Chess Puzzle Rating Prediction Using ML Techniques

CS 364M

Praveen Mogan, Tony Tian, Kevin Pham

Background

The motivation behind this project stems from inherent challenges associated with current methods used to rate chess puzzles on online platforms like Lichess and Chess.com. Presently, puzzle ratings are often determined through mass sampling, which introduces biases and inaccuracies, especially for newly discovered puzzles with unknown difficulty levels. These limitations have led to confusion and frustration among chess enthusiasts, as discussed in various online forums, an example: here.

Our goal is to address these shortcomings by developing a system that accurately predicts puzzle ratings using machine learning techniques. By utilizing a database of chess puzzles and employing Python's Chess library, we aim to transform textual puzzle representations into over-the-board formats. Through feature extraction and the creation of novel features/functions to capture puzzle complexity, we seek to provide a more empirical and accurate method for rating chess puzzles.

The scientific basis for our projects draws upon various research papers and concepts in the fields of chess analysis and machine learning. Papers such as "An Effective Approach in Endgame Chess Board Evaluation" by Trunkat M. and Kadlec R. (2017), "Robust Locally Weighted Regression and Smoothing Scatterplots" by Cleveland W.S. (1979), and "Regression Shrinkage and Selection via the Lasso" by Tibshirani R. (1996) provide valuable insights into chess board evaluation, regression techniques, and feature selection methods. Additionally, works like "Deep Learning and the Information Bottleneck Principle" by TishbyN. and Zaslavsky N. (2015) and "Multilayer feedforward networks are universal approximators" by Hornik K., Stinchcombe M., and White H. (1989) offer foundation knowledge on neural networks and their capabilities.

By synthesizing these research findings and leveraging modern machine-learning methodologies, our project aims to contribute to the advancement of chess puzzle analysis while addressing the existing limitations in puzzle rating systems. Through experimentation and evaluation, we aspire to develop a robust model that can accurately predict puzzle ratings, thereby enhancing the overall puzzle-solving experience for chess enthusiasts

Data cleaning

1. We filtered out all puzzles with fewer than 100 plays. This not only reduced the size of our dataset but also eliminated ratings that might not be accurate enough for our model to train on.
2. We one-hot encoded the “Themes” column. This column contains a list of themes/tactics, so we split them into individual features taking binary inputs.
3. We perceived that the first move in the “Moves” column is not part of the solution but rather the setup move for the chess puzzle. Therefore, we needed to remove the first move and update the sample’s FEN by converting the FEN to a board representation, pushing the move, and replacing the original FEN code with the FEN of our new board.

Feature Extraction

1. Material_sacrifice: We believe that puzzles involving sacrifice/exchange sacrifice tend to be more difficult thus having a higher puzzle rating than the ones that don't. We defined the "exchange period" as a sequence of capturing moves. For each exchange period, if the total piece value loss is greater than the opponent's, we increase the material_sacrifice feature for this puzzle by the difference. The piece value for each piece follows that of chess.com
2. Puzzle_length: the length of the solution to the puzzle. We believe that the longer puzzles tend to be more difficult
3. Avg_num_captured: the number of opponent's pieces captured averaged by the length of the solution. We believe that the capturing moves tend to be easier to spot, making the puzzle easier
4. Avg_num_being_captured: the number of pieces being captured averaged by the length of the solution. Similar to material_sacrifice, players are generally unwilling to lose material, making puzzles that allow the opponent to capture multiple pieces harder.
5. Material_difference: the difference in the total piece value. It is easier to find moves when we have a significant material disadvantage because the moves must be sharp and thus easier to spot
6. Avg_num_bad_checks: the total number of possible but incorrect checks averaged by the length of the solution. Checks are tempting pitfalls, making the puzzle harder
7. Avg_num_bad_captures: the total number of possible but incorrect captures averaged by the length of the solution. Same idea with Avg_num_bad_checks
8. Avg_legal_moves: the number of moves that the player might consider throughout the solution averaged by the length of the solution. We made a very simplistic assumption that the number of moves a player will consider in each position = total number of legal moves in the position ^ 0.8. Our reason is that "The average branching factor for min-max in chess is about 35, but with the alpha-beta pruning and sorting, the program achieves a branching factor of around 25". However, since the player is making the moves knowing they are solving a chess puzzle, they can further prune the moves that are less forcing ($35^{0.8} \approx 17$)

We experimented with the "sharpness" of a position as suggested in the feedback of the proposal, but we didn't include it as one of our features because the sharpness is quantified by the metric called "WDL". We observe that the WDL given by the Stockfish engine always has higher entropies for the middle game e.g. (Win=990, Draw=10, Loss=0) than the end game which almost always produces (Win=1000, Draw=0, Loss=0) regardless of the rating of the puzzles. This is understandable as to the engine, regardless of the difficulties of the moves, the position is always clearer when there is a mate-in-20.

Models

Simple feed-forward Neural Network(MLP)

We believed that Neural Networks might outperform traditional ML approaches due to the hype surrounding them and their ability to capture complex relationships like a black box. We used the Adam optimizer and MSE loss, achieving an MAE of 344.54. Unlike traditional ML methods, we did not reduce dimensionality beforehand using techniques like PCA, as neural networks can learn any nonlinear mapping and are not limited by linear models.

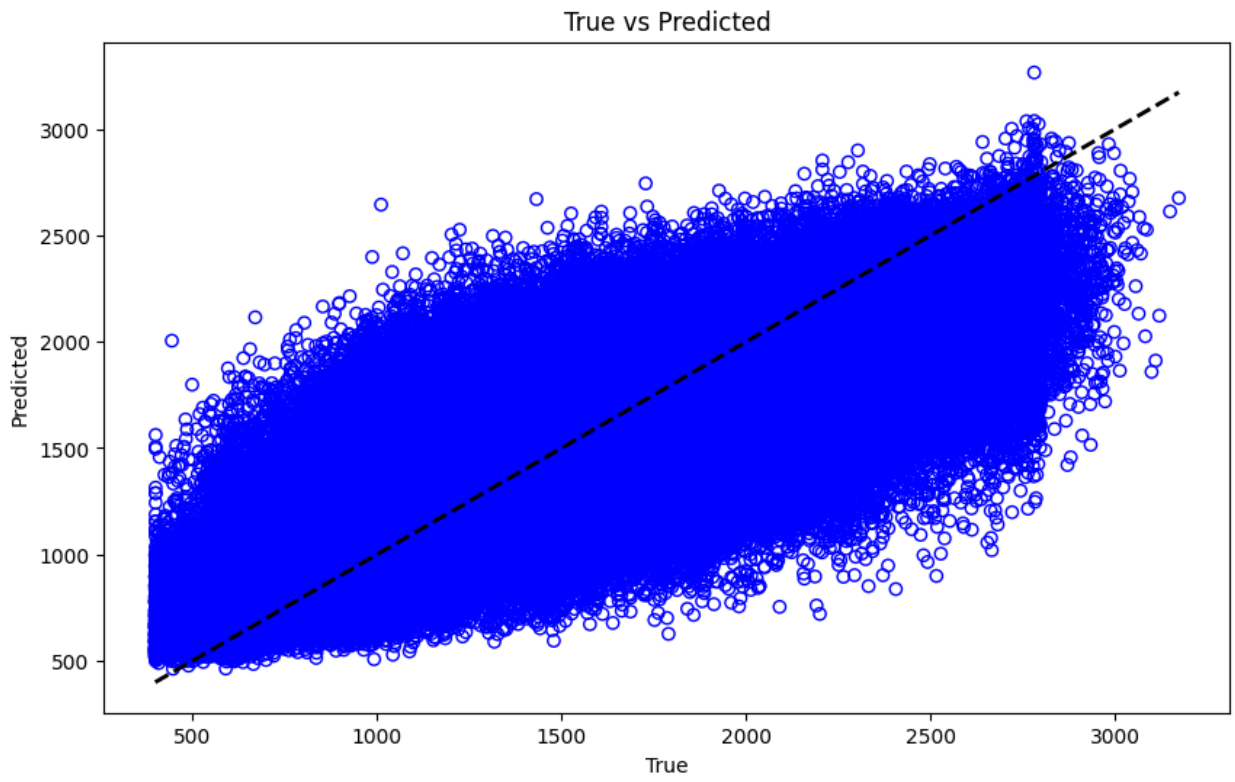
```
Exiting from training early
Test MSE loss: 127577.914
```

After adding some regularization and hyperparameter tuning, the MSE loss was decreased to 118707.562

```
EPOCH 24
 Train loss: 117016.11051
   Val loss: 118823.22084

EPOCH 25
 Train loss: 116923.41894
   Val loss: 118779.09801

EPOCH 26
 Train loss: 116840.05209
   Val loss: 118730.81949

EPOCH 27
 Train loss: 116767.11759
   Val loss: 118741.21837

EPOCH 28
 Train loss: 116690.89638
   Val loss: 118720.50184

EPOCH 29
 Train loss: 116620.24704
   Val loss: 118739.83535

EPOCH 30
Exiting from training early
Test MSE loss: 118707.562
```

Our results show that a simple MLP was actually outperformed by most traditional ML algorithms. It's worth noting that we only experimented with one neural network architecture, and optimizing the number of hidden layers and their widths is impractical due to the time-consuming nature of training these models.

The plot indicates homoscedasticity, suggesting consistent performance across puzzles with varying difficulty ratings. However, the predictions appear conservative, with higher predictions for lower-rated puzzles and vice versa.

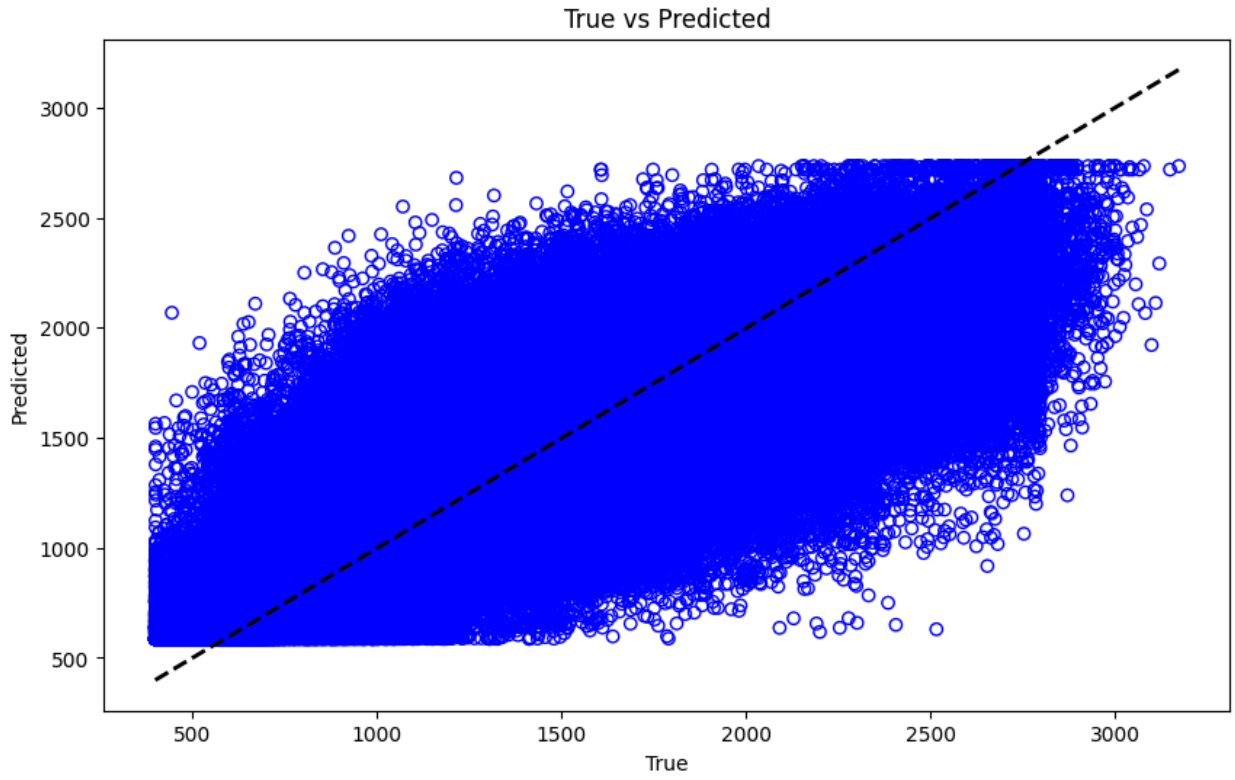


TabNet

TabNet is a deep learning architecture developed by Google researchers in 2019, specifically designed for problems where inputs and outputs are structured as data frames. We anticipate it will outperform simple feedforward Neural Networks. Utilizing the state-of-the-art deep learning architecture TabNet regressor, we achieved the lowest MAE among all models, with a test MAE loss of 273.305. While overfitting can be a concern for deep learning models, our dataset of 2.7 million samples and the built-in L1 regularization, known as "lambda_sparse," helped mitigate this issue.

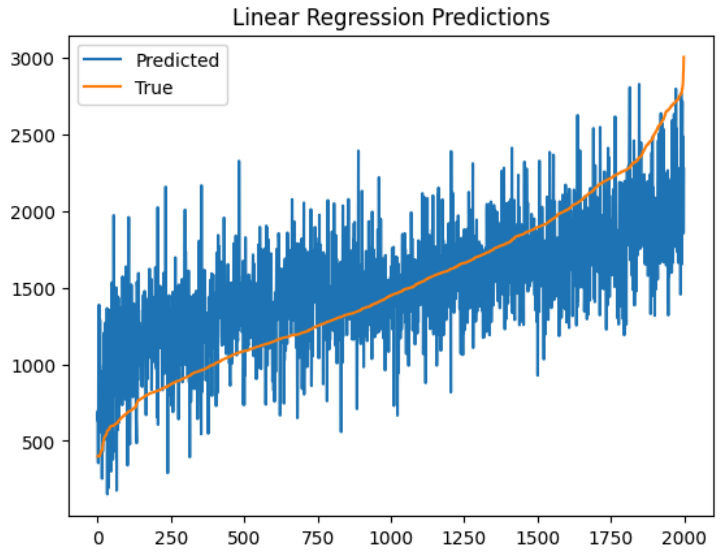
```
epoch 16 | loss: 275.76119| val_0_mae: 274.009521484375|  0:55:49s
epoch 17 | loss: 275.75598| val_0_mae: 273.8667297363281|  0:57:24s
epoch 18 | loss: 275.56604| val_0_mae: 274.0772705078125|  0:59:07s
epoch 19 | loss: 275.41983| val_0_mae: 273.2454528808594|  1:00:50s
Manually exiting from training early
Test MAE loss: 273.305
```

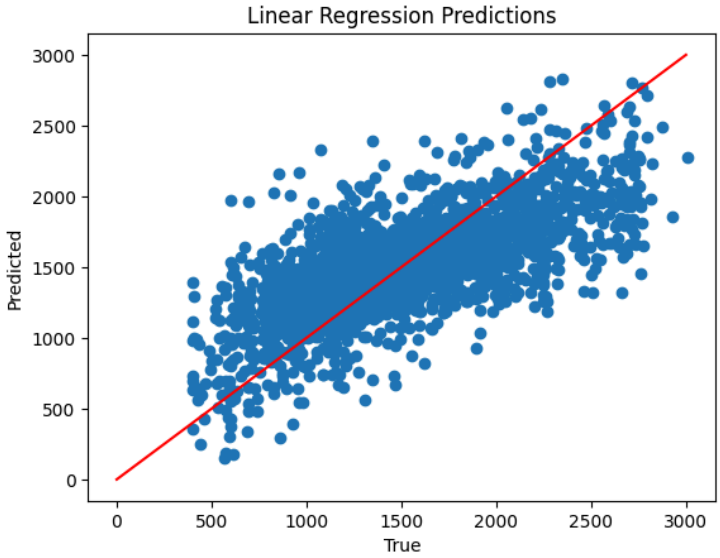
Nevertheless, observe the scatterplots below. The homoskedasticity phenomenon is still pervasive.



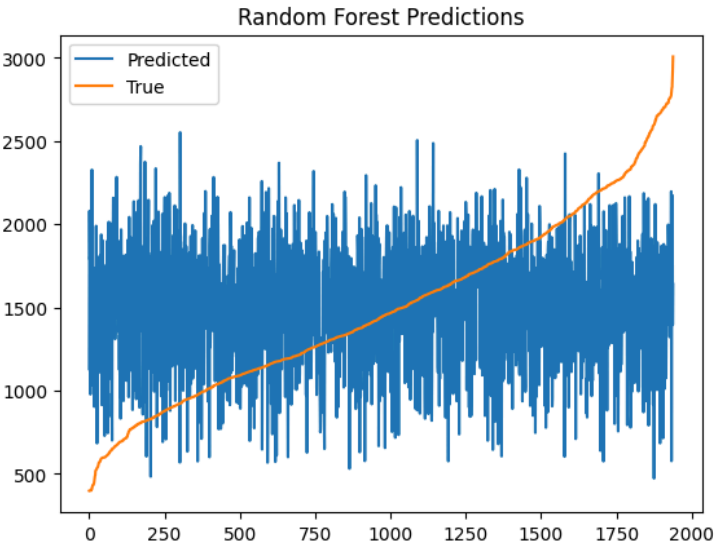
Scikit Learn Regression Models

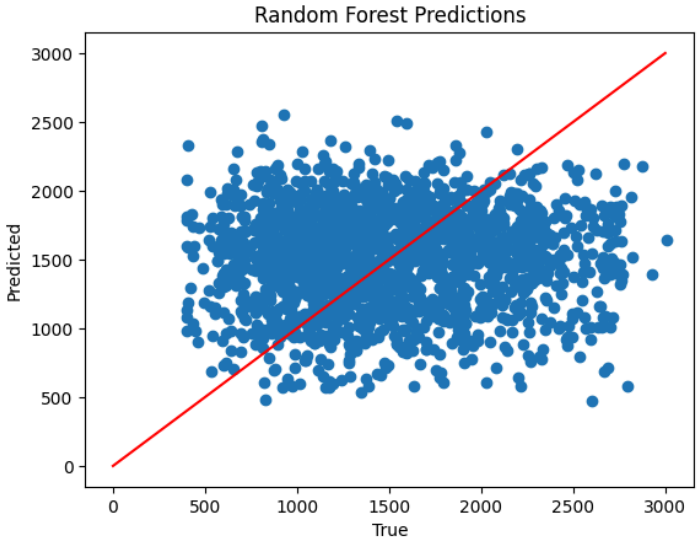
Afterwards, we explored various regression models in the Scikit-Learn library to tackle this problem. We were able to average a MAE of 310 across all models. Despite not performing optimally, we were surprised to see that linear regression had a MAE of 317 using a 100,000 record sample. This was around the median of the performance of the other regression schemes. For more detailed performances of each model, refer to the linked repository.





Overall, we found that models that tend to overfit such as random forest and polynomial regression tended to overfit the training data and had worse comparable testing errors, unable to find more general patterns in the dataset.

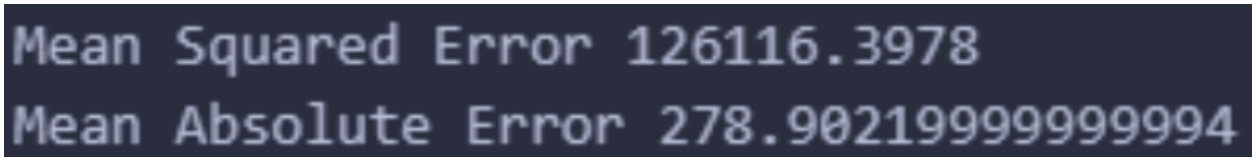




Custom Regression Models

We also implemented various custom regression models such as linear regression, polynomial regression, locally weighted polynomial regression, KNN Regression, and a regression Tree. Overall, the performance of these regression schemes was somewhat comparable to those in the Scikit learn implementation. However, we did notice a significant performance degradation as our models were not optimized for speed. In the future, we could look into vectorizing our implementation and more efficient distance computations using a K-D tree, as opposed to calculating distances on every request in KNN regression. Consequently, we had to reduce our dataset to 10,000 to 1,000 records depending on the model in order to allow the custom models to converge in a reasonable amount of time. As a result, even though some of the models performance is comparable to their respective scikit-learn counterparts, the smaller dataset may

provide greater opportunities for insignificant patterns to be learned by the models instead of larger themes. Regardless, the results were able to average around 320 MAE with an outlier being polynomial regression. We also found that implementing L2 regularization on top of polynomial regression did manage to bring the error down, but it was still well within the 6 digit range. An example of a surprisingly good result on KNN regression is shown below. Once again, we believe this is due to a small subset being considered and not an indication that our custom KNN regression managed to outperform the Scikit Learn version.



Once again, see the linked repository for a more detailed breakdown of each model's performance.

Conclusion

It became evident that the model's struggle to improve was not due to differing architectures, but rather the lack of distinct features to differentiate difficult puzzles from easier ones. This realization debunked our idea of combining the strength of different models via weighted voting e.g. if TabNet is accurate with puzzles with a rating > 2000 then if it predicts a puzzle rating > 2000 then give more weight to this prediction.

While there may be additional hidden features that Stockfish analysis could potentially uncover, dynamically analyzing 2.7 million samples is impractical time-wise. Even our static analysis took half an hour to compute. Assuming Stockfish requires at least 0.5 seconds to analyze each puzzle and its subsequent moves, it would take more than 2 weeks to complete the analysis.

In addition, it became clear as we were looking at the regression schemes that “vanilla” regression without regularization might cause overfitting problems. This became extremely apparent in our polynomial regression in the form of high coefficients and extremely large MSE/MAE values being reported at “convergence.”

Ultimately, we will need to experiment with different neural network model architectures, feature extraction, and regularization parameter tuning to reach lower errors than those presented in this paper.