Diego Izaguirre

Final Project

19 March 2024

Abstract

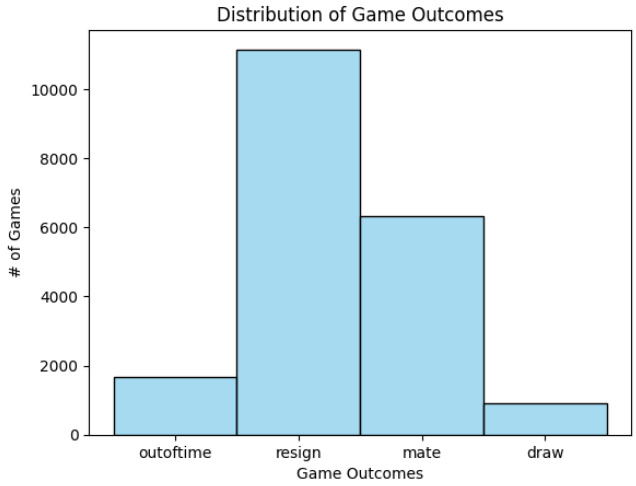
This project explores the predictive power machine learning when applied to chess games. I aim to discern the factors influencing game outcomes and predict the status of a victory. With a dataset from Lichess with over 20,000 games, I applied Random Forest, Gradient Boosting, and Support Vector Machines to predict whether a game would end in resignation or mate. My findings highlight the significance of early game aggression, time management, and player skill level, as evidenced by the importance of features like opening ply, piece moves, time, and player ratings. My conclusions also suggests that specific openings do not play as pivotal a role as one might think. Despite limitations such as the inability to capture move sequences and class imbalance, the models achieved an accuracy of up to 70% and an AUC of 0.74. These results could offer valuable insights for AI chess algorithms, chess training programs, and players' self-analysis, emphasizing the multifaceted nature of chess strategy.

Introduction

Chess has been one of the most studied topics in machine learning and artificial intelligence. Since it is primarily a game of patterns, studying chess provides good data for capturing these patterns using machine learning to learn from them. Studying this chess dataset can help figure out what meta-factors, openings, and other in game factors influence the outcome of a chess match. There are possible applications in helping people learn to play chess. Which pieces should be focused on the most? Which opening is the best? These are some examples of some questions that might be answered by studying chess. In my application of machine learning on chess data I plan to use random forest, gradient boosting, and support machine vectors to classify whether a chess match will end in a resignation or a mate.

My dataset was obtained from Kaggle and utilizes game data from Lichess - a free open-source internet chess server - which consists of 16 total features and 20,100 game observations. The features are: "id", "rated", "created_at", "last_move_at", "turns", "victory status", "winner", "increment_code", "white_id", "white_rating" "black_id", "black_rating", "moves", "opening_eco", "opening_name", "opening_ply". The data has a mix of categorical, continuous, and unique features. The target feature of this project is

"victory_status" which denotes how the match was won. This feature has 4 categories which are resign, mate, out of time, and draw. Though it would be interesting to utilize all four of these classes to make the classification models, the data is simply too unbalanced to make use of all of them. Out of time and draw account for only 13% of the data combined and resign and mate take up 56% and 32% respectively. This would make it difficult to correct the class imbalance without losing many observations. Such is the reason I am only using models to classify resignations and mates.



Literature Review

There is a lot of literature surrounding the topic of chess and machine learning. It is a very well-studied topic. The applications range from classification to regression and even the use of deep learning. One such paper uses various models to predict the ELO rating of a player [1]. The models they used were linear regression, neural networks, random forest, and XGBoost. For each game they collected the data and time, white and black ratings (Glicko-2), game result, time control, opening, list of moves, evaluation type, and evaluation score. They were able to get the best performance out of the XGBoost model which yielded an RMSE of 159. Since this paper is a regression problem and my project is a classification problem it is not possible to compare our models directly. Especially, since in this paper ELO rating is being predicted and not victory status. Despite this I found this paper helpful because there were some similarities to my own data that I saw. Especially since both of our data came from Lichess. As such I borrowed some ideas on preprocessing from this paper.

Another paper I read was one that opted to predict the outcome of chess match after twenty moves [3]. This paper is overly complex, and their data requires a ton of preprocessing to make it best for the models. They used support vector machines and random forest for this classification task. Rather than typical accuracy and cross validation scores the researchers in this paper chose to use cross-validation error rates and out-of-bag (OOB) error to evaluate their models. Their goal was to use a model that would generalize well and work with enormous quantities of data. In the end they considered the SVM to perform the best.

Much of the literature also focuses on cheating detection during online chess game. With the rise of the internet and playing chess online, it has also become easier to cheat by having a chess engine pick your moves for you. One such paper I read aims to use machine learning classifiers to classify such cheating in a chess game [2]. They tried doing this with discriminant analysis, logistic regression, and k-nearest neighbor. Part of their data was also obtained from Lichess just like mine, but it was aggregated with other features such as the number of blunders or good moves made by the players as given by the chess engine on Lichess. Though they were not able to build a model with usable performance. They conclude that while machine learning is a good tool for picking up anomalies it cannot be used to reliably judge games on its own.

Many studies also focus on analyzing the board rather that predicting something. The literature I have read focuses on using convolutional neural networks (CNN) to classify the game state from an image of the chess board [4]. They utilize data obtained from synthetic 3d models of a chess board under different lighting and board configurations. Their process is in 3 stages three main stages: (1) board localization, (2) occupancy classification, (3) piece classification [4]. With this they can classify whether the game is in the opening, middle game, or end game. This can prove valuable to people wanting to improve their chess game by obtaining analysis without having to go through the trouble of inputting all their chess moves into a program.

As is evident by the abundance of literature about machine learning applications in chess, there is a wide variety of models that are used. There is also a wide variety of applications for these models in chess. I decided to use random forest, gradient boosting, and SVM in my project because these were the models most used in papers like mine. Ones where they were trying to classify the outcome or if a move was good or bad for instance. I also saw many neural networks being used but I am hesitant to use those at this time simply because they can be hard to explain to others. Rather I decided to start of more simply with ensemble methods and SVM to gauge their performance before I decide to transition to neural networks.

Methodology

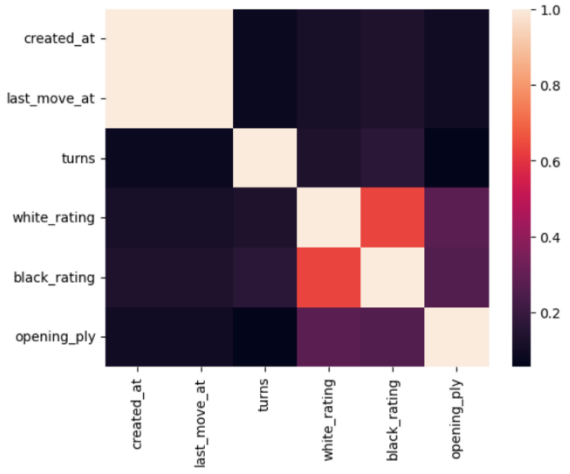
The data required some heavy preprocessing before I was able to work with it. The first thing that needed to be fixed was dropping the unique identifier columns. Features like “id”, “black_id”, “white_id”, “opening_name”, and “increment_code” were all dropped since they would provide no valuable information to the models. Additionally, since “draw” and “outoftime” observations would not be used to classify these were dropped as well. Duplicate games were also dropped as well.

Now starting with a clean dataset, I was able to begin some feature engineering and extraction on features that were not ideal for my models in their current state; “Opening_eco” and “moves” were two such features. “Opening_eco” consisted of codes that

uniquely identify the opening that was used in the game. For example, code D10 would correspond to the Slav Defense: Exchange Variation. Luckily these codes are sorted by a letter identifier at the beginning. An “A” indicates a flank opening, a “B” indicates a semi open game, a “C” indicates an open game/ French defense, a “D” indicates a closed/semi-closed game, and “E” indicates an Indian defense. These identifiers were used to bin this feature into 5 opening types as described.

The “moves” feature consisted of strings of a sequence of chess moves in chess notation. For example, “d4 d5 c4 c6 cxd5 e6 dxe6 fxe6…” and so on. This is a particularly challenging feature to use. Instead of getting rid of it though I extracted the number of times each piece moved from these moves. That way I could keep some of the information from this feature. Though admittedly the pattern that the moves were made in is lost. In chess notation a move is denoted by its piece followed by coordinates on the board. The letter “N” indicates a knight, “K” indicates the king, “B” indicates a bishop, “R” indicates a rook, and “Q” indicates a queen. Pawns were not included because typically there is no letter identifying them as the piece moved. The moves were counted based on these identifiers and new features that contained the number of moves for each piece were generated.

The features “last_move_at” and “created_at”, which denote the start and end time of the game, are colinear as indicated by the correlation plot of continuous variable to the right. Interestingly the times for these features are formatted in Unix time, but in milliseconds rather than seconds. As such these values are rather large. Since they are collinear, I opted to combine them into one feature called “game_time” which is the total amount of elapsed time during the game. This not only removed the multicollinearity of the model but also yields a slightly more parsimonious model.



In addition to the feature extraction and engineering performed dummy variables were created for those categorical variables in the data such as the winner and the opening eco. Additionally, the values of the features were normalized using a min-max scaler because of the wide range of values. Particularly due to the differences between “opening_ply”, “game_time”, and both player ratings.

As mentioned, there is a slight class imbalance in the target. Resignation accounts for most of the observations at 56% and mates account for 36% of the observations. Since the imbalance is not too large and I have enough observations to afford it I just decided to under sample the majority class down to 32%. This class rebalance method is used only for the SVM and gradient boosting models. Random Forest has a built-in parameter *class_weights*

which when set to “balanced” will assign more weight to the minority class in order to correct a class imbalance.

In terms of feature selection two methods were used. The first is wrapper select via model. The second was univariate selection via mutual info classification. The following were the selected features for both:

SELECTION TYPE	SELECTED FEATURES
RANDOM FOREST (WRAPPER)	['turns', 'white_rating', 'black_rating', 'queen_moves', 'opening_ply', 'knight_moves', 'king_moves_moves', 'bishop_moves', 'rook_moves', 'game_time']
SVM (WRAPPER)	['turns', 'white_rating', 'queen_moves', 'opening_eco_Indian Defense']
GRADIENT BOOSTING (WRAPPER)	['turns', 'white_rating', 'black_rating', 'queen_moves', 'opening_ply', 'knight_moves', 'king_moves_moves', 'bishop_moves', 'rook_moves', 'game_time']
UNIVARIATE	['rated', 'turns', 'white_rating', 'black_rating', 'queen_moves', 'knight_moves', 'king_moves_moves', 'bishop_moves', 'rook_moves', 'game_time']

Note univariate selection utilized the k best features where k=9.

Optimal model parameters were obtained by running grid search on all the models. It was run using standard 5-fold cross validation. Below are the parameter grids for each model along with the selected parameters:

MODEL	PARAMETER GRID	SELECTED PARAMETERS
RANDOM FOREST	n_estimators: [10, 100, 150, 200, 500], min_samples_split: [1, 2, 3, 4, 5], criterion: ['gini', 'entropy', 'log_loss']	n_estimators: 200 min_samples_split: 5 criterion: log_loss
SVM	kernel: ["linear", "rbf", "sigmoid"], gamma: ["scale", "auto"], C: [0.01, 0.1, 1, 5]}	kernel: linear gamma: scale C: 5.0
GRADIENT BOOSTING	n_estimators: [100, 150, 200, 300], min_samples_split: [2, 3, 4, 5], loss: ['deviance', 'exponential', 'log_loss'],	n_estimators: 100 min_samples_split: 3 loss: exponential criterion: squared_error learning_rate: 0.2

	criterion: ['friedman_mse', 'squared_error'], learning_rate: [0.1, 0.2, 0.3]}

Additionally, 5-fold cross validation was used to run each of the models on top of the aforementioned preprocessing, parameter optimization, and class rebalancing. Each model was run 3 times. Once with a fill fit (all features), once with wrapper selection, and once with univariate selection.

Results

AUC and accuracy were used as performance metrics for all the models built. AUC measures how good the model is at distinguishing between classes. Accuracy provides a look at the correct predictions divided by the total number of predictions. Below is a compilation of all the models and their performance metrics.

FULL FIT	Accuracy	AUC
RF	0.70	0.74
SVM	0.66	0.69
Grad Boost	0.69	0.73

WRAPPER	Accuracy	AUC
RF	0.69	0.73
SVM	0.64	0.67
Grad Boost	0.68	0.72

UNIVARIATE	Accuracy	AUC
RF	0.69	0.73
SVM	0.65	0.68
Grad Boost	0.68	0.72

The SVM underperforms in all regards. It averages to be 6 points lower in both AUC and accuracy compared to the random forest and gradient boosting models. It also takes exponentially longer to run than the other two models because a linear kernel was used. Gradient boosting and random forest both perform similarly across all runs of the models, but random forest outperforms by just a little bit. Additionally, worth noting is that not a lot of performance is lost with either of the feature selection methods, indicating that the features taken out of the model did not contribute much to its predictive power. Looking at the scores themselves accuracy averages around 0.69 and AUC 0.72. This indicates that the models are predicting the outcome correctly 69% of the time and that the models generally have a 72% chance of correctly classifying a randomly chosen instance. These scores can be considered good, but there is room for improvement in the model. Something else worth mentioning is that the features selected with the models were not all consistent with each other. The wrapper method on the SVM model continuously selected less features than the other wrappers and univariate selection. Additionally, it also selected the Indian Defense dummy variable when the others did not select any opening eco dummy variables. Though it seems this is not the reason for the SVM models poor performance as when it was run with the univariate selection (which was more in line with the other wrappers) it still performed relatively poorly.

Discussion

After running the models, though they perform well, there is still some room for improvement; Particularly because of some of the limitations of this project. Firstly, the 'moves' feature was a huge limitation. It is difficult to utilize unique strings as features unless running a sophisticated deep learning or transformer model, something way outside the scope of this class. Rather than losing that feature outright, the decision to extract the number of moves for each piece caused the patterns contained in that sequence of moves to be lost. I consider this to be the reason that the model did not perform as well as it could have because all the move features did end up being selected into the final models. If the pattern could have been kept the models would have performed better. Another limitation was that the original 4 classes in the target were unbalanced. The original plan was to utilize all four classes but with a classification scheme approach. Unfortunately, a small number of games end in a draw or out of time win. As consequence those two classes were dropped since it would have been difficult to rebalance without using smote heavily or losing a lot of the data though under sampling. Had the classes been a little bit more balance or there been enough observations to afford under sampling the project could have utilized the models for a more comprehensive approach.

Conclusions

So, what can be concluded from these models? The models perform well so the features selected by the feature selection contribute a lot to the classification of the target. By looking at the features that were selected it is possible to distinguish what factors may be important in determining the outcome of a game. For example, opening ply and all the piece moves were selected in the models. Opening ply is the number of moves that were made in the opening of the game. This could indicate that the level of aggressiveness in the opening is important in the opening of the game and could influence the outcome. Another feature selected was time. If someone is running against the clock, they may be more likely to resign, especially as the game runs longer. Player ratings were also selected, if two players are more evenly matched it more likely for the game to end in a resignation. Or it also might simply be the case that higher rated players often know when they have been beaten and rather than lose to checkmate resign gracefully.

Another thing to think about is why none of the opening eco types were selected in the features selection? Do openings matter? Since none of the openings were selected to be in the model it might mean that the opening you play holds less importance than you think it does. There is likely no "meta" opening that will win you the game or give you an outright advantage. What matters is how you develop your pieces after the opening and how you adapt to your opponents' moves.

In conclusion, the importance of opening ply suggests that a strong start can set the tone for the rest of the game. Similarly, the significance of time indicates that the ability to maintain composure under the clock can be as decisive as most strategies. Additionally, it's not about finding a one-size-fits-all opening but about understanding the complex interplay of aggression, timing, and skill that guides the game's outcome. These insights could be utilized in several practical ways. For one, it can be used in AI (Artificial Intelligence) chess programs. Incorporating these features could lead to more human-like and adaptable algorithms that react based on time, and the aggressiveness of the opposing player. Another is chess training, which based on these findings could emphasize the importance of mastering openings, developing time management skills, and understanding the psychological aspects of the game. Finally, players can use these insights for self-analysis, identifying their strengths and weaknesses in these areas to improve their overall gameplay.

Works Cited

[3] H. Apolo and R. Pulido, "Predicting the Outcome of a Chess Game by Statistical and Machine Learning techniques," 2016. Available: https://upcommons.upc.edu/bitstream/handle/2117/106389/119749.pdf?sequence=1&isAllowed=y

[1] P. Avva and J. Hanke, “GUESS THE ELO 1 Guess the Elo -Predicting Chess Player Rating,” 2022. Available: https://pranavavva.com/docs/guess-the-elo.pdf

[2] M. Hoque, “Classification of Chess Games: An Exploration of Classifiers for Anomaly Detection in Chess - ProQuest,” *www.proquest.com*, May 2021. https://www.proquest.com/docview/2539890690?pq-origsite=gscholar&fromopenview=true&sourcetype=Dissertations%20&%20Theses (accessed Feb. 12, 2024).

[4] G. Wölflein and O. Arandjelović, “Determining Chess Game State from an Image,” *Journal of Imaging*, vol. 7, no. 6, p. 94, Jun. 2021, doi: https://doi.org/10.3390/jimaging7060094.