CS464 Final Report: Chess ELO Predictor

Enes Koç, Ege Aydın, İlter Onat Korkmaz, Arnisa Fazla and Furkan Kerim Çabaş
CS464 Section 01 - Group 02
Department of Engineering, Bilkent University

I. Introduction

In chess one of the rating methods that calculate the relative skills of the players to each other is the ELO rating system. It is calculated according to the outcome of the previous games the players have played, and ratings of opponents as well [1]. The objective of this project was to develop a classifier that classifies the ELO ratings of the players, given the individual games from the online chess server, Lichess.

We used a dataset of 1 million raw chess games in PGN notation taken from Lichess [2]. For the preprocessing, Stockfish Engine was used to analyze each move and extract the move scores and best moves for each player. Then their statistics were calculated. After that PCA was used to get a better insight into the data. Openings and move scores are the data used in the model.

The final model consisted of MLP with loss function MSE, two dropout layers each having ReLU activation function and two batch normalization layers. We got 87% accuracy for binary classification, however the model did not find a significant correlation between our feature set and the player ELO's, and performed poorly in regression. In order to increase accuracy of the model, Stockfish with more depths should be used and a model which can take variable length inputs should be developed. Then the model can be given the move scores data instead of their statistics which causes an excessive amount of information loss, and be trained to find the patterns in the data that might possibly lead to an improvement in the regression results.

II. Problem description

The ELO score was invented to rank chess players in terms of their absolute skills. Therefore, the estimation of ELO scores is also a problem of estimating the absolute chess skill of players. Given that the ELO score is determined by the result of the player's previous games, there is no direct connection between the moves he made in a single game and his/her absolute chess skill. However, we can hypothesize that high-ranked players make very good moves and low-rank players cannot. This means that in a way, we try to determine whether this hypothesis is true or not. After generating the features, the first two principal components are plotted to see if there is a tangible relation. The following questions which are taken from the competition are going to be answered in this work.

- Do a player's moves reflect their absolute skill? [3]
- Does the opponent matter? [3]
- How closely does one game reflect intrinsic ability? [3]
- How well can an algorithm do? [3]
- Does computational horsepower increase accuracy?[3]

III. Methods

A. Preparing the dataset

The input data had 1 million games where each game's ELO ranking was distributed normally. From that 1 million games we selected a subset such that the ELO ratings were distributed uniformly. This meant selecting all the games with low and high ratings but only selecting a random subset of midrank games. We performed this step because we noticed that if we don't, the model performs very poorly. When we oversample the data we observed overfitting. Since we already had more than enough data, just taking a uniform subset worked the best.

B. Features

The dataset we used was composed of different games between different people with their respective ELOs. This dataset is in PGN format where each game is in SAN format. In order to do some prediction from the data, extracting some kind of features from these games is necessary. Each chess game in the PGN file is extracted one by one and various features are extracted from the SAN string.

1) ECO Codes: One of the main features of any chess game is its opening. If a player is playing above the normal, then he/she tends to play more professional chess openings. In the case of beginners, they tend to play more basic openings. To take the opening data of games, we used the Encyclopaedia of Chess Openings (ECO) codes that are standardized opening codes in chess. There are 500 ECO codes, but we consider them in 13 main categories. So we one hot encoded our ECO codes for each game and created a feature vector length 13 that corresponds to the opening played in the game.

2) Move Scores: Each move is analyzed and their strengths were quantified using a chess engine called Stockfish. Stockfish chess engine gives a centipawn score to each move, which stands for an advantage of 1/100 of a pawn per unit [4]. However, the CP score does not linearly correlate with the situation the player is in. In order to get a better metric about move scores, a Method Called WDL is Used. WDL stands for "Win-Draw-Lose" and shows the rate of people who won their games from the given CP score. This value is normalized such that it gives 1 for white wins, 0 for black wins, and 0.5 for a draw. This gave us a good metric about determining how good each move is and by observing the changes in the score, it was possible to make a guess about how good a player is. However, there were two problems with using the games move by move. The first one was that all the games are of varying lengths, and therefore contain move score arrays of varying lengths. Since the data couldn't be rescaled or interpolated like image data, we couldn't give the move scores directly as input to train our model. How we processed this kind of different length ”series” data is explained in the succeeding parts. The second problem was the excessive amounts of computation power required to obtain these scores. We used the stockfish engine asset to a specific depth of 8. This means that the engine takes at most 8 moves after each move into consideration when analyzing each move. Running the Stockfish engine on our dataset of 1 million games with an average length of 80 moves took us 2 days with 5 of our computers running 24 hours. Since Stockfish can also work multithreaded, it has used 100% of our CPUs making us unable to perform any other task while it is running. However, when we got the scores, they were much less than optimal, because of the depth we used. Normally stockfish is run at depth 20 or more and its strength increases as its depth increases. However, at that depth, it takes around 1 second per move with our processing power. This made it impossible for us to use more depth and as a result, we got inadequate and partly missing data.

3) Best Moves: One other thing stockfish can do is to find the best possible moves to play and their respective move scores given a specific board. We used that option to generate the 5 best possible moves for every move and compare it to the played move. In one hot encoding, This gave an array with the length of 5 per move. Preprocessing steps of this kind of series data will also be explained in detail in the following parts.

4) Game Scores: Other sets of features taken into account are some intrinsic game features such as the first move the queen is played, total promotion count, total check count, the move that first check occurred, etc. The total number of them is 24. These values can be easily reached using the chess module in Python, which offers functions specifically made for move generation and validation [5].

5) Statistics Extraction from Series Data: In order to use the series data we have in our model, we calculated some statistical values from the given series such as mean, median, standard deviation, maximum, minimum, and used them to train our model. Although using these directly does not give us all the information about each game, we couldn't find a better way since this was not time-series data and all the methods used for dealing with variable input size were about time data or image data. Also, we saw some improvement when the series data was split into equal-sized parts and their statistics were extracted from each part.

C. Model

A neural network was used as the model. The neural network predicted the average and difference of ELOs for White and Black. Two hidden layers were used with the number of nodes determined through trial and error. Our initial metric for the loss function was Mean Absolute Error but then decided to use Mean Squared Error after experimenting with both and finding out Keras minimized MSE in fewer steps than MAE, with roughly the same accuracy. Two dropout layers, L1 and L2 kernel regularization are used to prevent overfitting. Also, two batch normalization layers were used since using them decreases the mean absolute error. Activation functions using ReLU showed much better results than functions like Sigmoid. We also noticed that using leaky ReLU further decreased the MAE. The hidden layers have 401 and 28 neurons respectively. The full structure of the model is given in the Appendix B Figure 7. Other than NN, XGB Regressor is used and compared which is used in this project [6]. The final parameters for it are decided as estimators=250, and learning rate=0.05.

IV. RESULTS

When doing regression, we got a MAE score of 144 which was better than previous work on this subject. Ideally, the prediction of all the games with the same ELO should have a gaussian distribution with a mean of real ELO and minimal variation. However, when the real ELOs are compared with predictions on a scatter graph, we see that our prediction is not distributed as desired (Appendix B Figure 1). The “prediction means” do not follow the real ELOs. However, there is a sharp transition at 1550 that can be easily seen. When we do a binary classification to predict whether the player's ELO is higher or lower than 1550, we get accuracy as high as 87%. The false positives and false negatives were balanced so the accuracy score alone is a satisfactory metric for the goodness of the model (Confusion matrix Appendix B Figure 6). We also plotted the AUROC curve and compared it with the previous work on this topic (Appendix B Figure 8). For XGBRegressor, the accuracy rate is 84.7% which is lower than NN. The MAE score of the model is 148. The confusion matrix made for NN is made for this model and it is in Appendix B Figure 5.

V. DISCUSSION

We are using features we generated to predict the ELO of the players. These features mainly come from the Stockfish chess engine. Since we couldn't search with enough depth, our features were imperfect and were not sufficient to give a correlation between players' game and ELO score. We also noticed when the difference of ELO between players is large, features gave contradicting results as a low ELO player played like a grandmaster when playing a total beginner. This caused the features of some low and high ELO scored games to appear similar and they were affecting the output of the model in a wrong way. This led us to calculate the difference in ELOs between white and black players and select only the games between similarly ranked players. One game may be sufficient to see the skills of a player, however only evaluating the move scores is not enough. Finally, we noticed that our dataset was composed of a variety of games where some players play seriously, some don't, some games were 1-minute bullet chess, some were classical, etc. Better results could probably be achieved with a more specific dataset, more data, more depth in the chess engine. For the comparison between NN and XGB Regressor, the following can be said. The accuracy, precision, and F1 measures of NN are higher than XGB Regressor but the recall measure of XGB Regressor is higher than NN. The XGB Regressor is better at finding the negative values which correspond to the ELOs smaller than 1550. This model is a type of decision tree. Therefore, it can be said that the decision tree-based model performs a better job on the lower ELO scores. Nevertheless, the accuracy of the NN is higher than it and so, NN is preferred in this work.

VI. Conclusion

The questions that are asked in the problem definition are answered. First of all, with the current analyzing resources, the absolute skill of the players could not be found well by looking at the one game of the player. Our feature set gives an 87% accuracy for binary classification, so this shows that our methods extract the ELO information in some boundaries. But in regression, it does not work as expected. In order to avoid this problem, the higher-depth engines should be used by utilizing more resources. Secondly, the opponents' ELO difference leads to very unsatisfactory results. In this work, only the games in which players have close ELO scores are taken into consideration. It is because the move scores do not give any clear correlation between features and ELO differences. To overcome this problem, additional features from the games should be extracted. However, the model can distinguish between high and low ELOs well. Furthermore, we got a MAE score lower than previous works on this topic. Therefore our model is successful for this job. However, to make it better in the regression, additional resources are needed.

References

[1] Wikipedia contributors, *Elo rating system — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Elo_rating_system&oldid=1021657251, [Online; accessed 14-May-2021], 2021.

[2] E. Zhang, *Raw chess games (pgn)*, https://www.kaggle.com/ironicninja/raw-chess-games-pgn, [Online; accessed 14-May-2021], Jan. 2021.

[3] *Finding elo*, https://www.kaggle.com/c/finding-elo, [Online; accessed 14-May-2021].

[4] Fandom contributors, *Centipawn*, https://chess.fandom.com/wiki/Centipawn, [Online; accessed 14-May-2021].

[5] *Python-chess: A chess library for python*, https://python-chess.readthedocs.io/en/latest/, [Online; accessed 14-May-2021].

[6] Y. M. Palenzuela, *Model training.ipynb*, https://github.com/elyase/kaggle-elo/blob/master/Model\%20Training.ipynb, [Online; accessed 14-May-2021].

Appendix A
Contributions

Ege Aydın:

Data preparation	Multithreaded Stockfish engine communication, game analysis, uniformization
Feature generation	Generating all of the features, PCA
Training	Finding optimal meta-parameters for the neural network
Testing	AUROC curve and various test metrices

Enes Koç:

Data preparation	The Stockfish WDL points are prepared for the algorithms.
Training	Various Models such as XGB Regressior, NN are tried in order to compare them.
Research	Research is done to see important aspects of the data, and find suitable models.

İlter Onat Korkmaz:

Document preparation	Participated in the preparation of the documents such as the Proposal, the Progress Report and the Final Report.
Model preparation	Determining the structure of the models.
Training	Training NN models.
Feature extraction	Arranging the feature extraction.
Model Evaluation	Prepared the confusion matrices.

Arnisa Fazla:

Data preparation	Stockfish CP and WDL points preparation per move
Training	Trained an NN model taking an instance of the board per move as a one dimensional array as input. The board instances were generated using Python chess library and converted to one-hot-encoding format; and then flattened to a one dimensional array.

Furkan Kerim Çabaş:

Data preparation	The Stockfish WDL points are prepared.
Document preparation	Participated in proposal documentation.

Appendix B
Figures and Tables

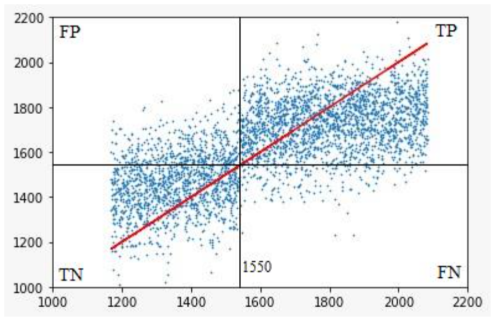


Figure 1. Distribution of Predicted Result

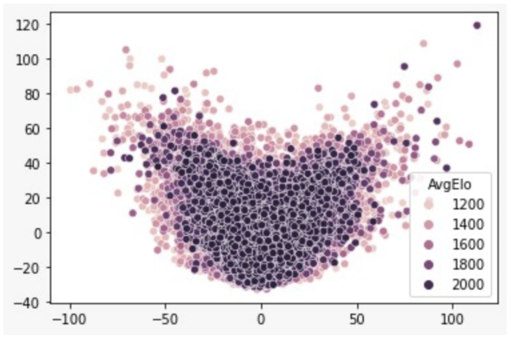


Figure 2. Distribution of Data

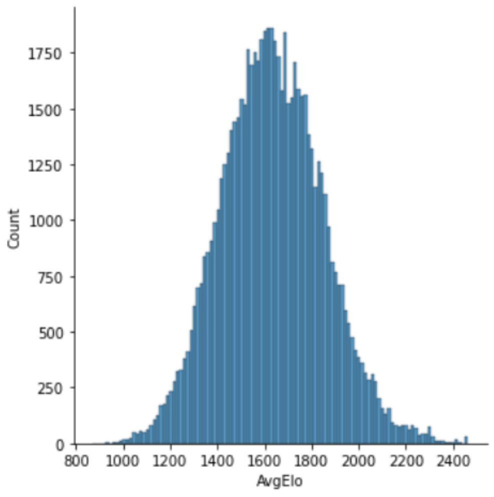


Figure 3. The Dataset Before Uniformization

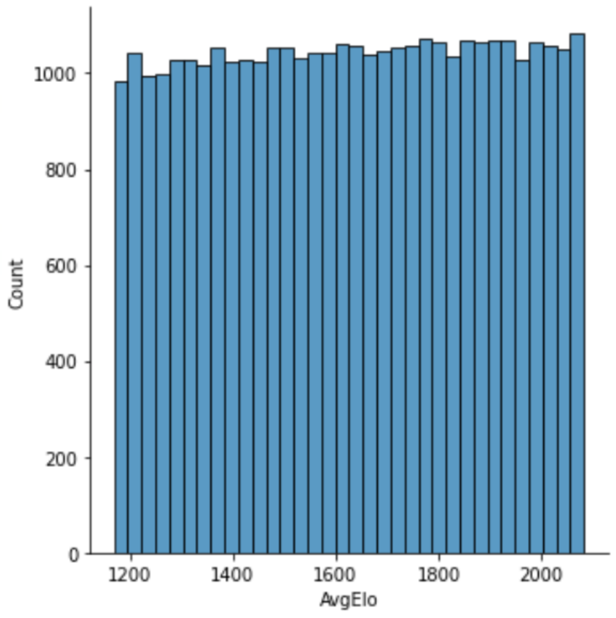
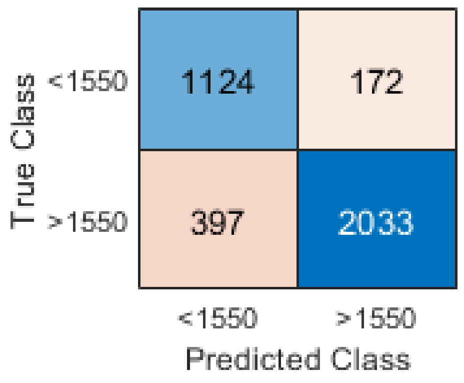


Figure 4. The Dataset After Uniformization



	<1550	>1550
<1550	1124	172
>1550	397	2033

True Class
Predicted Class

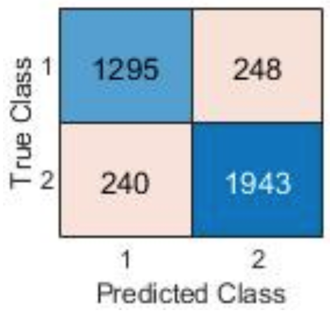
Figure 5. The Confusion Matrix for XGB Regression

Accuracy = 84.7%

Precision = 73.9%

Recall = 86.7%

F1 = 79.8%



	1	2
1	1295	248
2	240	1943

True Class
Predicted Class

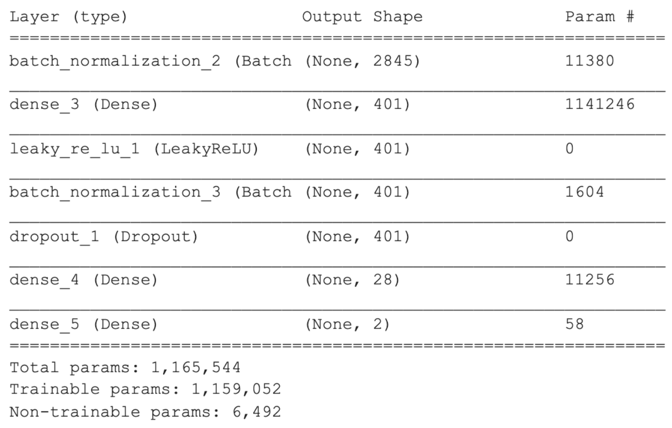
Figure 6. The Confusion Matrix for Neural Network

Accuracy = 87.0%

Precision = 83.9%

Recall = 84.4%

F1 = 84.4%



```
Layer (type)                 Output Shape              Param #
=================================================================
batch_normalization_2 (Batch (None, 2845)              11380
_________________________________________________________________
dense_3 (Dense)              (None, 401)               1141246
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 401)               0
_________________________________________________________________
batch_normalization_3 (Batch (None, 401)               1604
_________________________________________________________________
dropout_1 (Dropout)          (None, 401)               0
_________________________________________________________________
dense_4 (Dense)              (None, 28)                11256
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 58
=================================================================
Total params: 1,165,544
Trainable params: 1,159,052
Non-trainable params: 6,492
```

Figure 7. The Neural Network Summary

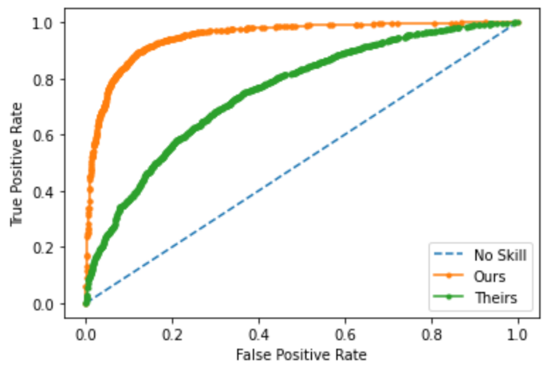


Figure 8. AUROC Curves and binary classification over regression