Categorization of Cheaters in Chess

Group 215 - Brian Coy

November 2025

1 Introduction and Problem Statement

The presence of dishonest players on a site like Chess.com lowers the trust of honest players to be treated fairly and to accurately judge their actual skill in the game. Unfortunately, players have been caught using chess engines during play to cheat and win games that they would not otherwise have the ability to win. While Chess.com does have its own system of detecting "fair play violations," they have to balance the high cost of false accusations and tend to trust the players more often than assume the worst.

This project aims to develop a machine learning framework for identifying players who use bots to cheat on games. Using a combination of unsupervised and supervised machine learning models, I will attempt to categorize players into most-likely honest and most-likely dishonest players so that the likely dishonest ones can be investigated.

The analysis relies on understanding how player rankings work. The basic idea of the ELO rating system in chess is that each player has a numerical rating. The difference between two players generally indicates the probability that one player wins against another. For example, a player with a 1500 rating should beat a 1300 player more often than not, and a 2500 rated player should nearly never lose against a 700 rated player.

In general, the move accuracy of a player should also coincide with their ELO rating, with higher ranked players having a higher move accuracy. By analyzing game data, we can look for specific patterns where low-rated players consistently display the high accuracy of a chess engine for a large portion of a game.

2 Data Collection and Methodology

The primary data source for this project was the Chess.com public API. We utilized a custom Python script to iterate through player archives, collecting details such as player names, current ratings, game outcomes, and historical game records.

The original proposal outlined a plan to download the full sequence of moves in Portable Game Notation (PGN) and use a local engine, such as Stockfish, to independently calculate the accuracy of every move. The initial estimates suggested a dataset of thousands, or perhaps hundreds of thousands, of games would be sufficient for the model.

However, the actual data collection phase resulted in a significantly larger dataset than anticipated. The final scrape included 22,087 players and a total of 12,220,862 games.

It became immediately apparent that running a local engine analysis on over 12 million games was not possible given the time and hardware limits. As a result, the we adjusted to rely on the accuracy scores provided directly by the Chess.com API. This pivot allowed us to analyze the entire dataset without the issues of trying local processing.

To handle the large amount of data efficiently, it was stored in a local SQLite database. We further shrunk the dataset by filtering out older records where accuracy data was null. The analysis focused exclusively on 10-minute ”Rapid” games, as this time control provides a cleaner signal for correlating rating with accuracy than blitz games, and likely should have more variation in accuracy than longer games.

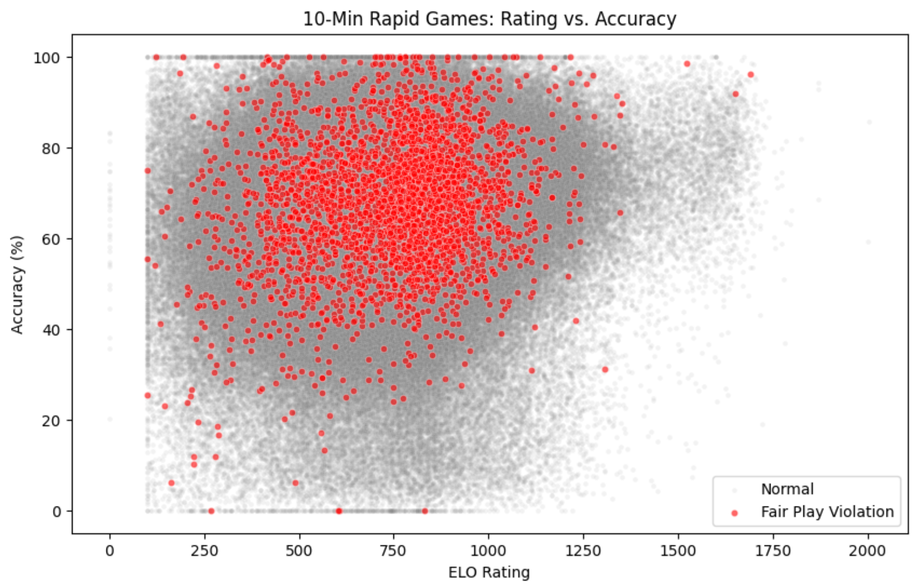
2.1 Exploratory Data Analysis

To test the hypothesis that cheaters exhibit a distinct ”signature” of low rating while still having high accuracy, we visualized the relationship between ELO rating and game accuracy. We plotted ELO rating on the x-axis and accuracy percentage on the y-axis, overlaying games played by users flagged with ”fair play violations” (red) against those with normal status (gray).

While we hoped for a clear separation, the scatter plot revealed a massive overlap between the two groups. As seen in Figure 1, the ”cheater” data points are not isolated in the top-left quadrant (low ELO, high accuracy) but are instead distributed widely throughout the general population of players.

The lack of an obvious separation of honest players and cheaters is an excellent reason to use machine learning on this problem, as it is not as simple as just looking for a clear line separating the cheaters.

Figure 1:



3 Modeling and Results

To tackle the classification problem, we employed three distinct machine learning approaches. We began with K-Means Clustering (unsupervised) to see if cheaters would naturally separate from honest players. When this proved insufficient, we pivoted to Support Vector Machines (SVM) and Random Forest (supervised) to handle the complexity of the data.

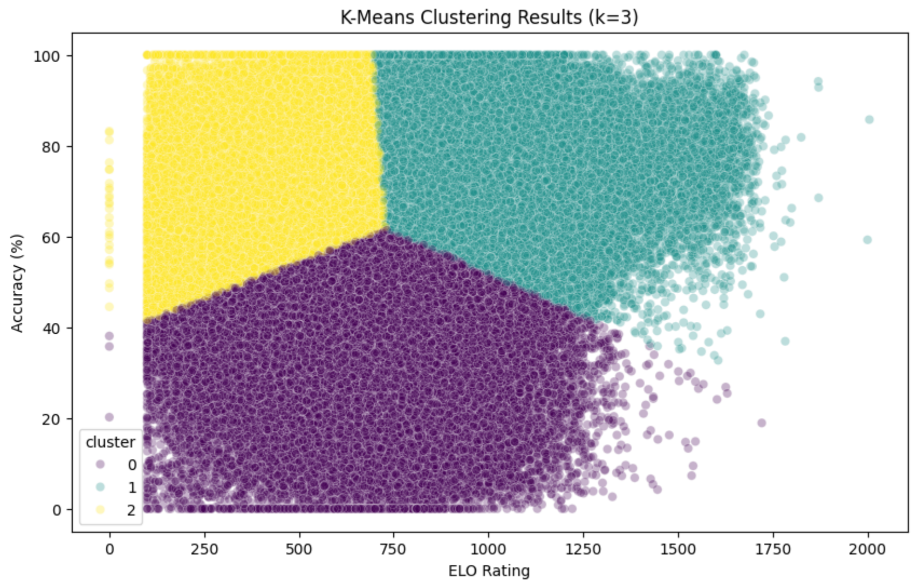
3.1 K-Means Clustering

We ran the K-Means algorithm with several number of clusters, starting with 2, and eventually reaching 8. The best results obtained were with k = 3, but because the fair_play_violation accounts were spread out through the normal accounts so evenly, it did not result in any ability to separate the cheaters from the non-cheaters.

Table 1: K-Means Clustering Analysis (k=3)

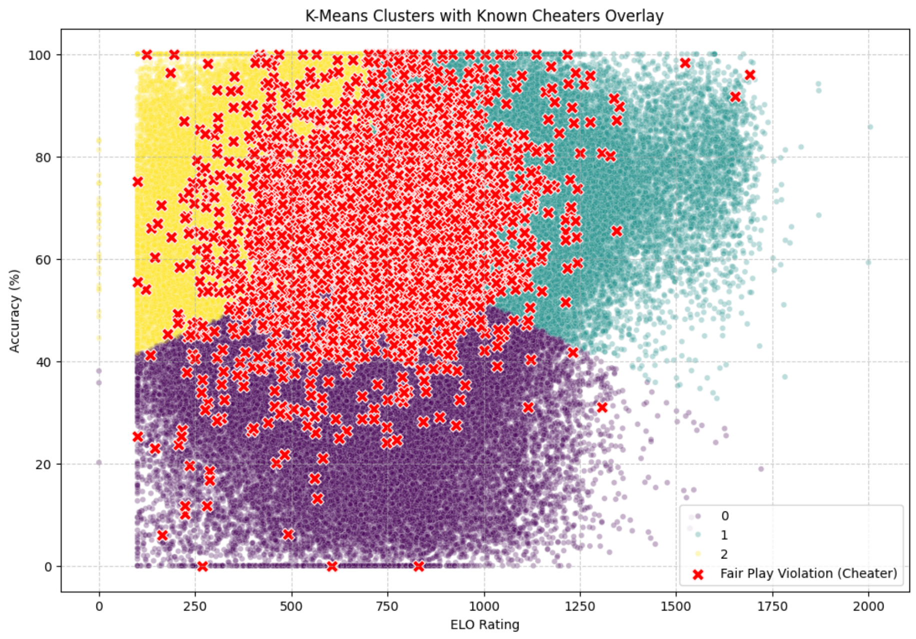
Cluster	Count	Mean Rating	Mean Accuracy (%)	Cheater Rate
0 (Beginner/Low)	193,323	704.28	46.29	0.35%
1 (Expert/High)	356,495	894.74	72.41	0.47%
2 (Suspicious)	230,278	546.73	70.80	0.42%

Figure 2:



The cluster of the results that we would suspect the most, actually accounted for the lowest percentage of fair_play_violation accounts.

Figure 3:



3.2 Support Vector Machine

Given the massive class imbalance (less than 1% of players were cheaters), we attempted to use SVM. This approach will only use the ”normal” data to learn the shape of honest play and then flags any data points that fall outside this boundary as anomalies.

The resulting visualization offers a clear explanation for the difficulty of this task. The blue region in the plot represents the ”normal” decision function learned by the model. The purple triangles represent data points the model successfully flagged as mathematical anomalies.

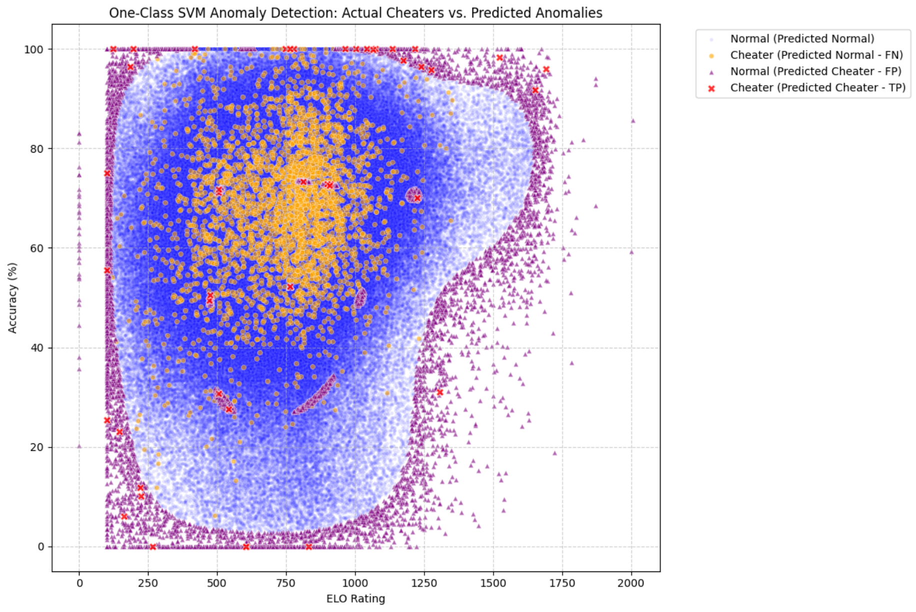
However, the actual cheaters (represented by orange dots) are not located at the fringes of the distribution. Instead, the vast majority of cheaters are clustered tightly in the center of the blue ”normal” zone. They mimic the statistical profile of average honest players almost perfectly.

The confusion matrix and classification report shows:
True Positives: The model correctly identified only 45 cheaters.
False Negatives: The model missed 3,261 cheaters, classifying them as normal.
False Positives: The model incorrectly flagged 9,193 honest players as potential cheaters.

Table 2: One-Class SVM Anomaly Detection Results

Metric	**Value**	**Description**
Confusion Matrix		
True Negatives (Normal)	767,597	Predicted Normal, Actually Normal
False Positives (Normal)	9,193	Predicted Cheater, Actually Normal
False Negatives (Cheater)	3,261	Predicted Normal, Actually Cheater
True Positives (Cheater)	45	Predicted Cheater, Actually Cheater
Performance Metrics (Cheater Class)		
Precision	0.00	
Recall	0.01	
F1-Score	0.01	
Overall Accuracy	0.98	

Figure 4:



With a recall score of 0.01 and precision of 0.00, the model was effectively unable to distinguish a cheater from a normal player. The graph confirms that while the SVM worked correctly by mathematical standards (identifying statistical outliers), "cheating" in chess does not manifest as a statistical outlier in terms of simple Rating vs. Accuracy.

3.3 Random Forest

Finally, we implemented a Random Forest Classifier. We hypothesized that an ensemble method might be able to find non-linear decision boundaries that the other models missed.

Unfortunately, the Random Forest model performed worse than the previous methods in terms of detecting the positive class. As seen in the confusion matrix, the model predicted "False" (Normal) for every single actual cheater in the test set.

True Positives: 0
False Negatives: 656
ROC AUC Score: 0.506

An AUC score of 0.5 indicates that the model is performing no better than random guessing. Because the dataset is so heavily imbalanced, the model achieved a high "accuracy" of 99% simply by classifying every single player as honest. The visualization confirms this failure, with the plot showing a sea of orange "False Negative" dots where the model failed to flag the cheaters hiding within the normal population.

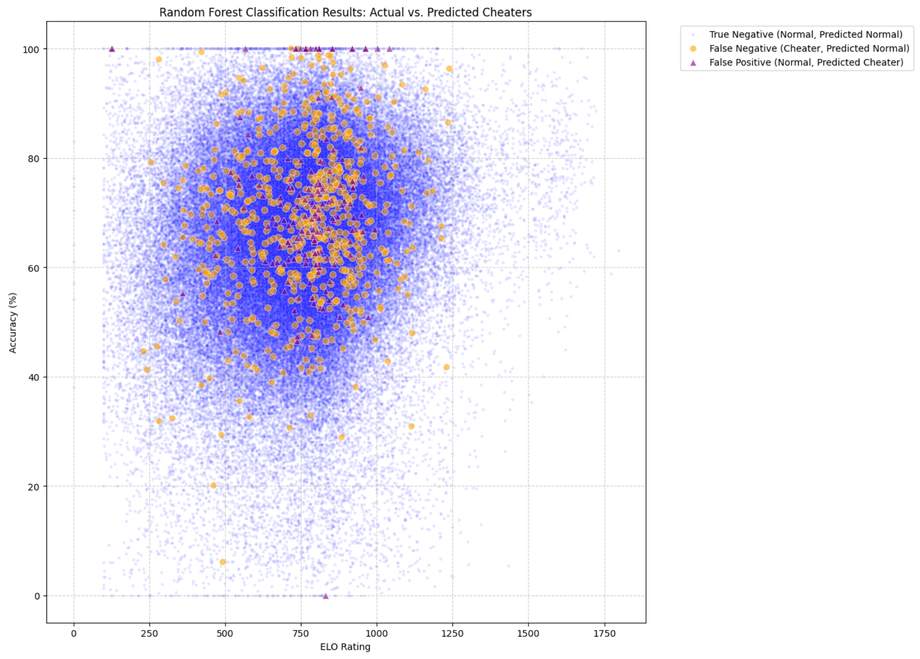
Table 3: Random Forest Classification Results

Metric	Value	Description
Confusion Matrix		
True Negatives (Normal)	155,193	Predicted Normal, Actually Normal
False Positives (Normal)	171	Predicted Cheater, Actually Normal
False Negatives (Cheater)	656	Predicted Normal, Actually Cheater
True Positives (Cheater)	0	Predicted Cheater, Actually Cheater
Performance Metrics (Cheater Class)		
Precision	0.00	
Recall	0.00	
F1-Score	0.00	
ROC AUC Score	0.5062	

4 Evaluation and Results

The point of this project was to determine if public game data, specifically ELO rating and game accuracy, could be used to build a reliable automated detection system for chess cheaters.

Figure 5:



The results of our analysis conclusively show that these two variables alone are insufficient for the task. Across all three modeling techniques, we encountered the same fundamental obstacle: there is no simple threshold for ELO vs accuracy between cheaters and honest players.

Exploratory Analysis revealed that cheaters do not consistently produce high-accuracy outliers; they often play with average accuracy or cheat only when necessary to win the game.

K-Means failed to cluster cheaters into a distinct group.

SVM and Random Forest failed to separate the classes, with the Random Forest model failing completely.

While the project did not result in a working detector, it provided a valuable insight into the complexity of the problem. Use of per-game summary statistics is not enough to catch cheating. Future work would require a full game analysis, as originally proposed, to detect the specific moments where a player's behavior diverges from human norms, rather than looking at their average performance over an entire game.