AR Tennis App: Detailed Odds Calculation Methodology

Prepared by Grok, xAI

August 20, 2025

This document provides a comprehensive explanation of the odds calculation system used in the AR Tennis application, including mathematical formulas, implementation details, and edge case handling.

Contents

1 Introduction

The AR Tennis application, built using Streamlit, facilitates tennis community management by tracking matches, player rankings, and court bookings. A key feature is the calculation of win probabilities (odds) for doubles and singles matches, along with suggesting balanced team pairings for doubles. These odds are derived from historical match data, aiming to promote fair and competitive matchups within the Arabian Ranches tennis community.

This document details the methodology behind the odds calculations, focusing on the **Performance Score** that quantifies player skill, the process for computing team and individual odds, and how the system handles edge cases. The explanations reference the relevant functions in the application's codebase (`ar new odds algo.py`) for transparency.

2 Key Concepts

The odds calculation system relies on the following core components:

- **Performance Score**: A weighted, normalized score that quantifies a player's skill and experience based on historical match data. It combines Win Percentage, Average Game Difference, and Matches Played.
- **Win Probability (Odds)**: The percentage likelihood of a team or player winning a match, derived by comparing Performance Scores.
- **Balanced Pairing (Doubles Only)**: For doubles matches with four players, the system evaluates all possible team combinations to suggest the pairing with odds closest to 50/50.
- **Format-Specific Rankings**: Odds calculations use rankings derived from either doubles or singles matches to ensure relevance to the match type.

The calculations are implemented in the following functions:

- `_calculate_performance_score(player_stats, full_dataset)`: Computes the Performance Score for a player.
- `calculate_enhanced_doubles_odds(players, doubles_rank_df)`: Calculates odds and suggests pairings for doubles.
- `suggest_singles_odds(players, singles_rank_df)`: Calculates odds for singles matches.

3 Data Preparation

Before odds are calculated, the application prepares player rankings using the `calculate_rankings(matches_df` function. This process involves:

1. **Filtering Matches**: Matches are filtered by type (`Doubles` or `Singles`) to create format-specific datasets (`doubles_matches_df` or `singles_matches_df`).
2. **Computing Player Stats**: For each player (excluding "Visitor"), the function calculates:
   - **Points**: 3 points for a win, 1 for a loss, 1.5 for a tie.
   - **Win %**: $\frac{\text{Wins}}{\text{Matches Played}} \times 100$.
   - **Matches Played**: Total matches played.
   - **Game Diff Avg**: Average game difference per match (positive for wins, negative for losses).

- **Games Won**: Total games won across all sets.
- **Cumulative Game Diff**: Sum of game differences across all sets.
- **Recent Trend**: Win/loss record for the last 5 matches.

3. **Generating Rankings**: A DataFrame (`rank_df`) is created, sorting players by Points, Win %, Game Diff Avg, Games Won, and Player name (alphabetically). Ranks are assigned with a trophy icon (1, 2, etc.).

The resulting `rank_df` contains columns like `Rank`, `Player`, `Points`, `Win %`, `Matches`, `Wins`, `Losses`, `Games Won`, `Game Diff Avg`, `Cumulative Game Diff`, and `Recent Trend`. Only `Win %`, `Game Diff Avg`, and `Matches` are used for odds calculations.

4 Calculating the Performance Score

The Performance Score is the foundation of the odds system, computed by `_calculate_performance_score(pl full_dataset)`. It normalizes three metrics and combines them with predefined weights.

4.1 Formula

The Performance Score is calculated as:

$$\text{Performance Score} = (w_{wp} \times wp_{norm}) + (w_{agd} \times agd_{norm}) + (w_{ef} \times ef_{norm})$$

Where:

- $w_{wp} = 0.50$: Weight for Win Percentage (emphasizes winning consistency).
- $w_{agd} = 0.35$: Weight for Average Game Difference (measures margin of victory/defeat).
- $w_{ef} = 0.15$: Weight for Experience Factor (rewards players with more matches).

4.2 Normalization Process

Each metric is normalized to a 0–1 scale relative to the entire dataset to ensure comparability.

4.2.1 Win Percentage Normalization (wp_{norm})

$$wp_{norm} = \frac{\text{Player's Win \%}}{\max(\text{Win \% across all players})}$$

- If $\max(\text{Win \%}) = 0$, $wp_{norm} = 0$ to avoid division by zero.
- This rewards players with a high win rate relative to the best performer.

4.2.2 Average Game Difference Normalization (agd_{norm})

$$agd_{norm} = \frac{\text{Player's Game Diff Avg} - \min(\text{Game Diff Avg across all players})}{\max(\text{Game Diff Avg}) - \min(\text{Game Diff Avg})}$$

- If $\max(\text{Game Diff Avg}) = \min(\text{Game Diff Avg})$, $agd_{norm} = 0.5$ to handle uniform datasets.
- Game Diff Avg reflects the average margin of victory or defeat per match, capturing dominance.

4.2.3 Experience Factor Normalization (ef_{norm})

$$ef_{norm} = \frac{\text{Player's Matches Played}}{\max(\text{Matches Played across all players})}$$

- If $\max(\text{Matches Played}) = 0$, $ef_{norm} = 0$.
- This accounts for experience, assuming more matches correlate with better skill exposure.

4.3 Edge Cases

- **No Match History**: If a player has no matches in the relevant format (e.g., no doubles matches), their stats (`Win %`, `Game Diff Avg`, `Matches`) are 0, resulting in a Performance Score of 0.
- **Uniform Dataset**: If all players have identical values (e.g., all `Win %` = 0), normalization defaults to safe values (0 for `Win %` and `Matches`, 0.5 for `Game Diff Avg`).
- **Visitor Players**: The "Visitor" player is excluded from rankings, so their Performance Score is implicitly 0.

5 Calculating Odds for Doubles

Doubles odds are computed by `calculate_enhanced_doubles_odds(players, doubles_rank_df)`, which also suggests the most balanced team pairing. The process is as follows:

5.1 Process

1. **Gather Player Scores**: Compute the Performance Score for each of the four selected players using doubles-specific rankings (`doubles_rank_df`). If a player is not in the rankings (no doubles history), their score is 0.
2. **Generate All Possible Pairings**: Use `itertools.combinations` to generate all ways to select two players for Team 1 (the remaining two form Team 2). This results in three unique pairings, as team order is irrelevant.
3. **Evaluate Each Pairing**:
   - **Team Scores**:

     $$\text{Team 1 Score} = \text{Performance Score of Player 1} + \text{Performance Score of Player 2}$$

     $$\text{Team 2 Score} = \text{Performance Score of Player 3} + \text{Performance Score of Player 4}$$

     $$\text{Total Score} = \text{Team 1 Score} + \text{Team 2 Score}$$

   - **Win Probabilities**:

     $$\text{Team 1 Odds (\%)} = \left(\frac{\text{Team 1 Score}}{\text{Total Score}}\right) \times 100$$

     $$\text{Team 2 Odds (\%)} = 100 - \text{Team 1 Odds}$$

     If Total Score $= 0$ (e.g., all players have no history), odds default to 50% each.

   - **Balance Score**: To find the most balanced pairing:

     $$\text{Balance Score} = |\text{Team 1 Odds} - 50|$$

     A lower balance score indicates a more even matchup.

4. **Select Best Pairing**: Choose the pairing with the minimum Balance Score. If multiple pairings tie, the first encountered is selected. The function returns a formatted string (e.g., "Team 1: PlayerA & PlayerB vs Team 2: PlayerC & PlayerD") and the odds.

5.2 Edge Cases

- **Fewer than Four Players**: Returns an error message: "Please select four players with doubles match history."
- **No History for All Players**: If all Performance Scores are 0, odds are set to 50/50.
- **Tied Balance Scores**: The first pairing with the minimum balance score is chosen arbitrarily.

The wrapper function `suggest_balanced_pairing(players, rank_df)` formats the output for display in the app.

6 Calculating Odds for Singles

Singles odds are computed by `suggest_singles_odds(players, singles_rank_df)`, which is simpler due to the lack of team pairings.

6.1 Process

1. **Gather Player Scores**: Compute the Performance Score for each of the two players using singles-specific rankings (`singles_rank_df`).
2. **Compute Odds**:

$$\text{Player 1 Score} = \text{Performance Score of Player 1}$$

$$\text{Player 2 Score} = \text{Performance Score of Player 2}$$

$$\text{Total Score} = \text{Player 1 Score} + \text{Player 2 Score}$$

$$\text{Player 1 Odds (\%)} = \left(\frac{\text{Player 1 Score}}{\text{Total Score}} \right) \times 100$$

$$\text{Player 2 Odds (\%)} = 100 - \text{Player 1 Odds}$$

   If Total Score $= 0$, odds default to 50/50.

6.2 Edge Cases

- **Fewer than Two Players**: Not applicable, as the app's logic ensures two players are selected for singles.
- **No History**: If both players have no singles matches, odds are 50/50.

7 Integration in the App

The odds calculations are integrated into the "Upcoming Bookings" section (Tab 4) of the AR Tennis app, displayed for bookings with sufficient players:

- **Doubles**: Requires four players. Displays "Suggested Pairing: Team1 (X.X%) vs Team2 (Y.Y%)" using `suggest_balanced_pairing`.
- **Singles**: Requires two players. Displays "Odds: Player1 (X.X%) vs Player2 (Y.Y%)" using `suggest_singles_odds`.

The rankings are computed on-the-fly:

- `doubles_matches_df = st.session_state.matches_df[st.session_state.matches_df == 'Doubles']`
- `singles_matches_df = st.session_state.matches_df[st.session_state.matches_df == 'Singles']`
- `doubles_rank_df, _ = calculate_rankings(doubles_matches_df)`
- `singles_rank_df, _ = calculate_rankings(singles_matches_df)`

If an error occurs (e.g., empty rankings), a warning is displayed, and no odds are shown.

8 Limitations and Assumptions

8.1 Assumptions

- **Team Performance**: In doubles, team performance is the sum of individual Performance Scores, ignoring potential synergy or partner-specific dynamics (though `partner_stats` exists for other purposes).
- **Format-Specific Data**: Calculations assume doubles rankings for doubles matches and singles rankings for singles matches, ensuring relevance but requiring sufficient match history.
- **Linear Probability**: Win probabilities are derived linearly from Performance Scores, assuming relative skill differences directly translate to win likelihood.

8.2 Limitations

- **Small Datasets**: With few players or matches, normalization may overemphasize outliers (e.g., a single strong performance).
- **No Recency Weighting**: Older matches have equal weight to recent ones, potentially missing current form.
- **No Head-to-Head Data**: Direct matchups between players are not factored into odds.
- **Visitor Players**: Always assigned a Performance Score of 0, treating them as unknowns, which may skew odds.
- **Simplified Doubles Model**: The sum of individual scores may not capture team chemistry or matchup-specific factors.

8.3 Potential Improvements

- **Partner Synergy**: Incorporate `partner_stats` (win/loss records with specific partners) into doubles odds calculations.
- **Recency Weighting**: Apply exponential decay to older matches to emphasize recent performance.
- **Advanced Models**: Use logistic regression or machine learning to model win probabilities, incorporating more variables (e.g., court surface, head-to-head).
- **Dynamic Weighting**: Adjust weights (w_{wp}, w_{agd}, w_{ef}) based on dataset size or match context.

9 Conclusion

The AR Tennis app's odds calculation system provides a data-driven approach to estimating win probabilities and suggesting balanced doubles pairings. By leveraging a weighted Performance Score based on Win Percentage, Average Game Difference, and Matches Played, the system ensures fair and competitive matchups. While robust, it has limitations due to its reliance on historical data and simplified assumptions about team dynamics. Future enhancements could incorporate more sophisticated models and additional factors to improve accuracy. For implementation details, refer to the functions under `# START: NEW COMPLEX ODDS CALCULATION FUNCTIONS` in `ar new odds algo.py`.