6-7-2024

Using Machine Learning Algorithms to Predict Outcomes of Chess Games Using Player Data

Sofia DeCredico
sld4732@rit.edu

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Masters of Science in Applied and Computational Mathematics

Using Machine Learning Algorithms to Predict Outcomes of Chess Games Using Player Data

By:

Sofia DeCredico

Rochester Institute of Technology

College of Science

Mathematics

Rochester, NY

June 7, 2024

Thesis Committee Members:

1. Dr. Nathan Cahill
2. Dr. Matthew Coppenbarger
3. Dr. Ernest Fokoue

Abstract

Chess is a two-player game, popular with a wide variety of people, ranging from people who play casually every once in a while with their family to professional players who make their living through playing and teaching. There are three outcomes that a chess game could have: player 1 (White) wins, player 2 (Black) wins, or both players draw. Using an online database called "The Week In Chess" which contains information about 2.5 million chess games, a variety of machine learning methods are tested to predict the outcomes of chess games. The features investigated as inputs for classification are: White's Win Rate, White's Draw Rate, Black's Win Rate, Black's' Draw Rate, Opening's Win Rate and Opening's Draw Rate. Then, the results of the classifiers using the full set of features and various subsets of features are compared in order to try and determine the best possible accuracy in classifying the game outcome.

Introduction

Chess is a well known game played by people of varying skill level, from people who know the basic rules of how the pieces move but don't have much of a strategy, to people who make their living playing and coaching and who spend many hours most days playing and studying the game in an effort to improve and play chess at the professional level. Every time a game is played, two people sit down at the chess board across the table from each other. Player one ("White") plays the white pieces and player two ("Black") plays the black pieces. The first phase of a chess game is called the "opening", which is followed by the middle game and then the end game. Where the opening ends and the middle game starts is a blurry line, however openings are set moves that players play to start the chess game and they are known by names. The game will end in one of three results, White can win, Black could win, or the game could end in a tie which is called a Draw.

This research will use different machine learning algorithms to predict, or classify, the outcomes of chess games without using the position evaluation. Features used for classification include: White's Win Rate, White's Draw Rate, Black's Win Rate, Black's' Draw Rate, Opening's Win Rate and Opening's Draw Rate. We will explore the effects of using different subsets of features as inputs on the accuracy.

Currently there are many available chess engines online that people can use to evaluate their chess games. One of the more well-known chess engines is Stockfish [1]. These engines can calculate the "position evaluation" at any particular point in the game. The position evaluation is a number that indicates the degree to which the current configuration of pieces (the "position") is favorable to each player. A positive position evaluation favors White, a negative position evaluation favors Black, and a position evaluation of zero is "even" (favors neither). The magnitude of the position evaluation indicates how strongly the current position favors the player in question. In general, if x is the position evaluation, then:

$x \leq |0.5|$ - the position is even,

$|0.5| < x \leq |1|$ - the position slightly favors that color,

$|1| < x \leq |1.5|$ - the position significantly favors for that color, and

$|1.5| < x$ - the position is winning for that color.

While these chess engines evaluate the actual position in the game, they don't actually predict the outcome of the game. Rather, they indicate how likely it is that a particular outcome will occur if both players play the best move from the current position to the end of the game. Even though the position evaluation itself does not predict the outcome of the game, it can be a useful tool in trying to predict the outcomes of games that are already in play.

There are some prior results on outcome prediction using position evaluations. In [2] different board positions and sequences of moves are used to try to predict the outcomes of chess games. In this paper, a "move" is considered as two "plys" or half-moves, one by White and the second by Black. Every game in the dataset is analysed ply by ply. For each ply, the move number is noted, along with the actual move and the position evaluation after that ply. Additionally, win scores are calculated, which is defined by the number of wins and draws over the total number of games where that move sequence was played. Multi-variate Linear Regression and Naive Bayes are compared to predict game outcomes. It was found that the win score had the highest significance in predicting the outcomes of the games and the move numbers and opening classification performed moderately well.

Another paper [3] uses deep learning to predict outcomes of chess games from different positions. They treat the moves made in a chess game as sequential data by converting them into vectors in various ways, and then they use neural networks to predict the game outcomes using the vectors that encode the sequences of moves. To predict the outcomes of the chess games, they represent each move using five features:

1. Piece - which piece is making the move
2. Column - which column the move is taking place in, a-h
3. Row - which row the move is taking place in, 1-8
4. Capture - whether or not the move captured an opponents piece
5. Check - whether or not the move put the opponent's king in check

For the algorithm to work, all the data vectors had to be the same length. So a length was chosen and any games that had more moves then the picked length had the moves after that number deleted and any games that came in under that move count had the extra moves needed filled in with zeros. Classifiers were trained using three different game lengths: 20, 50, and 80 moves. It was for two different models: Content Model and Context Match Fusion Model. For each model two different data storage methods were run, the bitmap and algebraic representation. For game lengths of 80 moves the Context Model gave accuracy results of 59.62% and 57.73% and the Context Match Fusion Model gave accuracy results of 68.9% and 67.03%.

Everything done in [2] and [3] was focusing on predicting game outcomes using position evaluations and/or sequences of moves, but without any information about the players playing the game. In [4], Diogo Ferreira looks at the players themselves but instead of predicting the outcomes of games they are using the outcome of games to predict players ratings. Their goal is to use the strength of player's opponents and the scores against those opponents to predict the strength of players. In order to do this, they expand upon the Bradley-Terry model which is an expression for $P(X > Y)$ which is the probability that player X wins when they play against player Y. The Bradley-Terry model says:

$$P(X > Y) = \frac{1}{1 + \frac{\gamma_Y}{\gamma_X}}, \tag{1}$$

where γ_X and γ_Y are the strengths of each player, these values must be positive. It can be noted here that when there are two players of equal strength we have $P(X > Y) = P(Y > X) = 0.5$, and in the case where player Y's strength is much higher than player X's, we have $P(X > Y) \approx 0$ and $P(Y > X) \approx 1$.

The previous methods for predicting the outcomes of chess games have been focused on the actual game being played on the board, using position evaluations and/or move sequences from the game in question. This research will focus on a different problem of predicting the outcomes of chess games using only information that is available before the start of the game. This allows us to focus mostly on the players themselves, so that we can attempt to predict game outcomes using the players' ratings and previous performances.

Methods

This research will explore the capabilities of four different machine learning algorithms to predict chess game outcomes based on information about the players prior to the start of the game: Naive Bayes, Decision Trees, AdaBoost, and Random Forests.

Preliminaries

The goal of each machine learning algorithm is to use a set of n features $X_1, X_2, ..., X_n$ to predict an output variable Y that takes on three values: $Y = -1$ corresponds to Black wins, $Y = 0$ to draw, and $Y = 1$ to White wins. Each machine learning algorithm can be thought of as a function, f, that maps an n-dimensional vector of sample features, $x = [x_1, x_2, ..., x_n]^T$ to an output sample y, and the function f may depend on a m-dimensional parameter vector $\theta = [\theta_1, \theta_2, ..., \theta_m]$.

Suppose we are given a set of data of the form $\{(x^i, y^i), i = 1, ..., k\}$ that contains features and outcomes from a set of k chess games. Training the machine learning algorithm involves identifying the best parameter vector θ according to a measure of the difference between $(f(x^i), \theta)$ and y^i across a subset of the data known as the training set.

The entire dataset is split into two subsets, a training set and a testing set. The training set is what is used to train the machine learning model as it learns what features result in which outcomes. Once the algorithm is trained it is then tested on the testing set. This is how the accuracy of the model can be seen since the outcomes of the testing data are known, the predicted outcomes are compared to the actual outcomes and accuracies are calculated.

In order to train a machine learning algorithm, the loss function (also referred to as the error function) is used. The loss function is the way that we can quantify the difference between the real outcomes and the predicted outcomes from the machine learning model. When a machine learning model is being trained its best parameters and hyperparameters have to be found and this can be done with the help of the loss function. The way to find the best values for the θ

vector is to find θ which minimizes the loss function over the entire training set.

How well a machine learning algorithm is performing can be quantified by its accuracy, A, which is defined as:

$$A = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \text{ x } 100\%. \tag{2}$$

The accuracy can be performed on both the training and the testing data subsets and a good model will be able to generalize. This means that the model is able to make predictions on data it has not seen before and does not get too specialized to the training set. This is called *over-fitting*, when the accuracy on the training set is very high but the model does poorly on testing sets. If the opposite occurs and the model does much better on the testing sets then the training sets this is called *under-fitting*. The model is a good fit and can generalize well if the accuracies on the testing and training sets are both high.

It should be noted that if the outcomes of chess games were to be guessed by random chance the accuracy would be 32.47% if each outcome was guessed proportionally to how often it appeared in the data since all three outcomes do not occur in one third of games.

Naive Bayes

The Naive Bayes algorithm is a supervised learning method based Bayes theorem, which is:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}, \tag{3}$$

where X and Y are events, $P(X)$ and $P(Y)$ are the independent probabilities of X and Y, $P(X|Y)$ is the probability of X given that Y is true and $P(Y|X)$ is the probability of Y given that X is true.

Naive Bayes is a relatively simple algorithm. It operates under the assumption of independence. So it assumes that the presence of a particular feature showing up in a class is independent of the other features also showing up in that class. To make classifications it calculates the probability of each class given the features and whichever class has the highest probability is chosen to be the class of that data point [5]. This is done by the formula:

$$Y = \arg\max_{Y_j}[P(Y_j) \text{ x } \prod_{i-1}^{n} P(x_i, Y_j)], \tag{4}$$

where arg max is an operation that returns the input, or "argument" that results in the maximum value from the given function, n is the total number of features, i is the index of the specific feature and j is the index of the specific class label.

In this research, a parameter optimization will be performed for a parameter “var_smoothing”. It is possible that some features will appear in the testing set but not the training set. When this is the case, with the current equations, that feature would contribute a zero probability when this is not necessarily the case. The var_smoothing is the amount that is added to each probability so that none of them are zero. This smoothing factor gets added into the calculation for $P(X_n|Y)$ which is within $P(Y|X_n)$. The equation is:

$$P(x_i|Y) = \frac{count(x_i, Y) + \alpha}{count(Y) + \alpha * F}, \tag{5}$$

where $count(x_i, Y)$ is the number of times the feature x_i appears in data points with class Y, $count(Y)$ is the total number of features across all data points with class Y, F is the total number of unique features, and α is var_smoothing.

Decision Trees

The decision tree algorithm is a divide and conquer type of method. The algorithm starts with the entire data set at the root node of the tree before any splits have occurred. From there, every non-leaf node is one feature test, which splits the data [6]. The data is split into different categories based on the criteria of that non-leaf node. This is a recursive process where from here the data set is given to each non-leaf node at the same level, a way to split the data is chosen and that split is used to divide the data into subsets. Each subset is the input of data for the next round of splits [7]. This process creates a tree-like structure where each internal node represents a feature and each branch represents a decision on based on that feature that gets the algorithm closer to a classification [6]. This will continue until one of the stopping criteria is met, whether the max depth has been hit, it has hit the minimum number of samples in a node, or all the data points in a node belong to the same class [7]. An example Decision Tree can be seen in Figure 1.

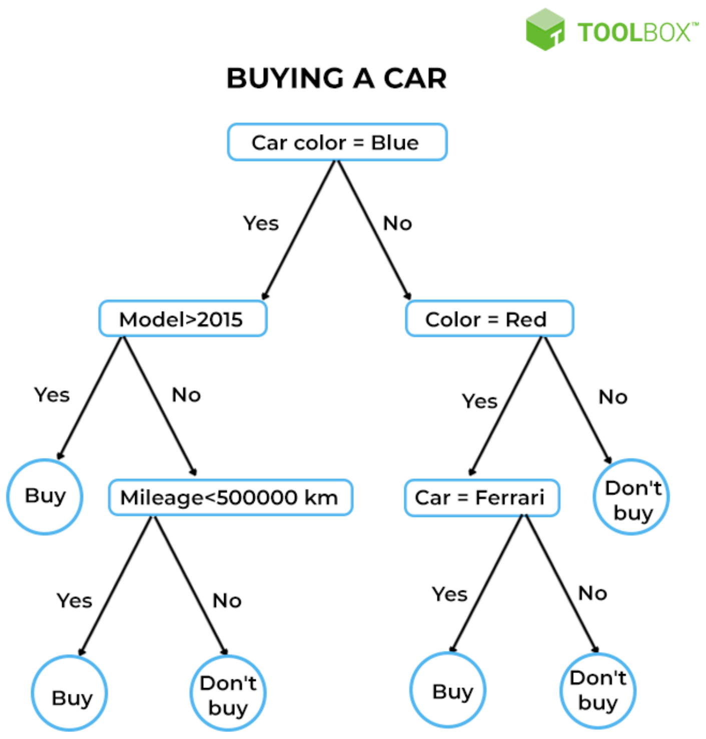


Figure 1: Decision Tree example for buying a car based on its color, make, model, year and mileage [8]

.

To use the decision tree to make predictions on new data points, the data is passed through the decision tree starting at the root node and following the path through the splits according to the feature values. When a leaf node is reached, the data point is assigned the class associated with that leaf node. This process repeats for every point in the data the algorithm is being tested on or making predictions on [7]. In this research a parameter optimization will be performed for the criterion, maximum depth of the tree and minimum number of samples in a node.

The criterion is the method for determining each split along the tree. One commonly used criterion is the information gain criterion, entropy. On a training set Y, the entropy of Y is defined as:

$$E(Y) = -\sum_{j=1}^{c} P_j log_2(P_j), \tag{6}$$

where p_j is the probability of class j occurring and c is the number of classes.

When the training set X, is split into subsets $X_1, ...X_k$ the entropy could be reduced and the amount that it is reduced is the information is:

$$G(X) = E(Y) - \sum_{j=1}^{c} \frac{N_j}{N} E(Y_j), \tag{7}$$

where N is the total number of samples in the original dataset, N_j is the number of samples in the j^{th} subset, and $E(Y_j)$ is the entropy of the variable within the j^{th} subset. The feature that results in the largest information gain is picked to be the splitting factor at that node [6].

One downside to the information gain criterion is that it tends to favor the features that have the largest number of possible values, even if those features are less relevant to the classification. This will cause problems in the case where we have a binary classification with one of the features being unique ID numbers. Taking this feature as a split will result in a large information gain where each data point in the training set will be categorised correctly; however, the classifier is not able to generalize because it will over-fit the training data [6].

Another popular criterion is the *gini index*, which quantifies how likely it is that the data is being misclassified. The lower the gini, the lower the chance of the data sample being classified incorrectly. The gini index is defined as:

$$I(Y) = 1 - \sum_{j=1}^{c} P_j^2, \tag{8}$$

where P_j is the probability of randomly selecting an element of class j from the set Y and c is the number of classes. Whichever feature results in the lowest gini value is picked to be the splitting factor at that node [6].

Ensemble Learning

The decision tree algorithm is an example of a weak learner algorithm. Another category of machine learning algorithms is ensemble learning methods. These methods use a weak learner as the base and fit multiple of those algorithms to the set of data to create an optimized algorithm. Decision trees are often used as the weak learners in ensemble learning methods. Instead of just fitting one decision tree algorithm to the data and relying on the fact that it made all the right decisions, ensemble learning methods look at multiple different decision trees that are fit to the data and combine them into a strong, optimized predictor [9].

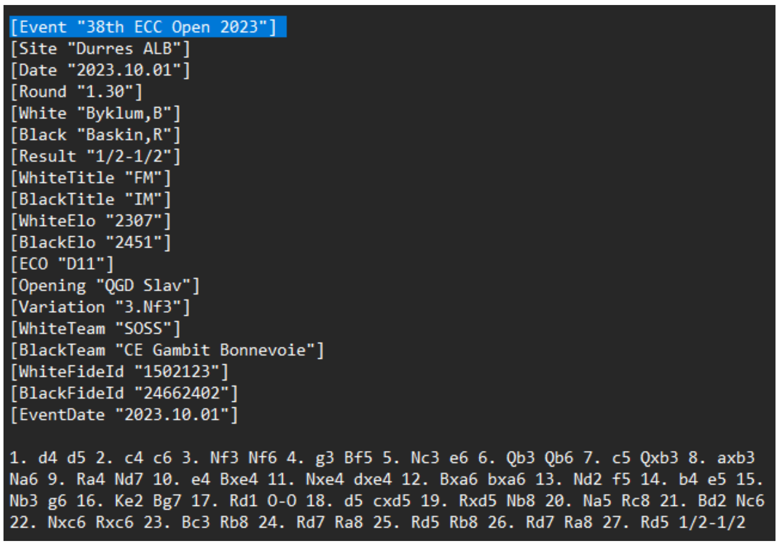
There are two categories of ensemble learning algorithms: sequential learners and parallel learners. Sequential learners are where the weak learners are generated sequentially and each algorithm is learning from the mistakes of the algorithms that were generated before. One example of a sequential learner ensemble learning algorithm is AdaBoost, which stands for Adaptive Boosting [9].

In AdaBoost, decision trees are created sequentially and each substantial decision tree learns from the previous one's mistakes and improves upon it [9]. It will fit a set amount of Decision Trees sequentially and the last one is the most optimized fit as it as learned from all the previous trees and that is the one that will make predictions. For the AdaBoost algorithm, two of the hyperparameters will be varied in the parameter estimation. The first hyperparameter is the number of weak learners that will be trained. Typically, the more weak learners being trained, the better the algorithm will perform; however, more weak learners significantly increase the run time. The other hyperparameter is the learning rate, which changes the impact or weight of each weak learner to the final optimized algorithm.

The other type of ensemble learning algorithm a machine learning algorithm could be is a parallel learner. In these algorithms the weak learners are trained and generated in parallel. An example of this type of algorithm is Random Forest [9]. A Random Forest is a collection of Decision Trees that, to make predictions, each individual tree "votes" for which class it believes the data should be classified as and whichever class with the most amount of votes is how that data point is classified. The options for parameter optimization on this algorithm are the same as on the Decision Tree algorithm [6].

Data

The data being used comes from [10]. This website and database is founded and updated weekly by Mark Crowther, who published the first issue on September 17th, 1994. Issues of TWIC come out every Monday and contain information about all the international chess games played at tournaments over the past week. Every issue is available for download as a PGN or CBV file; both file types are different ways of storing chess data which each have their own strengths and weaknesses. PGN files are readable by people and by python, while CBV files are not readable by people and are less straight forward to get information out of using python, and so I chose to use the PGN version of the data. Figure 2 shows an example of what an average game looks like stored in a PGN file.



```
[Event "38th ECC Open 2023"]
[Site "Durres ALB"]
[Date "2023.10.01"]
[Round "1.30"]
[White "Byklum,B"]
[Black "Baskin,R"]
[Result "1/2-1/2"]
[WhiteTitle "FM"]
[BlackTitle "IM"]
[WhiteElo "2307"]
[BlackElo "2451"]
[ECO "D11"]
[Opening "QGD Slav"]
[Variation "3.Nf3"]
[WhiteTeam "SOSS"]
[BlackTeam "CE Gambit Bonnevoie"]
[WhiteFideId "1502123"]
[BlackFideId "24662402"]
[EventDate "2023.10.01"]

1. d4 d5 2. c4 c6 3. Nf3 Nf6 4. g3 Bf5 5. Nc3 e6 6. Qb3 Qb6 7. c5 Qxb3 8. axb3
Na6 9. Ra4 Nd7 10. e4 Bxe4 11. Nxe4 dxe4 12. Bxa6 bxa6 13. Nd2 f5 14. b4 e5 15.
Nb3 g6 16. Ke2 Bg7 17. Rd1 O-O 18. d5 cxd5 19. Rxd5 Nb8 20. Na5 Rc8 21. Bd2 Nc6
22. Nxc6 Rxc6 23. Bc3 Rb8 24. Rd7 Ra8 25. Rd5 Rb8 26. Rd7 Ra8 27. Rd5 1/2-1/2
```

Figure 2: A typical game stored in a PGN file.

To process the data into a format that can be used by the machine learning algorithms, a python script was used that stepped through each game and saved any relevant information into a python data frame. Then, each data frame was saved as a csv file. According to the TWIC website, there are over 3.5 million games available in the database. After loading the data into a data frame, 2,613,091 games remained. The lost games are due to an error in the format of the PGN file that made the game unreadable by Python.

Preprocessing

Since many of the games are missing necessary information, some fields are loaded into the data frame with NaN's (Not-A-Numbers), which causes errors for the algorithms. Pre-processing was started by dropping all of the rows of data that have a "NaN" in a relevant column. After this step 2,438,831 games remain in the data frame.

After removing the NaN's from the data frame a few games had extremely high ratings (in the 10's of thousands) listed for one of the players. Since the highest rating even achieved by a human is 2882, this likely was just a typo or something strange occurring in the process of loading the data into the data frame. Any rows where either White or Black's rating was recorded as being over 2950 was dropped.

Another issue was how game outcomes were recorded. The full set of unique strings used to indicate game outcomes in the TWIC database is:

['1-0' '1/2-1/2' '0-1' '*' '109' '0-0' '–+' '1/2-1/2'' '0-1 ff'
'0–0' '(+)-(-)' '1-0 ff' '+/-' '1/2 1/2' '-/+' '-' '1-O' '00-1']

We are only interested in the first three options for chess game outcomes, which are White wins which is written as "1-0", Black wins which is written as "0-1", or the players Draw, which is written as "1/2-1/2". These three game outcomes are the options that are recorded for the vast majority of the games in the TWIC database. Any result with an f after it such as "1-0 ff" is usually signaling that one of the players forfeited that game, so the game was never played and we can ignore it. The results that look like "+/-", "-/+" or something similar might be representing that which ever player has the plus is the winning player; however, the documentation about what these options mean is unclear, and so those games will be dropped as well. The remaining options are also unclear from the documentation, and so those games will also be dropped, so that all that remains in the data will be the three possible results.

In total, pre-processing resulted in 163,623 games being dropped from the data, leaving 2,436,468 games remaining.

After the data was pre-processed, I randomly split the whole data set into a training set and a testing set using an 80/20 split. This resulted in 1,949,174 games in the training set and 487,294 games in the testing set.

Features

The features I want to explore for classifying game outcomes are White's Rating, Black's Rating, White's Win Rate, White's Draw Rate, Black's Win Rate, Black's Draw Rate, Opening's Win Rate, and Opening's Draw Rate. Of these, White's Rating and Black's Rating are available for each game directly in the data base. The win and draw rates for White, Black and the Opening must be computed, from all the games either played by that player or where that opening was played. They are calculated separately for the training and testing set as follows:

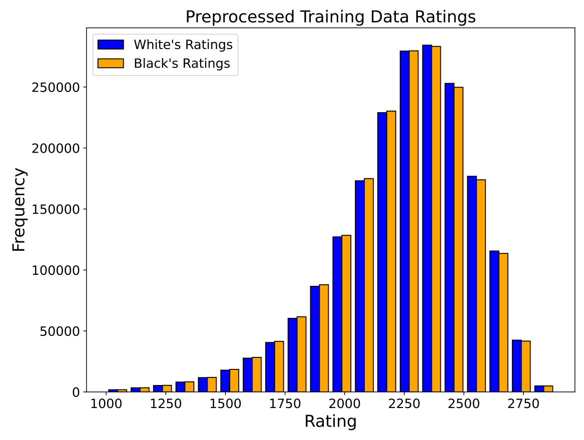
$$\text{White's Win Rate} = \frac{\text{Number of games that player won as white}}{\text{Number of games that player played as white}} \tag{9}$$

$$\text{White's Draw Rate} = \frac{\text{Number of games that player drew as white}}{\text{Number of games that player played as white}} \tag{10}$$

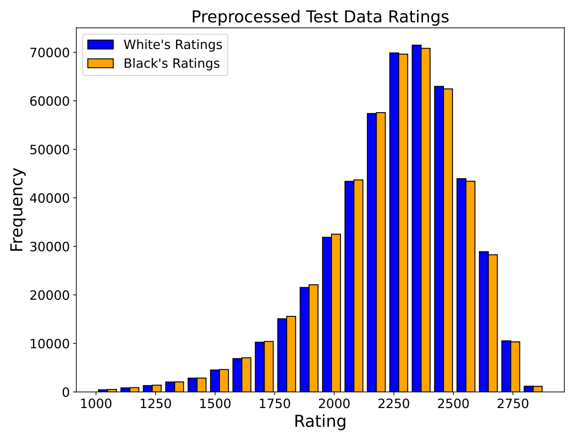
The Black and Opening rates are calculated the same, just replacing White with either Black or Opening.

Since the data frames are organized by game and not by player or opening, I had to write a script that would pull out all the unique values for White, Black and the Opening. For each unique value in each column it would then pull just the games out of the data frame they played in, or that opening was played in and calculate the win and draw rates. From there that rate was added into the data frame anytime that player or opening was listed the game data. This was performed separately on the training and testing set to preserve as much independence as possible.

After calculating the win and draw rates there were a few examples where a win or draw rate came through as a NaN and so, those games were also dropped from the data. This only dropped five rows from the training data and two rows from the testing data so in proportion to the size of the data the 80/20 split between testing and training data was maintained.

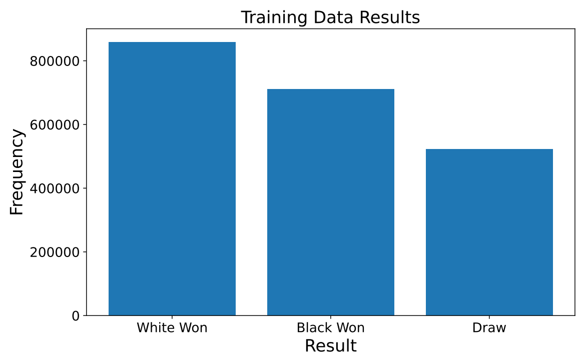


(a) Training Data

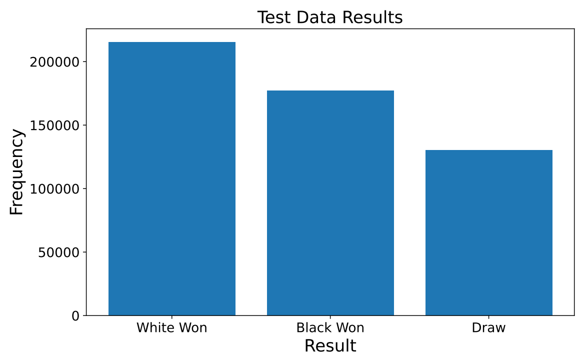


(b) Test Data

Figure 3: Histogram of White and Black's ratings separated into the training and the testing data after the data was pre processed.



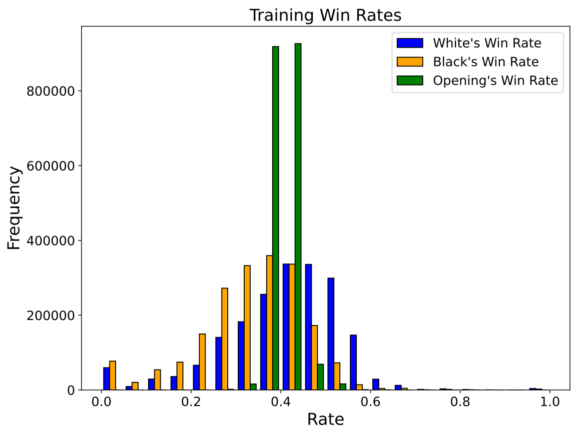
(a) Training Data



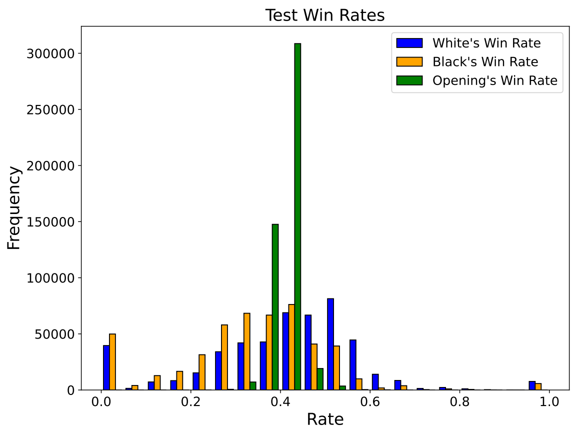
(b) Test Data

Figure 4: Histogram of the Results of all chess games separated into the training and the testing data after the data was pre processed.

Figures 3 and 4 show that the testing and training data has a good split for both the ratings of the players and the results of the games. The number of games in each category is different between the training and testing data which makes sense as the training data is 80% of the data and the testing data is 20%. Although the ratio of player's ratings and games results is not exactly the same between the testing and the training data the shape of the histograms is the same meaning that the split is good and both the testing and training data are good representations of the overall data as a whole.

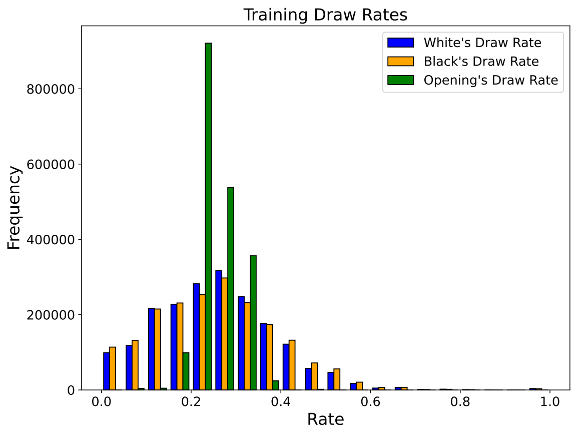


(a) Training Data

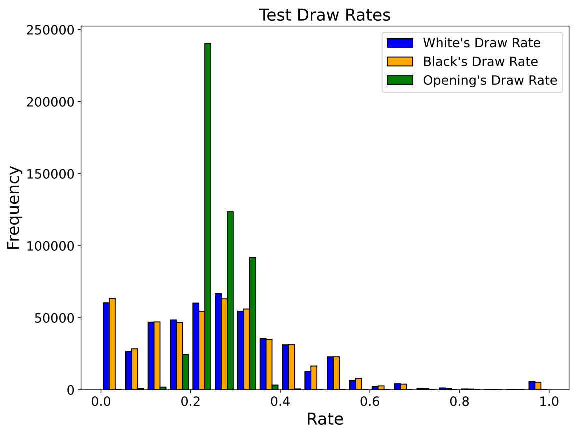


(b) Test Data

Figure 5: Histogram of the win rates of all chess games separated into the training and the testing data after the data was pre processed.



(a) Training Data



(b) Test Data

Figure 6: Histogram of the draw rates of all chess games separated into the training and the testing data after the data was pre processed.

Figures 5 and 6 yield interesting results as they loosely resemble a bell curve however both are skewed right with the Draw rates being more skewed. In the Draw rates both White and Black have very similar frequencies while in the win rates there is more of a difference between White and Black's frequencies. This makes sense as when one person wins the other will lose, however, when one person draws the other person also draws.

Although this data set has good information for the algorithm and a good number of entries, it also comes with a lot of limitations. As can been seen from the histogram in Figure 3, the majority of the players are in the rating range of 2250-2500 with the data being skewed left. This is not representative of the average tournament chess player. The average rating of an active FIDE rated player is around 1750. For context, in the United States rating system I am currently rated at 1492 which puts me in the 84.6 percentile of all USCF rated players. This data has the average player being someone in the 2250-2500 range, while IM Nazi Paikidze who fits into that category with a US chess rating of 2412 is in the 99.9 percentile of all USCF rated players.

This data is not representative of the general competitive chess population as a whole, it only representing the games that have been played at tournaments that have been recorded to TWIC which is majority professional chess players and events. The average player who is competitive but just mostly plays for fun will not have games recorded to this database.

Within competitive chess there are many different time controls. Games range from bullet (1-3 minutes per player) and blitz (3-10 minutes per player) chess which is over in minutes to classical chess where games can go on for five or six hours. These different time controls play into different strengths and weakness in each player and the majority of players are stronger in some time controls than others. This data does not note the time control of the event in the given information and because of that all the different time controls played at the various events have been grouped into one.

Results

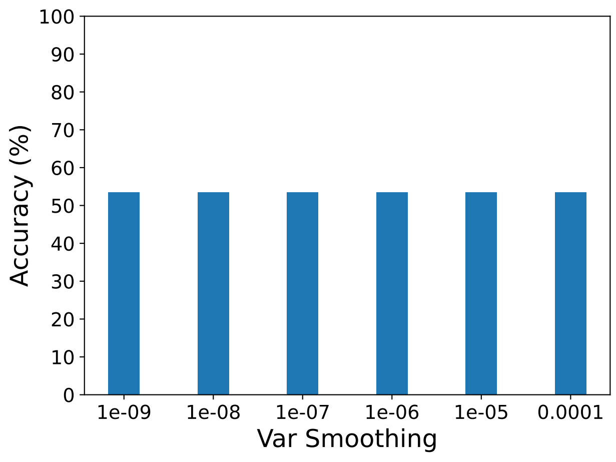
For all of the results three different sets of features will be used as input to predict the game outcome:

1. White's Rating and Black's Rating
2. White's Rating, Black's Rating, White's Win Rate, Black's Win Rate, White's Draw Rate, Black's Draw Rate

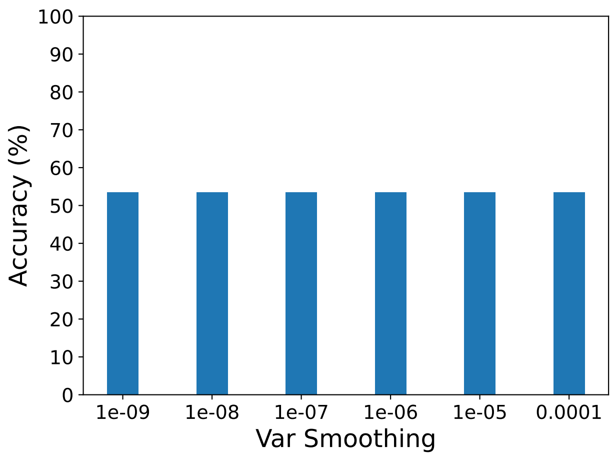
3. White's Rating, Black's Rating, White's Win Rate, Black's Win Rate, Opening's Win Rate, White's Draw Rate, Black's Draw Rate, Opening's Draw Rate

For each machine learning algorithm, graphs of the parameter optimization for each set of features will be shown first. Following that will be a table comparing game outcome prediction accuracy using all the possible pairs of input features. Using the best parameters found for the second feature set, the algorithm will be run using just pairs of features, comparing the accuracies when running the algorithm for each feature with each other feature. The parameters used for this will be the best parameters found for the second feature. There will also be a table showing the results of parameter optimization for a ten fold validation on each algorithm.

Naive Bayes

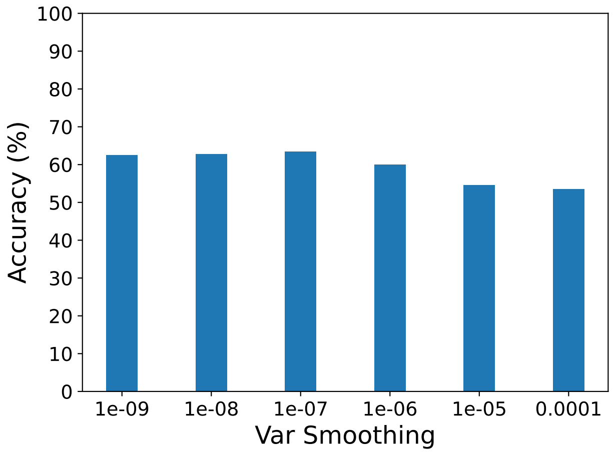


(a) Testing Data

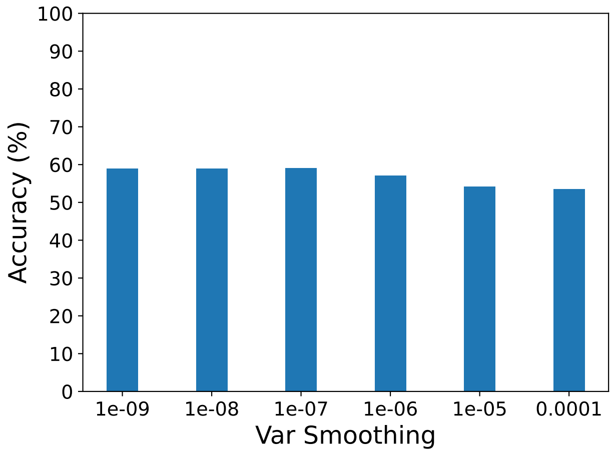


(b) Training Data

Figure 7: Bar graph of parameter optimization for Naive Bayes. a) shows testing on test data and b) shows testing on training data. The features used are White's Rating and Black's Rating. The highest accuracy for testing data is 53.54% which came from smoothing values of 10^{-9}, 10^{-8}, 10^{-7} and 10^{-6}. The highest accuracy for training data is 53.44% which came from the same smoothing values.

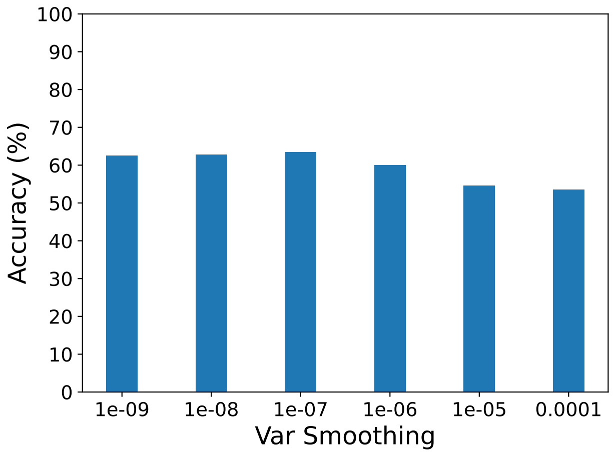


(a) Testing Data

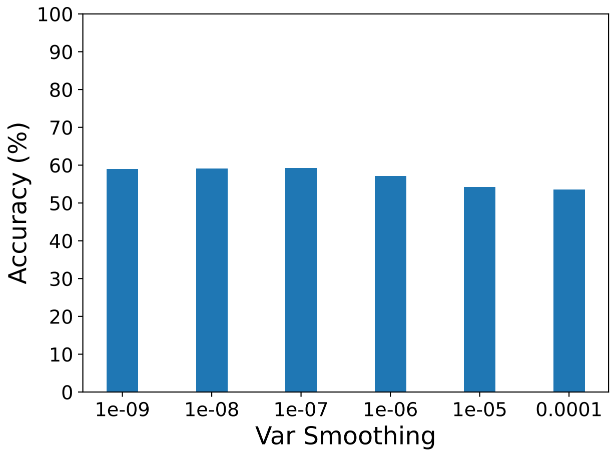


(b) Training Data

Figure 8: Bar graph of parameter optimization for Naive Bayes. a) shows testing on test data and b) shows testing on training data. The features used are White's Rating, Black's Rating, White's Win Rate, Black's Win Rate, White's Draw Rate and Black's Draw Rate. The highest accuracies are 63.38% for testing data and 63.43% for training data which came from a smoothing value of 10^{-7}.



(a) Testing Data



(b) Training Data

Figure 9: Bar graph of parameter optimization for Naive Bayes. a) shows testing on test data and b) shows testing on training data. The features used are White's Rating, Black's Rating, White's Win Rate, Black's Win Rate, Opening's Win Rate, White's Draw Rate, Black's Draw Rate and Opening's Draw Rate. The highest accuracies are 59.10% for testing data and 59.18% for the training data which came from a smoothing value of 10^{-7}.

It is slightly easier to see the best parameter in Figures 8 and 9 compared to Figure 7. For Figure 7, smoothing values of 10^{-9}, 10^{-8}, 10^{-7} and 10^{-6} all yield the same accuracy of 53.43% for the test data and 53.44% for the training data. In Figures 8 and 9 the best parameter is a smoothing value of 10^{-7} which results in the accuracies of 63.38% and 63.43% for the test data and 59.10% and 59.18% for the training data.

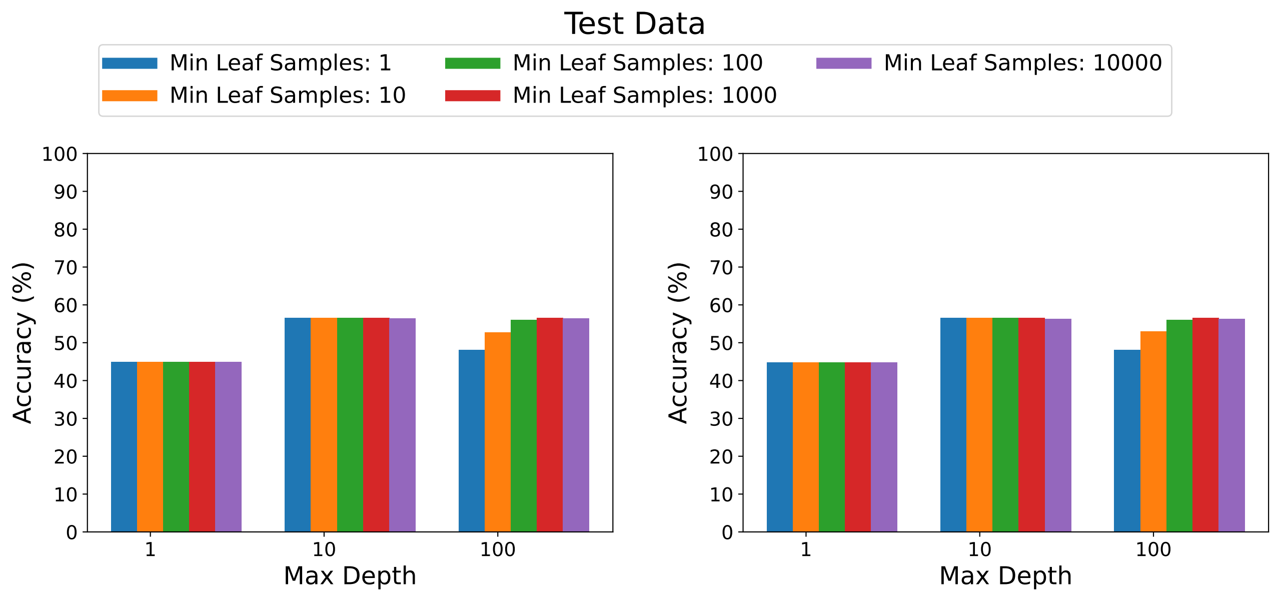
	White's Rating	Black's Rating	White's Win Rate	Black's Win Rate	White's Draw Rate	Black's Draw Rate
White's Rating		53.44, 53.43	47.21, 48.20	51.58, 51.90	47.49, 49.62	49.64, 51.45
Black's Rating	53.44, 53.43		49.97, 50.53	45.39, 46.52	50.08, 52.27	47.26, 49.05
White's Win Rate	47.21, 48.20	49.97, 50.53		51.19, 52.64	49.43, 52.00	50.37, 53.07
Black's Win Rate	51.58, 51.90	45.39, 46.52	51.19, 52.64		48.54, 51.81	48.40, 51.12
White's Draw Rate	47.49, 49.62	50.08, 52.27	49.43, 52.00	48.54, 51.81		50.13, 53.61
Black's Draw Rate	49.64, 51.45	47.26, 49.05	50.37, 53.07	48.40, 51.12	50.13, 53.61	

Figure 10: Predicted game outcome accuracy using pairs of features for Naive Bayes algorithm. All numbers are percentages. In each box the top number is the training data accuracy and the bottom number is the testing data accuracy. No individual feature pair performs as well as all the features together but the pairs with one feature from white and one feature from black tend to perform better then when both features are from the same player.

Smoothing Factor	Testing Data			Training Data		
	min	mean	max	min	mean	max
10^{-9}	67.98	68.09	68.23	67.98	68.09	68.23
10^{-8}	67.99	68.12	68.25	68.00	68.12	68.25
10^{-7}	68.11	68.21	68.38	68.11	68.21	68.38
10^{-6}	66.14	66.32	66.44	66.14	66.32	66.44
10^{-5}	56.27	56.58	56.73	56.27	56.58	56.73
10^{-4}	53.48	53.79	54.05	53.48	53.79	54.05

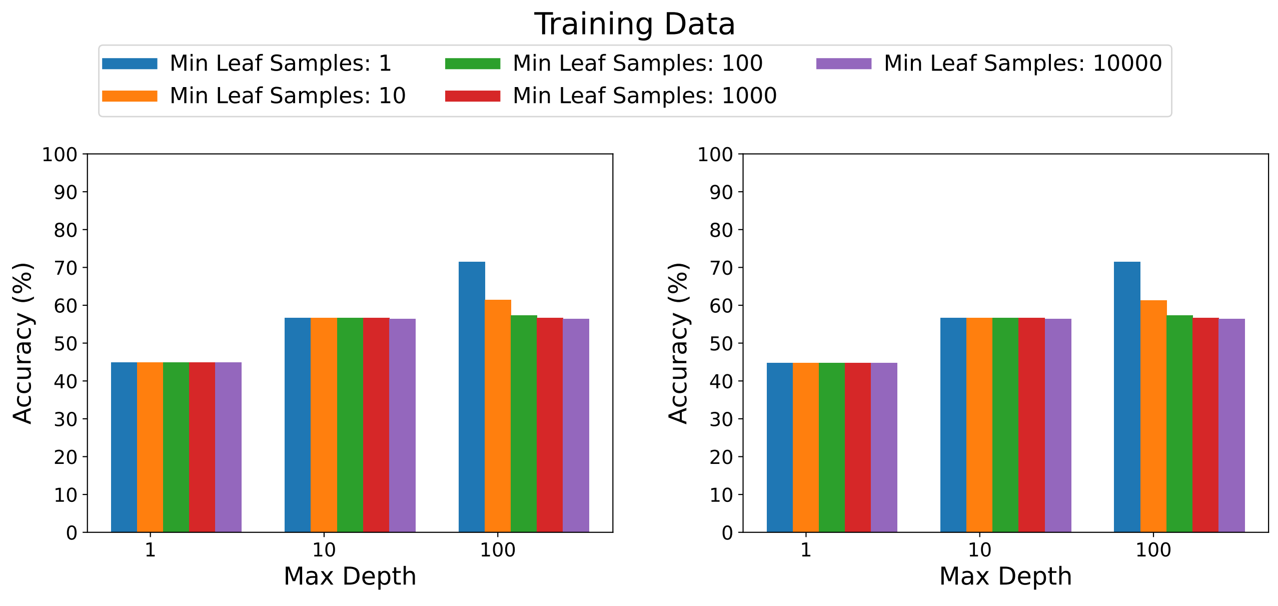
Figure 11: Predicted game outcome accuracy from ten-fold validation using all features from second feature set for Naive Bayes algorithm. All numbers are percentages.

Decision Tree



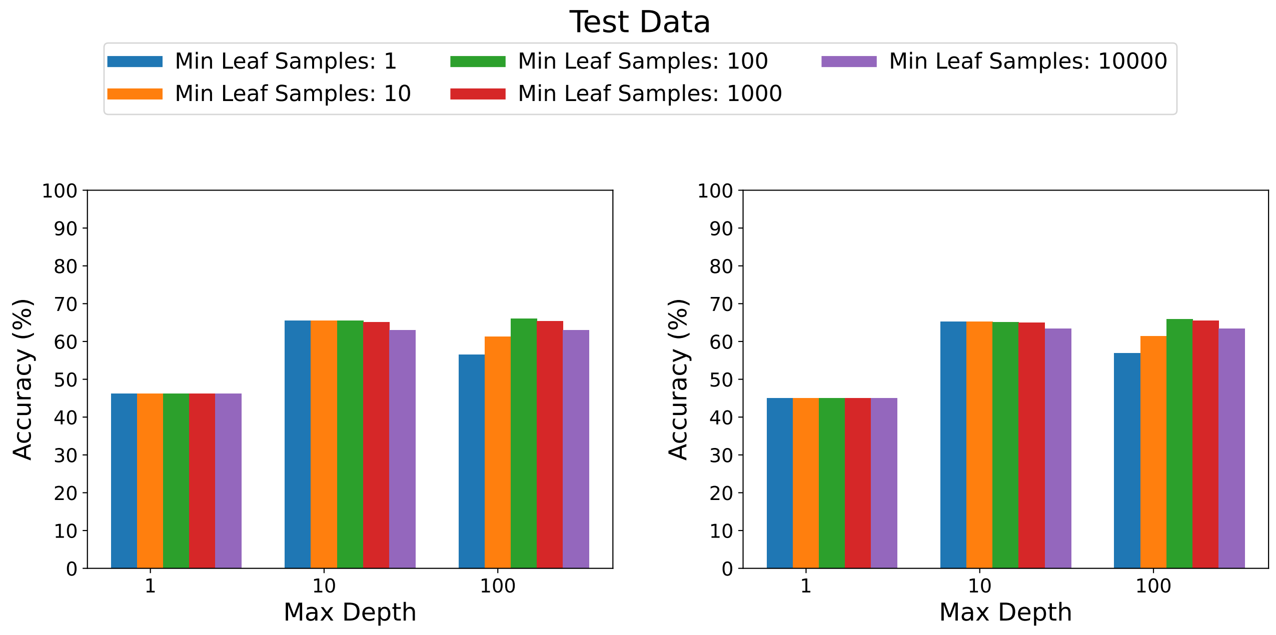
(a) Gini (b) Entropy

Figure 12: Bar graph of hyperparameter optimization for Decision Tree testing on the test data. a) shows with criterion=gini and b) shows criterion=entropy. The features used are White's Rating and Black's Rating. The best accuracy is 56.52% which comes from the hyperparameters {criterion=entropy, max depth=100, minimum leaf samples=1000}.



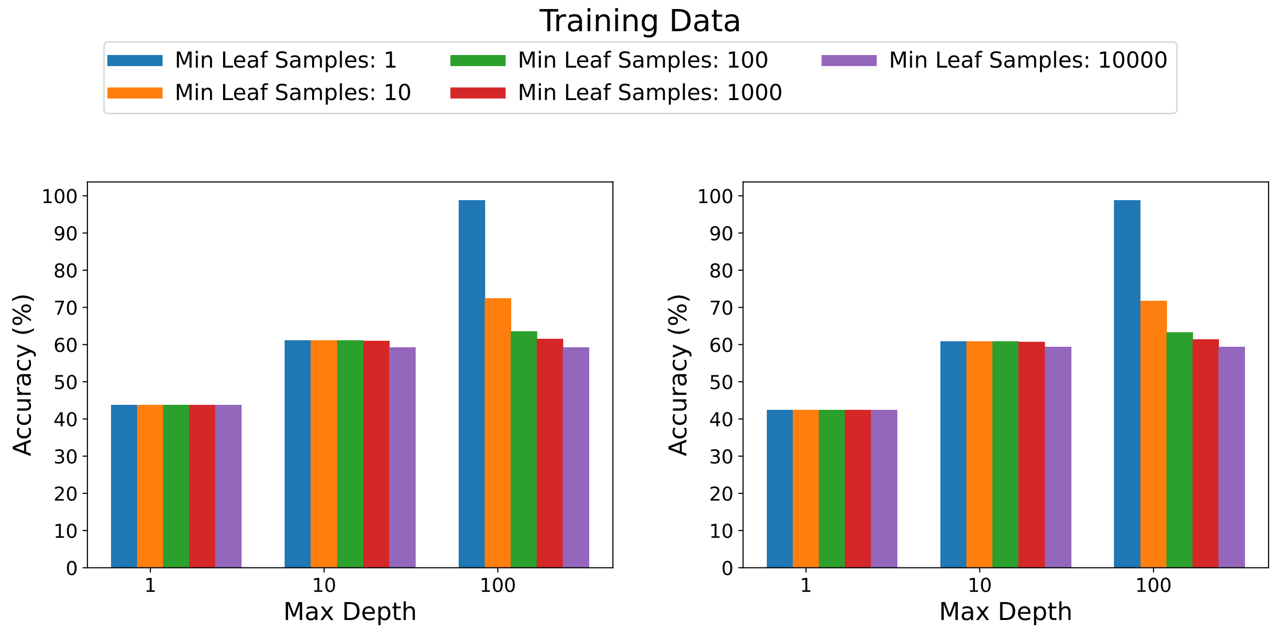
(a) Gini (b) Entropy

Figure 13: Bar graph of hyperparameter optimization for Decision Tree testing on the training data. a) shows with criterion=gini and b) shows criterion=entropy. The features used are White's Rating and Black's Rating. The best accuracy is 71.44% which came from the hyperparameters {criterion=gini, max depth=100, minimum leaf samples=1}.



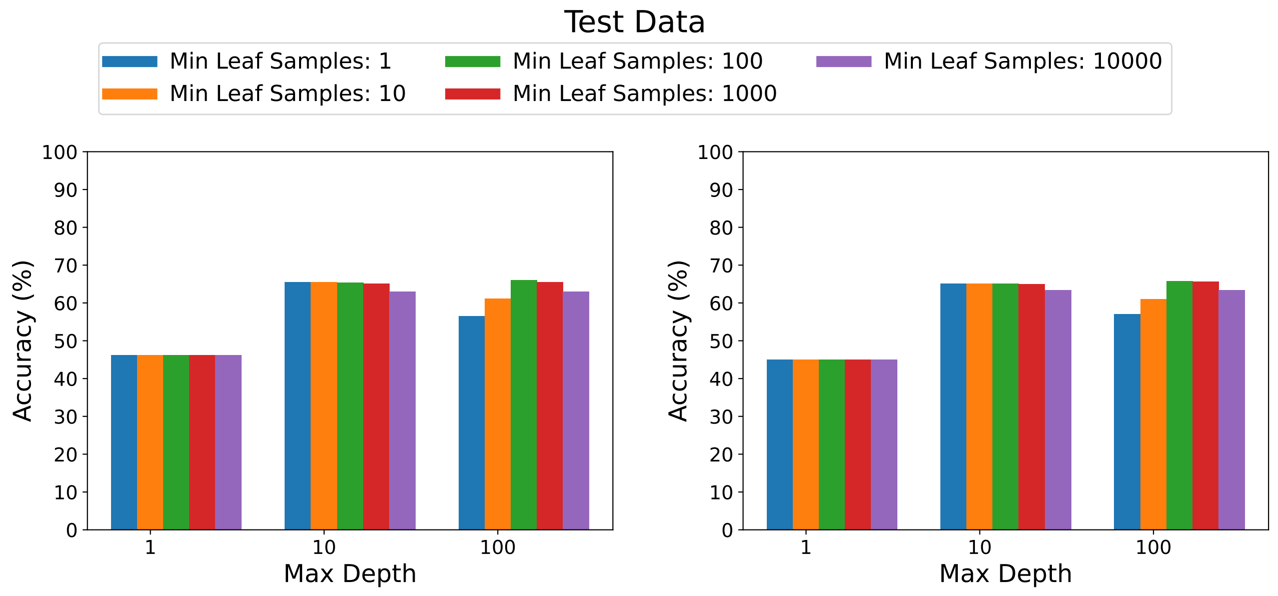
(a) Gini (b) Entropy

Figure 14: Bar graph of hyperparameter optimization for Decision Tree testing on the test data. a) shows with criterion=gini and b) shows criterion=entropy. The features used are White's Rating, Black's Rating, White's Win Rate, Black's Win Rate, White's Draw Rate, and Black's Draw Rate. The best accuracy is 66.03% which came from the hyperparameters {criterion=gini, max depth=100, minimum leaf samples=100}.



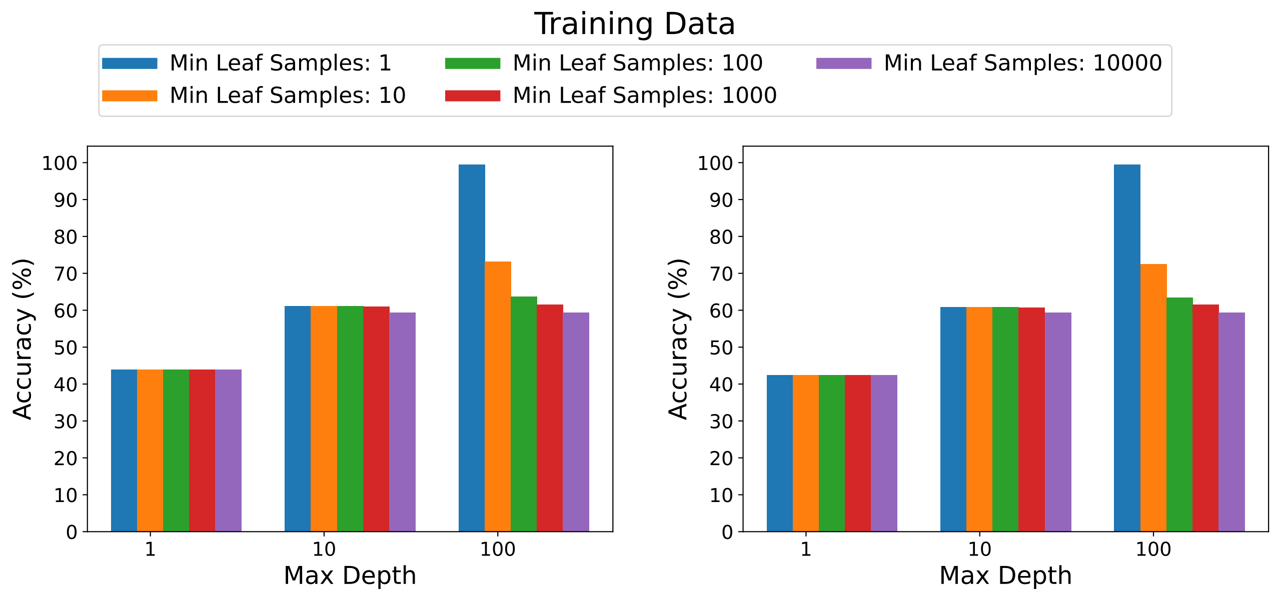
(a) Gini (b) Entropy

Figure 15: Bar graph of hyperparameter optimization for Decision Tree testing on the training data. a) shows with criterion=gini and b) shows criterion=entropy. The features used are White's Rating, Black's Rating, White's Win Rate, Black's Win Rate, White's Draw Rate, and Black's Draw Rate. The best accuracy is 98.79% which came from the hyperparameters {criterion=gini, max depth=100, minimum leaf samples=1}.



(a) Gini (b) Entropy

Figure 16: Bar graph of hyperparameter optimization for Decision Tree testing on the test data. a) shows with criterion=gini and b) shows criterion=entropy. The features used are White's Rating, Black's Rating, White's Win Rate, Black's Win Rate, Opening's Win rate, White's Draw Rate, Black's Draw Rate and Opening's Draw Rate. The best accuracy is 66.00% which came from the hyperparameters {criterion=gini, max depth=100, minimum leaf samples=100}.



(a) Gini (b) Entropy

Figure 17: Bar graph of hyperparameter optimization for Decision Tree testing on the training data. a) shows with criterion=gini and b) shows criterion=entropy. The features used are White's Rating, Black's Rating, White's Win Rate, Black's Win Rate, Opening's Win rate, White's Draw Rate, Black's Draw Rate and Opening's Draw Rate. The best accuracy is 99.49% which came from the hyperparameters {criterion=gini, max depth=100, minimum leaf samples=1}.

In Figures 12-17 the results achieved from criterion=gini and criterion=entropy are very similar. The differences between the two cannot be seen just by appearance of the graphs. For the test data, in Figure 12 entropy slightly edges out gini with the best hyperparameters being {criterion=entropy, max depth=100, minimum leaf samples=1000} which results in an accuracy of 56.52% for the test data and 56.72% for the training data. In Figures 14 and 16 gini comes out on top with best hyperparameters of {criterion=gini, max depth=100, minimum leaf samples=100} which results in accuracies of 66.03% and 66.00% for the test data and 63.54% and 63.66% for the training data. For all three sets of features the hyperparameters {criterion=gini, max depth=100, minimum leaf samples=1} outperforms every other hyperparameter set on the training data by far as can be seen in Figures 13, 15, and 17, although this is one of the worse performing hyperparameter sets for the testing data.

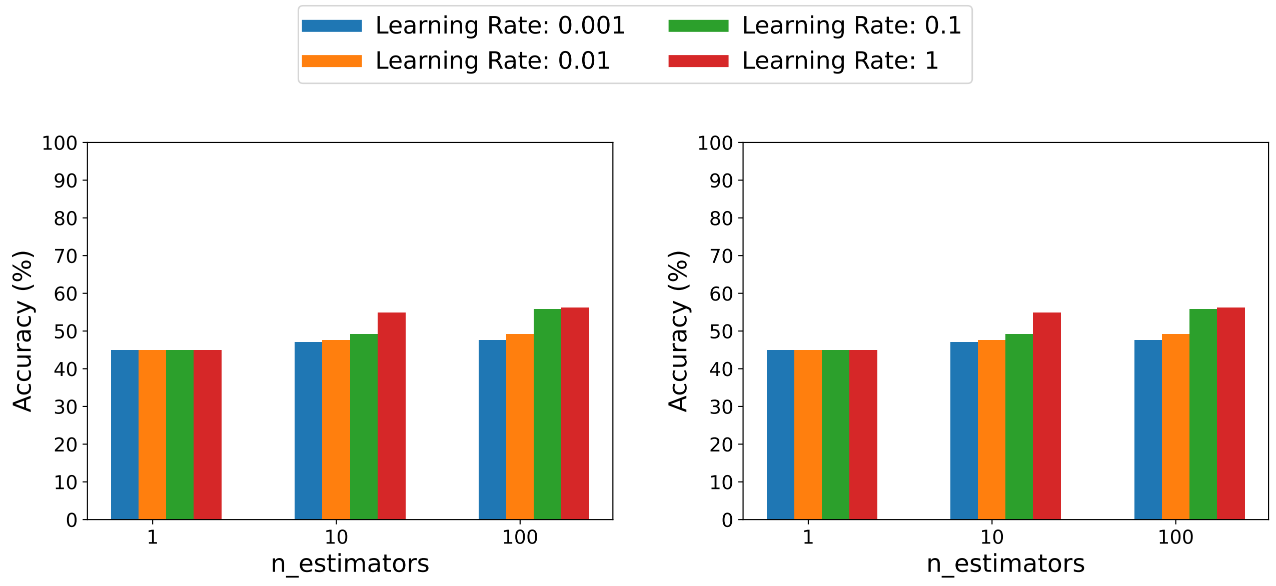
	White's Rating	Black's Rating	White's Win Rate	Black's Win Rate	White's Draw Rate	Black's Draw Rate
White's Rating		57.36, 56.00	49.86, 49.83	55.09, 52.84	49.65, 48.63	55.81, 52.53
Black's Rating	57.36, 56.00		55.42, 53.12	49.37, 48.24	55.75, 51.33	49.60, 48.30
White's Win Rate	49.86, 49.83	55.42, 53.12		54.90, 55.58	50.37, 54.58	55.08, 55.84
Black's Win Rate	55.09, 52.84	49.37, 48.24	54.90, 55.58		54.25, 52.20	49.49, 54.39
White's Draw Rate	49.65, 48.63	55.75, 51.33	50.37, 54.58	54.25, 52.20		53.78, 54.24
Black's Draw Rate	55.81, 52.53	49.60, 48.30	55.08, 55.84	49.49, 54.39	53.78, 54.24	

Figure 18: Predicted game outcome accuracy using pairs of features for Decision Tree algorithm. All numbers are percentages. In each box the top number is the training data accuracy and the bottom number is the testing data accuracy. No individual feature pair performs as well as all the features together but the pairs with one feature from white and one feature from black tend to perform better then when both features are from the same player.

Criterion	Max Depth	Min Leaf Samples	Testing Data			Training Data		
			min	mean	max	min	mean	max
gini	1	1	51.64	51.81	52.33	51.64	51.81	52.33
		10	51.64	51.81	52.33	51.64	51.81	52.33
		100	51.64	51.81	52.33	51.64	51.81	52.33
		1000	51.64	51.81	52.33	51.64	51.81	52.33
		10000	51.64	51.81	52.33	51.64	51.81	52.33
	10	1	69.82	70.04	70.26	69.82	70.04	70.26
		10	69.80	70.02	70.25	69.80	70.02	70.25
		100	69.75	69.98	70.21	69.75	69.98	70.21
		1000	69.59	69.81	70.00	69.59	69.81	70.00
		10000	68.22	68.36	68.47	68.22	68.36	68.47
	100	1	99.70	99.73	99.74	99.70	99.73	99.74
		10	92.22	93.27	92.36	92.22	93.27	92.36
		100	77.42	77.58	77.71	77.42	77.58	77.71
		1000	71.29	71.37	71.45	71.29	71.37	71.45
		10000	68.22	68.37	68.47	68.22	68.37	68.47
entropy	1	1	44.74	45.12	46.68	44.74	45.12	46.68
		10	44.74	45.12	46.68	44.74	45.12	46.68
		100	44.74	45.12	46.68	44.74	45.12	46.68
		1000	44.74	45.12	46.68	44.74	45.12	46.68
		10000	44.74	45.12	46.68	44.74	45.12	46.68
	10	1	69.08	69.30	69.49	69.08	69.30	69.49
		10	69.08	69.30	69.49	69.08	69.30	69.49
		100	69.07	69.28	69.48	69.07	69.28	69.48
		1000	68.97	69.22	69.43	68.97	69.22	69.43
		10000	67.68	68.10	68.37	67.68	68.10	68.37
	100	1	99.70	99.73	99.74	99.70	99.73	99.74
		10	93.17	93.23	93.30	93.17	93.23	93.30
		100	77.06	77.16	77.21	77.06	77.16	77.21
		1000	71.10	71.21	71.32	71.10	71.21	71.32
		10000	67.83	68.23	68.48	67.83	68.23	68.48

Figure 19: Predicted game outcome accuracy from ten-fold validation using all features from second feature set for Decision Tree Algorithm. All numbers are percentages.

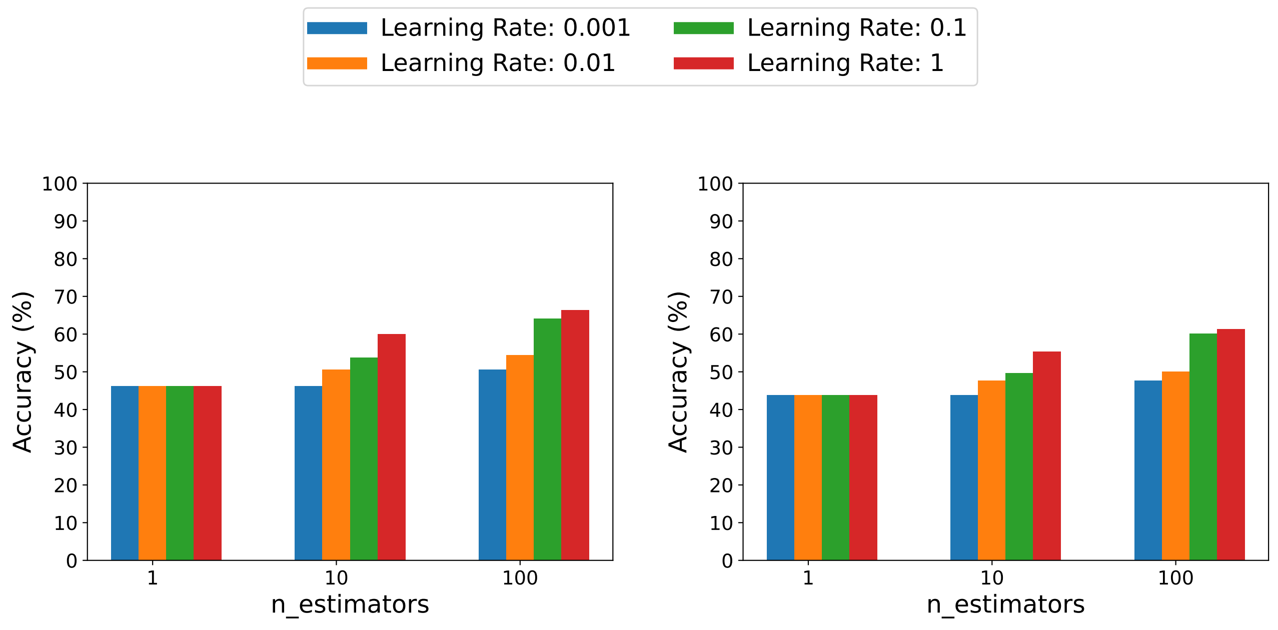
Ada Boost



(a) Testing Data

(b) Training Data

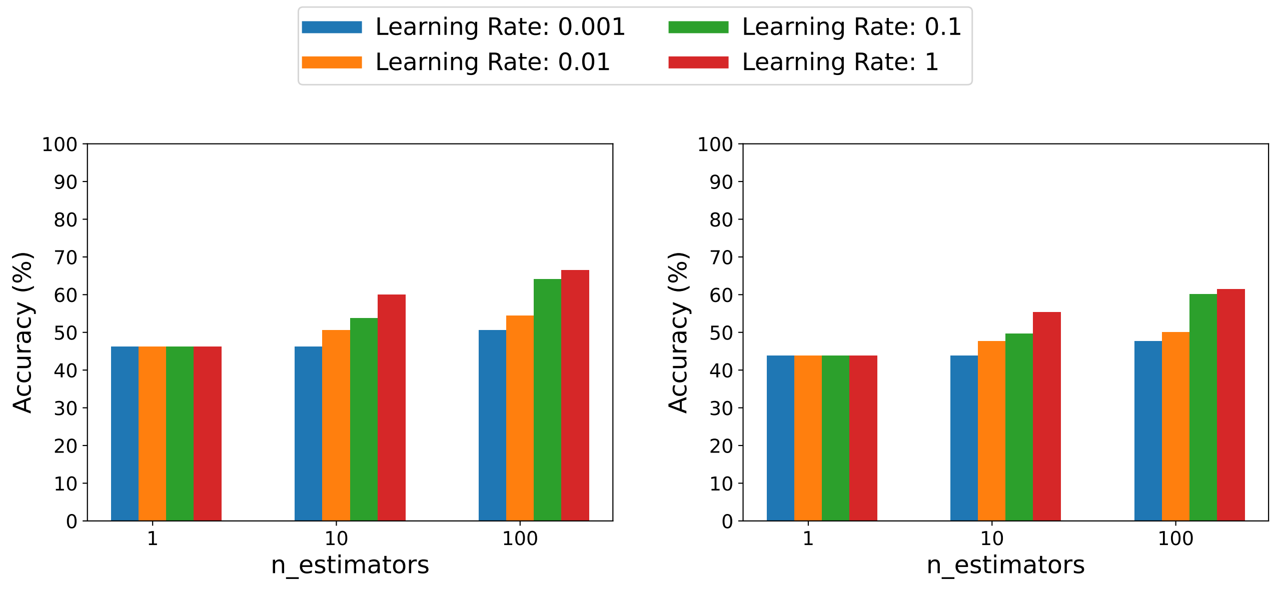
Figure 20: Bar graph of hyperparameter optimization for Ada Boost. a) shows testing on test data and b) shows testing on training data. The features used are White's Rating and Black's Rating. Both the testing and training data share the best hyperparameters which are {number of estimators=100, learning rate=1} which gave accuracies of 56.18% for testing data and 56.24% for training data.



(a) Testing Data

(b) Training Data

Figure 21: Bar graph of hyperparameter optimization for Ada Boost. a) shows testing on test data and b) shows testing on training data. The features used are White's Rating, Black's Rating, White's Win Rate, Black's Win Rate, White's Draw Rate, and Black's Draw Rate. Both the testing and training data share the best hyperparameters which are {number of estimators=100, learning rate=1} which gave accuracies of 66.32% for testing data and 61.34% for training data.



(a) Testing Data

(b) Training Data

Figure 22: Bar graph of hyperparameter optimization for Ada Boost. a) shows testing on test data and b) shows testing on training data. The features used are White's Rating, Black's Rating, White's Win Rate, Black's Win Rate, Opening's Win rate, White's Draw Rate, Black's Draw Rate and Opening's Draw Rate. Both the testing and training data share the best hyperparameters which are {number of estimators=100, learning rate=1} which gave accuracies of 66.45% for testing data and 61.46% for training data.

The best hyperparameters were found to be number of estimators:100, learning rate:1. Both the test and train data in all three feature sets share these best hyper-parameters. The accuracies found from these hyperparameters are 56.18% and 56.24% in Figure 20, 66.32% and 61.34% in Figure 21, and 66.45% and 61.46% in Figure 22.

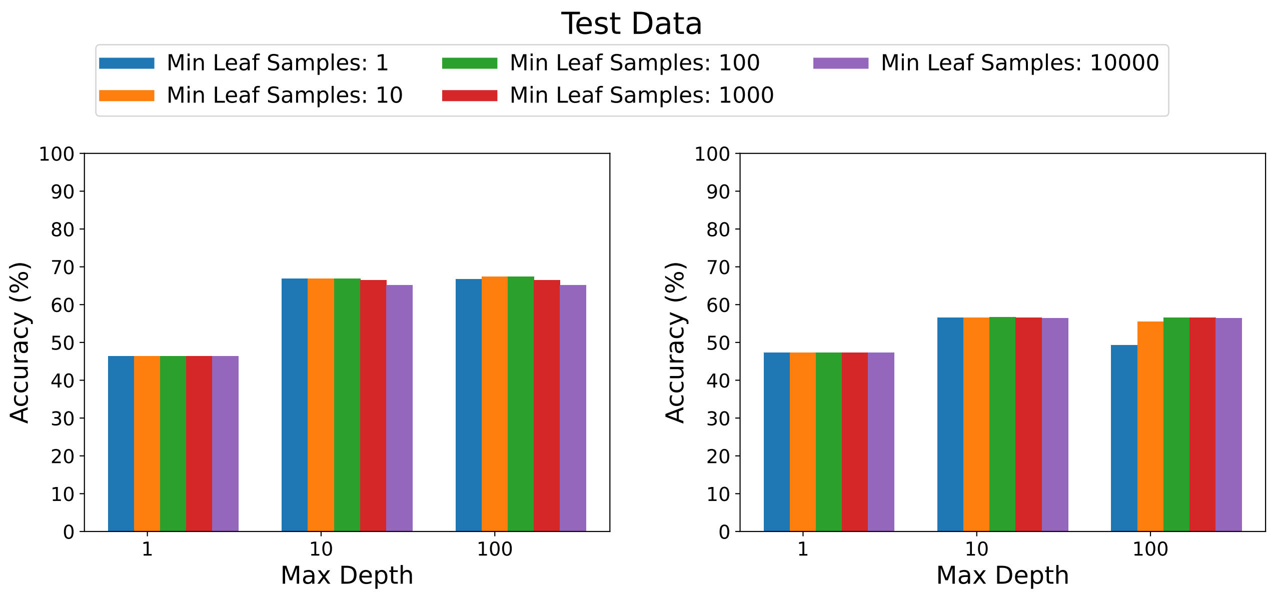
	White's Rating	Black's Rating	White's Win Rate	Black's Win Rate	White's Draw Rate	Black's Draw Rate
White's Rating		56.24, 56.18	48.26, 51.05	52.09, 53.78	47.90, 49.91	51.20, 53.37
Black's Rating	56.24, 56.18		51.15, 50.85	47.50, 49.80	51.09, 53.34	47.92, 49.67
White's Win Rate	48.26, 51.05	51.15, 50.85		52.61, 56.04	50.29, 54.52	51.30, 55.85
Black's Win Rate	52.09, 53.78	47.50, 49.80	52.61, 56.04		49.78, 54.33	49.40, 53.33
White's Draw Rate	47.90, 49.91	51.09, 53.34	50.29, 54.52	49.78, 54.33		50.75, 54.24
Black's Draw Rate	51.20, 53.37	49.92, 49.67	51.30, 55.85	49.40, 53.33	50.75, 54.24	

Figure 23: Predicted game outcome accuracy using pairs of features Ada Boost algorithm. All numbers are percentages. In each box the top number is the training data accuracy and the bottom number is from the testing data accuracy. No individual feature pair performs as well as all the features together but the pairs with one feature from white and one feature from black tend to perform better then when both features are from the same player.

n estimators	Learning Rate	Testing Data			Training Data		
		min	mean	max	min	mean	max
1	0.001	51.64	51.81	52.33	51.64	51.81	52.33
	0.01	51.64	51.81	52.33	51.64	51.81	52.33
	0.1	51.64	51.81	52.33	51.64	51.81	52.33
	1	51.64	51.81	52.33	51.64	51.81	52.33
10	0.001	51.64	51.81	52.33	51.64	51.81	52.33
	0.01	51.76	51.99	52.48	51.76	51.99	52.48
	0.1	59.65	59.98	60.42	59.65	59.98	60.42
	1	61.62	62.74	63.90	61.62	62.74	63.90
100	0.001	51.66	51.88	52.40	51.66	51.88	52.40
	0.01	60.13	61.05	61.42	60.13	61.05	61.42
	0.1	68.42	68.57	68.81	68.42	68.57	68.81
	1	69.89	70.26	70.55	69.89	70.26	70.55

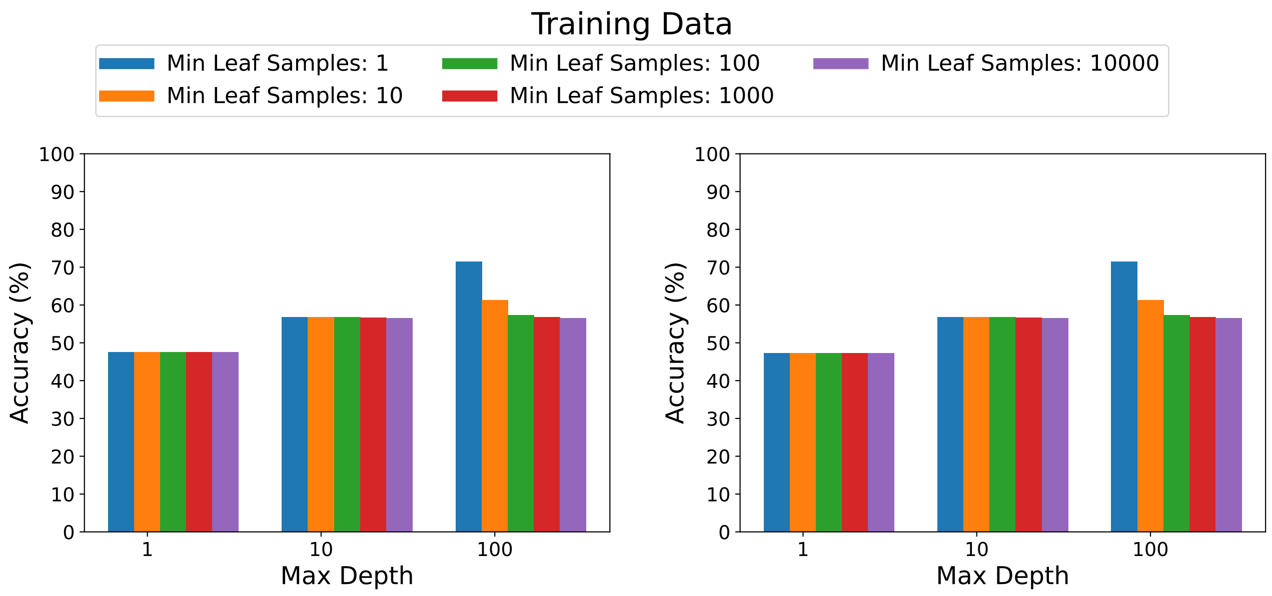
Figure 24: Predicted game outcome accuracy from ten-fold validation using all features from second feature set for Ada Boost algorithm. All numbers are percentages.

Random Forest



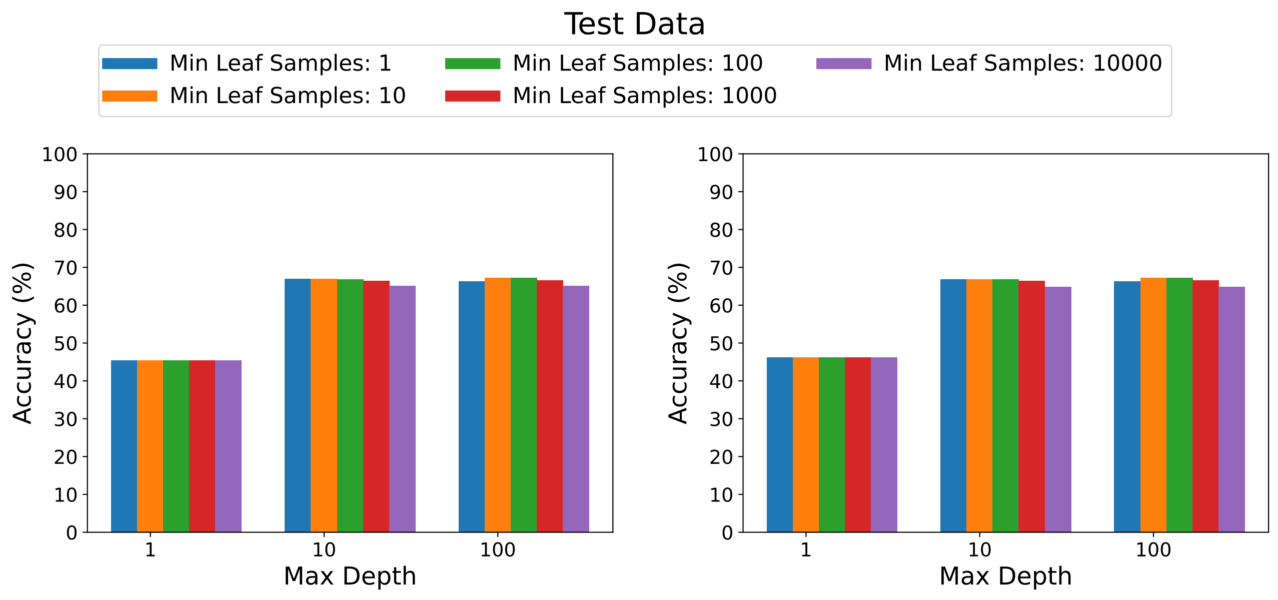
(a) Gini (b) Entropy

Figure 25: Bar graph of hyperparameter optimization for Random Forest testing on the test data. a) shows with criterion=gini and b) shows criterion=entropy. The features used are White's Rating and Black's Rating. The best accuracy is 56.60% which came from the hyperparameters {criterion=gini, max depth=10, minimum leaf samples=10}.



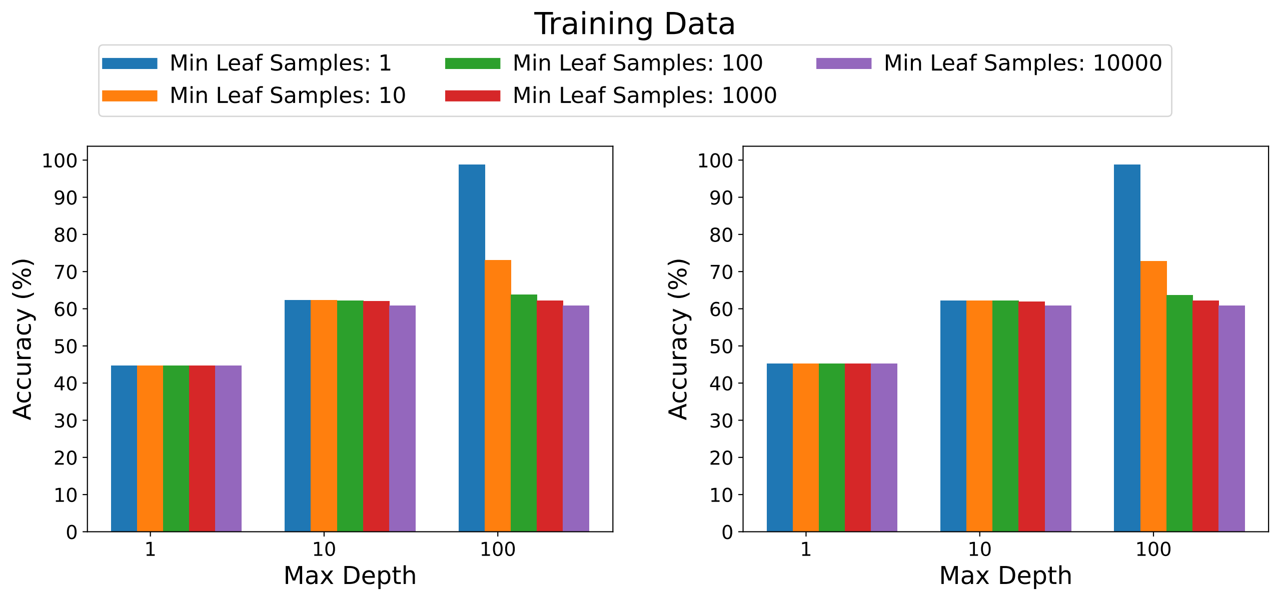
(a) Gini (b) Entropy

Figure 26: Bar graph of hyperparameter optimization for Random Forest testing on the training data. a) shows with criterion=gini and b) shows criterion=entropy. The features used are White's Rating and Black's Rating. The best accuracy is 71.45% which came from the hyperparameters {criterion=entropy, max depth=100, minimum leaf samples=1}.



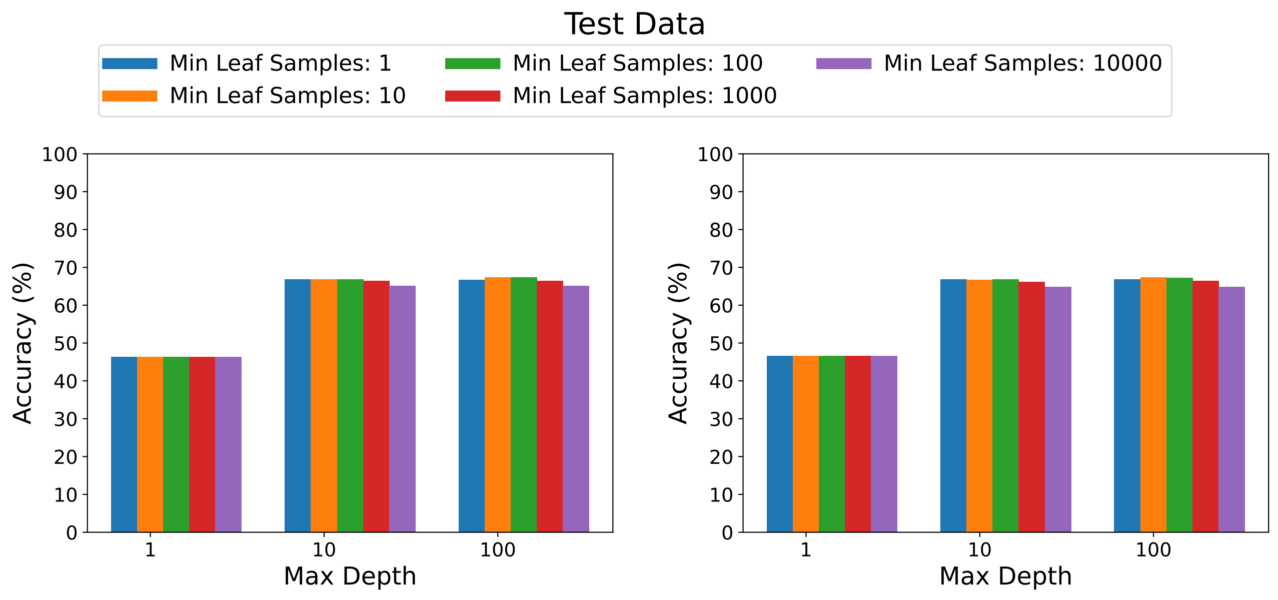
(a) Gini (b) Entropy

Figure 27: Bar graph of hyperparameter optimization for Random Forest testing on the test data. a) shows with criterion=gini and b) shows criterion=entropy. The features used are White's Rating, Black's Rating, White's Win Rate, Black's Win Rate, White's Draw Rate and Black's Draw Rate. The best accuracy is 67.27% which came from the hyperparameters {criterion=gini, max depth=100, minimum leaf samples=100}.



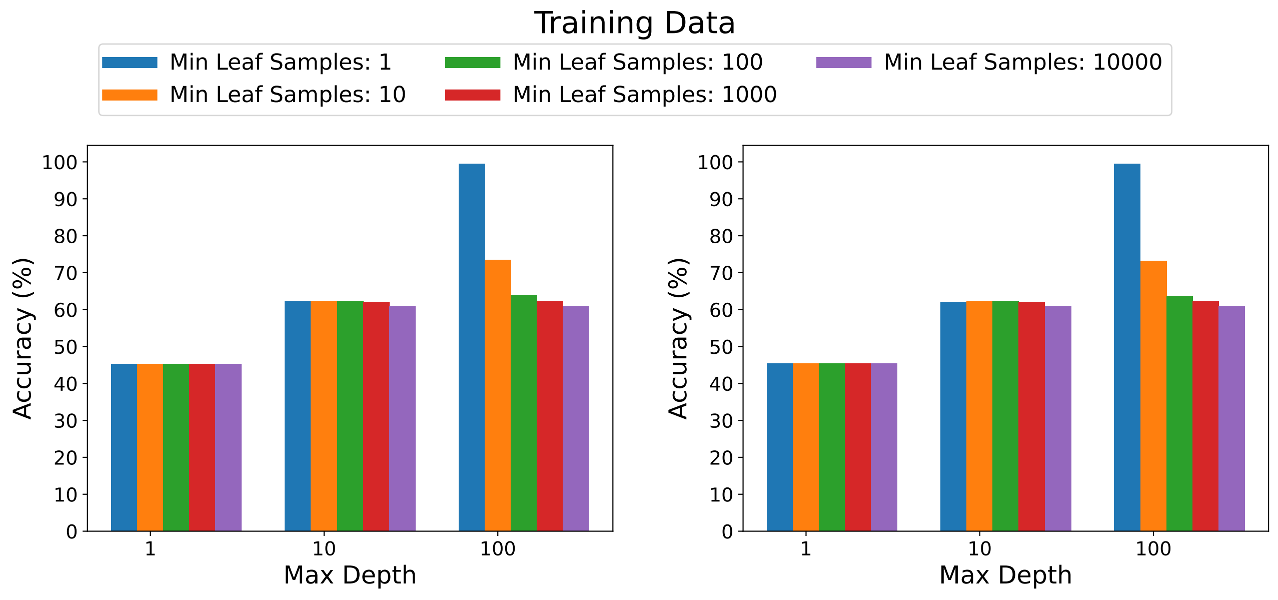
(a) Gini (b) Entropy

Figure 28: Bar graph of hyperparameter optimization for Random Forest testing on the training data. a) shows with criterion=gini and b) shows criterion=entropy. The features used are White's Rating, Black's Rating, White's Win Rate, Black's Win Rate, White's Draw Rate and Black's Draw Rate. The best accuracy is 98.79% which came from the hyperparameters {criterion=entropy, max depth=100, minimum leaf samples=1}.



(a) Gini (b) Entropy

Figure 29: Bar graph of hyperparameter optimization for Random Forest testing on the test data. a) shows with criterion=gini and b) shows criterion=entropy. The features used are White's Rating, Black's Rating, White's Win Rate, Black's Win Rate, Opening's Win Rate, White's Draw Rate, Black's Draw Rate and Opening's Draw Rate. The best accuracy is 67.41% which came from the hyperparameters {criterion=gini, max depth=100, minimum leaf samples=10}.



(a) Gini (b) Entropy

Figure 30: Bar graph of hyperparameter optimization for Random Forest testing on the training data. a) shows with criterion=gini and b) shows criterion=entropy. The features used are White's Rating, Black's Rating, White's Win Rate, Black's Win Rate, Opening's Win Rate, White's Draw Rate, Black's Draw Rate and Opening's Draw Rate. The best accuracy is 99.49% which came from the hyperparameters {criterion=entropy, max depth=100, minimum leaf samples=1}.

All three feature sets have different best hyperparameters for the test data as can be seen in Figures 25, 27, and 29. In Figure 25, the best hyperparameters are criterion=gini, max depth=10, minimum leaf samples=10 which got an accuracy of 56.60% and those same hyperparameters give an accuracy of 56.77% for the training data. In Figure 27, the best hyperparameters are criterion=gini, max depth=100, minimum leaf samples=100 which got an accuracy of 67.27% and those same hyperparameters give an accuracy of 63.78% for the training data. In Figure 29, the best hyperparameters are criterion=gini, max depth=100, minimum leaf samples=10 which got an accuracy of 67.41% and those same hyperparameters give an accuracy of 73.54% for the training data. In Figures 26, 28 and 30, it can be seen that the hyperparameters criterion=entropy, max depth=100, minimum leaf samples=1 results in the higher accuracy for the training data with accuracies of 71.45%, 98.79%, and 99.49%, respectively. It is interesting to note that here, like in the Decision Tree algorithm, the testing data has different hyperparameters that give the best accuracy for all three different sets of features while for the training data the same hyperparameters are best across all three feature sets.

	White's Rating	Black's Rating	White's Win Rate	Black's Win Rate	White's Draw Rate	Black's Draw Rate
White's Rating		57.28, 56.57	49.69, 50.88	53.69, 53.76	49.47, 49.74	54.20, 53.91
Black's Rating	57.28, 56.57		54.65, 54.04	49.18, 49.60	54.04, 53.70	49.42, 49.44
White's Win Rate	49.69, 50.88	54.65, 54.04		54.46, 56.18	50.39, 54.58	54.51, 56.51
Black's Win Rate	53.69, 53.76	49.18, 49.60	54.46, 56.18		53.95, 55.42	49.48, 53.39
White's Draw Rate	49.47, 49.74	54.04, 53.70	50.39, 54.58	53.95, 55.42		53.84, 55.06
Black's Draw Rate	54.20, 53.91	49.42, 49.44	54.51, 56.51	49.48, 53.39	53.84, 55.06	

Figure 31: Predicted game outcome accuracy using pairs of features Random Forest algorithm. All numbers are percentages. In each box the top number is the training data accuracy and the bottom number is the testing data accuracy. No individual feature pair performs as well as all the features together but the pairs with one feature from white and one feature from black tend to perform better then when both features are from the same player.

Criterion	Max Depth	Min Leaf Samples	Testing Data			Training Data		
			min	mean	max	min	mean	max
gini	1	1	55.23	55.99	57.51	55.23	55.99	57.51
		10	55.23	55.99	57.51	55.23	55.99	57.51
		100	55.23	55.99	57.51	55.23	55.99	57.51
		1000	55.23	55.99	57.51	55.23	55.99	57.51
		10000	55.23	55.99	57.51	55.23	55.99	57.51
	10	1	71.86	71.97	72.09	71.86	71.97	72.09
		10	71.82	71.95	72.03	71.82	71.95	72.03
		100	71.76	71.85	71.94	71.76	71,85	71.94
		1000	71.36	71.49	71.60	71.36	71.49	71.60
		10000	70.17	70.25	70.33	70.17	70.25	70.33
	100	1	99.70	99.73	99.74	99.70	99.73	99.74
		10	99.61	99.64	99.66	99.61	99.64	99.66
		100	78.51	78.64	78.78	78.51	78.64	78.78
		1000	71.97	72.09	72.19	71.97	72.09	72.19
		10000	70.17	70.25	70.35	70.17	70.25	70.35
entropy	1	1	50.09	51.53	52.33	50.09	51.53	52.33
		10	50.09	51.53	52.33	50.09	51.53	52.33
		100	50.09	51.53	52.33	50.09	51.53	52.33
		1000	50.09	51.53	52.33	50.09	51.53	52.33
		10000	50.09	51.53	52.33	50.09	51.53	52.33
	10	1	71.41	71.54	71.68	71.41	71.54	71.68
		10	71.48	71.53	71.67	71.48	71.53	71.67
		100	71.41	71.50	71.60	71.41	71.50	71.60
		1000	71.24	71.31	71.46	71.24	71.31	71.46
		10000	70.05	70.18	70.30	70.05	70.18	70.30
	100	1	99.70	99.73	99.74	99.70	99.73	99.74
		10	99.65	99.67	99.68	99.65	99.67	99.68
		100	78.25	78.36	78.48	78.25	78.36	78.48
		1000	72.03	72.12	72.24	72.03	72.12	72.24
		10000	70.07	70.18	70.30	70.07	70.18	70.30

Figure 32: Predicted game outcome accuracy from ten-fold validation using all features from second feature set for Random Forest Algorithm. All numbers are percentages.

Discussion

In Figures 10, 18, 23 and 31, the algorithm was run comparing each pair of features which shows that while any pair of features results in a higher accuracy then random chance, using all of the features results in a higher accuracy then any of the pairs. Additionally, something interesting to note is that when using one feature from White and one feature from Black, the accuracies will generally be higher than if using both features from either White or Black, even if it is the same type of feature. So using White's Win Rate and Black's Draw Rate yields a higher accuracy then using White's Win Rate and White's Draw Rate. Even though in both cases a win rate and a draw rate are used, having both players represented in the data gives a higher accuracy.

Comparing Figures 8 and 9, 14 and 16, 21 and 22, and 27 and 29, in all four pairs both graphs look virtually identical, the main difference being one graph including the opening's rates and the other excluding them. The accuracies when the opening's rates are included are technically higher, but only by a few tenths of a percent. Including the opening rates as features makes it so that the algorithms can only be used after the game has been played since we need to know what opening the game was. Not including these rates gives us two benefits; it allows us to have an algorithm that is usable before a game as been played and since it gets ride of two features the run time of the algorithm will improve.

In theory, using information about the opening played in the game should improve the accuracy. Many openings are known to have a better track record for either white or black and some variations of certain openings are known to be basically be played out straight into a drawing position. Adding in the opening's win and draw rate should be adding useful information that would increase the accuracy of at least one of the algorithms by a more substantial amount than a few tenths of a percent. So, then, why isn't it?

A likely reason for this has to do with the fact that the openings go by names, and not numbers like players ratings, and the way that they are put into the database. For some openings it is very clear which series of moves results in that opening title. However, the majority of openings have a parent name but also many sub-variations that are caused by certain moves that deviate from the main line but they still fall under the same parent opening. Openings are also always changing and evolving. Chess professionals are finding new moves to play in different openings and creating new variations every year. Additionally, there are many openings that go by multiple different names.

This dataset has games that date back to the 1990's, which is a long time is the chess world.

While openings that were played back then are still being played today, there are different most popular openings, new variations that were not seen before, and some openings that have started going by different names over time. When the games are being put into the database, it is likely that sometimes they are just being entered according to the parent name for the opening, sometimes they are being entered as the very specific variation, sometimes as an abbreviation of the opening name, or sometimes with extemporaneous punctuation marks. So when calculating the rates, all of these openings are interpreted as different openings, when in reality they all stem from the same opening.

From here, there are multiple different routes that could be taken with future work. This research has been focused more on the players playing the game as opposed to the position on the board while the game is in progress. A continuation of this work could be to add in features about the position evaluation. One option would be to pick a set number of moves into the game and add the position evaluation at that point in the game for every game in the data set. The difficulty for that would be picking the point at the game to take that position evaluation. Taking the position evaluation too early the game would not be very valuable since it is too early in most games to give us much information, there is still much of the game to be played and the majority of position evaluations too early in the game will be very close to zero. Whereas, the further out into the game you choose to take that position evaluation, the more games there will be that have ended before they ever got that far into the game.

Another option would be to focus less on any particular position evaluation and instead look at the plot of the position evaluation over the whole game. Most people are stronger in some parts of the game than others. So if someone has a habit of not having a great position during the middle game but they typically go into endgames and are very strong in those, that pattern would show up over all of their games in the plot of position evaluation. This would essentially be a more detailed feature compared to just the players win and draw rates.

Additionally, as mentioned in the limitations of the data, the time control of the game is not currently a feature. If the time control the game was played at was available, the time control could be added as a feature. The other option would be to have different datasets where each time control category would have its own training and testing data set. In order to have data that includes the time control, one possibility would be to find different data to start with that includes the time control. The data from the TWIC database does include the name of the chess tournament the game was played at it and the date so with some internet research it would likely be possible to find out the time control of the games. However with as many data points as this data has, doing this by hand would be nearly impossible.

Currently, the algorithm is just predicting the result of the chess game: White wins, Black wins, or the players Draw. Another path for future work would be to compute posterior probabilities. Instead of just predicting who will win, the probability that the win occurs could be estimated. So currently the algorithm might just predict that white will win a certain game. For example, it might be useful to be able to predict there is a 60% chance of White winning, a 10% chance of Black winning, and a 30% chance of the game ending in a Draw.

Conclusion

It was found across all four machine learning algorithms used in this research that using the ratings of the players as features will result in around a 20% increase in accuracy over random chance. Adding the win and draw rates of the players to the players ratings will increase the accuracy by about another 10%. When the opening's win and draw rates are added in as a feature, it appears that there is no significant change in accuracy. Some of the algorithms had their accuracy go up by only a few tenths of a percentage and some of the algorithms actually had their accuracy go down by about the same amount.

When comparing the different feature pairs and the resulting accuracy, none of the pairs perform as well as using all the features, although some feature pairs perform better than others. Typically a pair that had information about both the White and Black player gave a better accuracy then a pair where both features were about one player. This makes sense as chess is a two player game, so the algorithm having information about both players would be able to do better then just knowing information about one player.

This research was successful in the sense that it accomplished what it set out to accomplish and found a way to predict the outcomes of chess games with a higher accuracy then random chance. Even so, there is still room for improvement within the algorithm and features that could be added to improve the accuracy.

The code used in this research is available on GitHub at “sofia-decredico-thesis”.

References

[1] https://stockfishchess.org/.

[2] P. Lehana, S. Kulshrestha, N. Thakur, and P. Asthana, "Statistical analysis on result prediction in chess," *International Journal of Information Engineering and Electronic Business*, vol. 11, no. 4, p. 25, 2018.

[3] R. Dreżewski and G. Wątor, "Chess as sequential data in a chess match outcome prediction using deep learning with various chessboard representations," *Procedia Computer Science*, vol. 192, pp. 1760–1769, 2021.

[4] D. R. Ferreira, "Predicting the outcome of chess games based on historical data," Tech. rep., IST-Technical University of Lisbon (November 2010), Tech. Rep., 2010.

[5] M. Banoula, "Understanding naive bayes classifier," *simplilearn*, 2023.

[6] Z.-H. Zhou, *Ensemble Methods Foundations and Algorithms.* Taylor & Francis Group, LLC, 2012.

[7] S. Dash, "Decision trees explained - entropy, information gain, gini index, ccp pruning," *Towards Data Science*, 2022.

[8] https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-decision-tree/.

[9] V. Kurama, "A guide to adaboost: Boosting to save the day," *Paperspace*, 2020.

[10] https://theweekinchess.com/twic.