Tennis match forecasting; how AI can beat book makers on their own platforms

Patrick Schall* and Vahid Toomani†

(Dated: March 8, 2024)

* patrick.schall@gmail.com

† vahidtoomani2002@gmail.com

CONTENTS

I. INTRODUCTION

In 1943, Walter Pitts and Warren McCulloch laid the foundation for artificial intelligence (AI) and its subfield, machine learning, with their first mathematical model of a neural network [**?**]McCulloch and Pitts, 1943).

Seven years later, Alan Turing posed the fundamental question: "Can machines think?" He developed the Turing test, originally known as the imitation test, which assesses whether a machine exhibits intelligent behavior. In his seminal work "Computing Machinery and Intelligence" [**?**]Turing, 1950), Turing didn't settle for a simple definition of machines or thinking; instead, he asked a more profound question: "Can machines do what we, as thinking entities, can do?"

Over the past 70 years, fueled by ever-growing computational power, increased storage capacity, and an exponentially expanding pool of data, machine learning (ML) and deep learning algorithms have revolutionized the field of data science [**?**]Sarker, 2021, McCarn Deiana, Tran, et al., 2021). ML models can predict whether a customer will cancel their contract with a telecommunication provider, determine the authenticity of an unknown painting, verify the origin of wine analyzed in a laboratory, and even classify tumors as benign or malignant. Recent breakthroughs include autonomous driving cars, defeating the world's best Go player (a game more complex than chess and reliant on intricate pattern recognition), and predicting the 3D structure of proteins from their amino acid sequences [**?**]Silver er al., 2017, Jumper et al., 2021).

One significant advantage of machine learning models lies in their ability to forecast the future with a high degree of certainty. Today, ML models are employed to predict weather patterns, estimate fuel consumption for new cars, and optimize maintenance intervals for machinery. A practical application of ML is in the prediction of sports betting outcomes. By analyzing historical data, player performance, and other relevant factors, ML models can provide valuable insights for informed betting decisions.

A. Elo system

The Elo rating system, named after its creator Arpad Elo, is a method for calculating the relative skill levels of players in two-player games, such as chess, Go, and various sports competitions. Originally devised for chess, the Elo system has since found applications in a wide array of competitive endeavors where pairwise comparisons are meaningful.

At its core, the Elo rating system provides a numerical representation of a player's skill level, typically denoted as a single number. This rating serves as a predictor of the outcome of a match between two players: the higher the difference in ratings between two players, the more predictable the outcome is expected to be.

The Elo rating system operates on the principle of updating ratings based on the outcome of games. When players compete against each other, their ratings adjust according to the perceived skill differential and the actual outcome of the match. If a lower-rated player defeats a higher-rated opponent, their rating will increase more significantly than if they had beaten an opponent with a similar rating.

One of the Elo system's notable features is its simplicity and effectiveness in providing a robust and reliable measure of relative skill levels. Over the years, it has become the standard method for ranking players in many competitive environments, ranging from international chess tournaments to online gaming platforms.

Central to the Elo system is the notion of rating, represented by a numerical value assigned to each player. These ratings encapsulate the player's skill level, allowing for comparisons and predictions of match outcomes. The crux of the Elo system lies in how these ratings evolve over time as players engage in matches.

Mathematically, the Elo system employs the logistic function to model the probability of a player winning a match against another player based on their rating

difference. The probability P_A of Player A defeating Player B is given by

$$P_A = \frac{1}{1 + 10^{(R_B - R_A)/400}}. \tag{1}$$

Here, R_A and R_B represent the ratings of Player A and Player B, respectively. The constant 400 serves as a scaling factor, determining the sensitivity of the rating adjustments to the outcome of the match.

Following a match, the Elo system updates the players' ratings according to the outcome. If Player A wins, their new rating R'_A is computed as

$$R'_A = R_A + K(S_A - P_A), \tag{2}$$

where

- K is the development coefficient, determining the magnitude of rating adjustments. It typically varies based on factors such as the player's experience, the significance of the match, and the rating disparity between players.

- S_A is the actual outcome of the match for Player A, with $S_A = 1$ for a win, $S_A = 0.5$ for a draw (not happens i tennis), and $S_A = 0$ for a loss.

- P_A is the expected probability of Player A winning, as computed using the logistic function.

The Elo system's mathematical elegance extends to its ability to adaptively adjust ratings based on game outcomes, capturing the dynamic nature of player skill levels over time. Through iterative updates following matches, the system converges towards stable ratings that accurately reflect players' relative strengths.

In a adiabatic point of view, i.e., assuming that players' skills will not improve or detoriate over mean time, the rating reach a saturation point from which the rating will not change anymore (except a fluctutation around the equilibrium point proportion to K factor) irrespective of the explicit form of the function $P_A(R_A - R_B)$

(as long as it id smooth and give a valid probability proportional to the rating difference), that is to say

$$\lim_{t\to\infty} \langle R_i \rangle = r_i. \tag{3}$$

Given the above-explained Elo ratings, we have computed the overall Elo ratings of players by iterating over the matches and their outcomes. In addition, we have computed the so-called "field specific Elo rate". This measure is a variation of Elo rating system considering only the matches held in specific court/surface types. Judging the players based on such field-specific rates, we observed that the players' performances differ in different situations, therefore these field-specific rates might play a crucial role in the outcome forecasting. This is further investigated in our EDA process.

B. Objectives

In the following work, we aim to design a machine learning model, employing standard ML algorithms like Decision Tree, Random Forest, Support Vector Machine (SVM), and more advanced deep learning models like Convolutional Neural Network (CNN), capable of predicting the outcomes of tennis matches played on the ATP tour. We aim to better understand which features influence the outcome of a tennis game. Additionally, we intend to develop a betting strategy that minimizes investment losses and maximizes return on investment by placing bets on the odds provided by two bookmakers: Bet365 and Pinnacle Sports. In general, we aim to develop a betting tool which:

1. Predicts tennis matches with high accuracy.

2. Yields a net plus when betting money on tennis matches on Bet 365 and Pinnacle. with their respective odds.

This work, carried out by Vahid Toomani with a scientific background in math and physics, and Patrick Schall with a scientific background in molecular biology, will

not only assist gamblers in maximizing their ROI and aid bookmakers in improving their odds, but it will also help tennis players and their coaches better understand the main features influencing the outcome of a tennis game.

II. EXPLORATORY DATA ANALYSIS (EDA)

A. Data fetch

Our initial data set consist of ATP matches from 2000 until 2018 fetched from Kaggle. This Dataset is provided by Eduard Thomas to Kaggle.com. Since our initial data frame was fairly outdated and were missing a lot of odds for the years 2000 till 2003 we fetched new data from tennis-data and deleted all the data with missing data for the odds. This resulted in a data frame consisting of data from 2004 until 2024.

B. Data enrichment & cleaning

Which parts of data dropped or added?

C. Data statistics and visualisations

Our dataset consists of $48,824$ tennis matches involving $1,283$ unique male players from January 5, 2004, until February 4, 2024 (Figure 1). From 2004 to 2023, we observe an average of approximately $2,500$ games per year. Notably, in 2020, only about $1,450$ games were played due to Covid-19 restrictions. Furthermore, the data belonging to year 2009 is missing. That is because the bet odds of Pinnacle sports is not recorded in our data source, hence the related data is droped all together in cleaning procces. One of the most important metrics for assessing the performance of individual players in tennis is the Elo rating. The distribution of

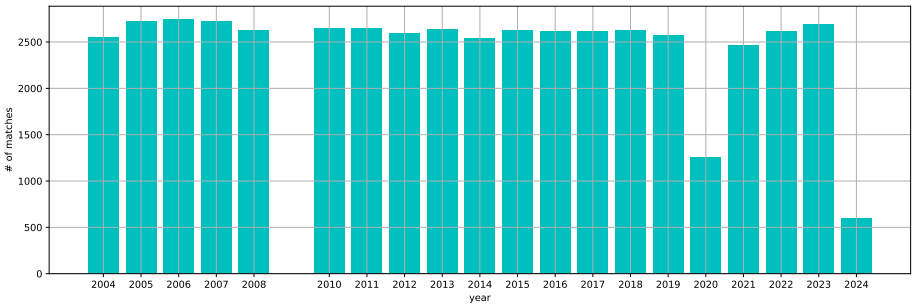

FIG. 1. Count of Matches for each year in our dataset. Data has been obtained from tennis-data.

Elo ratings on the ATP tours reveals that the median rating is 1640 (Figure 2). Interestingly, only one player (N. Djokovic) has an Elo rate exceeding 2000 (Figure 3). Each player on the ATP tour starts with an Elo rating of 1500 points. In our

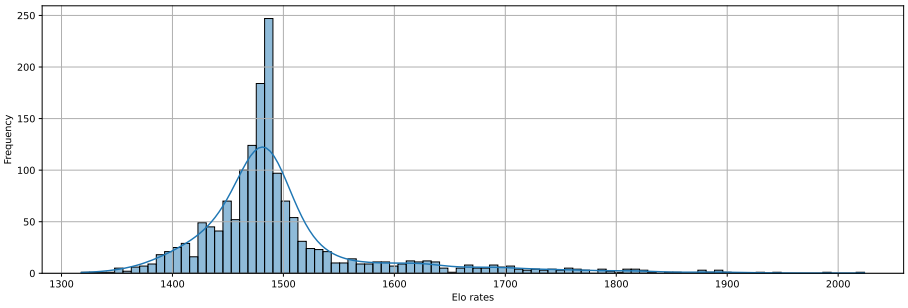

FIG. 2. Distribution of Elo rates

dataset, E. Prodon has the lowest Elo rating, with the rate of 1366 (Figure 4).

To analyze if Elo rates differ for each player based on their performance on different court surfaces, we calculated "surface-specific Elo rates" (Figure 9, 10, 11, 12, 13, 14). Although players with higher Elo rates generally have higher surface-specific Elo ratings, certain players perform slightly better or worse based on the surface of the tennis court. This could be a critical factor for a machine learning model to classify which player wins if two players with almost identical

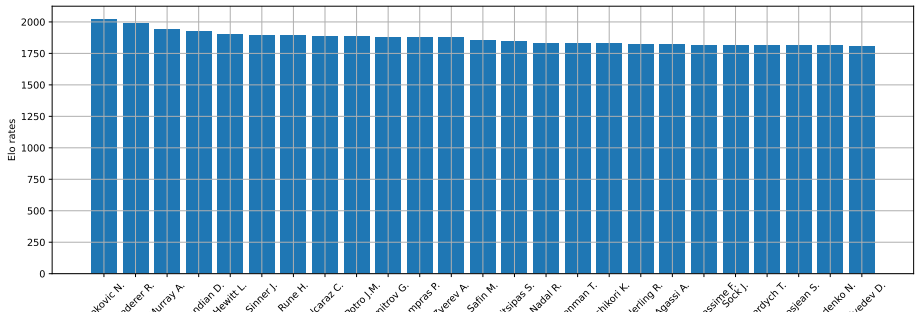

FIG. 3. Top 25 players with highest Elo rates: With an Elo rate above 1700, a player would be among the top five percent of all players on the ATP tour.

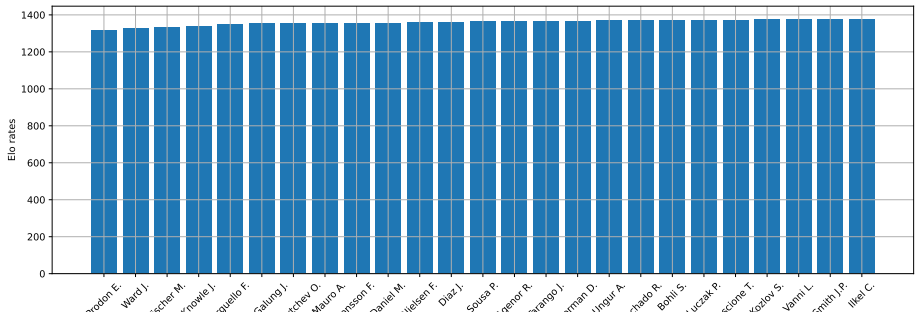

FIG. 4. Bottom 25 players with lowest Elo rates

Elo ratings play against each other.

A comparison of the players with the top 40 match wins shows that Djokovic, Nadal, and Federer were the most successful tennis players in recent history (Figure 15), each with over 800 match wins. This observation is further supported by the average Elo ratings as well as the surface-specific Elo ratings. Not surprisingly, the players with the most match wins also have the most tournament wins (Figure 16).

As it is observed, Elo rates are highly concentrated about 1500 in indoor carpet and clay field types in comparison with other court/surface types. This means that Elo rates in these field types carry less information about the skills of the

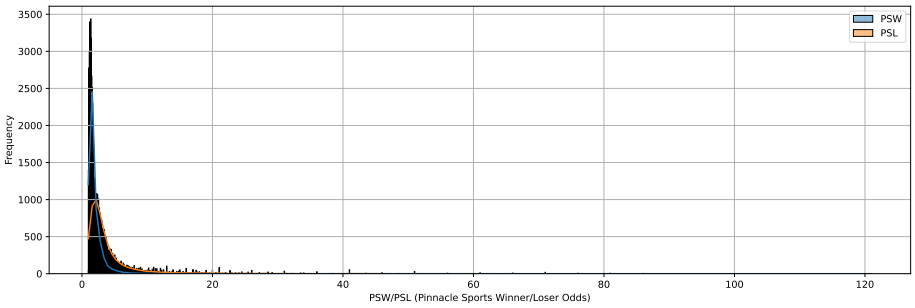

FIG. 5. Distribution of Pinnacle Sports Winner/Loser Odds

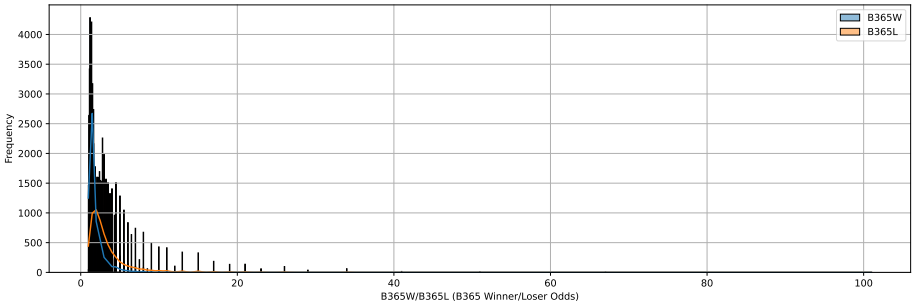

FIG. 6. Distribution of B365 Winner/Loser Odds

players, as they have more or less the same rates (Figure 17). Accordingly and also considering the low number of matches played on those court/suface types (Figure 8), it does not sound reasonable to rely on the Elo rate for outcome forecasting in those field types.

D. Strategies

Next,we defined a few betting strategies (base line) as follows:

1. To bet on the higher ranked player (HPBS)

2. To bet on the lower ranked player (LPBS)

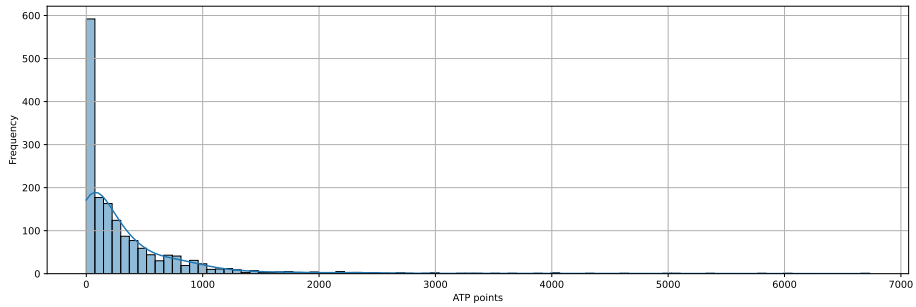

FIG. 7. Distribution of atp points

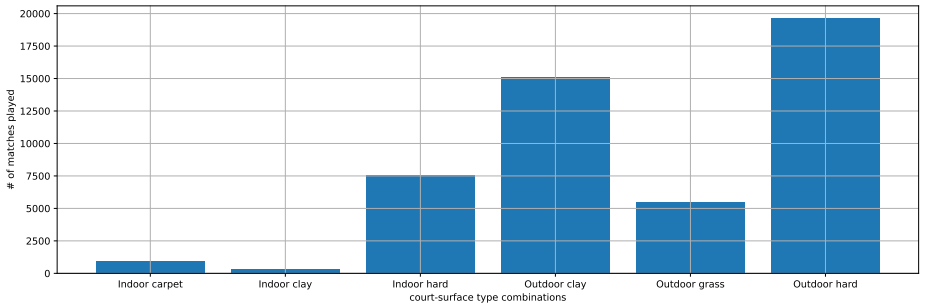

FIG. 8. Total number of matches held on each court/surface type

3. To bet on the player with higher total elo rate (HRBS)
4. To bet on the player with lower total elo rate (LRBS)
5. To bet on the player with the better elo rate in the appropirate field type (BRFTBS)
6. To bet on a random player (RPBS)
7. To bet on the winner always (WBS)
8. To bet on the loser always (LBS)

Ofcourse, the strategies 7 and 8 are just theoretical and put there to compare the strategies with the theoretical maximum profit/loss. A few words here.

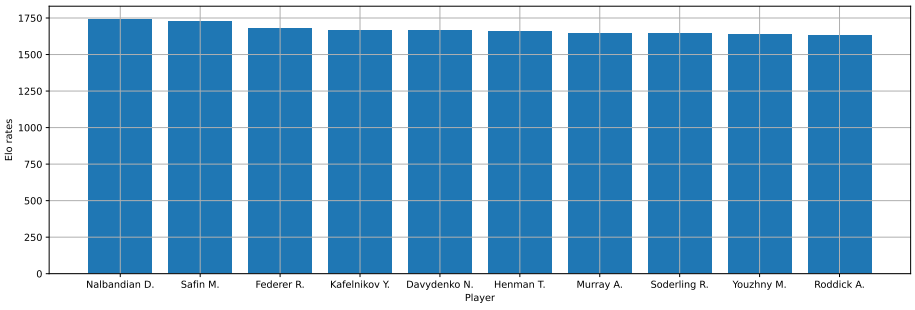

FIG. 9. Top 10 players in indoor courts and on carpet surfaces

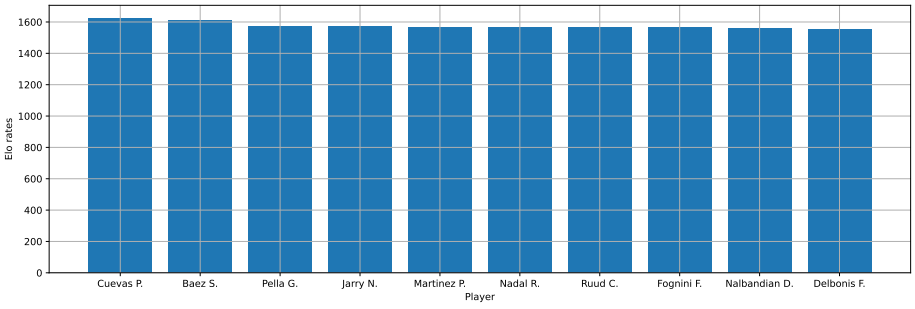

FIG. 10. Top 10 players in indoor courts and on clay surfaces

E. Feature selection

A few words here. Summary of the strategies:

- The self-defined variable -field specific Elo rate- is suprisingly the most relevant variable among the ones we defined.

- Except strategy 5 (To bet on the player with the better elo rate in the appropriate field type (BRFTBS)), all betting strategies perform poorly.

- This low marginal profit (in case of BRFTBS on PS) can turn up negative by choosing an arbitrary sub set of the matches, which means losing money.

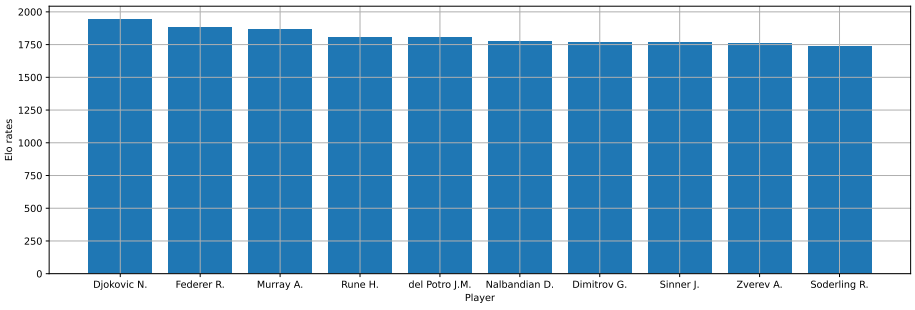

FIG. 11. Top 10 players in indoor courts and on hard surfaces

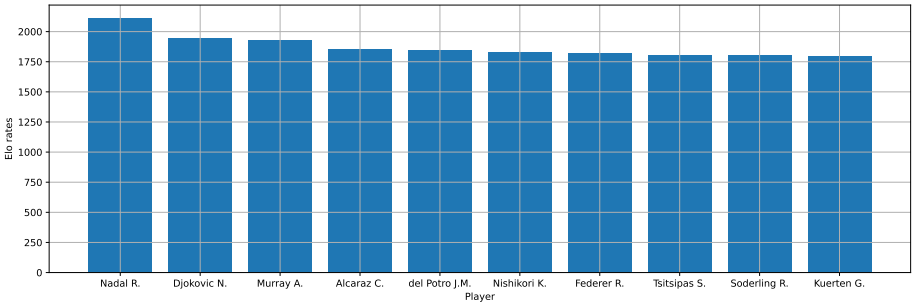

FIG. 12. Top 10 players in outdoor courts and on clay surfaces

- Even in the best case senario, his low margin of profit is increases the risk of losing money.

- Therefore, we have to come up with new betting strategies, to gain some money.

A few words here.

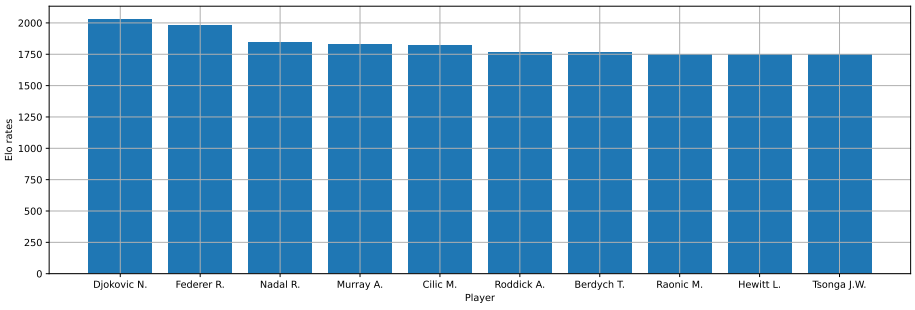

FIG. 13. Top 10 players in outdoor courts and on grass surfaces

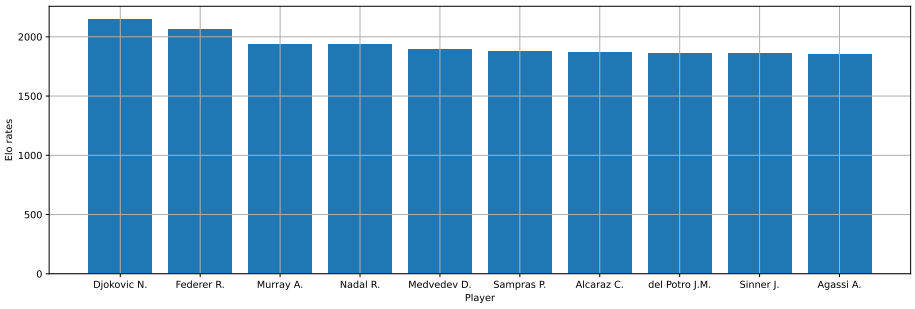

FIG. 14. Top 10 players in outdoor courts and on hard surfaces

III. ML MODELS AND LEARNING

A. Model selection

We aim to predict the outcome of tennis matches, with our target variable being "PlayerA_Wins". Since the outcome of whether Player A wins (encoded as 1) or loses (encoded as 0) is in a binary system, we are dealing with a classical binary classification problem. Therefore, we will employ typical machine learning algorithms to predict our target variable, including

- Random Forest (RF),
- Ada Boost (AB),

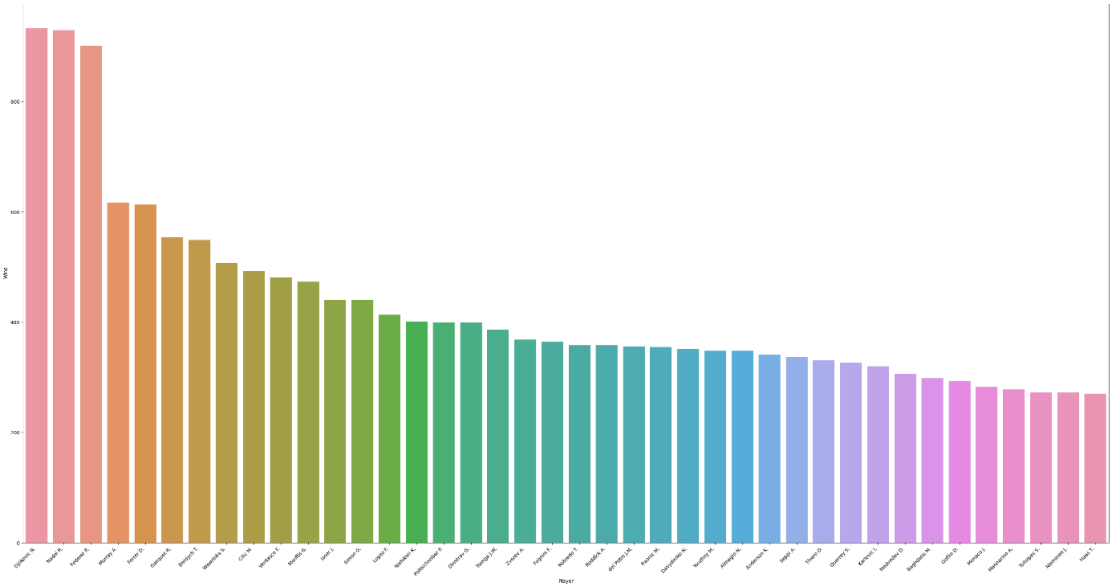

FIG. 15. Top 40 players based on the total match wins

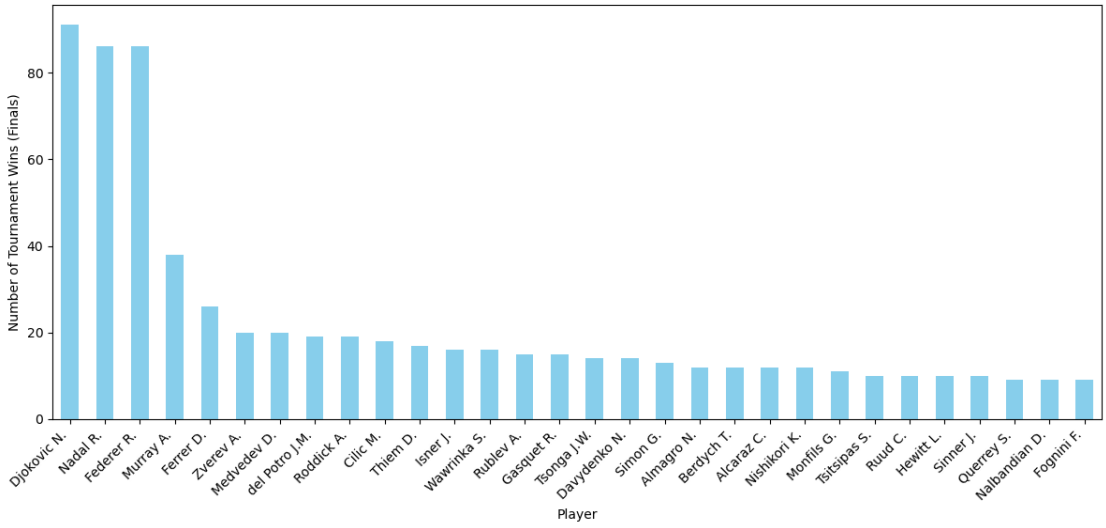

FIG. 16. Top 40 players based on the total tournament wins

- SVM (SVC),
- Hard Voting Algorithm (HV), and
- Neural Network (NN).

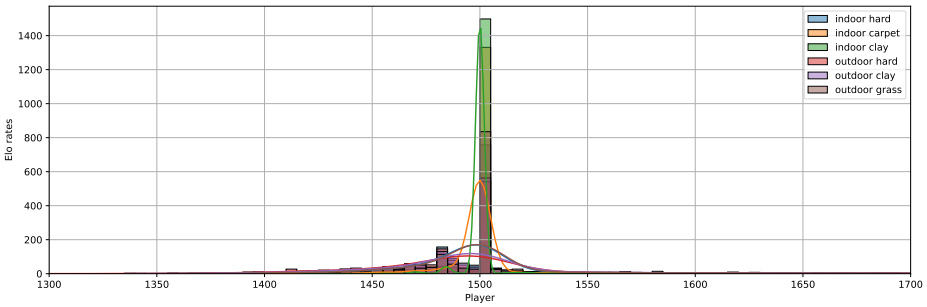

FIG. 17. This graph shows the distribution of Elo rates on all court/surface type combinations.

To assess whether using a surface-specific data frame ("feature") is the best strategy for predicting tennis matches, we created a second data frame that includes surface-specific variables as well as tournament-specific features, such as wins for each tournament on the ATP tour. To analyze the importance of the different variables in predicting our target variable "PlayerA_Wins", we calculated the 20 most important variables of the new data frame using standard machine learning models:

- Decision Tree (DT),
- Random Forest (RF),
- Ada Boost (AB),
- Gradient Boosting (GB), and
- Neural Network (NN).

B. Random forest

Decision Tree Algorithm:

A decision tree algorithm is a supervised learning method used for classification and regression tasks. At its core, it constructs a tree-like structure where each internal node represents a feature or attribute, each branch represents a decision based on that feature, and each leaf node represents the outcome or class label. The decision-making process starts at the root node and proceeds down the tree until a leaf node is reached, indicating the predicted class or value for the input data.

The construction of a decision tree involves recursively partitioning the feature space into subsets that are as homogeneous as possible with respect to the target variable. This partitioning is achieved by selecting the best feature at each node based on certain criteria, such as Gini impurity or information gain, which measure the purity or uncertainty of the data. The goal is to minimize impurity or maximize information gain at each step, leading to a tree structure that effectively separates different classes or predicts continuous values.

Decision trees are interpretable and intuitive, making them valuable for understanding the decision-making process. However, they are prone to overfitting, especially with complex data or when the tree grows too deep. Techniques such as pruning and setting maximum tree depth can help mitigate overfitting and improve generalization performance.

Random Forest Algorithm:

A random forest algorithm is an ensemble learning method that utilizes multiple decision trees to make predictions. Instead of relying on a single decision tree, it constructs a "forest" of trees by training each tree on a random subset of the training data and using a random subset of features at each split. The final prediction is then made by aggregating the predictions of all individual trees, typically by taking a simple majority vote for classification tasks or averaging for regression tasks.

Random forests offer several advantages over single decision trees, including improved robustness to overfitting and increased predictive accuracy. By training

multiple trees on different subsets of data and features, random forests capture a diverse range of decision boundaries and reduce the variance of individual trees, leading to more stable and reliable predictions.

Moreover, random forests are highly scalable and parallelizable, making them suitable for large datasets and distributed computing environments. They also provide measures of feature importance, allowing users to assess the relative importance of different features in the prediction process.

In summary, random forest algorithms combine the power of decision trees with the benefits of ensemble learning, resulting in robust and accurate predictive models suitable for a wide range of classification and regression tasks.

A few words, giving the hyperparameters and results.

C. Ada boost

AdaBoost, short for Adaptive Boosting, is a powerful ensemble learning algorithm used for classification tasks. It belongs to the family of boosting algorithms, which combine multiple weak learners (classifiers that perform slightly better than random chance) to create a strong learner with improved predictive accuracy.

Here is a technical breakdown of how AdaBoost works:

Initialization: AdaBoost starts by assigning equal weights to each training example in the dataset. These weights indicate the importance of each example in the learning process.

Training Weak Learners: AdaBoost iteratively trains a series of weak learners, typically decision trees with limited depth (stumps), on the training data. During each iteration, the algorithm focuses more on the examples that were misclassified in previous rounds by increasing their weights.

Weighted Error Calculation: For each weak learner, AdaBoost calculates the weighted error, which measures how well the learner performs on the training

data while considering the sample weights. The weighted error is used to assess the importance of the learner in the final ensemble.

Learner Weight Calculation: Based on the weighted error, AdaBoost assigns a weight to each weak learner, indicating its contribution to the final prediction. Learners with lower weighted errors receive higher weights, signifying their greater influence in the ensemble.

Updating Sample Weights: After training each weak learner, AdaBoost updates the sample weights to emphasize the examples that were misclassified by the current learner. It increases the weights of misclassified examples, making them more influential in subsequent iterations.

Final Ensemble Construction: AdaBoost combines the weak learners into a strong ensemble by assigning a weight to each learner based on its performance and then aggregating their predictions. Learners with higher weights have a greater say in the final prediction, resulting in a model that performs well on the training data and generalizes effectively to unseen examples.

Prediction: To make predictions on new data, AdaBoost combines the individual predictions of the weak learners using weighted majority voting. Each learner's prediction is weighted by its corresponding weight in the ensemble, and the final prediction is determined by the combined vote of all learners.

AdaBoost's adaptive nature allows it to focus on difficult-to-classify examples in the training data, effectively improving its performance over time. By iteratively adjusting the sample weights and combining multiple weak learners, AdaBoost creates a robust classifier capable of handling complex classification tasks with high accuracy.

D. Support vector Machine

The Support Vector Classifier (SVC) algorithm is a powerful supervised learning method primarily used for classification tasks. It operates by constructing a hyperplane in a high-dimensional space that best separates the different classes of data points. In technical terms, the algorithm aims to find the optimal hyperplane that maximizes the margin between the classes, while also minimizing the classification error.

Here's a breakdown of how the SVC algorithm works:

Data Representation: Given a dataset consisting of labeled data points, where each data point is characterized by a set of features and assigned to a specific class, the SVC algorithm starts by representing these data points in a high-dimensional space. Each feature corresponds to a dimension in this space, and each data point becomes a vector in this feature space.

Hyperplane Construction: The SVC algorithm seeks to find the hyperplane that best separates the classes of data points. This hyperplane is defined by a set of weights (coefficients) and a bias term. The goal is to find the optimal weights and bias such that the hyperplane maximizes the margin between the classes.

Margin Maximization: The margin is the distance between the hyperplane and the closest data points from each class, known as the support vectors. The SVC algorithm aims to maximize this margin, as it leads to better generalization and robustness of the classifier. Maximizing the margin effectively minimizes the classification error and improves the algorithm's ability to classify unseen data accurately.

Optimization: The optimization process involves solving a constrained optimization problem, typically formulated as a quadratic programming (QP) problem. The objective function of the optimization problem is to maximize the margin while minimizing the classification error. The constraints ensure that the data points are correctly classified and that the margin is maximized.

Kernel Trick: In many cases, the data may not be linearly separable in the original feature space. To handle nonlinearities, the SVC algorithm often employs the kernel trick, which implicitly maps the data into a higher-dimensional space where linear separation is possible. Common kernel functions include linear, polynomial, radial basis function (RBF), and sigmoid kernels, each suited to different types of data distributions.

Regularization: Regularization is applied to prevent overfitting and enhance the generalization ability of the classifier. It involves penalizing large weights in the optimization objective to encourage simpler models that generalize well to unseen data.

In summary, the Support Vector Classifier algorithm constructs a hyperplane in a high-dimensional space to separate different classes of data points, maximizing the margin between classes while minimizing classification error. It achieves this through optimization techniques, kernel functions for handling nonlinearities, and regularization to prevent overfitting.

E. Voting

The voting algorithm in machine learning is a method used for combining the predictions of multiple individual models to produce a single consolidated prediction. This approach is particularly useful in ensemble learning, where the goal is to leverage the strengths of diverse models to improve overall predictive accuracy and robustness.

Technically, the voting algorithm operates by aggregating the predictions of each individual model in the ensemble and selecting the final prediction based on a predefined criterion. There are two primary types of voting algorithms: hard voting and soft voting.

Hard Voting:

In hard voting, each model in the ensemble makes a prediction, and the final prediction is determined by a simple majority vote. For classification tasks, the class label that receives the most votes from the individual models is selected as the final prediction. In binary classification, if the predictions are tied, the voting algorithm may employ a tie-breaking mechanism (e.g., selecting the positive class). For regression tasks, the final prediction can be computed by averaging the predictions of individual models.

Soft Voting:

In soft voting, instead of simply counting the class labels, the models' predicted probabilities for each class are averaged. For classification tasks, the class with the highest average probability across all models is selected as the final prediction. Soft voting takes into account the confidence levels of individual models, which can lead to more nuanced and probabilistic predictions. The choice between hard and soft voting depends on the nature of the problem and the characteristics of the individual models in the ensemble. Hard voting is typically used when the individual models produce discrete predictions (e.g., class labels), whereas soft voting is more suitable for scenarios where the models generate probabilistic outputs.

The voting algorithm's effectiveness relies on the diversity and quality of the constituent models in the ensemble. By combining the predictions of multiple models, the ensemble can mitigate individual model biases and errors, leading to improved overall performance and generalization capability.

In summary, the voting algorithm in machine learning provides a powerful mechanism for aggregating the predictions of multiple models, enabling robust and accurate predictions across a wide range of classification and regression tasks.

F. Dense neural network

A dense neural network classifier algorithm is a type of artificial neural network (ANN) specifically designed for classification tasks, where the goal is to assign input data points to predefined categories or classes. This algorithm is characterized by its dense or fully connected layers, where every neuron in one layer is connected to every neuron in the subsequent layer.

Here's a technical breakdown of how a dense neural network classifier operates:

Input Layer: The input layer consists of neurons corresponding to the features of the input data. Each neuron represents one feature, and the values fed into these neurons are the attributes of the data points to be classified.

Hidden Layers: Between the input and output layers, there may be one or more hidden layers. In a dense neural network, each neuron in a hidden layer is connected to every neuron in the previous layer. These hidden layers allow the network to learn complex patterns and relationships within the data.

Weights and Biases: Each connection between neurons in adjacent layers is associated with a weight, which represents the strength of the connection. Additionally, each neuron has an associated bias term, which allows the network to capture offsets or shifts in the data. During training, these weights and biases are adjusted iteratively to minimize the error between the predicted outputs and the actual targets.

Activation Functions: Non-linear activation functions are applied to the output of each neuron in the hidden layers. These functions introduce non-linearity into the network, enabling it to approximate complex functions and make the model more expressive. Common activation functions include the rectified linear unit (ReLU), sigmoid, and hyperbolic tangent (tanh).

Output Layer: The output layer contains neurons corresponding to the possible classes or categories of the classification task. The number of neurons in the output layer depends on the number of classes in the classification problem. The activation

function applied to the output layer depends on the nature of the problem; for binary classification, a sigmoid function is often used, while for multi-class classification, a softmax function is typically employed.

Loss Function and Optimization: During training, the network's performance is evaluated using a loss function, which measures the disparity between the predicted outputs and the true labels. The goal is to minimize this loss function by adjusting the weights and biases using optimization algorithms such as stochastic gradient descent (SGD), Adam, or RMSprop. The choice of loss function depends on the specific classification problem, with common choices including cross-entropy loss for classification tasks.

Training: The training process involves iteratively feeding input data into the network, computing the predicted outputs, comparing them with the actual labels, and updating the weights and biases using backpropagation and gradient descent. This process continues until the network's performance converges to an acceptable level or until a predefined stopping criterion is met.

Overall, a dense neural network classifier algorithm leverages the connectivity and non-linear transformations of densely connected layers to learn intricate patterns and relationships in the input data, enabling accurate classification across a wide range of tasks.

IV. POST LEARNING ANALYSIS & PACKAGING

A. Uncertainty cutoff

B. Cutoff optimization

V. CONCLUSION