How Accurately Can a Machine Learning Model Predict Chess Game Results?

Jason Covey, Brown University Data 1030 final project

https://github.com/jasonchess/JasonData1030Project/tree/main

Introduction

Predicting the results of competitions has long been both a challenge and a hobby for sports and game enthusiasts alike. Supporters of their team want their city of favorite player to succeed, and the betting industry reported $13.7 billion[1] in revenue in 2024, as millions try their hand at predicting scores, results, winners, performances from individual players and more. As a former competitive chess player, I'm curious about how well a machine learning algorithm can make predictions for chess games between two players. This was the motivation behind this project, which uses a data set[2] from kaggle, and trains multiple algorithms to try and predict the outcomes of these given chess games. This data set contains ~20,000 chess games played online by players of all skill levels on the popular chess website LiChess.com, and tracking things such as both players chess rating, the opening played, the result, and more. This dataset is non-iid due to having repeated players in multiple of the games.

EDA

The target variable for this project is the *winner* column of the data frame. This is either the word 'white', 'black' or 'draw', indicating either the winner of the game or that the game ended in a tie. As can be seen in fig. 1, ~5% of the games in this data set ended in draws, while !46% were wins for black and ~49% were wins for white. The person with the white pieces in chess starts with an advantage as they play first, and because this data set is from a variety of skill levels, the amount of draws is expected to be fairly low. Higher rated players' experience leads them to have more balanced games that end more often in draws than the average chess player, so if we were to look at games from only top level chess players we would certainly not expect this distribution of results.

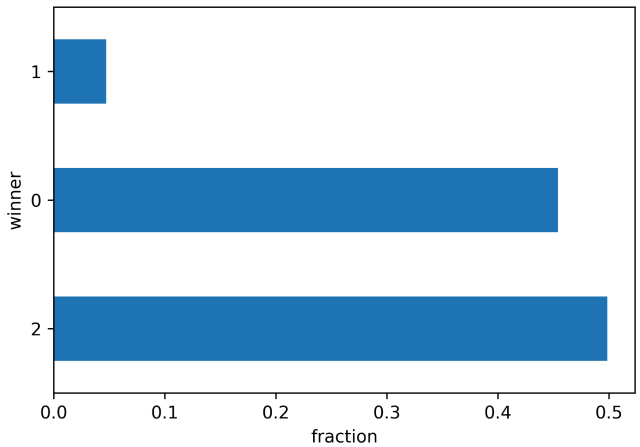

Fig. 1: Histogram of winner column

In addition, two important columns for this ML task will be the ratings of both players, as well as the rating difference between the two players. These can be seen in figures 2a and 2b, while figure 3 shows a distribution of the winners based on the difference in rating between the two players. It is important to note the spike in players rated exactly 1500 for both white and black, which is caused by 1500 being the default rating for players on LiChess. For players who have a rating of exactly 1500 and have no previous games in the dataset, we cannot have confidence that 1500 is their exact rating.

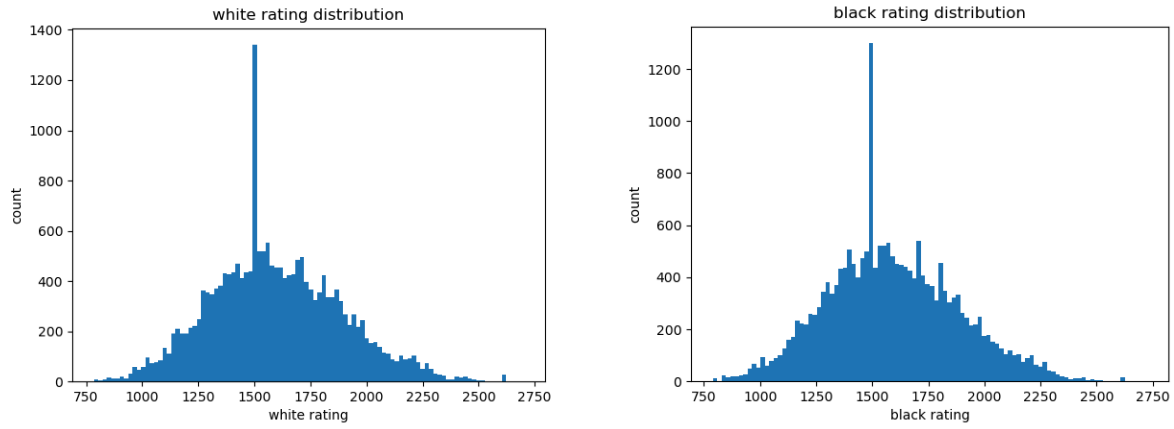

Fig. 2a shows the distribution of ratings of white, 2b shows the ratings of black

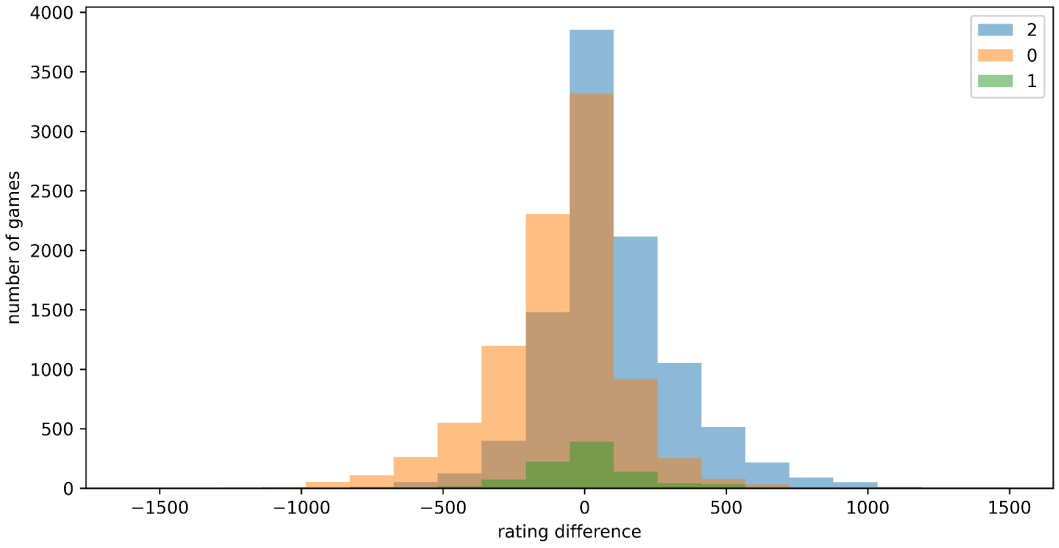

Fig. 3 shows the amount of results the games had as a function of the difference in ratings between the two players. A negative number means that black is higher rated, and a positive number means white is higher rated

As we can see, as the difference in rating will likely be a key factor in the algorithms, as the player with the higher rating is more likely to be the better player and so more likely to win.

During the EDA phase, two new features were also created for each player using the group structure of this data set. A ‘heatness tracker’ saved the result of the player’s last game in the dataset to measure their confidence with the color of pieces they were playing, and an overall tracker saved what their win and loss percentage was with that color to see how good they are with their colored pieces.

Methods

The 3 encoders used for this project are the categorical encoder, the ordinal encoder and the standard scaler. These 3 can cover all the data types found in this data set, and table 1 shows which encoder was used for each feature.

Preprocessor	Features
One Hot Encoder (categorical)	'opening_eco', 'opening_ply', 'eco_letter'
Ordinal Encoder (ordinal)	'rated', 'prev_res_w', 'prev_res_b'
Standard Scaler (continuous)	'turns','white_rating', 'black_rating', 'time_mins', 'time_inc', 'rating_diff', 'win_frac_w', 'loss_frac_w', 'win_frac_b', 'loss_frac_b'

Table 1 Features and the appropriate encoder for the category

The ordinal encoder makes sense for rated, as players tend to take rated games more seriously than unrated, so our order is [False, True]. For the previous result category, the ordinal encoder allows us to impute an empty value as its own category for players’ first game in the dataset, and then a loss is worse than a draw is worse than a win for each given player. Those features, as well as the win and loss fraction for white and black all contain missing values that can either be imputed, or put into models which can handle missing values such as random forests or XGBoost. For splitting, a stratified k-fold with 20% of the points in the test set is a logical choice for this data set to ensure the minority class of draws is represented in each split, and doing the k-fold allows more models to be built for a given amount of test data. A reference image for what this splitting pattern looks like is shown in figure 4.

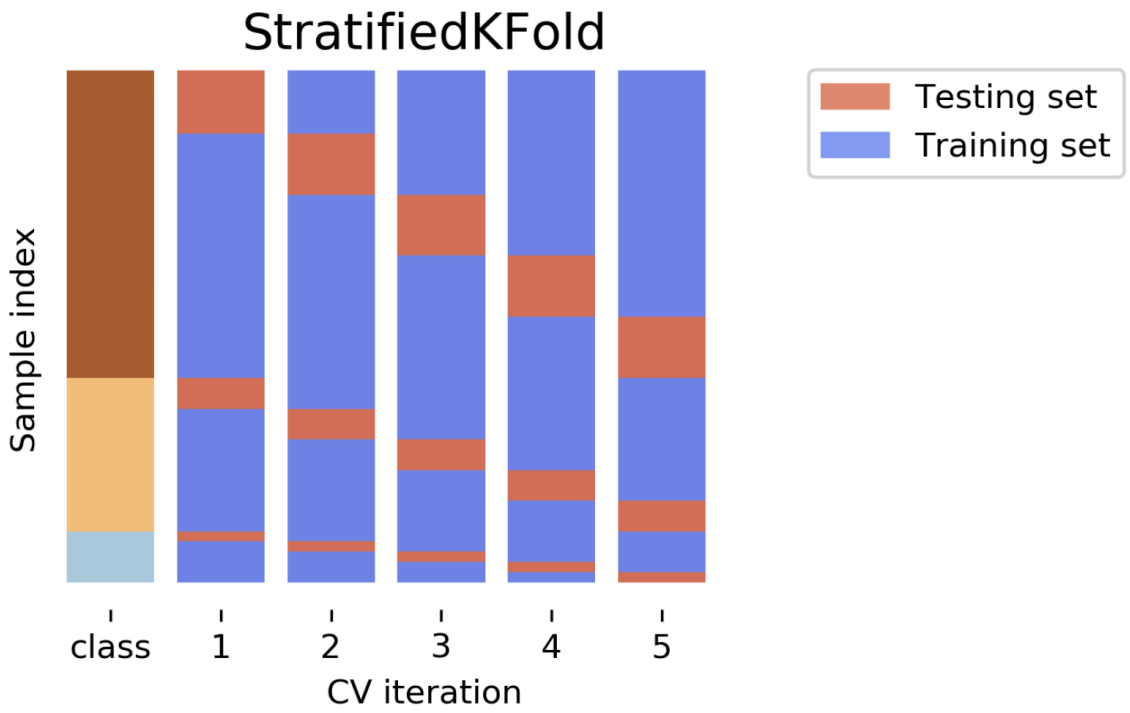

Fig. 4[3] A visual representation of how a stratified kfold splits a set into train and test data

The four models used in this project are a logistic regression, a random forest classifier, a k-nearest neighbors classifier, and XGBoost using both SciKitLearn's package and the XGBoost package. These models are all capable of performing multiclass classification and handling missing values in continuous features either using an imputer (knn's built-in imputer), the reduced-features approach (logistic legression), or dont have issues with missing data (the tree-based methods).

Model	Tuned Parameters
Random Forest	N_estimators: 8, 10 Max_depth: 10, 25, 100
K Neighbors Classifier	N_neighbors: 5, 7, 10 Weights: uniform, distance
XGBoost Classifier	Learning_rate: 0.03 N_estimators: 100 Subsample: 0.66
Logistic Regression	Penalty: l1 C: 0.1, 0.25, 1 Max_iter: 2500

Table 2 Used models and the tuned parameters in final project; other values were tried for many categories but were removed due to convergence warnings or serious runtime delays

The uncertainties of the evaluation metrics due to splitting are seen in the standard deviations for the model's performances on the test set, which can be seen graphically in the next section, but qualitatively k-nearest neighbors had the widest variance, followed by logistic regression, scikitlearn's XGBoost, and random forest classification. For all but knn, the variance in test score accuracy is small compared to the overall accuracy of the models.

Results

For this problem, accuracy score is the most obvious choice of metric, as the goal of the model is to predict who won the game, and in future steps would be to predict who would win a game. Accuracy tells us exactly how frequently the model predicts what the result of the game will be, which is the most important metric for the given question.

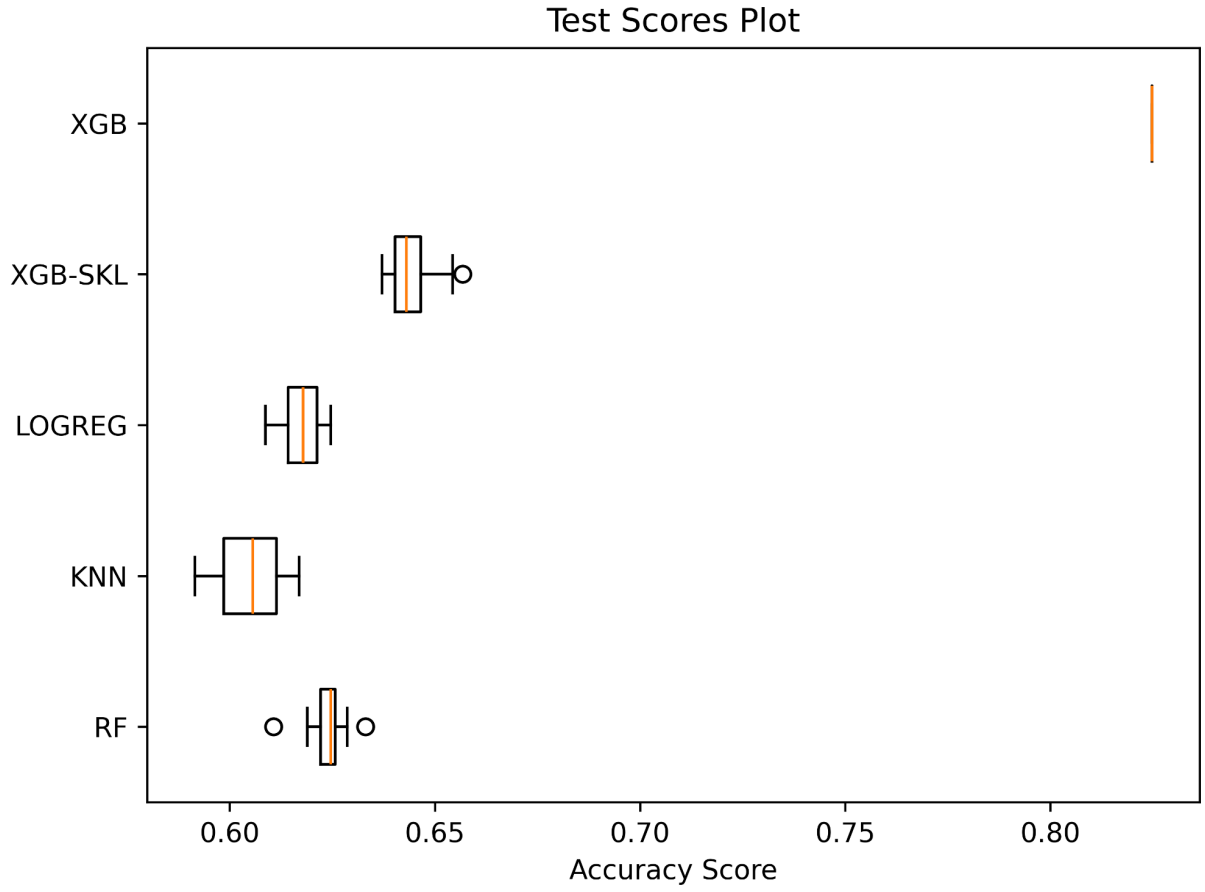

Fig 5: Boxplot of accuracy scores of various models. The mean of the accuracy scores is the orange bar, while the box represents the 25-75% distribution

With 49% of points in the majority class, the baseline accuracy for this project is 49%, and all models performed above this by at least 11%. XGBoost's package performed by far the best of

the tested models with over 82% accuracy, and a weighted f1 score of 81% compared to a baseline of 32%. A confusion matrix of XGBoost's predictions can be seen in figure 6.

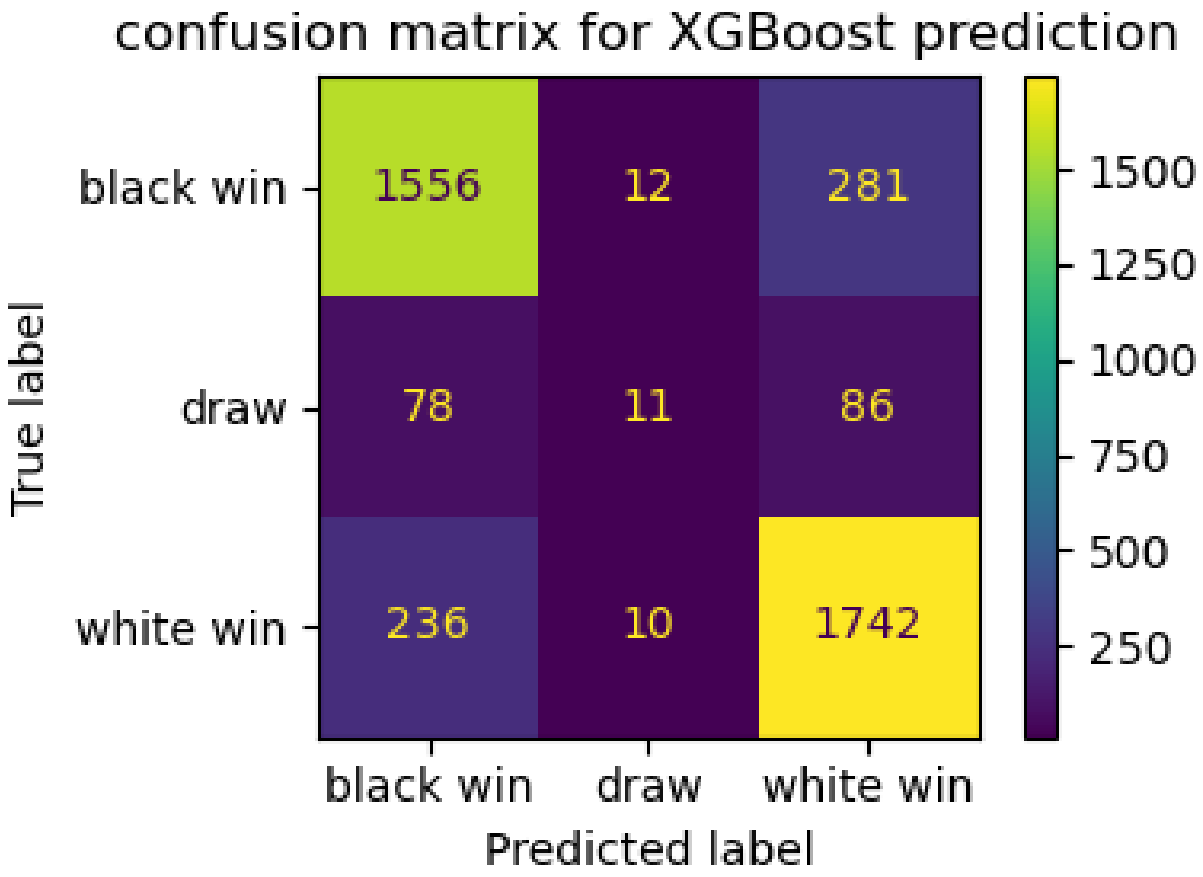

Fig. 6 Confusion matrix for XGBoost

XGBoost is a powerful tooll which performed very well on the data, though perhaps with more processing power and therefore the ability to run more samples and tune more hyperparameters, the other models could've performed better.

Additionally, global and local feature importance metrics were evaluated. For global feature importance, XGBoost's gain, weight and cover metrics were applied to find the feature perturbation importance, and for local feature importance the SHAP method was used.

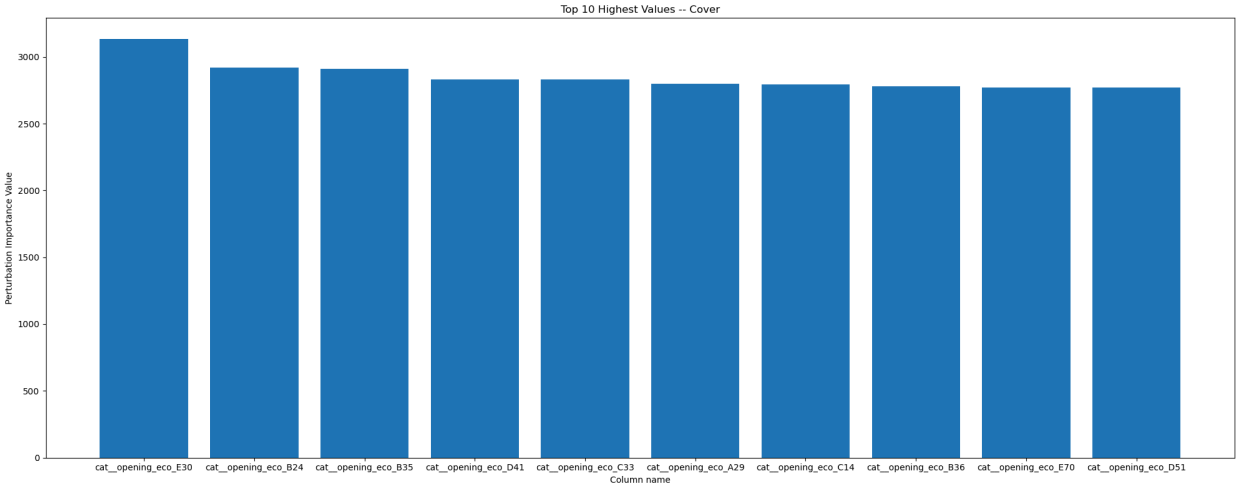

Fig. 7 XGBoost cover perturbation feature importance chart

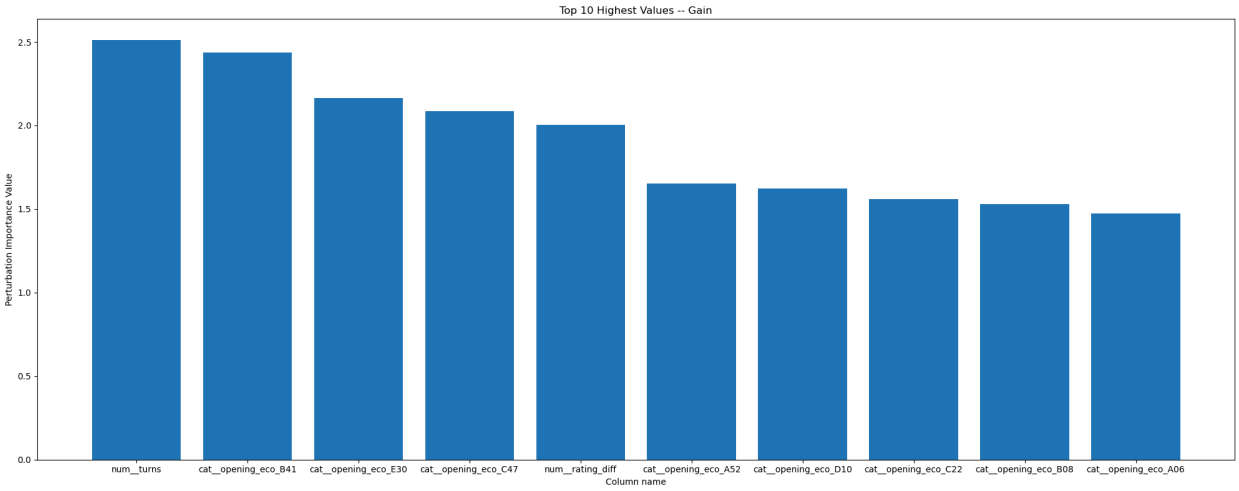

Fig. 8 XGBoost gain perturbation feature importance chart

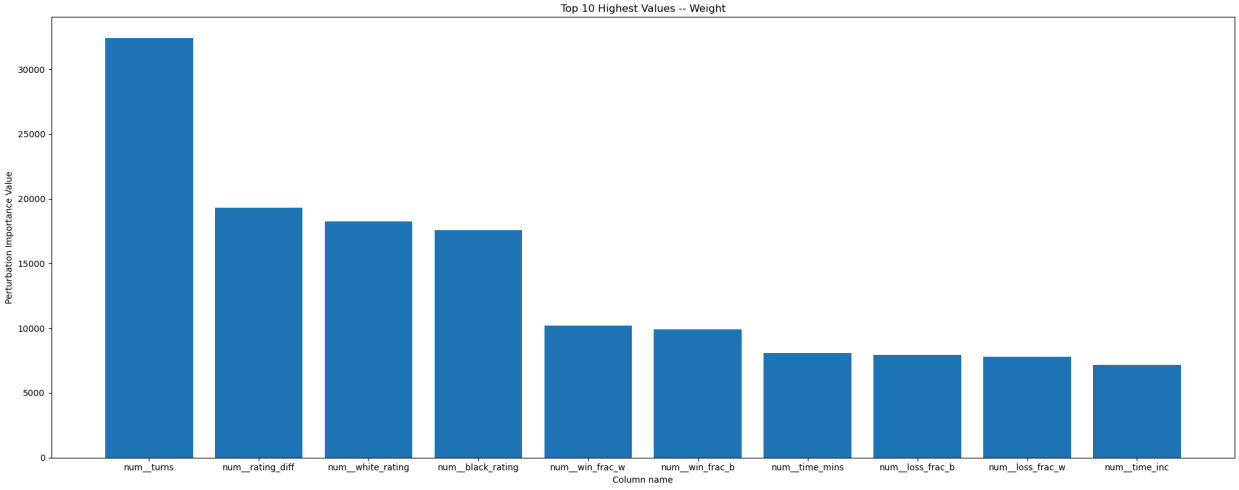

Fig. 9 XGBoost weight perturbation feature importance chart

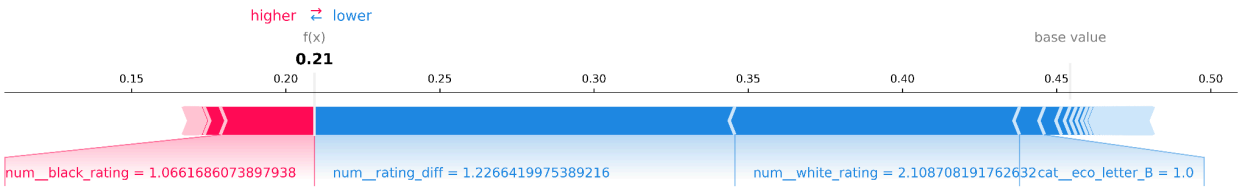

Fig. 10 SHAP local feature importance force plot for data point 100

A very interesting feature about the calculated global feature importances is that two of XGBoost's metrics calculated high perturbation feature importance values for many of the opening categorical data. This is surprising because one might assume that the rating of the players and the difference in rating between them would be one of if not the most important features, as it is supposed to measure how good a player is at chess. This could be a result of the way the method calculates the perturbation feature importance, or a product of the dataset. Since there are many different openings, some of them are probably only played a handful of times in this dataset, and so perhaps when they are played there's a large correlation between that and the winner of the game. On the other hand, the weight method of calculating the feature importance favored thenumber of turns played in the game, as well as the ratings and rating differences of the players, and how they'd done in previous results. This makes more sense logically as someone's previous results and estimated skill at the game should be predictors of how they do in any given game, but it's interesting to see that different metrics produce very different findings.

Outlook

There are a few things that, given more time and resources, could be interesting to improve or change the model. Firstly, more processing power would allow more hyperparameters tested to potentially scan for better performance particularly in the scikitlearn packages. Predicting results of a game like chess is inherently hard, made even more difficult that some games or sports by the fact that both teams can win and there can be draws, but I'd be curious to see how well a model can do at its peak. The hyperparameters mentioned above were the best performing of those tested, but there still may be more combinations or values that produce better results.

Another way to slightly change this project would be to only use higher level chess games. As mentioned in the EDA portion, stronger players draw more frequently than lower rated players. There are many different questions one could ask in the realm of comparing results between different data sets, such as: Do in person games vs online games have an effect on the predictions? What about longer vs shorter time controls? It would be interesting to see how the models react to these different variables.

Lastly, the models that were used in this project technically predict the winner of the game 'after' it's played, as the model uses information like the number of turns or the opening played. If we

removed both these features, we could ask the model to make predictions before the game, basically asking it out of these two players with the following data playing white and black, who do you predict will win, or what are the odds of each results? Or perhaps we could put back in the opening played, as the first few moves of the game often happen quickly, and ask the model to make predictions early into the game about the end result. These are all very interesting questions that with more time could be really valuable to explore.

References

[1]www.espn.com/espn/betting/story/_/id/43922129/us-sports-betting-industry-posts-record-137b-revenue-24
[2]www.kaggle.com/datasets/datasnaek/chess
[3]amueller.github.io/aml/04-model-evaluation/1-data-splitting-strategies.html