerical format. This is done by one hot encoding the board with the piece that is in each square [8]. As there are 12 different kinds of pieces on the board, this gives us a 768 element array (8 x 8 x 12) [9].



Board Representation with Hot Encoding

This representation of the board is generated by using the python-chess library which gives us the piece at a given square for the board. The output classes are generated by the chess move which is given by the 'move.from_square' and 'move.to_square' attributes of a move.

With this information we are creating two datasets for two models, one having the board representation and the square it moved from and the other having the board representation and the square it moved to. The board representation is taken after every move is performed for a game and the dataset is generated for all the games in the main dataset.

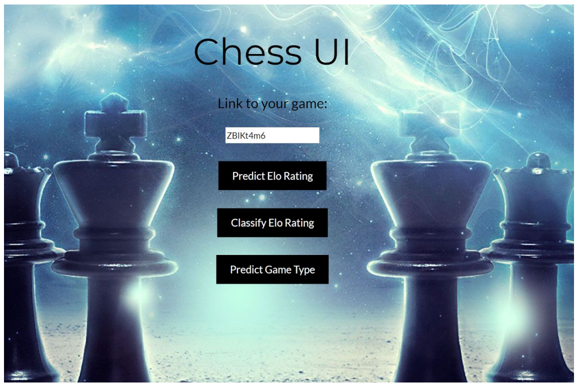
The system takes these two datasets and make two models, namely 'move_from_model' 'move_to_model'. These models will classify the input (board) into one of the 64 classes (positions on the board). We use the outputs of these two models and predict the next move from the given board.

The models are trained for the input data (boards) only for the winning player. Therefore the model only takes the board as input if the final game resulted in a win for that player.

To ensure the model follows the rules of the game, we check if the predicted move is legal at the given board state. If it is illegal then the next best move is chosen. The validation accuracy for 'move_from_model' was 30.42% and for 'move_to_model' 24.38%.

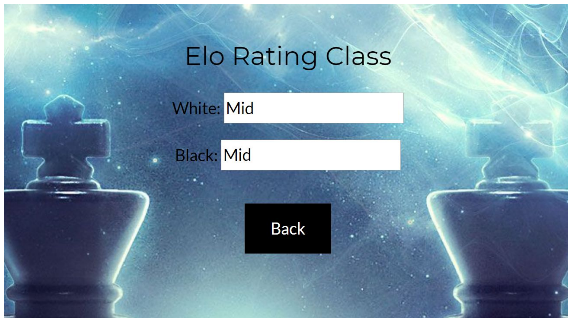
5 User Interface

We have created a user interface to help users interact with the model and get predictions using Flask. Flask is a is a lightweight WSGI web application framework [10]. Every game on Lichess has a unique ID in the URL which differentiates each game. We take this unique ID as input and from this we get the entire game in PGN format. We extracted this game and converted it to CSV. We then extracted detailed features and performed pre-processing in the same way as it was performed to build the model. The final result will be one dataframe tuple. For the UI, we use the best model for each prediction which we had generated and saved. We give this tuple as input to the model and get the desired output.



User Interface to enter game link

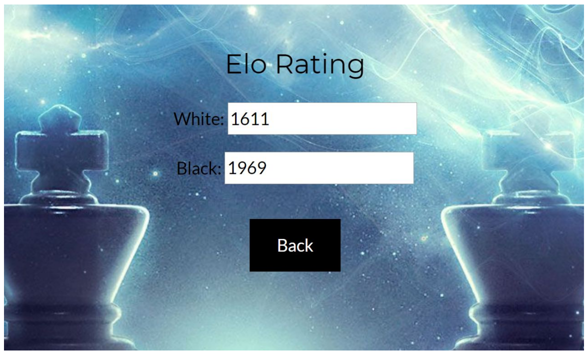
The following predictions were generated in the UI:



White: Mid

Black: Mid

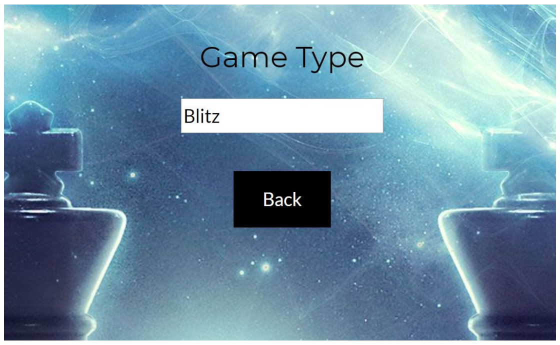
ELO Rating Class Prediction Output



White: 1611

Black: 1969

ELO Rating Prediction Output



Blitz

Game Type Prediction Output

6 Conclusion

In this project we have successfully developed models to predict the Elo Rating of the white and black player using Classification and Regression techniques. We have also predicted the Game Type from the transcript with very good accuracy. Further, we also performed sub-transcript level analysis to predict final game result based on board position as input. We have also made model to predict the next move given the board state. As an extra effort, we have tried to generate a UI to help the user interact with the model and get predictions.

7 References

[1] https://en.wikipedia.org/wiki/Chess

[2] https://en.wikipedia.org/wiki/Elo_rating_system

[3] Paras Lehana, Sudhanshu Kulshrestha, Nitin Thakur and Pradeep Asthana - Statistical Analysis on Result Prediction in Chess [https://search.proquest.com/openview/5108034a9fa8212970332620bc1ace84/1?cbl=2026670&pq-origsite=gscholar]

[4] Lichess Game Database [https://database.lichess.org/]

[5] Standard: Portable Game Notation Specification and Implementation Guide

[6] https://www.cs.kent.ac.uk/people/staff/djb/pgn-extract/

[7] https://www.tensorflow.org/api_docs/python/tf/estimator/DNNClassifier

[8]https://erikbern.com/2014/11/29/deep-learning-for-chess.html

[9]https://towardsdatascience.com/predicting-professional-players-chess-moves-with-deep-learning-9de6e305109e

[10] https://www.palletsprojects.com/p/flask/