# Project Report: Matchmaking System for Age of Empires IV

## Overview

This project implements a matchmaking system for **Age of Empires IV** (AoE4) using **machine learning (ML)** techniques to provide fair and engaging player experience. By leveraging advanced ML models, the system predicts match outcomes and clusters players based on their performance metrics and in-game characteristics. The primary goal is to create a system that enhances gameplay by ensuring balanced and competitive matches. Age of Empires IV, a real-time strategy game, requires players to manage resources, build civilizations, and compete in dynamic battles. Matchmaking plays a critical role in maintaining player retention and satisfaction by ensuring fair challenges based on skill levels.

This report outlines the methodology, datasets used, feature engineering strategies, model training processes, and evaluation results, demonstrating the system's potential for real-world application. Additionally, it highlights the importance of integrating predictive analytics with clustering techniques to address the complexities of skill-based matchmaking in competitive gaming environments. By combining these approaches, the project aims to improve the matchmaking experience for players across all skill levels.

---

## Objectives

1. **Predict Match Outcomes:** Train a classification model to predict match outcomes between players.
2. **Cluster Players:** Group players into clusters based on performance metrics to create balanced matchmaking.
3. **Matchmaking API:** Design a matchmaking function and expose it through a simple API.
4. **Frontend Application:** Develop a user-friendly web interface to display matchmaking results and player statistics in an interactive and visual manner.

---

## Dataset Description

- Two datasets were used for this project and these datasets were originally retrieved from the AoE4World:
  https://aoe4world.com/dumps

1. **Games Dataset** (JSON):
    - Contains details of ranked games, including player performance, civilizations used, match results, and more.
    - Example structure:

```
{
 "game_id": 137248000,
 "started_at": "2024-07-18T18:04:10.000Z",
 "finished_at": "2024-07-18T18:34:00.000Z",
 "teams": [
  [{"profile_id": 8313452, "result": "loss", "mmr": 992}],
  [{"profile_id": 14416582, "result": "win", "mmr": 984}]
 ]
}
```

2. **Leaderboard Dataset** (CSV):
    - Includes player rankings, ELO ratings, win counts, and other player-specific metrics.

---

**Data Preprocessing**

**Steps:**

1. **Flatten Game Data:**
    - Extract team and player information into a structured format.
    - Resulting structure:

      **profile_id game_id mmr civilization result**

2. **Merge Leaderboard Data:**
    - Combine leaderboard features (e.g., rating, win_rate) with game-specific features.

3. **Handle Missing Data:**
    - Imputed missing MMR and win rates with default values.
4. **Derived Features:**
    - win_rate: Calculated as wins_count / games_count.
    - rating_diff: Difference in ratings between players in a match.
        1. Aggregated statistics:
           avg_mmr_diff_10: average MMR difference in the last 10 games
        2. avg_mmr_diff_25: average MMR difference in the last 25 games
        3. avg_mmr_diff_50: average MMR difference in the last 50 games
        4. avg_mmr_diff_75: average MMR difference in the last 75 games
        5. avg_mmr_diff_100: average MMR difference in the last 100 games
        6. avg_opp_mmr: average opponent MMR
        7. avg_game_length: average game length in seconds
        8. common_civ: most common civilization, based on player's entire game history

---

**Feature Engineering**

The following features were engineered:

- **Match-Level Features:**
    - Player A vs. Player B metrics (e.g., rating_A, win_rate_A, rating_B, win_rate_B).
    - Differences in performance metrics (e.g., rating_diff, win_rate_diff).
- **Cluster Features:**
    - Clustering was performed using **KMeans** with 25 clusters to group players based on scaled features.

---

**Model Training and Evaluation**

**Predicting Match Outcomes**

- **Objective:** Predict whether Player A wins the match.
- **Algorithm:** XGBoost (eXtreme Gradient Boosting)

**Workflow:**

1. **Data Split:**
    - Training: 80%
    - Testing: 20%
2. **Features:**
    - Numerical features were normalized.
    - Target: outcome_A (binary: 1 for win, 0 for loss).
3. **Model Training:**
```
classifier_model = XGBClassifier(
  n_estimators=100,
  learning_rate=0.1,
  max_depth=6,
  random_state=42
)
classifier_model.fit(X_train, y_train)
```
4. **Results:** AUC: **73%**

**Feature Importance:**

Top features influencing predictions:

1. avg_mmr_diff_100_A
2. avg_mmr_diff_100_B
3. avg_mmr_B
4. avg_mmr_A

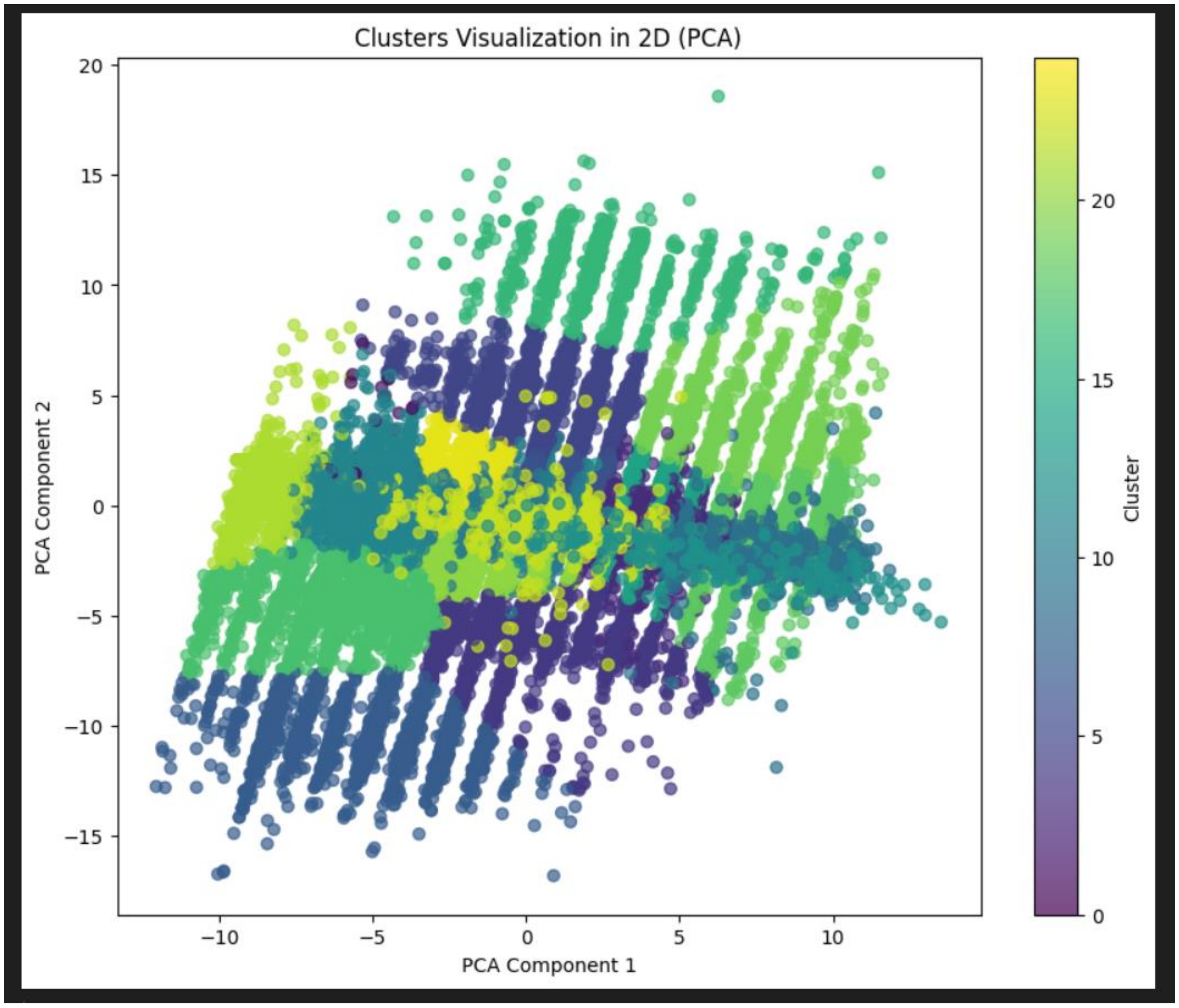
**Player Clustering**

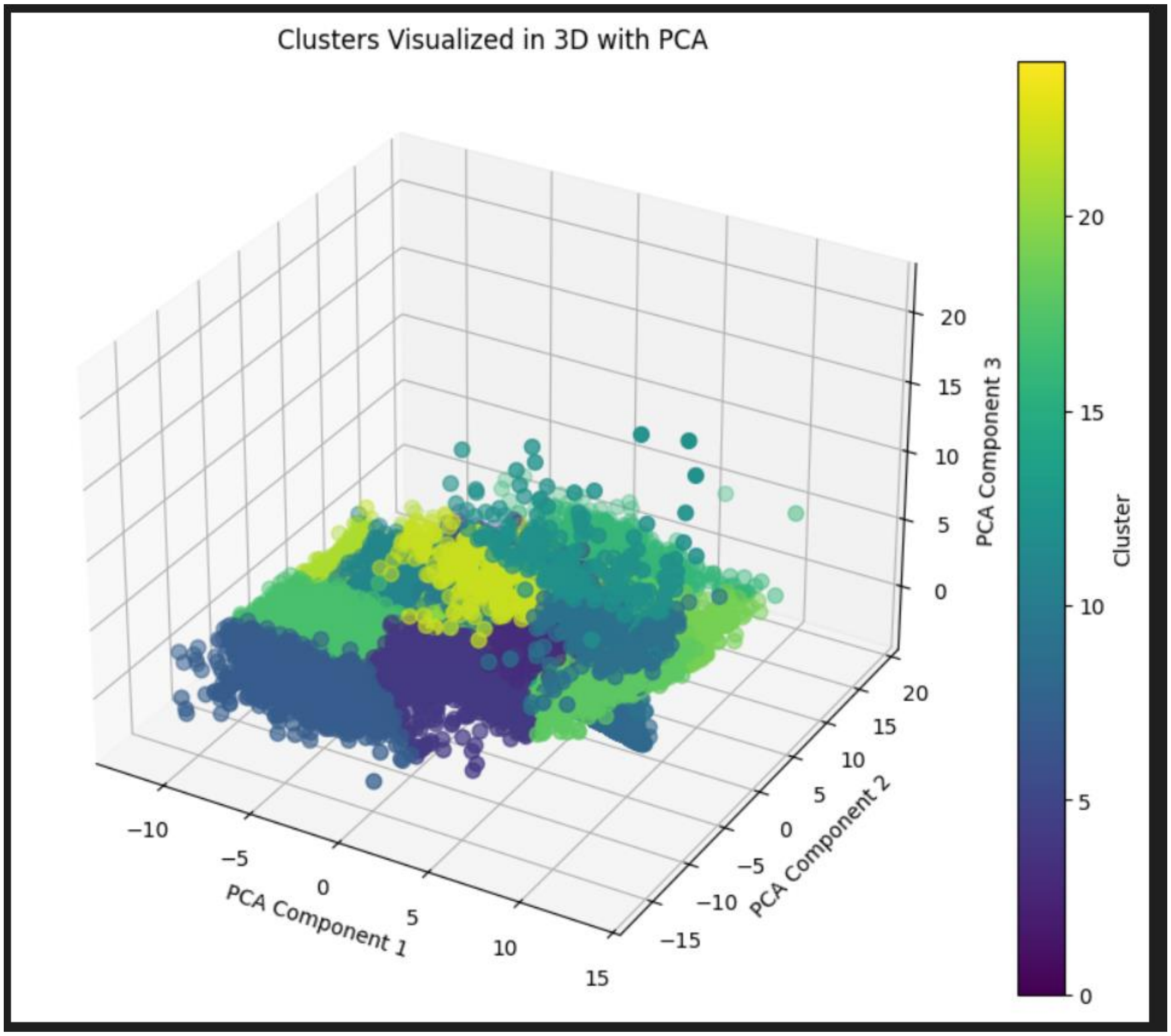
- **Objective:** Group players into clusters for balanced matchmaking.

- **Algorithm:** KMeans

**Workflow:**

1. **Input Features:** Normalized player metrics (e.g., rating, win_rate, games_count).
2. **Number of Clusters:** 25
3. **Visualization:**
    - Reduced dimensions using PCA for 2D and 3D visualization.
    - Results showed distinct clusters based on player characteristics.





## Matchmaking System Design

### Functionality

The matchmaking function:

1. Assigns new players to clusters based on their features.
2. Finds potential matches within the same cluster by:
    - Minimizing rating differences.
    - Optimizing fairness using the classification model to predict balanced outcomes.

    - If a balanced match is not found, use a fallback ELO-based approach to pair players with similar ratings.

**Matchmaking API**

The matchmaking system is powered by a set of well-defined APIs designed to manage the matchmaking process, predict match outcomes, and retrieve player information. These APIs ensure seamless communication between the frontend and backend systems and offer robust functionality for both developers and end-users.

**Available Endpoints**

**1.** Add Player to Matchmaking Queue

- **Endpoint:** POST /api/matchmaking/queue
- **Description:** Adds a player to the matchmaking queue.
- **Request:**

```
{
  "playerId": 123
}
```

- **Response:**

```
{
  "success": true,
  "error": null
}
```

**2. Remove Player from Queue**

- **Endpoint:** POST /api/queue/remove
- **Description: Removes a player from the matchmaking queue.**
- **Request:**

```
{
  "playerId": 123
}
```

- **Response:**

  ```
  {
    "success": true,
    "error": null
  }
  ```

**3. Predict Match Outcome**

- **Endpoint:** POST /api/matchmaking/predict
- **Description:** Predicts the probability that Player 1 wins against Player 2.
- **Request:**

  ```
  {
    "player1": 123,
    "player2": 456
  }
  ```

- **Response:**

  ```
  {
    "success": true,
    "data": 0.65,
    "error": null
  }
  ```

**4. Pair Player with Opponent**

- **Endpoint:** POST /api/matchmaking/pair
- **Description:** Pairs a player with the best-matched opponent from the queue and returns detailed match data.
- **Request:**

  ```
  {
    "playerId": 123
  ```

```
}
```

- **Response:**

```
{
  "success": true,
  "data": {
    "player1": {
      "profileId": 123,
      "name": "Player123",
      "rank": 5,
      "rating": 1200.5,
      "gamesCount": 300,
      "winsCount": 180,
      "lastGameAt": "2024-12-01",
      "rankLevel": "Gold 2",
      "country": "US",
      "avgMmr": 1185.4,
      "avgOppMmr": 1190.2,
      "avgGameLength": 1500,
      "commonCiv": "English",
      "winRate": 0.60,
      "inputType": "keyboard"
    },
    "player2": {
      "profileId": 456,
      "name": "Player456",
      "rank": 6,
      "rating": 1210.2,
      "gamesCount": 290,
      "winsCount": 175,
      "lastGameAt": "2024-11-30",
      "rankLevel": "Gold 2",
      "country": "UK",
      "avgMmr": 1200.3,
      "avgOppMmr": 1198.5,
      "avgGameLength": 1480,
      "commonCiv": "Ottomans",
      "winRate": 0.58,
      "inputType": "keyboard"
    },
    "player1WinProb": 0.52
  },
  "error": null
}
```

**5. Get Player by ID**

- **Endpoint:** GET /api/players/{player_id}
- **Description:** Retrieves detailed information about a specific player by their ID.
- **Response:**

```
{
    "success": true,
    "data": {
        "profileId": 123,
        "name": "Player123",
        "rank": 5,
        "rating": 1200.5,
        "gamesCount": 300,
        "winsCount": 180,
        "lastGameAt": "2024-12-01",
        "rankLevel": "Gold 2",
        "country": "US",
        "avgMmr": 1185.4,
        "avgOppMmr": 1190.2,
        "avgGameLength": 1500,
        "commonCiv": "English",
        "winRate": 0.60,
        "inputType": "keyboard"
    },
    "error": null
}
```

**6. Get Paginated List of Players**

- **Endpoint:** GET /api/players
- **Description:** Retrieves a paginated list of players with sorting and filtering options.
- **Query Parameters:**
  - page (int): The page number.
  - pageSize (int): The number of players per page.
  - orderBy (str): Column to sort by.
  - filter (str): Filter players by name.
- **Response:**

```
{
  "success": true,
  "data": [
    {
      "profileId": 123,
      "name": "Player123",
      "rank": 5,
      "rating": 1200.5,
      "gamesCount": 300,
      "winsCount": 180,
      "lastGameAt": "2024-12-01",
      "rankLevel": "Gold 2",
      "country": "US",
      "avgMmr": 1185.4,
      "avgOppMmr": 1190.2,
      "avgGameLength": 1500,
      "commonCiv": "English",
      "winRate": 0.60,
      "inputType": "keyboard"
    }
  ],
  "error": null
}
```

**Frontend Application**

To enhance usability, a frontend web application was developed to display matchmaking results. This app allows users to:

- View matched player details and fairness scores in real-time.
- Access insights about player rankings, win rates, and historical performance.

The frontend app is built with modern web technologies like **Angular** and **PrimeNG**. It connects the matchmaking API to fetch and display results, providing an intuitive interface for both casual players and competitive users.

---

**Future Improvements**

**Improvements:**

1. **Advanced Feature Engineering:** Add real-time player activity data.
2. **Hyperparameter Tuning:** Improve model performance using GridSearchCV or Bayesian Optimization.
3. **Fairness Metrics:** Include additional fairness constraints (e.g., latency, APM).

---

**Conclusion**

This project demonstrates the use of ML to enhance matchmaking in AoE4. The combination of clustering and match outcome prediction offers a scalable solution for creating balanced matches. The integration of a clustering mechanism ensures that players are grouped with competitors of similar skill levels, promoting a fair and challenging gameplay environment. Additionally, the use of predictive modeling adds another layer of precision by forecasting match outcomes and balancing pairings based on historical performance data.

Future iterations can focus on refining the system for higher accuracy and fairness. Enhancements could include incorporating real-time player statistics, addressing regional matchmaking preferences to reduce latency, and adding dynamic features such as adaptive clustering that evolves with player performance over time. By implementing these improvements, the matchmaking system can better align with player expectations, further enriching the overall gaming experience.