How Good is a Chess Player?
Prediction of Elo rankings based on Play-by-Play records

1 Overview

Chess has always been a fascinating combination of strategy and skill. Both during, and especially after a game, there are endless opportunities to analyze how the players went about their moves, what a board state at an instant in the game represented, which move turned the tables in favor of the winner and so on. The strength of chess players is translated into their respective Elo ratings, with players above a certain grade to be categorized as Grand Masters. Given that the performance of the players are mapped to Elo rankings, it should be possible to foretell the Elo rankings of a player by looking at the set of moves made by them in their games. Our main aim as part of this analysis is to create a model that predicts just that, the Elo rankings of the players involved in the game within a certain degree of confidence. Lichess is an online chess server that enables players to play online and analyze their games. All this data is available as part of a humongous repository which would form the essential dataset for this endeavor. Building up on our initial proposal and the mid-sem report, we employ extensive feature engineering to create novel attributes and quantify the corresponding moves of both a player and their opponent (White and Black). These features are then used for regression analysis by applying them to linear and random forest regression models and the results obtained are analyzed. As an additional exercise we applied the relative strength of the moves of each player to a deep learning model to classify their rankings as part of a particular range of players as well as try to predict the type of game a player was involved in. The rest of the report is divided into the following sections:

- Background
- Dataset
- Exploratory Data Analysis and Preprocessing
- Feature Engineering
- Methods
- Extended Work
- Results
- Conclusion

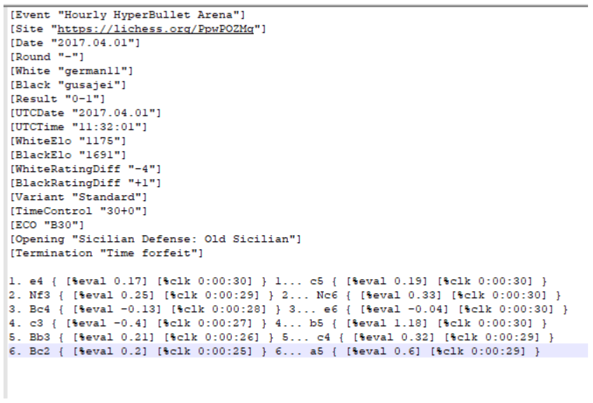
2 Background

Elo scores were initially created as a means to rank chess players. Since then, they have been introduced in a lot of other fields and games like baseball, basketball and even video games. The primary concept is the zero-sum nature of the Elo score where the ratings between two players serve as a predictor of the outcome of a match. The difference between the ratings of the winner and loser determines the total number of points gained or lost after a game.[5]
Coming back to the roots of Elo ratings with Chess, the players at every level, students and analysts of the game keep a record of the moves played in the games they were a part of. The moves are represented in a standard format called the 'Algebraic Chess Notation' and generally chess enthusiasts maintain such a record in a scoring sheet. Thankfully, with the advent and popularity of online chess tournaments, many online portals have come to the fore which have huge repositories of game data in Portable Game Notation files (PGN) and these form the basis of our analysis. Lichess.org(Lic) is one such portal, that enables players to both play online and analyze their games through the embedded Stockfish Chess Engine(Sto). Given that the PGN data has all the moves, Elo Rankings of both the involved

players and other metadata, this becomes the perfect recipe for a machine learning problem where in given the moves and metadata of a set of games, we employ a model to predict the Elo Rankings of the players involved. This seems logical as higher the ranking of a player, the better his/her skills are and more informed their choice of moves would be.

3 Dataset

As mentioned previously, we used a PGN files from Lichess which has game data of 100,000 games. The particular dataset used for this project is from February 2013. One of the challenges was to get this data in a readable format that could be used for data analysis. We created a Python based pipeline to generate a CSV file for the game data which can then be directly loaded as a Pandas DataFrame. We added extra steps to gather additional information from the moves itself during this conversion of PGN to CSV. This was done to ease our work on the bits which might not be straight forward once we moved to the CSV format. eg. Some of the critical pieces of information we captured was the number of pieces attacking the 4 center squares (E4, E4, D4 and D5) for both Black and White at any giving instant as we traverse the moves in the PGN file. This was relatively easy thanks to the python-chess API(pyt) and would not have been straightforward later during feature engineering.



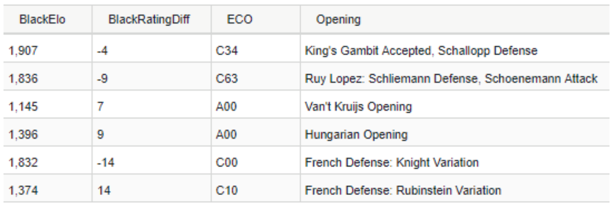
```
[Event "Hourly HyperBullet Arena"]
[Site "https://lichess.org/PpwPOZMq"]
[Date "2017.04.01"]
[Round "-"]
[White "germanll"]
[Black "gusajei"]
[Result "0-1"]
[UTCDate "2017.04.01"]
[UTCTime "11:32:01"]
[WhiteElo "1175"]
[BlackElo "1691"]
[WhiteRatingDiff "-4"]
[BlackRatingDiff "+1"]
[Variant "Standard"]
[TimeControl "30+0"]
[ECO "B30"]
[Opening "Sicilian Defense: Old Sicilian"]
[Termination "Time forfeit"]

1. e4 { [%eval 0.17] [%clk 0:00:30] } 1... c5 { [%eval 0.19] [%clk 0:00:30] }
2. Nf3 { [%eval 0.25] [%clk 0:00:29] } 2... Nc6 { [%eval 0.33] [%clk 0:00:30] }
3. Bc4 { [%eval -0.13] [%clk 0:00:28] } 3... e6 { [%eval -0.04] [%clk 0:00:30] }
4. c3 { [%eval -0.4] [%clk 0:00:27] } 4... b5 { [%eval 1.18] [%clk 0:00:30] }
5. Bb3 { [%eval 0.21] [%clk 0:00:26] } 5... c4 { [%eval 0.32] [%clk 0:00:29] }
6. Bc2 { [%eval 0.2] [%clk 0:00:25] } 6... a5 { [%eval 0.6] [%clk 0:00:29] }
```

Figure 1: Sample PGN Data

As can be seen in the above sample data, we get the set of moves for every ply along with the strength of the move represented by eval(more on that later),and metadata for the game and players involved.

4 Exploratory Data Analysis and Pre-processing



BlackElo	BlackRatingDiff	ECO	Opening
1,907	-4	C34	King's Gambit Accepted, Schallopp Defense
1,836	-9	C63	Ruy Lopez: Schliemann Defense, Schoenemann Attack
1,145	7	A00	Van't Kruijs Opening
1,396	9	A00	Hungarian Opening
1,832	-14	C00	French Defense: Knight Variation
1,374	14	C10	French Defense: Rubinstein Variation

Figure 2: Lichess Data imported as CSV

4.1 Important Attributes

Most of the data obtained from Lichess is self-explanatory but it would be useful to elaborate some important attributes that have been used for the analysis of this work.

(i) **Event:** The variant of game in question. After the above pre-processing, 4 variants remained in our dataset, viz. Blitz/Bullet/Classical/Correspondence. As mentioned earlier, we did merge similar variants played under different different event types into one variant.

(ii) **ECO (Encyclopedia of Chess Openings)**(ECO): A 3 character code which indicates the kind of opening sequence of moves in the game. The format is <alphabet> followed by 2<numerals>. The alphabet indicates the type of opening sequence and the numerals represent the sub-type. This can be considered a more structured and precise way of representing the opening move in comparison to the free text way of specifying an opening. eg. The ECO classifies the 'English Opening' under A10 to A39.

(iii) **Termination:** The different ways in which the game can end viz. Terminated/Normal/Unterminated/Rule Infraction/Death/Emergency/Time forfeit.

(iv) **Numeric Annotation Glyph (NAG):** We also retrieve the comments or NAGs from the PGN file. Unfortunately, for the dataset we picked from Lichess we only came across 3 distinct values of NAGs in the all the games we did parse, a poor move, a questionable move or a blunder. The notation does allow for positive moves like good, very good and interesting move too. We allot scores, albeit negative scores based on how many of these poor moves were made. These are allotted after the game retrospectively.

(v) **Center/Diagonal Squares Controls:** We also gather information of how many pieces of White and Black are attacking the central squares E4, E5, D4 and D5. Based on some analysis and readings we concluded that this generally indicates how well either player is playing as control over center squares is widely regarded as indicator of dominance in the game. This is definitely something players strive for during the opening and during the middle phase of the game. But this is not true during the finishing stages of the game. This can be seen from the visualization in figures 3 and 4. Players with higher ELO generally have slightly higher center control than players with lower ELOs. Also note, that mean center control in the end phase of games is more skewed towards 0 compared to the open and mid phases. Based on this analysis, we too split our centre control scores into 3 separate attributes, one for each phase.

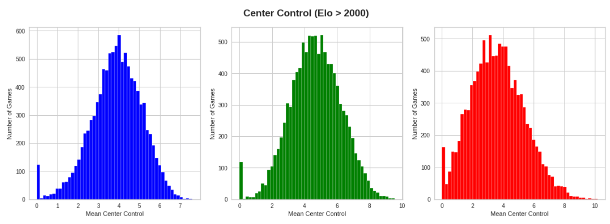


Figure 3: Center Control Over Phases For Players with ELO $>$ 2000

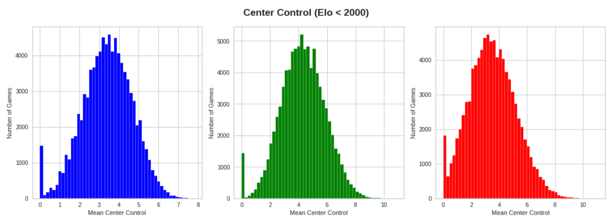


Figure 4: Center Control Over Phases For Players with ELO $<$ 2000

4.2 Preprocessing

For pre-processing we started out by first imputing the comments column in the CSV itself and thereafter removing some of the unwanted attributes like Site, Date, Black Player, White Player and Round from the created dataframe as they were of no use in determining the player rankings. Also, to maintain the homogeneity in the data, similar event types were merged for better results, eg. 'Rated Blitz Game' and 'Rated Blitz Tournament' were combined as 'Blitz'. Coming to the actual moves sequence, as they are given separately for each ply of moves, we combined them together in a single list to represent the entire state of a game. Additionally, the eval information was provided only for 15% of the games so we discarded those instead choosing to obtain that data from the source (Chess Engine) itself and for all the moves that we were analysing.

4.3 Challenges

(i) **Familiarity with the PGN format:** Even though we had played chess before, we were not aware the extent to which its used as a potential research tool for looking at problems. We familiarized ourselves with the various Chess openings, game variants, what each specific symbol in a PGN means, how Chess Engines actually work for a given specific depth etc.

(ii) **Handling the volume of data:** Given the huge repository of Chess PGNs available at Lichess with each file running into GBs worth of data, we had to figure out a way to process the files effectively without hitting a roadblock in the first stage itself. To get around this, we built a Python pipeline which would take a PGN file and in addition to parsing the moves and dumping the data to a CSV, also performed some computations on it in conjunction with the Python Chess API to give us the features we needed.

(iii) **Computational Deficiency:** We had a way to convert the source data, our PGN files, into a format that we could easily analyse, CSV files with extracted information. But we struggled to convert the PGNs to CSVs as it the conversion took hours to achieve. As a rough figure, it would take us 12-15 hrs to parse through 50k games on our personal laptops as well as Google Colab. Given that these computations had nothing to do with matrices or tensor calculations, having the extra GPU power did not help. In order to overcome this challenge, we used the Seawulf cluster and split our workload over 13 different nodes. Each node was responsible for converting data for 10k games. The added computational power and ability to process the data on several nodes in parallel effectively reduced the lead time. Finally, we were able to extract data for close to 125k games in close to 2 hours.

(iv) **Choice of Model:** In the initial stages while building our baseline model, we were in two minds whether to approach the problem as a classification or discrete value prediction (Regression). The classification approach would make our life a bit easier with ranges of ranks to predict but we figured that with real in-depth analysis the regression models should display good results as well. In the end we decided to use a hybrid approach and create two different models just to ensure that our regression results complement the ones we obtain from classification as well.

5 Feature Engineering

Since the moves cannot be directly fed into machine learning models, most of our efforts in this project were spent on intelligent feature engineering to quantify the moves and infer as much information as we can about the strategy of the players from them.

First of all, our main goal was prediction of Elo rankings based on the player moves, hence we divided the half moves between White and Black.This meant that rather than treat the games as a single entity, we created two rows for each game, one for Black and another for White with the corresponding features created from their half moves and their corresponding Elo Rankings as the labels.

Based on our knowledge of the game, online sources and the 5-year old Kaggle challenge (Kag), we created 58 features, which can be broadly categorized as 'Move Count' based, 'Board Control' based and 'Statistics based', some of which are explained below:

I **Moves Count Based:**

(i) **Opening Move:** Based on whether the Opening Sequence was a 'Defence' or 'Attack/Reversed Defence', the categorical column was set to 1 for Black or White respectively.

(ii) **(Player/Opponent)Captures:** The number of pieces captured by a player and the number of pieces lost. Captures by the opponent are represented with a negative value to punish the player.

(iii) **Checks/Check Mates:** The number of times player forces a 'check' on the opponent or finishes them off with a mate.

(iv) **Moves made by Significant Pieces:** The number of times significant pieces, Queen(Q), King(K), Knight(N), Bishop(B) and Rook(R) move. We also capture the first time these pieces were moved. (We did NOT mark the first move for each of the Bishops/Knights/Rook individually for the first move.)

Our choice of this too was based on some preliminary analysis. In figures 6 and 6 we plot number of times Queen and King moved against the Elo rating of the player.

While players with higher Elo generally move the King more than players with lower Elo, it is the opposite when it comes to the Queen. Similar can also be said about the Bishop, Knight and Root, though the effect is not as prominent as that of the Queen and the King.

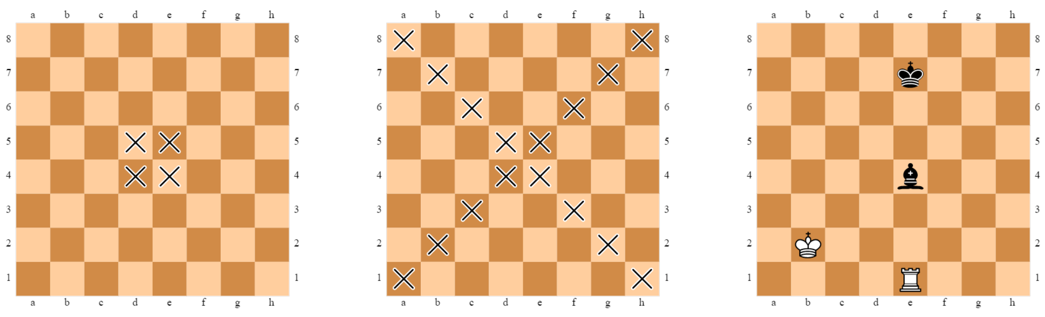
(v) **Questionable Moves:** Referencing the 'Comments' column from the previous section, some of the moves are annotated with a score from 2/4/6 which indicate a Poor Move/Blunder/Questionable Move respectively. We count the number of instances such a move was played by the player as well as the opponent.

II **Board Control Based:**

(i) **Special Moves:** Castling(Both Queen side and King side) and which are represented as 'O-O-O/O-O'.

(ii) **Promotion:** Promotion of pawn to special pieces represented by '=X' where X is the piece to which the pawn transforms too.

(iii) **Board Center Control:** These are some of the most important calculated and derived features that we have used. If the game has more than 10 moves,



(a) Center Control Squares (b) Diagonal Control Squares (c) Pinned Piece - Black Bishop

Figure 5: Board Control Based Feature Representation

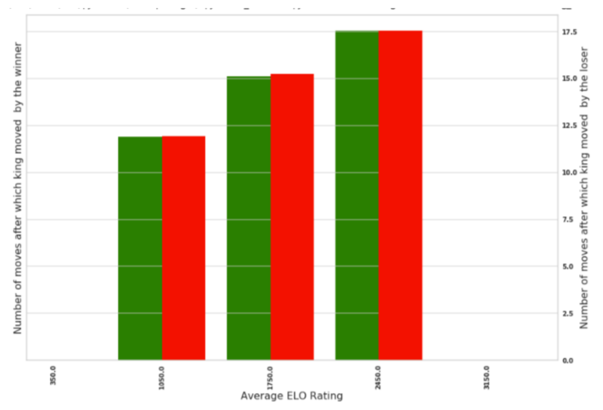


Figure 6: Num of moves for Queen vs Elo

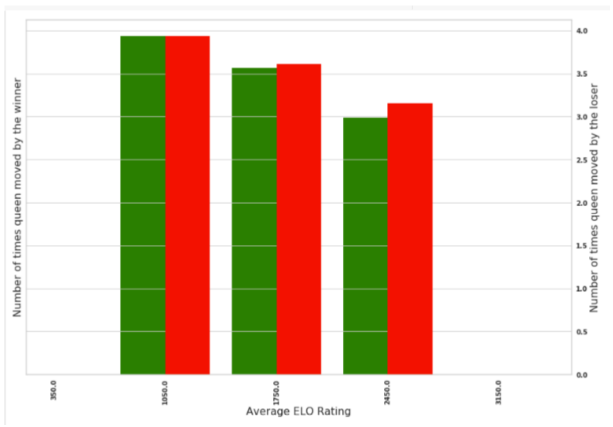


Figure 7: Num of moves for King vs Elo

we divide the game into three phases-Open, Mid and End moves. We define the Open-Phase as the first 10 moves, the Mid-Phase to be moves 11 to 28 and move 29 on wards is the End Phase. 10 moves is generally regarded as the number of moves used to define the opening phase. We chose 28 for end of the Mid-phase based on the average move number when the King is first moved in the game. Then, by the use of python-chess, we calculate the count of attackers on the center squares (4 middle squares) which are considered the most important as far as controlling the board space area is concerned.

(iv) **Board Diagonals Control:** Similar to Center Control, we also define the Diagonal Control based on the number of pieces that are attacking the diagonal squares. Also, we split the Diagonal Control metrics into 3 phases like as discussed above.

(v) **Number of Pinned Pieces:** This feature captures the number of pieces that are pinned to the King. These pieces cannot move without putting the King under a check and hence are 'pinned' to their square. It is an indicator of how many choices a player has at any stage. Having more pieces pinned is generally an indicator where the player might get locked down and that of a poor Elo rating.

III Statistics Based:

(i) **StockFish Analysis:**As mentioned earlier, we discarded any eval information provided with the PGN as eval was marked only for a few games and we wanted it for as many games as possible to make useful predictions. The official term for eval is ACPL (Average Centi Pawn Loss per move) which is an indicator how good or bad a move was in measures of Centi Pawn units which is generally considered a value of 1/100. We obtained the corresponding values for about 100k games and used them as features for our analysis.

(ii) **Mean Player/Opponent Gain:** From the evals that we obtained above, we can calculate the Gain or Loss per move for a player by measure the change in evals between plys. We calculated the mean of the Gain thus obtained which proved to be a good addition for our model.

(iii) **Difference Prediction between Elo Ratings of Players:** Based upon our various readings throughout the Project phase, we came across (Pre) where an algorithm was designed to predict the Elo rating difference for 2 players based on a single/set of games. The essence of the algorithm is still in eval but rather than depend only the eval, a series of evals are used to create a probability distribution which is in turn used to calculate a parameter 'D' which is the **perceived** difference between the ratings of two players. The distribution of perceived Elo difference over all games can be seen in 8. Notice, that number of games for which predicted Elo differences small are less is high. This is inline with what we expect - players with similar ratings are more like to play against each other.

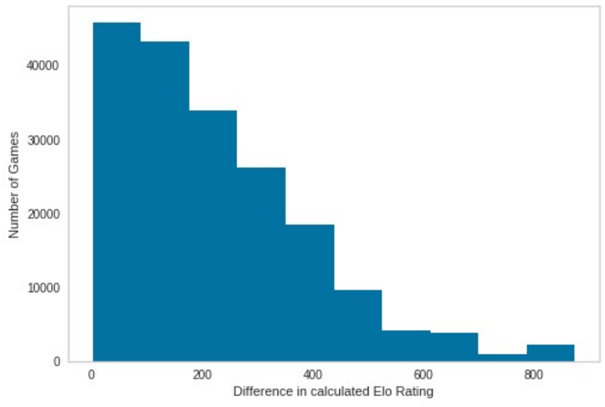


Figure 8: Distribution of Perceived Elo Rating Difference

Finally, we end up using the features as seen in 10. As per feedback from our Mid-Project Report we can see the correlation matrix in 9 as requested.

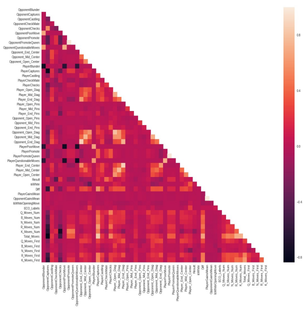
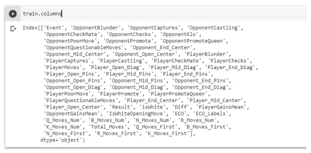


Figure 9: Correlation Matrix of Features Used



```
train.columns
```

```
Index(['Event', 'OpponentBlunder', 'OpponentCaptures', 'OpponentCastling',
       'OpponentCheckMate', 'OpponentChecks', 'OpponentElo',
       'OpponentPoorMove', 'OpponentPromote', 'OpponentPromoteQueen',
       'OpponentQuestionableMoves', 'Opponent_End_Center',
       'Opponent_Mid_Center', 'Opponent_Open_Center', 'PlayerBlunder',
       'PlayerCaptures', 'PlayerCastling', 'PlayerCheckMate', 'PlayerChecks',
       'PlayerMoves', 'Player_Open_Diag', 'Player_Mid_Diag', 'Player_End_Diag',
       'Player_Open_Pins', 'Player_Mid_Pins', 'Player_End_Pins',
       'Opponent_Open_Pins', 'Opponent_Mid_Pins', 'Opponent_End_Pins',
       'Opponent_Open_Diag', 'Opponent_Mid_Diag', 'Opponent_End_Diag',
       'PlayerPoorMove', 'PlayerPromote', 'PlayerPromoteQueen',
       'PlayerQuestionableMoves', 'Player_End_Center', 'Player_Mid_Center',
       'Player_Open_Center', 'Result', 'isWhite', 'Diff', 'PlayerGainsMean',
       'OpponentGainsMean', 'IsWhiteOpeningMove', 'ECO', 'ECO_Labels',
       'Q_Moves_Num', 'B_Moves_Num', 'N_Moves_Num', 'R_Moves_Num',
       'K_Moves_Num', 'Total_Moves', 'Q_Moves_First', 'B_Moves_First',
       'N_Moves_First', 'R_Moves_First', 'K_Moves_First'],
      dtype='object')
```

Figure 10: Training Features

6 Methods

6.1 Baseline:

Given the nature of the problem at hand, the most natural approach is to try a regression model to fit the features that we created as described earlier. We tried a Linear regression model based on the basic features that could be directly obtained from the moves sequence to have a good baseline models to build upon.
The combination of 13 features considered for the baseline model were:

(a) Result

(b) isWhite

(c) TotalMoves

(d) Number of Checks/Mates

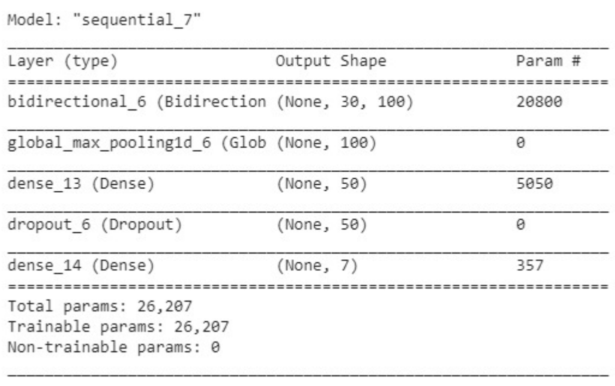
(e) Poor/Questionable/Blunder Moves of Player/Opponent (From Commentary)

The results obtained from our baseline model as outlined in the Results section helped us gain confidence enough that starting out with regression was the right approach and we could go from 13 basic features to a 58 features rich model which gave us considerable improvements over the baseline.

6.2 Intermediate and Final:

After finalizing our approach we set to work to improve our baseline model in an incremental fashion. By the time we reached the mid-project report stage we had 39 hand crafted features with the statistical features being saved for the final iteration. Finally, with all the features worked out we applied them to regression models of varied complexity starting from Linear to Random Forest Regressor and XG Boost Regressor.

7 Extended Work - Deep Learning:



```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_6 (Bidirection (None, 30, 100)           20800
_________________________________________________________________
global_max_pooling1d_6 (Glob (None, 100)               0
_________________________________________________________________
dense_13 (Dense)             (None, 50)                5050
_________________________________________________________________
dropout_6 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_14 (Dense)             (None, 7)                 357
=================================================================
Total params: 26,207
Trainable params: 26,207
Non-trainable params: 0
_________________________________________________________________
```

Figure 11: Bi-LSTM Model Architecture

As mentioned in the future work of our previous iteration from mid-sem project report, we tried to pursue the ambitious goal of applying deep learning models to the move sequence to replicate a sort of time-series analyses. The architecture consisted of a state of the art Bidirectional LSTM layer with MaxPooling concatenation and finally a Dense layer with 'softmax' activation for classification. The sequence of gains was limited at 30 after data analysis and for players with lesser moves the gains sequence was padded and for those with greater number of moves truncated. For our purpose, we used the sequence of gains scored by players in their games to predict :

7.1 Classification of Elo Ratings:

Given our regression results we divided the player ranking in our dataset into four ranges:

1. < 1000
2. $1000 - 1500$
3. $1500 - 2000$
4. > 2000

With a 500 ranks range we suspected we would get decent enough accuracy for predicting classes. Sure enough, as described in the Results section we obtained over 60% accuracy with the dominating classes showing the best accuracy.

7.2 Classification of Game Variant:

With a Dense layer classification model architecture in place, we extended our scope to include a basic prediction model for Game Type or 'Events' in the dataset. As mentioned in the Pre-processing section, we combined all the games into the following four classes:

1. $Blitz$
2. $Bullet$
3. $Classic$
4. $Correspondence$

Since using the Clock information provided in the PGN would directly correlate to the game variant and we were only interested in the move sequence analysis, we applied the same approach as in Elo Rankings classification to treat this is a Time-Series based prediction task. Again, since most of the entries in the dataset were for Blitz and Bullets the results as in the next section reflected that. Even after applying random resampling and upsampling of the minority classes we couldn't see any improvements in the results. Maybe this should be something that needs further investigation.

8 Results

8.1 Regression over Elo Ratings

As part of our experiments the models were initially run on a smaller dataset of 20000 games. Given that we split the predictions to be separate, we made predictions with a Mean Absolute Error(MAE) of around 155 for Linear Regression and 150 for Random Forest Regressor. We then pumped more data in, in hopes of improving the MAE but only managed a small bump. Feeding the model with 10 times the initial number of games, the MAE was closer to 147.37. We also performed predictions using XGBoostRegressor which gave us an MAE of 148.83. The results are displayed in table 1 and 2 for baseline and final results for our main task.

8.2 Range of Elo

Given that we were getting of RMSE of LR, we wanted to try classification-based approach by trying to classify the in the ranges of ELO. We get an accuracy of 64% by analyzing the games based on time-series data. The results can be seen in 3. The

	LR	RFR	XGBR
RMSE	205.08	204.61	204.45
MAE	163.44	163.24	163.08

Table 1: Baseline Results (LR: LinearRegressor, RFR: RandomForestRegressor, XGBR: XGBoostRegressor)

	LR	RFR	XGBR
RMSE	192.12	185.81	187.56
MAE	152.74	147.37	148.83

Table 2: Final Results (LR: LinearRegressor, RFR: RandomForestRegressor, XGBR: XGBoostRegressor)

models created are vanilla model with little complexity giving an excellent accuracy of 64%. Further enhancements can be made in this direction to improve the accuracy from 64%.

8.3 Game Type Classification

We tried to classify the games based on the game type. We purposely kept out the time field since there is a direct correlation between type Blitz and the time of the game. Results can be seen in 4

9 Conclusion

Starting with working knowledge of chess, this was an enlightening experience. Our main task was the prediction of Elo rankings of players based on their moves in chess games obtained from Lichess.com. From a baseline model with RMSE of 205 we were able to bring the prediction within a confidence of RMSE of 187 through smart feature engineering and use of available chess engine APIs. Further, we were also able to apply a deep learning model to the task of classifying Elo ratings and also the Game variant with baseline level accuracy. Those prove to some degree that various aspects of chess move sequences can be modelled as Time Series analysis to obtain appreciable results.

Elo Range	Precision	Recall	F1	Support
1000-1500	0.55	0.21	0.3	21319
1500-2000	0.65	0.91	0.76	38488
<1000	0	0	0	135
>2000	0	0	0	1906
Accuracy			0.64	61848
Macro Avg	0.30	0.28	0.27	61848
Weighted avg	0.59	0.64	0.58	61848

Table 3: Elo Range Classification Results

Game Type	Precision	Recall	F1	Support
Blitz	0.34	0.97	0.5	21319
Bullet	0.77	0	0	38488
Classic	0.01	0.04	0.01	135
Corres.	0	0	0	1906
Accuracy			0.34	61848
Macro Avg	0.28	0.25	0.13	61848
Weighted Avg	0.59	0.34	0.18	61848

Table 4: Game Type Classification Results

References

Determining the Strength of Chess Players Based on Actual Play. https://pdfs.semanticscholar.org/f761/d248b256704d3a0a2645aa43895164c6fa14.pdf. [Online; Accessed 06-Dec-2019].

ECO. https://en.wikipedia.org/wiki/Encyclopaedia_of_Chess_Openings. [Online; Accessed 14-Nov-2019].

KaggleFindingElo. https://www.kaggle.com/c/finding-elo/. [Online; Accessed 14-Nov-2019].

Lichess. https://lichess.org/. [Online; Accessed 14-Nov-2019].

NAG. https://en.wikipedia.org/wiki/Numeric_Annotation_Glyphs. [Online; Accessed 14-Nov-2019].

Protable Game Notation. https://en.wikipedia.org/wiki/PortableGameNotation. [Online; Accessed 14-Nov-2019].

python-chess. https://github.com/niklasf/python-chess. [Online; Accessed 14-Nov-2019].

Stockfish. https://stockfishchess.org/. [Online; Accessed 14-Nov-2019].