

Basketball Player Performance Analysis

My name is Adaora Onwumelu and this project is part of my Applied Algorithms class with Professor Ghosh. The goal of this project is to analyze basketball player performance using various algorithms such as clustering, classification, and regression.

Overview

This project aims to develop a comprehensive system for analyzing basketball player performance. The system incorporates various algorithms and techniques to predict player performance, categorize players into distinct classes, and cluster players based on their playing profiles. Additionally, the system includes modules for computing various performance metrics and visualizing the results to aid decision-making for coaches, scouts, and team management. The implemented algorithms include:

Features:

- **Regression Algorithm:** Utilizes regression analysis to predict player performance metrics, such as points per game (PTS), based on relevant features.
- **Classification Algorithm:** Categorizes players into distinct classes (e.g., Star Player, Average Player, Bench Player) based on their performance metrics using classification techniques.
- **Clustering Algorithm:** Groups players into clusters based on their performance metrics, such as minutes played (MIN) and points (PTS), to identify different player profiles or playing styles.
- **Performance Analysis Module:** Computes various performance metrics for each player, including PTS, rebounds (REB), assists (AST), steals (STL), blocks (BLK), and shooting percentages.

Implementation

The project is implemented in C programming language. Each algorithm has its own source code file:

```
- `clustering_algorithm.c`  
- `classification_algorithm.c`  
- `regression_algorithm.c`  
- `player_performance_analysis.c`
```

Each source code file loads input data from a file provided by the user, performs the respective analysis, and prints the results to the screen.

Usage

To run each algorithm:

1. Compile the source code file using a C compiler.

```
gcc classification_algorithm.c -o classification_algorithm
gcc clustering_algorithm.c -o clustering_algorithm
gcc regression_algorithm.c -o regression_algorithm
gcc player_performance_analysis.c -o player_performance_analysis
```

2. Execute the compiled program.

```
./classification_algorithm
./clustering_algorithm
./regression_algorithm
./player_performance_analysis
```

3. Follow the prompts to enter the name of the input data file.

Player_data.txt

Data Collection: Gather basketball player data from various sources, including publicly available datasets and proprietary team databases.

Data Preprocessing: Clean and preprocess the data to handle missing values, normalize features, and encode categorical variables.

Algorithm Implementation:

- Implement regression models for predicting player performance metrics.

- Develop classification algorithms for categorizing players into performance classes.

- Design clustering algorithms to group players based on their playing profiles.

Performance Analysis:

- Compute performance metrics for each player using the analysis module.

- Visualize the results using plots and charts to facilitate understanding and decision-making.

Efficiency Evaluation:

- Conduct experiments using both synthetic minimal data and real WNBA data to evaluate the efficiency and scalability of the algorithms.

- Measure runtime performance, monitor memory usage, and analyze scalability across varying problem sizes.

Input Data Screenshots

The image shows a C++ IDE with a project named 'FINALPROJECT'. The main file is 'player_performance_analysis.c'. The code defines a structure for player statistics and a function to read data from a file. The main function prompts the user for a file name, reads the data, and prints a table of player performance metrics. The terminal output shows the following table:

Player	Team	PTS	REB	AST	STL	BLK	FGM	3PM	FTM	GP
Brandon Stewart	New York Liberty	22.8	9.5	3.8	1.4	2.2	55.7	31.0	81.2	48
Sabrina Ionescu	New York Liberty	17.8	5.8	5.4	1.8	0.3	42.3	44.8	87.2	38
Kierstan Bell	Las Vegas Aces	2.7	1.8	8.5	8.4	8.1	34.6	24.4	64.8	38

```
EXPLORER
  CLASSIFICATION.py
  player_data.c
  regression_algorithm.c
  clustering_algorithm.c
  clustering_plot.py
  classification.py
  regression_plot.py
  player_performance_analysis.c
  player_performance_analysis2.c

OPEN EDITORS
  Untitled-1
  classification_...
  player_data.txt
  regression_alg...
  clustering_alg...
  clustering_plo...
  classification_...
  performance_...
  regression_plo...
  player_perfor...

FINAL PROJECT
  classification_al...
  classification_al...
  classification_p...
  clustering_algo...
  clustering_algo...
  clustering_plo...
  performance_plo...
  player_data.txt
  player_perform...
  player_perform...
  regression_algo...
  regression_algo...
  regression_plo...

1 // classification.py
2 # Import necessary libraries
3 import sys
4 import os
5 import numpy as np
6 import pandas as pd
7 from sklearn.preprocessing import StandardScaler
8 from sklearn.linear_model import LinearRegression
9 from sklearn.metrics import mean_squared_error, r2_score
10
11 # Load the input file
12 def load_data(file_name):
13     """Load the input file and return the data as a dictionary of lists"""
14     data = {}
15     with open(file_name, 'r') as f:
16         lines = f.readlines()
17         for line in lines:
18             parts = line.strip().split(',')
19             player_name = parts[0]
20             team = parts[1]
21             pts = float(parts[2])
22             features = parts[3:]
23             data[player_name] = {'team': team, 'pts': pts, 'features': features}
24     return data
25
26 # Define the input file name
27 input_file = 'player_data.txt'
28
29 # Load the data
30 data = load_data(input_file)
31
32 # Extract the features and target variable
33 num_players = len(data)
34 num_features = len(data[list(data.keys())[0]]['features'])
35
36 # Create a list of features and target variable
37 features = []
38 target = []
39 for player_name in data:
40     features.append(data[player_name]['features'])
41     target.append(data[player_name]['pts'])
42
43 # Standardize the features
44 scaler = StandardScaler()
45 features = scaler.fit_transform(features)
46
47 # Create a linear regression model
48 model = LinearRegression()
49 model.fit(features, target)
50
51 # Predict the points for each player
52 predicted_points = []
53 for i in range(num_players):
54     predicted_points.append(model.predict(features[i])[0])
55
56 # Print the results
57 print("Player Information:")
58 for i in range(num_players):
59     player_name = list(data.keys())[i]
60     team = data[player_name]['team']
61     pts = data[player_name]['pts']
62     predicted_pts = predicted_points[i]
63     print(f"{player_name}: {team}, PTS: {pts} (Predicted: {predicted_pts})")
64
65 # Main function
66 def main():
67     """Main function to run the program"""
68     # Load the data
69     data = load_data(input_file)
70
71     # Extract the features and target variable
72     num_players = len(data)
73     num_features = len(data[list(data.keys())[0]]['features'])
74
75     # Create a list of features and target variable
76     features = []
77     target = []
78     for player_name in data:
79         features.append(data[player_name]['features'])
80         target.append(data[player_name]['pts'])
81
82     # Standardize the features
83     scaler = StandardScaler()
84     features = scaler.fit_transform(features)
85
86     # Create a linear regression model
87     model = LinearRegression()
88     model.fit(features, target)
89
90     # Predict the points for each player
91     predicted_points = []
92     for i in range(num_players):
93         predicted_points.append(model.predict(features[i])[0])
94
95     # Print the results
96     print("Player Information:")
97     for i in range(num_players):
98         player_name = list(data.keys())[i]
99         team = data[player_name]['team']
100         pts = data[player_name]['pts']
101         predicted_pts = predicted_points[i]
102         print(f"{player_name}: {team}, PTS: {pts} (Predicted: {predicted_pts})")
103
104 # Run the main function
105 if __name__ == '__main__':
106     main()
107
108 PROBLEMS | DEBUG CONSOLE | TERMINAL | SQL CONSOLE
109 OUTPUT
110 Enter the name of the input file: player_data.txt
111 Player information:
112 Player: Aja Wilson, Team: Las Vegas Aces, PTS: 22.80 (Predicted: 24.72)
113 Player: Breanna Stewart, Team: New York Liberty, PTS: 23.80 (Predicted: 25.97)
114 Player: Sabrina Ionescu, Team: New York Liberty, PTS: 17.80 (Predicted: 21.28)
115 Player: Kirsten Bell, Team: Las Vegas Aces, PTS: 3.70 (Predicted: 6.26)
116 alexandra@hooras-MacBook-Air:~/FinalProject$
```

The screenshot displays a C++ IDE with a project named 'FINALPROJECT'. The main file, 'clustering_algorithm.cpp', contains the implementation of a clustering algorithm. The code defines a 'Player' struct and a 'Cluster' struct. It reads data from 'player_data.txt' and 'player_perform.txt'. The algorithm iterates through the data, calculating distances and updating centroids until convergence. The output is printed to the console, showing the final clusters and their centroids.

```

// clustering_algorithm.cpp
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include <algorithm>
#include <limits>

using namespace std;

struct Player {
    string name;
    int team;
    double MIN;
    double PTS;
};

struct Cluster {
    int num_players;
    double centroid_MIN;
    double centroid_PTS;
};

// Function to calculate the distance between two players
double euclidean_distance(const Player &p1, const Player &p2) {
    return sqrt((p1.MIN - p2.MIN) * (p1.MIN - p2.MIN) + (p1.PTS - p2.PTS) * (p1.PTS - p2.PTS));
}

// Function to find the closest cluster for a given player
int find_closest_cluster(const vector<Player> &players, const vector<Cluster> &clusters) {
    int closest_cluster = 0;
    double min_distance = INFINITY;

    for (int j = 0; j < clusters.size(); j++) {
        float distance = euclidean_distance(players[i], players[j], clusters[j].centroid_MIN, clusters[j].centroid_PTS);
        if (distance < min_distance) {
            min_distance = distance;
            closest_cluster = j;
        }
    }

    return closest_cluster;
}

// Function to update the centroid of a cluster
void update_centroid(struct Player players[], struct Cluster clusters[]) {
    for (int i = 0; i < K; i++) {
        float sum_MIN = 0.0;
        float sum_PTS = 0.0;

        for (int j = 0; j < clusters[i].num_players; j++) {
            int player_index = clusters[i].players_indices[j];
            sum_MIN += players[player_index].MIN;
            sum_PTS += players[player_index].PTS;
        }

        if (clusters[i].num_players > 0) {
            clusters[i].centroid_MIN = sum_MIN / clusters[i].num_players;
            clusters[i].centroid_PTS = sum_PTS / clusters[i].num_players;
        }
    }
}

// Function to print the final clusters
void print_clusters(struct Player players[], struct Cluster clusters[]) {
    for (int i = 0; i < K; i++) {
        cout << "Cluster " << i << ": " << endl;
        cout << "Centroid: MIN = " << clusters[i].centroid_MIN << ", PTS = " << clusters[i].centroid_PTS << endl;
        cout << "Players: " << endl;

        // Keep track of printed players
        int printed_players[MAX_PLAYERS] = {0};

        for (int j = 0; j < clusters[i].num_players; j++) {
            int player_index = clusters[i].players_indices[j];
            if (!printed_players[player_index]) {
                cout << "from " << players[player_index].name << ", team: " << players[player_index].team << ", MIN: " << players[player_index].MIN << ", PTS: " << players[player_index].PTS << endl;
                printed_players[player_index] = 1;
            }
        }

        cout << "Printed " << clusters[i].num_players << " players" << endl;
    }
}

int main() {
    // Read data from files
    vector<Player> players;
    vector<Cluster> clusters;

    // ... (Data reading code) ...

    // Run the clustering algorithm
    while (true) {
        // Find closest cluster for each player
        for (int i = 0; i < players.size(); i++) {
            int closest_cluster = find_closest_cluster(players, clusters);
            clusters[closest_cluster].players_indices[clusters[closest_cluster].num_players++] = i;
        }

        // Update centroids
        update_centroid(players, clusters);

        // Print clusters
        print_clusters(players, clusters);

        // Check for convergence
        bool converged = true;
        for (int i = 0; i < clusters.size(); i++) {
            if (clusters[i].num_players > 0) {
                double old_MIN = clusters[i].centroid_MIN;
                double old_PTS = clusters[i].centroid_PTS;
                // ... (Convergence check logic) ...
            }
        }

        if (converged) break;
    }
}

```

The terminal output shows the following results:

```

adornaxia@Adornas-MacBook-Air FinalProject % ./clustering_algorithm
Enter the name of the input file: player_data.txt
File opened successfully.
Cluster 1:
Centroid: MIN = 11.88, PTS = 3.78
Players:
Kiersten Bell from Las Vegas Aces: MIN = 11.88, PTS = 3.78

Cluster 2:
Centroid: MIN = 32.18, PTS = 26.93
Players:
Alyssa Wilson from Las Vegas Aces: MIN = 38.78, PTS = 22.88
Breanna Stewart from New York Liberty: MIN = 34.18, PTS = 23.88
Sabrina Ionescu from New York Liberty: MIN = 35.58, PTS = 37.88

```

```
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

// Read player information from the file
while (fgets(header, sizeof(header), input_file) != NULL) {
    sscanf(header, "%d%T", &num_players, %N);
    // ... [rest of the code] ...
}

fclose(input_file);

// Classification based on PTS (Simple example)
print("Classification based on PTS\n");
for (int i = 0; i < num_players; i++) {
    if (players[i].PTS == 20.0) {
        print("%s from %s: PTS = %.2f\n", players[i].name, players[i].team, players[i].PTS);
        print("Classification: Star Player\n");
    } else if (players[i].PTS == 18.0) {
        print("%s from %s: PTS = %.2f\n", players[i].name, players[i].team, players[i].PTS);
        print("Classification: Average Player\n");
    } else {
        print("%s from %s: PTS = %.2f\n", players[i].name, players[i].team, players[i].PTS);
        print("Classification: Bench Player\n");
    }
}

return 0;
```