Centrality and Performance Analysis in Chess Player Network: Correlations with Win Rates and Strategic Influence

Introduction

The intriguing world of competitive chess is not merely a battleground of individual prowess but also a complex network of interactions, where the strategic positioning of players within this network potentially influences their success and resilience. This project aims to dissect these dynamics by employing centrality measures to understand how a player's network position correlates with their game performance, particularly focusing on win rates and strategic influence.

You can obtain the data set from kaggle- [Analyzed Chess Games Dataset (Lichess) 2024]

Objective

The primary goal is to explore the relationship between various centrality metrics within the chess player network—such as PageRank, betweenness centrality, and closeness centrality—and the performance outcomes of players. We hypothesize that players who are central in this network, as identified by these metrics, might exhibit higher win rates and more consistent performance.

Methodology

Data Collection and Preparation:

We utilize a dataset comprising game records from a popular online chess platform. Each record includes details such as player IDs, game outcomes, and ELO ratings.
Data is structured into a suitable format for analysis using the Rust programming language, leveraging its powerful data handling capabilities.
You can obtain the dataset from here -

Graph Construction

A directed graph is constructed where nodes represent players and edges represent games, with the direction pointing from the winner to the loser. This structure effectively captures the flow of victories and defeats among players.

Centrality Calculation

1. PageRank: Calculates the importance of each player based on their win/loss record, considering the quality of opponents. This reflects how often a player wins against other strong players.

2. Betweenness Centrality: Measures the frequency at which a player lies on the shortest path between other players, indicating their strategic role as a connector within the network. Players with high betweenness may influence the flow of competition significantly.

3. Closeness Centrality: Assesses how quickly a player can reach all other players in the network through direct or indirect paths. This metric reflects a player's accessibility and influence over the network's dynamics.

4. In-Degree Centrality: Counts the number of incoming edges to a node. For this network, it represents the number of losses a player has, as each incoming edge signifies a game lost to another player. Higher in-degree might indicate players who participate a lot but lose more frequently.

5. Out-Degree Centrality: Counts the number of outgoing edges from a node. This metric indicates the number of wins a player has, as each outgoing edge represents a victory over another player. Players with high out-degree are often more dominant in the network.

6. Unweighted Degree : This is a simple count of both incoming and outgoing edges for each node, providing a measure of overall activity or participation of a player in games, regardless of whether they resulted in a win or loss.

7. Weighted Betweenness and Closeness Centrality:

    a. Purpose: These metrics adapt traditional centrality measures by incorporating the weight of connections, such as the importance of matches or player strength differences. Weighted betweenness centrality identifies players who act as critical connectors in significant games, thus influencing the competitive flow, while weighted closeness centrality measures how centrally a player can access or be accessed within the network, factoring in the quality of their interactions.

b. Impact: Both metrics highlight key strategic players: those with high weighted betweenness are crucial in influencing match outcomes across strong competitors, and those with high weighted closeness are well-integrated, capable of impacting the network through significant connections.
c. Strategic Insight: These weighted measures provide a nuanced view of player roles and dynamics, aiding in strategic planning, training focuses, and understanding competitive interactions within the chess community.

Performance Metrics

1. Win Rates: Calculated by dividing the number of games won by the total number of games played. This provides a straightforward measure of a player's success rate.

2. ELO Rating Changes: Tracks the changes in players' ratings over time. ELO ratings are a standard measure of a player's competitive skill level, with changes indicating improvement or decline in performance relative to other players.'

3. Total Games Played: Sum of in-degree and out-degree centrality, offering a clear view of how active a player is within the chess network.

Implementation

Analysis.rs

This file serves as a core component of a chess game analysis system, utilizing data manipulation and network analysis techniques to examine player performances and interactions within a dataset of chess games. It employs libraries like petgraph for graph construction and analysis, and polars for handling data frames efficiently.

Function Descriptions in analysis.rs

1. read_games_from_dataframe(df: &DataFrame) -> Result<Vec<Game>, Box<dyn Error>>
    - Reads and parses game data from a dataframe to create a vector of Game structs containing detailed information about each game.

2. build_graph(games: &[Game]) -> DiGraph<String, u32>
    - Constructs a directed graph from the game data, where nodes represent players and directed edges represent game outcomes.

3. calculate_pagerank(graph: &DiGraph<String, u32>) -> HashMap<NodeIndex, f64>
   Computes the PageRank for each player in the graph to quantify their importance based on their victories and defeats.

4. calculate_betweenness_centrality(graph: &DiGraph<String, u32>) -> HashMap<NodeIndex, f64>
   - Calculates betweenness centrality for nodes, highlighting players who frequently connect other players through the shortest game paths.

5. calculate_closeness_centrality(graph: &DiGraph<String, u32>) -> HashMap<NodeIndex, f64>
   - Measures closeness centrality to determine how quickly and efficiently a player can reach other players in the network.

6. export_centrality_data(centrality_scores: &HashMap<NodeIndex, f64>, graph: &DiGraph<String, u32>, filepath: &str) -> Result<(), Box<dyn Error>>
   - Exports calculated centrality scores for each player to a specified file, providing a record of their network influence.

7. export_performance(performance: &HashMap<String, PlayerPerformance>, filepath: &str) -> Result<(), Box<dyn Error>>
   - Outputs the performance data for each player, detailing games played, won, lost, and drawn, along with their total rating change and win rate.

8. track_player_performance(games: &[Game]) -> HashMap<String, PlayerPerformance>
   - Tracks and aggregates individual player performance across all games, updating records for wins, losses, draws, and rating changes.

9. calculate_in_out_degree_centrality(graph: &DiGraph<String, u32>) -> HashMap<NodeIndex, (usize, usize)>
   - Computes the in-degree and out-degree centrality for each player, providing insights into how many games they've lost or won.

10. calculate_weighted_centrality(graph: &DiGraph<String, u32>) -> (HashMap<NodeIndex, f64>, HashMap<NodeIndex, f64>)
    - Calculates both weighted betweenness and closeness centrality, taking into account the strength and significance of game outcomes in the player network.

11. export_in_out_degree_centrality(in_out_degree_centrality: &HashMap<NodeIndex, (usize, usize)>, graph: &DiGraph<String, u32>, filepath: &str) -> Result<(), Box<dyn Error>>
    - Exports the in-degree and out-degree centrality measures for each node to a file, useful for further analysis or reporting.

12. export_weighted_centrality(weighted_betweenness: &HashMap<NodeIndex, f64>, weighted_closeness: &HashMap<NodeIndex, f64>, graph: &DiGraph<String, u32>, filepath: &str) -> Result<(), Box<dyn Error>>
    - Saves weighted centrality scores, providing a nuanced view of player centrality that factors in the importance of connections.

13. export_mean_mode_metrics(mean_mode_metrics: &HashMap<String, (f64, f64, f64, u32)>, filepath: &str) -> Result<(), Box<dyn Error>>
    - Outputs a composite view of players' win rates, draw rates, average rating changes, and total games played to a specified file.

14. calculate_mean_mode(games: &[Game]) -> HashMap<String, (f64, f64, f64, u32)>
    - Aggregates and computes mean statistics for player performance, combining win rates, draw rates, and rating changes into a comprehensive metric.

Column_info.rs

Thefile column_info.rs in this project is designed to read and analyze CSV files to extract and display information about the columns within these files. It is particularly useful for data validation and schema discovery in datasets.

Function Description in column_info.s

1. pub fn print_column_info(subset_files: &[&str]) -> Result<(), Box<dyn Error>>
    - This function accepts a slice of string slices (&[&str]), each representing a file path to a CSV file, and returns a Result which on success is empty (()) and on error contains a boxed (Box<dyn Error>) error.

2. File Handling:
    - Opens each file specified in the subset_files array using a BufReader for efficient reading. This method is beneficial for handling potentially large files by reading them into memory in manageable chunks.

3. Data Processing:
    - Reads the entire content of each file into a string and splits it into lines, effectively separating each row of the CSV file.

4. Header Parsing:
    - Extracts the first line from the file, assuming it contains the headers (column names) of the CSV, and splits this line by commas to individual column names.

5. Column Analysis:
    - Iterates over each column name extracted from the header:

6. Column Identification:
    - Outputs the name of each column.

7. Data Type Determination:
    - Analyzes the data in each column to infer the data type based on the content of each field. Checks if fields can be parsed as integers (i64) or floating points (f64), defaulting to string (String) if neither parsing attempt succeeds.

8. Null Check:
    - Determines if any field in the column is empty or contains only whitespace, indicating a null or missing value.

9. Output:
    - For each file, prints the detailed information about each column including its name, inferred data type, and whether it contains null values.

Data_distrubution.rs

The distribute_data function is designed to partition and distribute a dataset across multiple files for efficient data management and analysis.

1. Function:
    - Handles the distribution of data rows across multiple output files, ensuring an efficient and balanced division.

2. Initial Setup:
    - Checks for the presence of data and exits early if none exists, optimizing processing time and resource use.

3. Header Management:
    - Identifies and writes relevant column headers to each output file based on a predefined list of necessary columns, ensuring consistency across all files.

4. File Handling:
    - Creates buffered writers for each file path provided, facilitating fast and efficient data writing.

5. Data Partitioning:
    - Distributes data evenly across the specified files, adjusting for any imbalance in the division to ensure each file receives a fair share.

6. Data Writing:

- Selects and writes specific column data for each row into the appropriate files, maintaining data integrity and structure.

7. Completion Status:
    - Completes the data writing process, flushes all buffers, and outputs a status message indicating successful distribution.

Strategy_analysis.rs

The classify_games_by_eco function categorizes chess games based on their ECO (Encyclopaedia of Chess Openings) codes. It is designed to organize games into groups for easier analysis and retrieval based on opening strategies.

1. pub fn classify_games_by_eco(games: &[Game]) -> HashMap<String, Vec<&Game>>
    - Accepts a slice of Game objects and returns a HashMap where each key is an ECO code string and the value is a vector of references to games with that ECO code.

2. Game Grouping Logic:
    - Iterates over each game in the provided slice, retrieves the ECO code of the game, and groups games by their ECO codes in a hashmap. This allows for quick access and analysis of games based on their openings.

3. Data Structure and Efficiency:
    - Uses a HashMap to efficiently categorize games, ensuring that games can be quickly added and retrieved based on their ECO code. The use of references (&Game) minimizes memory usage by avoiding data duplication.

4. Utility and Application:
    - This function is particularly useful for chess analysts and enthusiasts who want to study game strategies or track performance variations across different types of openings.

5. Test Coverage
    - Included in the same module is a test (test_classify_games_by_eco) that validates the functionality of classify_games_by_eco:
        - Setup: Constructs a mock list of Game objects with specific ECO codes.
        - Execution: Applies the classify_games_by_eco function to the list.
        - Assertions: Checks that the function correctly groups games by ECO codes and accurately counts the number of games in each group.

Main.rs

The main.rs file serves as the entry point for a Rust application that handles the analysis and distribution of chess game data. It orchestrates the workflow by coordinating various modules and functions.

Function Descriptions in main.rs

1. Directory;
    - Determines the current working directory and prepares paths for input and output files.

2. Column Information Processing (column_info::print_column_info):
    - Extracts and displays column information from input CSV files, aiding in understanding data structure.

3. Data Combination (combine_csv_files):
    - Merges data from multiple CSV files into a single dataset while preserving the header from the first file.

4. Data Distribution (data_distribution::distribute_data):
    - Distributes combined data into multiple output files based on the provided header, ensuring even distribution.

5. Game Data Analysis (perform_game_data_analysis):
    - Analyzes distributed data files by performing statistical and graph-based analyses to derive insights into player performance and game strategies.

Functions

1. combine_csv_files:
   Combines multiple CSV files into one dataset, handling headers and ensuring all data is included from each file.

2. perform_game_data_analysis:
   Conducts comprehensive game data analysis, utilizing various metrics such as PageRank, betweenness centrality, closeness centrality, and player performance from analysis modules. Outputs results to specific files.

Output

The results of this project are detailed in various CSV files, each highlighting different facets of chess game analysis based on the implemented algorithms and methodologies. Here's a breakdown of each file and what it represents:

1. Centrality Scores (pr_scores.csv, btw_scores.csv, cls_scores.csv, in_out_degree.csv, weighted_centrality.csv):
    - These files list the top nodes (players) based on different centrality measures like PageRank, betweenness, closeness, in/out-degree, and weighted centrality. The files are formatted to pair each node with its corresponding centrality score, demonstrating the strategic positioning and influence of players within the chess network.

2. Player Performance (player_perf.csv):
    - This file captures the comprehensive performance metrics for players, including games played, wins, losses, draws, total rating changes, and win rates. It reflects the effectiveness of player strategies and overall competitive performance.

3. Game Analysis (mean_mode_metrics.csv):
    - This file provides an analysis of game outcomes related to various chess openings, classified by their ECO codes. It offers insights into the effectiveness of different openings in relation to player performance metrics.

Key Influencers

Through detailed analysis, several key influencers are identified based on their consistent presence across multiple centrality measures:

- Nodes with High Centrality in All Measures: Players appearing across all centrality measures are pivotal within the network, demonstrating high degrees of influence and strategic significance. They are likely the most skilled or strategically adept players, exhibiting a strong command over their games and opponents.

- Nodes with High Centrality in Specific Measures: Some players show high centrality in specific measures but not in others, indicating specialized roles within the network. For example, a player might be a key hub in direct interactions (high degree centrality) but less central in terms of controlling game flow (lower betweenness centrality).

Correlation Between Centrality and Performance

1. The analysis extends to examining the correlation between a player's centrality in the network and their game performance:

2. Higher Performance Among Highly Central Nodes: Players with higher centrality scores tend to have better performance metrics, suggesting a strong correlation between a player’s network influence and their game outcomes.

3. Trust and Influence: Players positioned centrally in the network generally receive higher trust scores from peers, reflecting their perceived reliability and strength in competitive settings.

Conclusion

The project successfully implements a robust system for analyzing chess game data using Rust, demonstrating the language's efficiency and capability in handling data-intensive operations. By dividing the task into specific modules and functions, the application not only ensures clean and maintainable code but also leverages Rust's performance advantages, particularly in concurrent data processing and safety.

The utilization of various data analysis techniques such as PageRank, betweenness centrality, and closeness centrality allows for a deep understanding of player dynamics and game strategies. This is particularly valuable for players and coaches looking to refine strategies or for enthusiasts interested in statistical trends within chess games.

Future improvements could include integrating more sophisticated machine learning models to predict game outcomes based on opening moves and player history. Additionally, expanding the dataset and incorporating real-time data analysis could provide more comprehensive insights and enhance the application's utility in professional chess training and competition analysis.

Overall, the project serves as a strong foundation for further development in chess analytics, offering scalable solutions that can be extended to other domains requiring detailed network analysis and performance metrics in sports or other fields.

Citations

Lichess Dataset - Analyzed Chess Games Dataset (Lichess) 2024
Chat GPT - https://chat.openai.com/
Towards Data Science. "Notes on Graph Theory: Centrality Measurements."
Petgraph Library Documentation: https://docs.rs/petgraph

Rust Programming Language Official Documentation: "Rust programming language book".
Serde Serialization Framework: https://serde.rs
Polars Data Frame Library(User Guide):
https://github.com/pola-rs/polars-book/tree/main/docs/src/rust/getting-started
CSV Manipulation in Rust: https://docs.rs/csv.
Simple-pagerank Crate: https://crates.io/crates/simple-pagerank
RustWorkx-core for Network Analysis: https://docs.rs/rustworkx-core
W3schools(basic commands):https://www.tutorialspoint.com/rust/rust_slices.htm