How Good is a Chess Player?

CSE519: Data Science Fundamentals
Final Project Report

Introduction

Chess is a two-player strategy board game, which is one of the most popular board games in the world. There are hundreds of millions of chess lovers around the world, and it even has been included in the Olympics Games. In the competitive chess games, questions such as how to assess the difficulty of one game and how to rate the players' performances should be handled by a reasonable and fair solution according to the situation of certain games. It is quite challenging to rely on only humans to formulate these rules. Therefore, we aim to propose reliable methods to predict the ratings of chess players and to evaluate games given the final board situation.

With respect to the estimation of the players' level, there is a certain standard which is called Elo rate. A player's Elo rating is represented by a number which may change depending on the outcome of rated games played. After every game, the winning player takes points from the losing one. The difference between the ratings of the winner and loser determines the total number of points gained or lost after a game. The prediction we make is on Elo ratings provided by the dataset from lichess.

Regarding to the sub transcript level analysis, we propose an approach to evaluate the whole game situation from the final board position. In the problem, we give the definition of the quality of a game using Piece-Square Tables, in which the score of each move on different materials is considered.

Background

To better explore the possible useful information, we conduct some research about the rules about chess. Apart from the basic rules, there exists a threefold repetition of a position which gives the player to move the option to claim a draw, no matter whether the threefold repetition already occurred yet, or is about to occur after declaring the intended move in conjunction with the draw claim. If the same position is reached with the same player to move three times during a game, either player may immediately claim a draw. The procedure for claiming this draw varies somewhat between rule sets, but the rule itself is fairly standard across the board. This rule exists to stop games in which both sides are simply repeating moves. We use this to solve the problem about Elo rating.

Promotion in chess is a rule that requires a pawn that reaches its eighth rank to be immediately replaced by the player's choice of a queen, knight, rook, bishop of the same color. The new piece replaces the pawn, as part of the same move. Pawn promotion, or the threat of it, often decides the result in an endgame. The ability to promote is often the critical factor in endgames and thus is an important consideration in opening and middle game strategy. Almost all promotions occur in the endgame, but promotion in the middle game does happen. Castling is a move in the game of chess. It is an important goal in the opening, because it serves two valuable purposes: it often moves the king into a safer position away from the center of the board, and it moves the rook to a more active position in the center of the board.

What's more, we take the change of player's performance through the whole game into consideration as well. Players might perform differently as the game become intrigue, and that's when high-ranked players show their ability.

To evaluate the quality of the game from the knowledge of only final position, we firstly implement the evaluation function brought up by Tomasz Michniewski. Tomasz Michniewski's function contains 2 parts: piece values and piece-square tables. In our method, we define our own piece value and realize his definition of piece-square tables. It is intuitive that the importance of different materials varies a lot. Thus, it is necessary to consider them distinctly. We assign different weights by materials to every piece remaining on the board, which represent its initial value. Then, we give bonuses for pieces standing well and penalties for pieces standing badly by building tables for different material.

For pawns we simply encourage the pawns to advance. Additionally we try to discourage the move which leaves central pawns unmoved.

```
pawnEval = [[0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0],
            [5.0,  5.0,  5.0,  5.0,  5.0,  5.0,  5.0,  5.0],
            [1.0,  1.0,  2.0,  3.0,  3.0,  2.0,  1.0,  1.0],
            [0.5,  0.5,  1.0,  2.5,  2.5,  1.0,  0.5,  0.5],
            [0.0,  0.0,  0.0,  2.0,  2.0,  0.0,  0.0,  0.0],
            [0.5, -0.5, -1.0,  0.0,  0.0, -1.0, -0.5,  0.5],
            [0.5,  1.0, 1.0,  -2.0, -2.0,  1.0,  1.0,  0.5],
            [0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0]]
```

With knights we simply encourage them to go to the center. Standing on the edge is a bad idea and standing in the corner is a terrible idea.

```
knightEval = [[-5.0, -4.0, -3.0, -3.0, -3.0, -3.0, -4.0, -5.0],
              [-4.0, -2.0,  0.0,  0.0,  0.0,  0.0, -2.0, -4.0],
              [-3.0,  0.0,  1.0,  1.5,  1.5,  1.0,  0.0, -3.0],
              [-3.0,  0.5,  1.5,  2.0,  2.0,  1.5,  0.5, -3.0],
              [-3.0,  0.0,  1.5,  2.0,  2.0,  1.5,  0.0, -3.0],
              [-3.0,  0.5,  1.0,  1.5,  1.5,  1.0,  0.5, -3.0],
              [-4.0, -2.0,  0.0,  0.5,  0.5,  0.0, -2.0, -4.0],
              [-5.0, -4.0, -3.0, -3.0, -3.0, -3.0, -4.0, -5.0]]
```

For bishops, we avoid corners and borders. Additionally we prefer some squares and the central ones. The only idea about rooks is to centralize.

```
bishopEval = [[ -2.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -2.0],
              [ -1.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -1.0],
              [ -1.0,  0.0,  0.5,  1.0,  1.0,  0.5,  0.0, -1.0],
              [ -1.0,  0.5,  0.5,  1.0,  1.0,  0.5,  0.5, -1.0],
              [ -1.0,  0.0,  1.0,  1.0,  1.0,  1.0,  0.0, -1.0],
              [ -1.0,  1.0,  1.0,  1.0,  1.0,  1.0,  1.0, -1.0],
              [ -1.0,  0.5,  0.0,  0.0,  0.0,  0.0,  0.5, -1.0],
              [ -2.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -2.0]]
```

Generally with queen we mark places where normally people wouldn't like to have a queen.

```
queenEval = [[-2.0, -1.0, -1.0, -0.5, -0.5, -1.0, -1.0, -2.0],
             [ -1.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -1.0],
             [ -1.0,  0.0,  0.5,  0.5,  0.5,  0.5,  0.0, -1.0],
             [ -0.5,  0.0,  0.5,  0.5,  0.5,  0.5,  0.0, -0.5],
             [  0.0,  0.0,  0.5,  0.5,  0.5,  0.5,  0.0, -0.5],
             [ -1.0,  0.5,  0.5,  0.5,  0.5,  0.5,  0.0, -1.0],
             [ -1.0,  0.0,  0.5,  0.0,  0.0,  0.0,  0.0, -1.0],
             [ -2.0, -1.0, -1.0, -0.5, -0.5, -1.0, -1.0, -2.0]]
```

The table for King describes where the ending begins: the Both sides have no queens or Every side which has a queen has additionally no other pieces or one minor piece maximum.

```
kingEval = [[ -5.0, -4.0, -3.0, -2.0, -2.0, -3.0, -4.0, -5.0],
            [ -3.0, -2.0, -1.0,  0.0,  0.0, -1.0, -2.0, -3.0],
            [ -3.0, -1.0,  2.0,  3.0,  3.0,  2.0, -1.0, -3.0],
            [ -3.0, -1.0,  3.0,  4.0,  4.0,  3.0, -1.0, -3.0],
            [ -3.0, -1.0,  3.0,  4.0,  4.0,  3.0, -1.0, -3.0],
            [ -3.0, -1.0,  2.0,  3.0,  3.0,  2.0, -1.0, -3.0],
            [ -3.0, -3.0,  0.0,  0.0,  0.0,  0.0, -3.0, -3.0],
            [ -5.0, -3.0, -3.0, -3.0, -3.0, -3.0, -3.0, -5.0]]
```

At the end, we plus the initial value for a piece with the location value and gain the total value. Then we add up the value of each piece and form the quality of a board. The matrices we define above can be used on both white and black players. We calculate the value for both players and add them up. To show the validity of our method, we find the top 10 best games and gain 10 games randomly on the Chess.com and calculate their values with our method. It is quite obvious that our method is able to distinguish the good and bad games. Here shows the results:

Top 10 Best Games		Random Games	
Name	Score	Name	Score
Kasparov vs. Topalov, 1999	182	game1	88
Morphy vs. Allies, 1858	110	game2	89
Aronian vs. Anand, 2013	108	game3	86
Karpov vs. Kasparov, 1985	142	game4	94
Byrne vs. Fischer, 1956	142	game5	82
Ivanchuk vs. Yusupov, 1991	148	game6	91
Short vs. Timman, 1991	99	game7	93
Bai jinshi vs. Ding Liren, 2017	121	game8	85
Rotlewi vs. Rubinstein, 1907	103	game9	82
Geller vs. Euwe, 1953	122	game10	86

Dataset

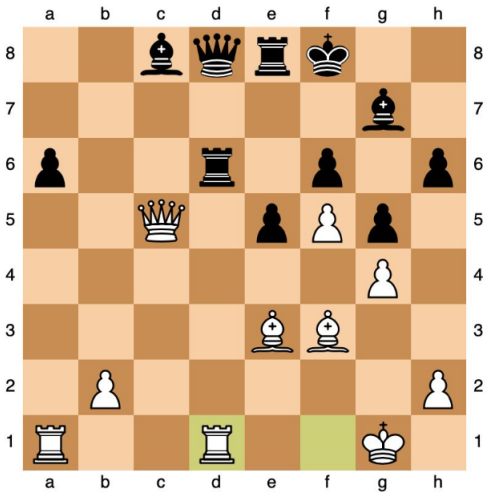
Our dataset is downloaded from the lichess game database. The raw dataset indicates the *Event, Site, UserName, Result, UTCDate, UTCTime, Elo ratings, RatingDiff, Title, ECO, Opening, TimeControl, Termination* and *Records*. We select and clean 153457 rows of data from all dataset and we use them to finish our project.

We also downloaded the pure computer chess data from CCRL (Computer Chess Rating List), which contains 32000 rows. However, this dataset only includes *Result, Records, ElO*. But This is enough for our computer chess player test

To obtain more helpful data, we extract the clock values, local evaluation values and board record values for each step and for each player of the game from the game record. We name them as *CLOCK, LOCAL EVAL, TRANSCRIPT*, and we process the clock values into a list of referenced time in seconds starting with 0. According to the record, we also generate the final position matrix using chess engine. Every material is represented as a letter based on its initial where capital indicates white pieces and lowercase indicates black pieces.

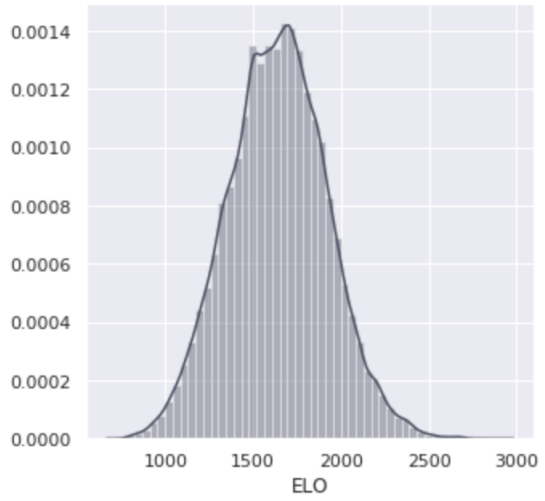
After getting the specific transcript, we calculate the plays of the game. Then we filter out the data with less than or equal to 3 rounds, and drop the data without time control information which means '-' in the pgn file to form one of our final data.csv table. In this case, each sample/row contains features of both sides. However, since the different research direction of the later analysis, we need the data from both sides to analyze the game but only need the features of one player to analyze or predict his or her performance. Therefore, we split the data according to the black and white and then concatenate them vertical to form our another dataset csv table such that each sample/row now only corresponding to one player.

In the records, the pieces other than pawns are named as K, Q, R, B, and N for king, queen, rook, bishop and knight respectively. The moves are represented by the piece notation concatenated with the coordinate of destination square. The vertical squares or files from white's left are labelled a through h while the horizontal rows of ranks are numbered 1 to 8 starting from white's side. Thus, for example, Nf3 indicates a move by knight to square f3. The captures are indicated by an 'x' inserted immediately before the destination square, for example, in Bxf3. Castling is indicated by using uppercase letter O as O-O for kingside and O-O-O for queenside. Checks are depicted by '+' appended after the move notation. End of the game or result is indicated by 1-0 for white win, 0-1 for black win and 12-12 for a draw. We use these ideas in the extraction of *CAPTURE, CAPTURED* and *CHECK* features.

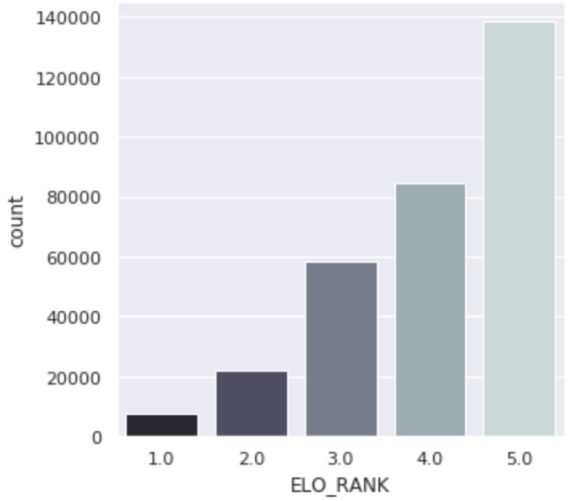


From the records, we delete the local value of each step and extract only the position of each step thus form the *TRANSCRIPT* feature. And from the *TRANSCRIPT*, we use the chess class provided by Python to generate the final position of game in Board type. The following picture simply shows

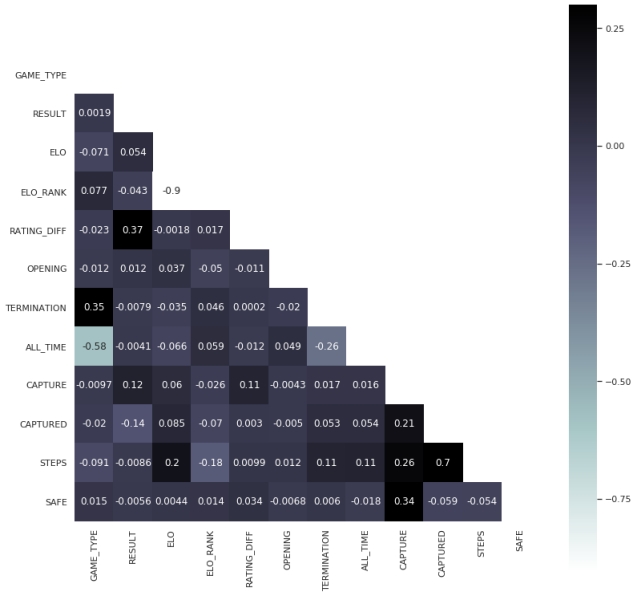
what we do to preprocess the data for quality prediction based on final position.



The picture above depicts the distribution of the Elo ratings. In the records of Elo rates, the lowest score is 776 and the highest one is 2706. Based on the distribution of Elo rates, we define the player with score over 2200 as advanced player; score within 2000 and 2200 as mid-advanced player; score within 1800 and 2000 as middle player; score within 1600 and 1800 as mid-beginner players and score under 1600 as beginner players. Following is the distribution of the *ELO_RANK*:



After preprocessing and expanding of the dataset, we try to explore the relation between each column. Unfortunately, the pairplot results are disappointing, but the heatmap shows some useful information:



We find some features are highly correlated with *GAME_TYPE*, which gives us inspirations a lot. However, it is quite lost when looking at the *ELO* and *ELO_RANK* because there is basically nothing relevant. Thus, feature engineering become crucial in this part.

Feature Engineering

Elo Prediction

Following the interested features proposed in the last report, we further extracted new interested features below:

- *CHECK*: number of times one color was attacked in the game.

- *VARIANT_WIN*: whether the winning side won abnormally.

- *INSUFFICIENT_MATERIAL*: when determining the result of the game, are there any missing material on the board that are necessary to win.

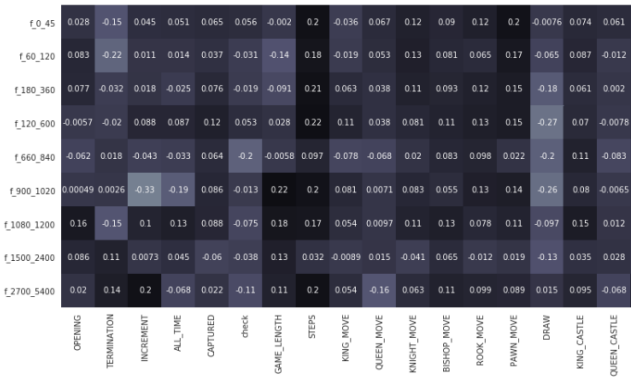
- *REPETITION*: whether there are 3 repetitions of same location from same side in the game, because in this case the game will draw automatically.

- *STALEMATES*: stalemate is a situation in the game of chess where the player whose turn it is to move is not in check but has no legal move. This feature represents if stalemate occurs, because when stalemate occurs, the game ends as a draw.

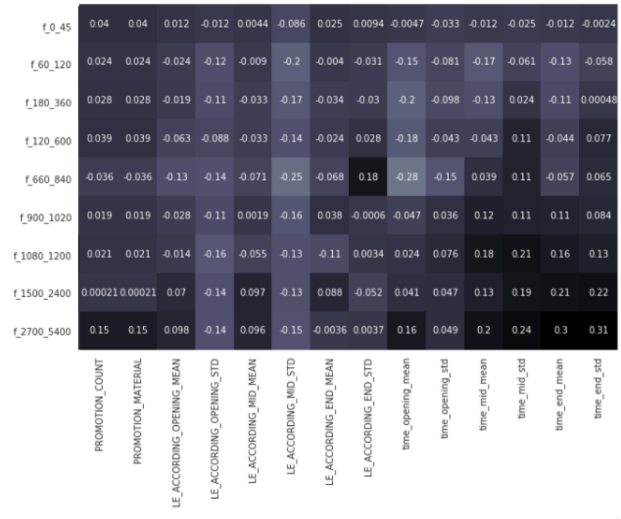
At the same time ,we also redefined the features which represent the game performance based on the period of game, the modified feature named LE_ACCORDING_OPENING/MID/END_MEAN/STD. Theses features are generated by the analysis of the statistics of the computer evaluation in the opening, middle and end sections of the game. We only keep \ACCORDING mode in the final and remove mode \SAME", the data tells us that split the game period accordingly by the steps gets better result of predicting Elo value.

After an in-depth analysis of the entire dataset, we found that most features fluctuate based on the fixed time length of the game, that is our feature TIME_CONTROL_A. This phenomenon leads us to separate the dataset based on the

TIME_CONTROL_A to predict Elo value. In fact we tried our baseline model on the whole dataset using the same features as we used for our best model, but the result is poor which gives us accuracy lower than 55%, even though we added new features, modified old features and removed inefficient features. So we tried a new way to build the model.

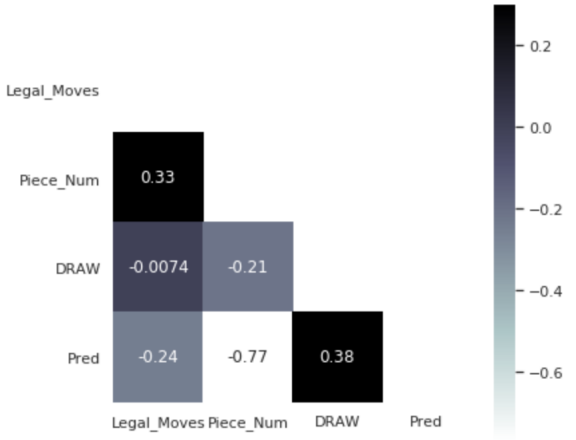


Observing the fixed time length of game given in our datasets, we got 33 specific game length which are roughly divided into 4 types, of which [0s, 15s, 30s, 45s] are the first type with interval 15s, the second type is [60s, 90s, 120s] with time interval 30s, the third type is [120s, 180s, 240s, 300s, 360s, 420s, 480s, 540s, 600s, 660s, 720s, 780s, 840s, 900s, 960s, 1020s, 1080s, 1140s,1200s] with time interval 60s, and the last type is [1500s, 1800s, 2100s, 2400s, 2700s, 3600s, 5400s] with time interval 300s. We defined the dataset f_#number1_#number2 as the game whose fixed time length is shorter than or equal to number1 and longer than or equal to number2. According to this definition, we have those datasets, f_0_45, f_60_120, f_180_360, f_120_600, f_660_840, f_900_1020, f_1080_1200, f_1500_2400, f_2700_5400. The figure below excerpts the Pearson correlation coefficients between interested features and Elo values grouped by TIME_CONTROL_A.



Quality Prediction

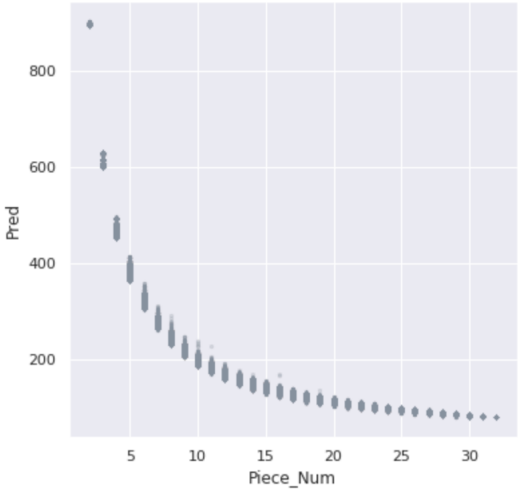
Given that in this problem, we can only extract features from the final board position, which means that most of the information from the dataset can not be used. we first tried to define the evaluation function with features from raw dataset such as the transformation of Elo rates and local evolution calculated by Chess Engine. However, the result is disappointing. Then, we implement several functions and gain some desirable features from the static final board position. From the limited usable data, we find the following three features are strongly correlated with the target(score value). The following heat map describes the correlation:



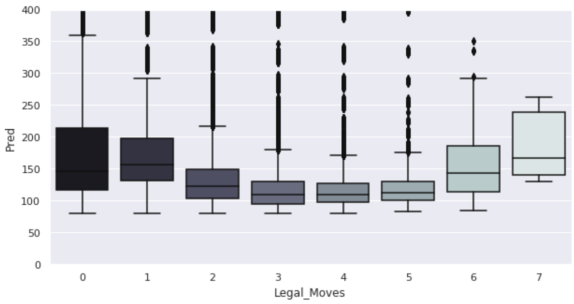
- *Piece_Num* is the number of the remaining pieces on the board.

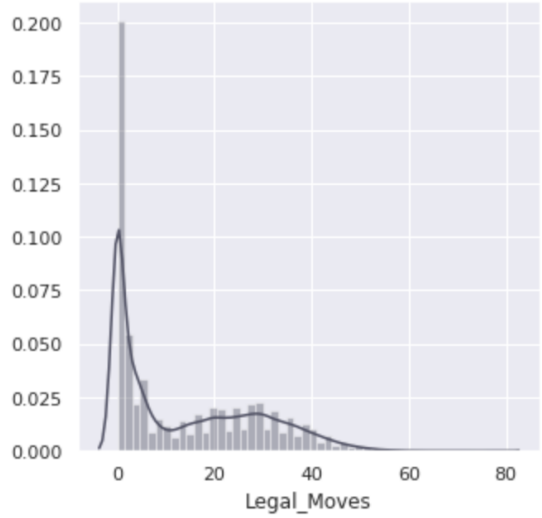
- *DRAW* is the result of a game ending in a tie. Usually, in tournaments a draw is worth a half point to each player, while a win is worth one point to the victor and none to the loser. A draw occurs when it appears that neither side will win.

- *Legal_Moves* indicated the sum of legal moves for each piece on current board.

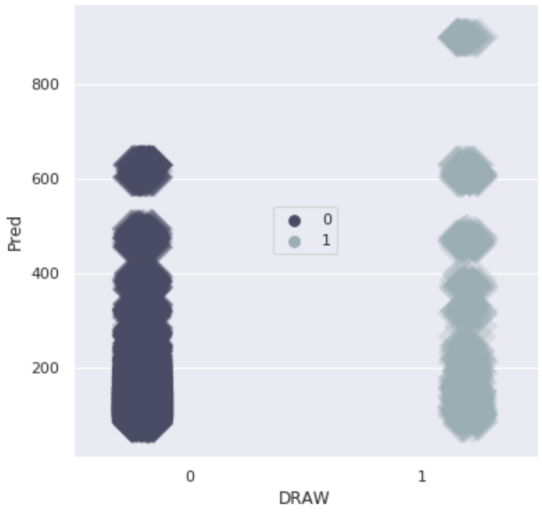


The scatter graph above shows the detailed correlation between the score of the game and the number of pieces. We can see that the score decreases as the number of pieces grows. The relation between the two variables can be recognized as logarithmic function. When a game is close to the end, the remaining pieces should be the important ones(those with high initial value). More pieces mean that lower average value for each piece, which partly accounts for the distribution.





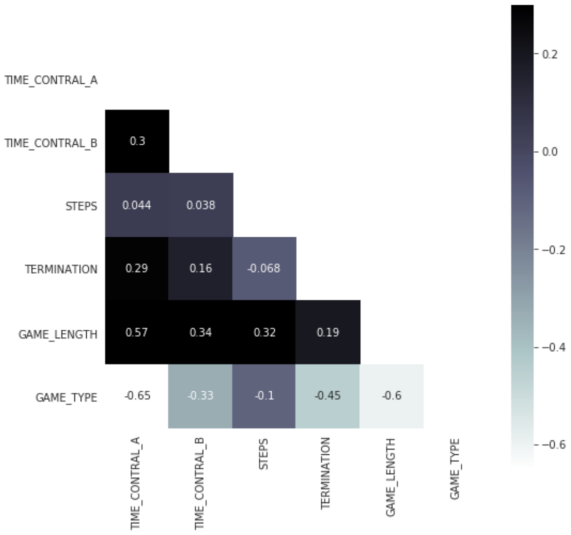
It is easy to notice that the score is higher for those games with less possible legal moves, especially for no available move. This is reasonable because players would try every possible way to win and the game only end when there is no other way. The small number of games with possible legal moves more than 50 can be ignored to some extent.



Sometimes, a draw represents the comparable ability of the two players.

Game Type Prediction

In this problem, we aim to predict the type of games based on the information from the data set. After data preprocessing and exploration of the raw data, we select five features. The following heat map shows the pearson correlation between those features and the game type:



- *TIME_CONTROL_A*: We extract this feature from the term TimeControl in the original data set. The original term is composed of two parts, for example '300+0'. TIME CONTROL A is the first part in the TimeControl.

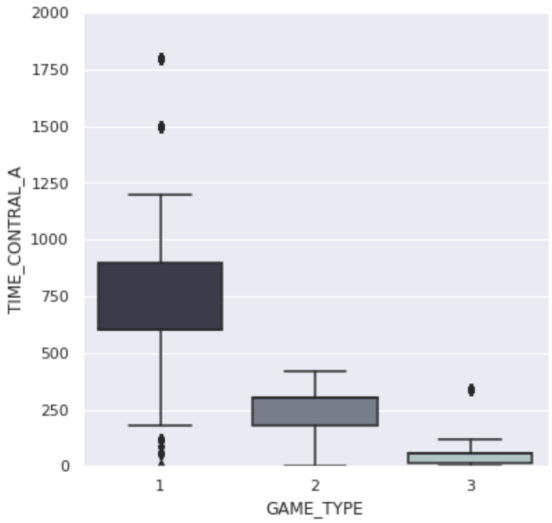
- *TIME_CONTROL_B*: This represents the second part in the TimeControl.

- *STEPS*: From the whole record of the game, we count the moves made by both two players and form the feature STEPS.

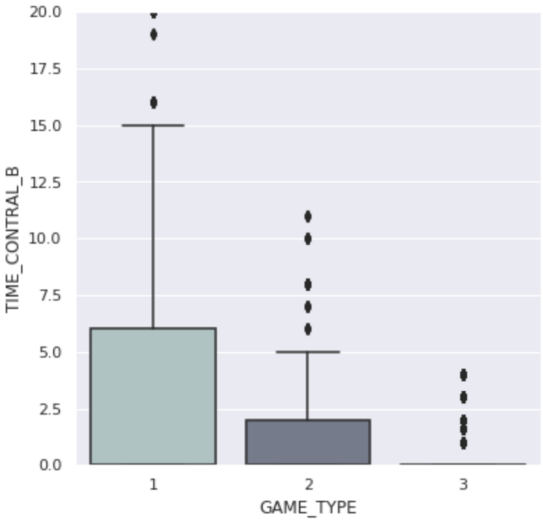
- *TERMINATION*: All of the games end with either normal or time forfeit. We represent these two types into 0 and 1 in TERMINATION.

- *GAME_LENGTH*: We use the clock time in the raw data and turn it into a time period represented in seconds.

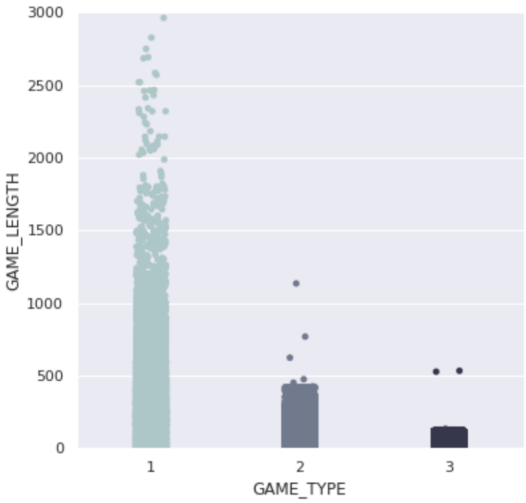
- *GAME_TYPE*: In this term: '1' represents 'Classical'; '2' represents 'Blitz'; '3' represents 'Bullet'.



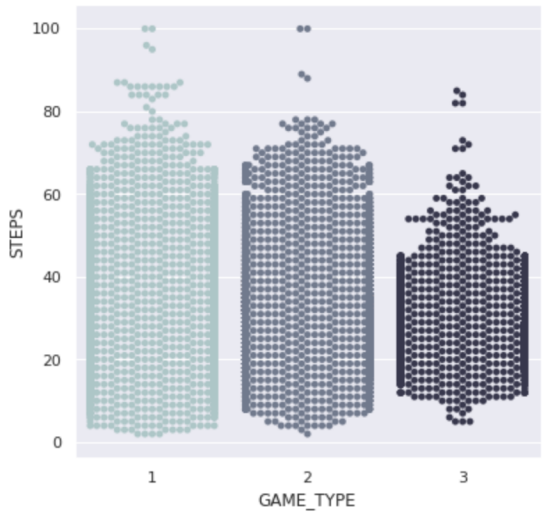
The box chart above shows how the *TIME_CONTROL_A* correlates with *GAME_TYPE*. It is not hard to see that these two are strongly correlated. In the classical type, the *TIME_CONTROL_A* is the highest and it is the lowest in the bullet.



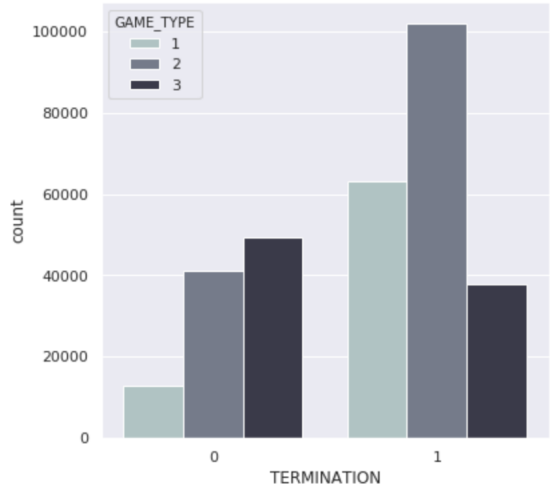
TIME_CONTROL_B is less important than the previous part, however it shows something as well. The highest bound for the three type still remains the same with the *TIME_CONTROL_A*.



GAME_LENGTH describes the time length of a chess game. It shows great relevance with the type we should predict. Classical games have the largest value, which ranges from 0 to over 3000. While Blitz games' value peaks at nearly 500 and Bullet games are less more. We can infer that the classical games need much more time than two other types.



From the swarm graph above, we can see that there are more steps in Classical and Blitz games than Bullet games. This can be explained by the time length of games: more time the player use, more steps they will make.



In the term of *TERMINATION* '0' represents 'Time forfeit' and '1' represents 'Normal'. It is quite interesting that more games end normally in type Classical and Blitz, however more games end with time forfeit in Bullet. This can be partially explained by the extremely limited time for Bullet games. For the other two types, the played might have more sufficient time to finish the game.

Human or AI?

In this part, we add an additional feature named *ROB* to define whether this row of data is a computer or not. The same pre-processing is handled on the rob data too, and the final features of rob dataset are basically same with the features used in Elo prediction (*ELO, TERMINATION, STEPS, All_MATERIAL_MOVE, DRAW, KING_QUEEN_CASTLE, PROMOTION, LOCAL_EVAL_STATISICS*). Apart from those, there are some new features are added to predict:

time_opening_mean, time_opening_std, time_mid_mean, 'time_mid_std, time_end_mean, time_end_std, ROB

Results

Elo Prediction

In the prediction of the Elo value, we divided this part into three branches–regression, classification and machine learning. Regression is to directly predict the Elo value using the regression approach and then convert the predicted result to Elo rank to evaluate its accuracy. The regression model contains Linear Regression, XGB Regression, ExtraTrees Regression, and ElasticNet model.

Another approach is to feed model as classifier, we pre-process the Elo value into Elo rank first , and do classification prediction on *ELO_RANK* variable. Classification models are Random Forest Classifier.

In addition, we also tried machine learning methods, H2ODeepLearning model and H2OAutoEncoder Model. In the deep learning model, we set it with dropout equals to 0.2 and trained it with cross validation. Using H2OAutoEncoder method is an innovative idea obtained from kaggle competition. The idea is the trainer encode the training data save the parameters in the trainer so that the test data goes through the trainer can be encoded first and then decoded according to the parameters, the output of decoding is the reconstructed test data with features as same as training data even though test data contains some missing features. So our prediction of Elo value will be the feature *RECONSTRUCT_ELO_RANK* in the reconstructed test data.

We mentioned above that in the analysis of Elo value, we divided it into 9 sub-datasets for research. According to this, our model is defined as :

First, identify the fixed time of game. Second, find corresponding sub-dataset, make sure the fixed time is between the time interval of sub-datasets. Then, find corresponding sub-model to predict Elo rates. The sub-model is named as f_#number1_#number2_#bestmodel meaning that the test data targeted by this sub-model has a fixed time between time number2 and time number1 and this sub-model has the best performance among other sub-models within this fixed time intervals. So the best performance sub-models of specific fixed time intervals are f_0_45_RandomForest, f_60_120_FandomForest, f_180_360_RandomForest, f_120_600_RandomForest, f_660_840_DeepLearning, f_900_1020_LinearRegressor, f_1080_1200_, f_1500_2400_RandomForest, f_2700_5400_AutoEncoder. The prediction result shows as following:

Table 1. The Best Results in Elo Prediction

Dataset	RSME	R2	Accuracy
f_0_45	0.8356	0.0775	65.90
f_60_120	0.9965	0.1509	67.22
f_180_360	1.0690	0.1329	64.22
f_120_600	0.7734	0.0799	77.73
f_660_840	1.0080	0.3865	67.74
f_900_1020	0.7080	0.1817	73.06
f_1080_1200	0.9764	0.1637	68.21
f_1500_2400	0.9746	0.1637	68.21
f_2700_5400	0.8528	0.0242	77.27

Quality Prediction

In the prediction of the quality from final position, we first try to solve the problem as regression one. We use a basic linear regression model as the baseline model and gain the accuracy as 64%. Then we try to solve it as a classification problem(we create four ranks for quality). Similarly, we use the linear regression model as baseline model and gain 82%. Then we try different classifiers and implement the grid searching techniques to find better parameters. Following shows the results:

Table 2. Quality Prediction From Final Position

Method	Accuracy	RMSE	R2
LinearRegression	86.02%	0.13	0.86
RandomForestClassifier	81.16%	0.18	0.80
XGBClassifier	94.69%	0.05	0.94
LGBMClassifier	94.70%	0.05	0.94
KNeighborsClassifier	94.42%	0.05	0.94

Game Type Prediction

The prediction of game types can be recognized as a classification problem. We use the following five methods to make the prediction. The results are desirable, the accuracy can be 99.81% when using xgboost classifier.

Table 3. Game Type Prediction

Method	Accuracy	RMSE	R2
LinearRegression	54.51%	0.30	0.55
RandomForestClassifier	98.68%	0.01	0.97
XGBClassifier	99.81%	0.002	0.99
LGBMClassifier	99.81%	0.002	0.99
KNeighborsClassifier	96.99%	0.03	0.95

Human or AI?

In the problem, We are going to detect whether the record is from a computer or a human, so this problem can be treated as a binary classification problem. Here we chose to use XGB Classifier, Random Forest Classifier, ElasticNet and Deep Learning to do prediction.

Table 4. Robot Detect

Method	RSME	R2	Accuracy(%)
RandomForestClassifier	0.0	1.0	100
XGBClassifier	0.0068	0.9997	99.99
ElasticNet	0.0059	0.9597	98.72
H2ODeepLearning	0.0072	0.9997	99.99

Conclusion

In the project – 'How good is a chess player?', we solved four problems: the prediction of Elo rate and game types from the entire chess records; the prediction of quality of games from only final position; distinguishing the human player and machine. The results are quite desirable. The accuracy values of prediction of game types, quality and player types are above 94%. For Elo problem, we get average accuracy value of 70%.

However, there are still possible work to do. We only collect the data in two months, which might be insufficient to find

more useful information and draw better pattern of data. In addition, the data we use are all with local evaluation value calculated by chess engine. We should try our method on less informative data. The features provided by chess engine can be counted in thousands, but we are not possible to try more. Due to the lack of knowledge about chess games, we are not able to make good definition of quality of games. It can be represented in more rigid and specific method.

Appendix

The following tables shows the accuracy of Elo prediction in 9 classes respectively.

Table 5. f_0_45

f_0_45	RSME	R2	Accurac(%)
LinearRegressor	0.8073	0.1967	64.68
RandomForestClassifier	0.8356	0.0775	65.90
XGBClassifier	0.7594	0.0588	65.70
ExtraTreesRegressor	0.7429	0.0134	64.25
ElasticNet	0.7913	0.1497	64.93
H2ODeepLearning	0.9558	0.4098	64.10
H2OAutoEncoder	0.9558	0.4098	64.10

Table 6. f_60_120

f_60_120	RSME	R2	Accuracy(%)
LinearRegressor	0.	0.0554	63.92
RandomForestClassifier	0.9965	0.1509	67.22
XGBClassifier	0.8627	0.1206	65.81
ExtraTreesRegressor	0.8913	0.0612	62.73
ElasticNet	0.9450	0.0551	63.95
H2ODeepLearning	1.0934	0.3855	65.46
H2OAutoEncoder	1.0934	0.38550	65.46

Table 7. f_180_360

f_180_360	RSME	R2	Accuracy(%)
LinearRegressor	0.9399	0.1200	58.97
RandomForestClassifier	1.0690	0.1329	64.22
XGBClassifier	0.9241	0.1491	62.70
ExtraTeesRegressor	0.9410	0.1179	59.59
ElasticNet	0.9417	0.1164	59.09
H2ODeepLearning	1.2076	0.4453	61.69
H2OAutoEncoder	1.2071	0.4445	61.69

Table 8. f_120_600

f_120_600	RSME	R2	Accuracy(%)
LinearRegressor	0.7747	0.1223	75.07
RandomForestClassifier	0.7734	0.0799	77.73
XGBClassifier	0.7421	0.1946	74.91
ExtraTreesRegressor	0.7358	0.2083	73.43
ElasticNet	0.7780	0.1148	74.91
H2ODeepLearning	0.9028	0.2535	75.16
H2OAutoEncoder	0.8531	0.1192	73.12

Table 9. f_660_840

f_660_840	RSME	R2	Accuracy(%)
LinearRegressor	1.1037	0.2097	38.18
RandomForestClassifier	0.8327	0.0536	65.35
XGBClassifier	1.1119	0.2278	45.45
ExtraTreesRegressor	1.0000	0.0068	49.09
ElasticNet	1.0617	0.1195	47.27
H2ODeepLearning	1.0080	0.3865	67.74
H2OAutoEncoder	1.0160	0.4085	66.13

Table 10. f_900_1020

f_900_1020	RSME	R2	Accuracy(%)
LinearRegressor	0.7305	0.1288	72.41
RandomForestClassifier	0.8524	0.0437	71.79
XGBClassifier	0.7080	0.1817	73.06
ExtraTreesRegressor	0.7125	0.1711	72.80
ElasticNet	0.7314	0.1267	72.28
H2ODeepLearning	1.0073	0.3355	68.84
H2OAutoEncoder	1.0073	0.3355	68.84

Table 11. f_1080_1200

f_1080_1200	RSME	R2	Accuracy(%)
LinearRegressor	0.9180	0.0674	56.43
RandomForestClassifier	1.0151	0.1981	58.06
XGBClassifier	0.9746	0.2031	57.85
ExtraTreesRegressor	0.9063	0.0403	68.21
ElasticNet	0.9561	0.1579	56.42
H2ODeepLearning	1.0807	0.3579	59.54
H2OAutoEncoder	1.1324	0.4910	55.72

Table 12. f_1500_2400

f_1500_2400	RSME	R2	Accuracy(%)
LinearRegressor	1.2951	0.5194	39.78
RandomForestClassifier	0.9746	0.1637	68.21
XGBClassifier	1.0875	0.0714	55.91
ExtraTreesRegressor	1.1731	0.2467	55.91
ElasticNet	1.2700	0.4610	45.16
H2ODeepLearning	1.1925	0.3748	65.55
H2OAutoEncoder	1.1972	0.3847	64.44

Table 13. f_2700_5400

f_2700_5400	RSME	R2	Accuracy(%)
LinearRegressor	1.9928	0.5351	45.82
RandomForestClassifier	0.7833	0.1767	70.45
XGBClassifier	1.01411	0.6449	68.57
ExtraTreesRegressor	0.6761	0.3689	71.42
ElasticNet	0.8280	0.0966	71.42
H2ODeepLearning	0.8528	0.0242	77.27
H2OAutoEncoder	0.9770	0.2806	75.00