Problem Chosen	2024 MCM/ICM Summary Sheet	Team Control Number
C		2427257

Summary

The recent victory of tennis star Alcarez over super Slammer Djokovic to win the 2024 Wimbledon men's singles title has been a hot topic in the community, with a number of upsets during the tournament, which have been fascinating to watch and have caused us to think about the momentum of the tennis tournament.

For Problem 1, we performed data processing and feature extraction on the dataset given in the question, and then established the **momentum metric function** to generalize the quantification of the player momentum, and also established the **performance evaluation function** determined by six parameters to measure specifically the The **expected winning percentage** was calculated and mapped to the real data score points, thus establishing the **objective function and constraints**. The ideal optimal value of the objective function and the corresponding six optimal parameters are calculated by **the BP parameter optimization process composed of the Gradient Descent Method**, and the performance excellence of the players at specific moments is precisely quantified by the determined functional form, and the visualization process is also given and analyzed from different perspectives.

For problem 2, we analyze the problem and convert it into the problem of determining **whether the flow of a tennis match is a Markov chain**, and construct an evaluation function in the form of a Markov chain on the basis of the model of problem 1, determine the parameters inside the function through a similar training process, and obtain the optimal solution of the objective function under the Markov chain, and we carry out **hypothesis testing** for the two models as well as the real data, and the final result is that **the Tennis program is a non-Markov chain process**.

For problem 3, we built a **differential difference model** based on the model we built earlier, seeking the approximate curve of **momentum metric function extreme value points**, which we consider to be the key nodes at which momentum **transformation** occurs. At the same time, we took the **remaining factors** in the dataset into account and built a **body force value model** to retrain the model for the players' individual data and to obtain the most relevant **four influencing factors** corresponding to the extreme points based on statistical analysis methods. With the dataset and the corresponding influencing factors, we give some **common strategies** to deal with players of different opponents in different fixture situations, which provide support for pre-match preparation and in-match decision-making.

For Problem 4, we conducted **migration tests** of the established model on different datasets, and established a **fit level evaluation model** to quantitatively analyze the performance level of the model on different datasets. For the underperforming datasets, we adjusted and changed the structure of the model considerations according to the actual situation, and identified some different considerations in similar datasets to improve the model's **generalization ability**.

Key words: momentum metric function performance evaluation function BP parameter optimization process Markov chain influencing factors migration tests

1 Establishment and Solution of Model for Problem 1

1.1 Content of the problem

Develop a model that captures the flow of play as points occur and apply it to one or more of the matches. Your model should identify which player is performing better at a given time in the match, as well as how much better they are performing. Provide a visualization based on your model to depict the match flow. Note: in tennis, the player serving has a much higher probability of winning the point/game. You may wish to factor this into your model in some way.

1.2 Analysis of the problem

The problem requires us to develop a model that dynamically captures the performance status of both players in each points of a tennis match and presents it in a quantitative form to show the performance status of the players. At the same time, we hope that this model is universally applicable to valuable tennis match scenarios. Therefore, we consider starting from the scoring points, based on which we explore and consider multiple factors, in order to establish a comprehensive mathematical model to quantify the performance of the players; on the other hand, for the optimal selection of some parameters in the model, we consider using the whole dataset given in the title to carry out training, and according to the results of the training to guide the calibration of the relevant parameters, which in fact forms a model tuning BP process.

1.3 Modeling

We believe that there exists a "momentum" of the athlete's own state, which reflects the athlete's ability to master the game, the degree of his or her own strength, and the ability to reverse an unfavorable situation in the course of the game. This likelihood is related to the influence of the events that the athlete is subject to, such as whether the point is scored or not, whether he/she has the right to serve or not, etc., and it has a significant positive correlation with his/her probability of scoring a point.

Furthermore, we build a winning performance model based on the conditions and flow of the matches, which determines whether a player is performing well or not based only on the score-dependent game state parameter X (note that we do not take into account factors that are closely related to the player's own characteristics, such as stamina level, resistance to stress, etc., in order to ensure the universality of the model), which consists of the following parameters:

Right to serve: $x_r : \{1, 0\}$, 1 means the right belongs to you, 0 means the right belongs to the opponent.

Points scored/missed: $N : \{1, 0\}$, 1 indicates a point scored, 0 indicates a point scored by the opponent.

Number of consecutive points scored/missed: x_1, positive integer, indicates the number of consecutive points scored or missed.

Whether or not the serving team made an error: $x_2 : \{0, 1\}$, 0 means no serving error, 1 means serving error.

Whether it is a game point or not: $x_g : \{1, 0\}$, 1 means it is a game point, 0 means it is not.

Whether it is a set point: $x_s : \{1, 0\}$, 1 means it is a set point, 0 means it is not.

Whether it is a direct service point: $x_3 : \{1, 0\}$, 1 means it is a direct service point, 0 means it is not.

ariable x above is time-dependent $x(t_n)$ and the state parameter $X(t_n) = (x_r, N, x_1, x_2, x_3, x_4, x_g, x_s)$

Define the momentum metric function $Q(t_n) = kQ(t_{n-1}) + F(X(t_{n-1}))$, where$n = 1, 2, 3Q(t_1) = 0$, t_1 denotes the time of the first scoring point, and the decay rate, $k \in (0, 1)$, is is used to measure the extent to which varying time away from that score point affects the momentum of the next scoring point, and we believe that the closer the event is to that scoring point, the more it will play a role in influencing the momentum of the next scoring point.

At the same time we define the predicted win rate

$$P = \frac{Q_1(t_n) - Q_2(t_n)}{Q_1(t_n) + Q_2(t_n)} \tag{1}$$

Representing the predicted win rate for that score point derived from the quantized momentum, we argue that the greater the difference in momentum between player one and player two when player one's momentum is higher than player two's at that score point, the greater the likelihood that player one should be victorious at that score point.

Under this definition, F denotes the upward tendency of the player's winning percentage at the n-1st score point due to various game performances, and is used by us to evaluate the player's performance. The performance evaluation function is defined as follows:

$$\begin{aligned} F(X(t_{n-1})) =& [(1 - x_r)\frac{q}{1-q} + x_r]\delta(N, 1)[1 + k_1(x_1 - 1) - k_2 x_r x_2 + k_3(x_g + x_s)] \\ & - k_4[\frac{q}{1-q}x_r + (1 - x_r)]\delta(N, 0)[1 + k_1(x_1 - 1) + k_5 x_2 x_r + k_3(x_g + x_s) + k_6 x_3] \end{aligned} \tag{2}$$

Among that we consider:

- the right to serve gives the player a greater advantage, we counted the average winning percentage q of the serving side in the dataset, and used the reward coefficient $\frac{q}{1-q}$ to amplify the performance score when the receiving side wins and the performance deduction when the serving side loses, here $q \approx 0.6731$;
- when scoring $N = 1$, $\delta(N, 1) = 1, \delta(N, 0) = 0$, only the first term of F is non-zero, at this time, if the other side serves the ball, you get the reward coefficient $\frac{q}{1-q}$, otherwise, you get 1, and the same when $N = 0$. That is, the first term of F is the positive rating when winning the game, and the second term is the negative rating when losing the game;
- The last part of the two terms of F determines the base value of the evaluation score, including the win/loss situation, consecutive points scored or lost, and the effect of errors on one's own serve. In addition, wins and losses at set and game points have a greater impact on the player than those in general;
- the scoring factors are related to each other, which in order to incorporate the examination of the model, such as the situation of the ball possession and the situation of the stronghold can reflect whether it is a break point or not.

Figure 1: Momentum Vision

On this basis, we define the optimization objective as

$$min\left(T=\frac{1}{N_{play}}\sum_{all}\sum_{t_n}\left(\frac{Q_1(t_n)-Q_2(t_n)}{Q_1(t_n)+Q_2(t_n)}-N(t_n)\right)^2\right) \tag{3}$$

The optimization goal is to make the predicted wins as close as possible to the actual wins. We include every scoring point of all games in the dataset in the training dataset, and parameterize it to obtain a universal evaluation model.

1.4 Solving and Analysis

First we processed and extracted the dataset provided by the question using python code, divided it into thirty-one complete matches, and extracted the required factors from each match, such as the right to serve, the number of serves, and so on, which in turn were transformed into the factors required for the performance evaluation function, such as consecutive points scored or lost, whether it was a set point or not, and so on. In turn, we achieve a complete reproduction of the above model and construct a functional form of the optimization objective, which is determined by six parameters, and whose function return value we expect to be as small as possible.

Using the Gradient Descent Method, the parameters were iteratively adjusted by BP, and the above objective function value was finally optimized to 0.4992, which indicates that the momentum predicted win rate conforms to the actual situation to approximately 0.65.

At the same time, we obtain a near-optimal solution for all k-parameters in the F function: $\vec{k} = (56.46, -26.96, 195.48, -190.65, 28.36, 49.67)$. Taking the final match played by Carlos Alcaraz against Novak Djokovic as an example, we plotted images of the variation of momentum $Q_1(t_n)$ and $Q_2(t_n)$ for both in 1

As you can see from the graph (where player 1 is Djokovic and player 2 is Alcarez), when the match first started, Djokovic was firmly in control of the match and thus dominated most of the points in the first set, however, as the match progressed, the momentum of the two athletes began to change, which allowed Alcarez started to fight back in the second set and took a hard fought set in a tie-breaker.

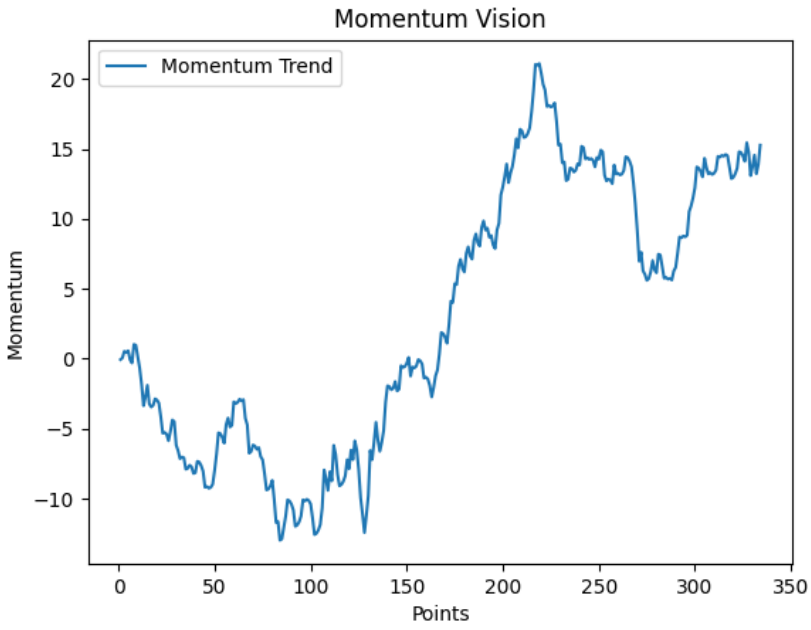


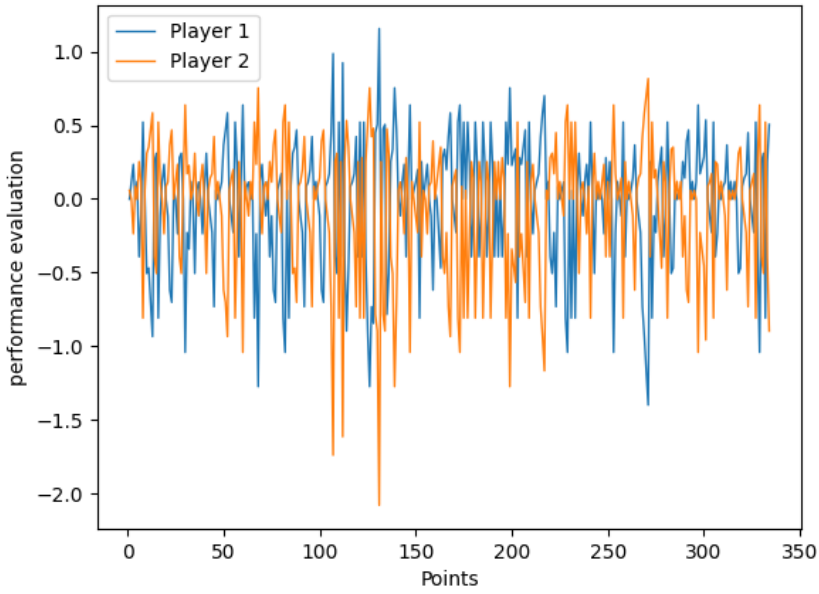
Figure 2: Momentum Vision

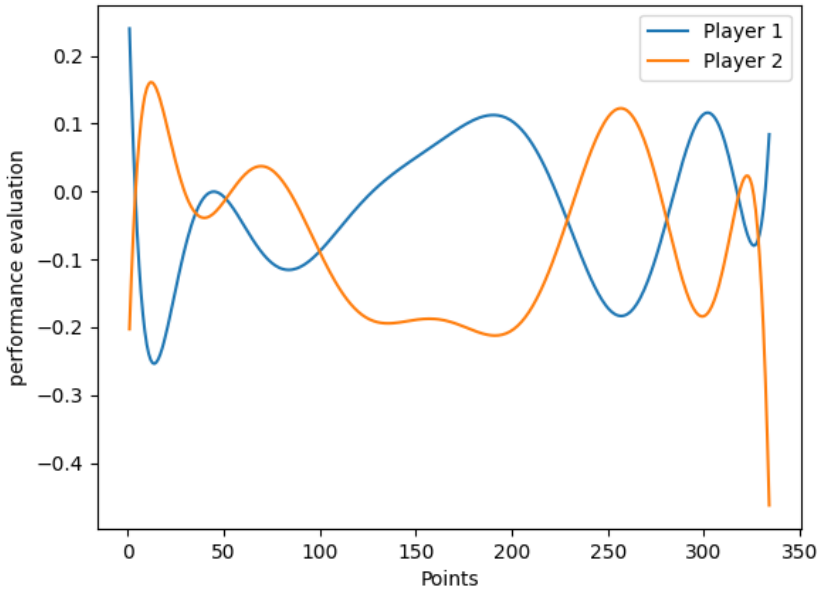
Thereafter, Djokovic began to gradually take control of the match and won the third set comfortably with a huge 5 game difference. Although Djokovic regained a set in the fourth set (which is reflected in a small peak in the blue line in the graph late in the match), the match ultimately tipped the scales in favor of Alcarez, which is a reflection of Alcarez's momentum.

At the same time, we visualized the difference between the momentum of the two athletes in the race 2 This image provides another, more intuitive perspective on the flow of the match, as we can easily find which player is performing better at a given moment in time, which is reflected in the plus and minus of the vertical axis, while the absolute value of the vertical axis value quantifies how well the player is performing. In addition, looking at the fluctuations of the curve, it is easy to see that some of the turning points of the curve reflect the tendency that players are trying to take the pace of the game into their own hands.

In addition, we plotted the image of the performance evaluation function $F(X(t_{n-1}))$ of the two players in the flow of the game, and at the same time, in order to better observe the trend of its value through the more violent fluctuations, we made a certain fit to the discrete sequence composed of the function value, in order to fully show its relative trend on the basis of filtering out the violent fluctuations, as shown in the figure 1.4 and 1.4 The curve formed by the values returned by the performance evaluation function can be seen in the graph, and its general trend and relative magnitude are similar to the course of the game analyzed earlier, but what can be observed in particular is that towards the end of the game, the value of Djokovic's performance evaluation begins to drop drastically, which is considered to be a sign that, on one hand, he has completely lost the control over the situation, and, on the other hand, it also reveals that there may be the influence of some other potential factors in the course of the process, and that's exactly what we will need to take into account later on.

All in all, these charts give us a visual representation of the flow of the game and the performances of both players, and provide different perspectives for us to get a fuller picture of the game.


Player 1
Player 2
performance evaluation
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
50
100
150
200
250
300
350
Points


Player 1
Player 2
performance evaluation
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
0
50
100
150
200
250
300
350
Points

2 Establishment and Solution of Model for Problem 2

2.1 Content of the Problem

A tennis coach is skeptical that "momentum" plays any role in the match. Instead, he postulates that swings in play and runs of success by one player are random. Use your model/metric to assess this claim.

2.2 Analysis of the Problem

The tennis coach believes that swings during a match are random and do not depend on so-called potential energy. According to our definition, momentum is the accumulated effect of a series of past events and conditions on whether or not the current point is scored, and this effect is reflected in the swings of each point. In other words, the coach believes that the result of the occurrence of the current scoring point event depends only on the current playing conditions and has nothing to do with the relevant events of the previous scoring points, i.e., the playing conditions of a tennis match are considered to be a Markov chain process; whereas, if momentum is considered to be present in the course of the match, i.e., the process is a non-Markov chain process. Therefore, in order to determine whether the tennis match is a Markov chain or not, based on the model of the first question, we delete the non-Markovian term $Q(t_{n-1})$ from the model, transform the performance function into the current performance function, and after replacing the state parameter $X(t_n)$ with $X'(t_n)$, we carry out the same training to obtain a Markov chain model representing the data and compare it with the original model to arrive at the final result. Comparison is made to arrive at the final result.

2.3 Modeling

Define the state space $S = \{(0,0,0),(1,0,0),...\}$, to denote the current scores (points, games, sets), and $Y(t_n)$ to denote the scores at the moment of t_n. That is, the Markov chain-based model is satisfied:

$$\begin{aligned} &P\{Y(t_n) = s(t_n)|Y(t_1) = s(t_1), Y(t_2) = s(t_2), ..., Y(t_{n-1}) = s(t_{n-1})\} \\ &\quad = P\{Y(t_n) = s(t_n)|Y(t_{n-1}) = s(t_{n-1})\} \end{aligned} \tag{4}$$

$$P\{Y(t_n) = s(t_n)\} = F_{unknown}(X'(t_n), Y(t_{n-1})) \tag{5}$$

where $X'(t_n) = (N(t_{n-1}), x_1(t_{n-1}), x_3(t_{n-1}), x_r(t_n), x_g(t_n), x_s(t_n), x_2 = 0)$, denotes the current state in the Markov chain.

In order to maintain formal consistency between the two models and to facilitate subsequent comparisons between the two models, we have selected the main state parameters from the first-question model, but have adopted a different composition for the scoring point. We consider that the state of the current scoring point consists of two components, the scoring situation of the previous point (winner/-consecutive number of points/serve direct), and the nature of the current scoring point as it would have been if the point had not been scored (set point/game point/right to serve).

Let $Q'(t_n) = F(X'(t_n))$, and replace the performance function in the first-question model with the current performance function, then the probability of winning can be expressed as (where a negative

probability denotes the opponent's winning percentage):

$$P\{Y(t_n) = s(t_n) = Y(t_{n-1}) + (1,0,0)\} = \frac{F(X_1'(t_n)) - F(X_2'(t_n))}{F(X_1'(t_n)) + F(X_2'(t_n))} \tag{6}$$

We can obtain an optimization objective similar in form to the first problem:

$$min\left(T' = \frac{1}{N_{play}} \sum_{all} \sum_{t_n} \left(\frac{F(X_1'(t_n)) - F(X_2'(t_n))}{F(X_1'(t_n)) + F(X_2'(t_n))} - N(t_n)\right)^2\right) \tag{7}$$

For the performance function $F(X'(t_n))$ of the trained Markov chain model, we compare the convergence value and convergence accuracy of its optimization objective with the results in the first question. In addition, we compare the accuracy of the two models in predicting the true sequence point by point, i.e., we take the result of any game from the data set as the true data, and its true win/loss result N as the data sequence, i.e., $N(t_n)$, and input the information of the scoring points point by point into the optimized Q-model and F-model, and output the predicted win/loss probability of the next scoring point according to the corresponding probability formula, and get a sequence of win/loss probability Sequence $Q\{P_{t_n}\}$ and $F\{P_{t_n}\}$, calculate and compare the size of the two respectively:

$$\sum_{t_n} (q_n - N(t_n))^2 \tag{8}$$

$$\sum_{t_n} (f_n - N(t_n))^2 \tag{9}$$

A comprehensive multi-dimensional comparison of the two models, combined with a comparison of the true values, can adequately determine whether the tennis schedule is a Markov chain or not.

2.4 Solving and Analysis

The BP parameter optimization process for this model is also carried out using the Gradient Descent Method, and the specific optimization process is the same as in Problem 1, with only the functional form being different. The objective is optimized to obtain the optimal $T' = 0.6371$, and the optimal parameter$\vec{k} = \left(94.54, 2, -97.16, -6.8 \times 10^{-3}, 20, 20\right)$

In the first question, the convergence value of the optimization objective is $T = 0.4992$. It can be found that T' is significantly larger than T and $\frac{T'-T}{\delta T} >> 1$ compared to the optimization result in the first question, thus the optimal prediction of the winning percentage of the in-the-moment performance model without the consideration of the antecedent tournament conditions is significantly inferior to the momentum-performance model with the consideration of the antecedent tournament conditions.

Thus, we are well-conditioned to believe that the match play condition of tennis is a non-Markov chain, i.e., there exists a momentum that affects the swings at each point in the match.

3 Establishment and Solution of Model for Problem 3

3.1 The First Problem

Coaches would love to know if there are indicators that can help determine when the flow of play is about to change from favoring one player to the other.

Using the data provided for at least one match, develop a model that predicts these swings in the match. What factors seem most related (if any)?

3.2 The Second Problems

Given the differential in past match "momentum" swings how do you advise a player going into a new match against a different player?

3.3 Analysis of the problems

For the first question, we believe that for a given match data, the momentum metric function we built can give the full momentum change process of the match flow. Based on this, we believe that the key swings in the match where relative momentum shifts occur are often the swings in which the players start to change the current unfavorable situation after accumulating momentum and regain the initiative of the match. To predict these swings in the match, we can build a difference difference model on the basis of the momentum metric function to find the extreme points of the approximate relative momentum curve, and we believe that incredible swings tend to occur at these points.

For the points with incredible swings, we take the rest of the factors mentioned in the dataset into consideration in the model, including players' physical strength, hitting area, etc. (at the same time, we established a model of players' physical strength with respect to the time), and by means of statistical analysis, we get the four most relevant factors for each player in the points with incredible swings. pointss the four most relevant factors.

Thus, we can use past game data to understand a player's momentum trend and the factors most associated with momentum change. For the players themselves, we should consider how to make reasonable use of these factors for pre-match preparation, how to make reasonable use of the rules to make momentum change in the course of the match, and how to adjust their own rhythm for offense and defense conversion; on the other hand, when facing a new player, we can analyze the data to reasonably find out his weaknesses, so as to sabotage the opponents' advantageous situation, and reasonably allocate physical strength to the players. On the other hand, when facing a new player, we can analyze the data to find out his weaknesses, so as to destroy the opponent's advantage on the court, reasonably allocate the physical energy to consume the opponent, and positively utilize the right to serve to increase the scoring rate.

3.4 Modeling

First of all, we build a difference difference model $\Delta Q(i, \delta q, j)$ based on momentum metric function, where i is the corresponding scoring point, δq is the size of the chosen difference, and j is the corresponding player. Since each player is affected by different relevant influences, the training dataset of the model should be the player's own dataset and the rest of the information is additionally extracted as one of the datasets for the statistical analysis, so there is:

$$\Delta Q(i, \delta q, j) = R(F_{Data(j)})_{[p_1, p_2, p_3, p_4]} \tag{10}$$

Where Data(j) is the corresponding dataset for player j, p_1 to p_4 are the four most relevant factors,R is the statistical analysis method used, specifically:

$$R = max(\sum_{i=1}^{4} p_i * F_i) \tag{11}$$

Furthermore, we model the physical strength of the player with respect to time $H(i,t)$ and consider the factors that influence the physical strength, including

- Physical strength at the previous time $H(i,t-1)$
- The running distance h at this scoring point
- The number of strokes at this point n
- The stroke speed v at this time point
- The rest time before this scoring point t

Combining the above factors there are:

$$H(i,t) = H(i,t-1) - \phi_1 h - \phi_2 n - \phi_3 v + \phi_4 t \tag{12}$$

3.5 Solving and Analysis

We take the data of the final match as an example for the difference difference score, and for the statistical analysis of the factors related to for the scoring points, we found that the four main influencing factors for Djokovic are [physical strength value, net points, serve rights, serve errors], while the four main factors for Alcarez are [consecutive points, serve rights, physical strength value, set points]. Combined with the previous analysis of the process of the final match, it is not difficult to find that Djokovic gradually lost the mastery of the game in the late stages of the match, and the performance point evaluation also declined rapidly, which may be related to the rapid decline of its physical strength value, while the net points are more important means of scoring, which can be seen that Djokovic has a more exquisite net skills; while Alcarez is less affected by the Alcarez, on the other hand, is less affected by the physical strength, so he can maintain a high level of competition, but it can be found that his scoring is more dependent on the right to serve, and it is more difficult to seize the initiative of the game in the set without the right to serve.

From the above examples, we can see that targeted training before the match and looking for the opponent's weaknesses, as well as actively adjusting the pace and using the rules of the game during the game can be more helpful to win the game, so we have summarized some common strategies for the game:

1. For the side affected by physical strength, the pace of the game should be appropriately accelerated, the game time should be shortened, and the physical strength should be actively utilized in the physical exhaustion of the rules to rest to obtain the recovery of physical strength.
2. For the opponent who is poor in a certain technical point, the data should be obtained from the data and targeted pre-match preparation.

3. For a team that is more dependent on the serve, it should give more stamina when it has the serve to improve its scoring.
4. For the team that is more affected by consecutive point losses, the rules should be used to break the trend of consecutive point losses.
5. For the team affected by service errors, reduce the amount of energy used when serving.
6. For players who are affected by physical strength, they may choose to give up some difficult balls to minimize the distance they have to run in order to maintain their physical strength.
7. For the opponent who is affected by the game point or match point game, he should pay more stamina value in these points.

The above strategies are partial strategies taken from the model, and actual matches should take into account fuller factors as well as more comprehensive strategies.

4 Establishment and Solution of Model for Problem 4

4.1 Content of the Problem

Test the model you developed on one or more of the other matches. How well do you predict the swings in the match? If the model performs poorly at times, can you identify any factors that might need to be included in future models? How generalizable is your model to other matches (such as Women's matches), tournaments, court surfaces, and other sports such as table tennis.

4.2 Analysis of the Problem

The problem examines the performance level of the model we built on the rest of the different types of datasets, and we should first build a pervasive level evaluation model to evaluate the performance level of the model. For the competition dataset where some of the factors are different but the basic kernel is similar, it is essentially migration learning to examine the migration ability of the model, and in the face of the fact that the results are not very good, we should be able to find out the factors that are more different between the two datasets, so as to make certain modifications to the key structure of the model to adapt it more to the specific situation, which is also a manifestation of the model's pervasive level.

4.3 Modeling

Considering the performance level of the model on different datasets, we build a universal level evaluation model $O(Data_i)$, which represents the performance level on the ith dataset, which is specified as:

$$O(Data_i) = \frac{\sum_t^{allData} \sqrt{|(F(t) - Data(t))|}}{\sum_t^{all} \sqrt{|Data(t)|}} \tag{13}$$

For datasets with poor model performance, we should restructure the relevant factors according to the actual situation and retrain the F-function to make it more adaptable to datasets with specific characteristics.

4.4 Solving and Analysis

We searched for different datasets and tested the models for different datasets with the universal level evaluation model, and found that it performed well on datasets such as Men's Singles Championships, Red Clay Courts, etc., while it did not perform well on datasets such as Women's Singles Grand Slam, Hard Courts, Table Tennis, etc., which suggests that the latter should be taken into account with different factors and focuses, e.g., in Women's Singles matches, serve advantage in women's singles matches, the greater relationship between stroke speed and points scored or lost in Hard Courts, the greater emphasis on individual skill in table tennis matches and the lesser importance of fitness values, and so on.

In summary, our model generalizes better on datasets with more similar features, while on datasets that are not similar, elements such as the function structure should be changed according to the actual situation.

A Code of problem 1

A.1 problem1.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os

# Calculate the winning percentage of the serving side
'''
q_num=0
for k in range(7284):
value_data=Parameters.get(k)
if(value_data[0]==value_data[1]):
q_num+=1
q_goal=q_num/7284
print(q_goal)
'''

# Consider Gradient Descent Method to form a BP parameter o

# Set server as the ball right, pointer_victor as the autho
# Whether serve is direct depends on p1_ace/p2_ace

# Specify the folder path
folder_path = '/Users/guyue  //Mathematical_modeling/2024 _

# Get all files in the folder
all_files = os.listdir(folder_path)

# Filter out CSV files
csv_files = [file for file in all_files if file.endswith('.

all_Parameters=[]
all_player1_target=[]
all_player2_target=[]

# Loop over CSV files
for csv_file in csv_files:
# Construct complete file paths
file_path = os.path.join(folder_path, csv_file)

# Use pandas to read CSV files
data = pd.read_csv(file_path)
```

```
target_column_name = ['server','point_victor','p1_sets','p1
Parameters = {}
num_data=[]

for i in range(1000):
row_index = i
if row_index < len(data):
for j in target_column_name:
col_name = j
element = data.at[i, j]
if(element=='0'):
element=0
elif(element=='15'):
element=1
elif(element=='30'):
element=2
elif(element=='40'):
element=3
elif(element=='AD'):
element=4
num_data.append(element)
Parameters[row_index]=num_data
num_data=[]

# The Parameters dictionary contains the required data for
player1_target = {}
player2_target = {}

# Target Data: Possession (1, 2) x_r Points Scored (1, 2) N
for i in range(1000):
value_num = Parameters.get(i)
if value_num is not None:

player1_num = [0, 0, 0, 0, 0, 0, 0]  # Initialize player1_nu
player2_num = [0, 0, 0, 0, 0, 0, 0]  # Initialize player2_nu

con_num = 1

if value_num[1] == 1:
player1_num[0] = 1
player2_num[0] = 0
else:
player1_num[0] = 0
player2_num[0] = 1
```

```
if value_num[1] == 1:
for t in range(i - 1, 0, -1):
value_before = Parameters.get(t)
if value_before[1] == 1:
con_num += 1
else:
break
else:
for t in range(i - 1, 0, -1):
value_before = Parameters.get(t)
if value_before[1] == 2:
con_num += 1
else:
break
player1_num[1] = player2_num[1] = con_num

if value_num[10] == 1:
player1_num[2] = player2_num[2] = 0
elif value_num[10] == 2 and value_num[1] == 1:
player1_num[2] = 1
else:
player2_num[2] = 1

if value_num[8] == 1 and value_num[9] == 0:
player1_num[3] = 1
player2_num[3] = 0
elif value_num[8] == 0 and value_num[9] == 1:
player1_num[3] = 0
player2_num[3] = 1
else:
player1_num[3] = 0
player2_num[3] = 0

if value_num[0] == 1:
player1_num[4] = 1
player2_num[4] = 0
else:
player1_num[4] = 0
player2_num[4] = 1

if (int(value_num[4]) == 3 and int(value_num[7]) < 3) or (i
player1_num[5] = player2_num[5] = 1
else:
player1_num[5] = player2_num[5] = 0
```

```
if ((int(value_num[3]) == 6 and (int(value_num[6]) == 6 or
(int(value_num[3]) == 5 and int(value_num[6]) == 4)) or ((i
player1_num[6] = player2_num[6] = 1
else:
player1_num[6] = player2_num[6] = 0

player1_target[i] = player1_num.copy()
player2_target[i] = player2_num.copy()
all_Parameters.append(Parameters)
all_player1_target.append(player1_target)
all_player2_target.append(player2_target)

def T_fun(initial_parameters):
all_sum=0
for p in range(len(all_Parameters)):
player1_target=all_player1_target[p]
player2_target=all_player1_target[p]
Parameters=all_Parameters[p]

# The objective function F
def F_fun(initial_parameters,n,t):
q=0.6731191652937946
if t==1:
player=player1_target[n]
else:
player=player2_target[n]
k1, k2, k3, k4, k5, k6 = initial_parameters
F=((1-player[4])*(q/(1-q))+player[4])*player[0]*(1+k1*(play
return F

# The objective function Q
def Q_fun(initial_parameters, m, t):
a_sum=0
if m == 1:
return 1
else:
if t == 1:
for f in range(m-1):
a_sum=a_sum*0.95+F_fun(initial_parameters, f, 1)
else:
for f in range(m):
a_sum=a_sum*0.95+F_fun(initial_parameters, f, 2)
return a_sum
```

```
sum=0
for l in range(1000):
if l in Parameters:
value_list=Parameters[l]
if value_list[1]==1:
N=1
else:
N=0
sum+=((Q_fun(initial_parameters, l+1, 1) - Q_fun(initial_par
else:
break
all_sum+=sum
return all_sum


def gradient(initial_parameters):
del_initial_parameters_1=[0,0,0,0,0,0]
del_initial_parameters_2=[0,0,0,0,0,0]
del_initial_parameters_3=[0,0,0,0,0,0]
del_initial_parameters_4=[0,0,0,0,0,0]
del_initial_parameters_5=[0,0,0,0,0,0]
del_initial_parameters_6=[0,0,0,0,0,0]
i=1
for a in initial_parameters:
if i ==1:
del_initial_parameters_1[0]=a+0.001

del_initial_parameters_2[0]=a

del_initial_parameters_3[0]=a

del_initial_parameters_4[0]=a

del_initial_parameters_5[0]=a

del_initial_parameters_6[0]=a

elif i==2:
del_initial_parameters_1[1]=a

del_initial_parameters_2[1]=a+0.001

del_initial_parameters_3[1]=a

del_initial_parameters_4[1]=a
```

```
del_initial_parameters_5[1]=a

del_initial_parameters_6[1]=a

elif i==3:
del_initial_parameters_1[2]=a

del_initial_parameters_2[2]=a

del_initial_parameters_3[2]=a+0.001

del_initial_parameters_4[2]=a

del_initial_parameters_5[2]=a

del_initial_parameters_6[2]=a

elif i==4:
del_initial_parameters_1[3]=a

del_initial_parameters_2[3]=a

del_initial_parameters_3[3]=a

del_initial_parameters_4[3]=a+0.001

del_initial_parameters_5[3]=a

del_initial_parameters_6[3]=a

elif i==5:
del_initial_parameters_1[4]=a

del_initial_parameters_2[4]=a

del_initial_parameters_3[4]=a

del_initial_parameters_4[4]=a

del_initial_parameters_5[4]=a+0.001

del_initial_parameters_6[4]=a

elif i==6:
```

```
del_initial_parameters_1[5]=a

del_initial_parameters_2[5]=a

del_initial_parameters_3[5]=a

del_initial_parameters_4[5]=a

del_initial_parameters_5[5]=a

del_initial_parameters_6[5]=a+0.001
i+=1

df_k1=(T_fun(del_initial_parameters_1)-T_fun(initial_parame
df_k2=(T_fun(del_initial_parameters_2)-T_fun(initial_parame
df_k3=(T_fun(del_initial_parameters_3)-T_fun(initial_parame
df_k4=(T_fun(del_initial_parameters_4)-T_fun(initial_parame
df_k5=(T_fun(del_initial_parameters_5)-T_fun(initial_parame
df_k6=(T_fun(del_initial_parameters_6)-T_fun(initial_parame
gradients = []
gradients.append(df_k1)
gradients.append(df_k2)
gradients.append(df_k3)
gradients.append(df_k4)
gradients.append(df_k5)
gradients.append(df_k6)
return np.array(gradients)


# Gradient Descent Method
def gradient_descent(initial_params, learning_rate, max_ite
params = np.array(initial_params)
iteration = 0

while iteration < max_iterations:
grad = gradient(params)

params = params - learning_rate * grad

# Calculate the van of the gradient as a stopping condition
gradient_norm = np.linalg.norm(grad)

print(f"Iteration␣{iteration␣+␣1},␣Parameters:␣{params},␣Gr

if gradient_norm < tolerance:
```

```
            break

            iteration += 1

            print(T_fun(params))

            return params

            # Initial parameters, learning rate, maximum number of itera
            initial_params = [ 110.07021533,  26.13556517,  230.1001945
20,   20]
            learning_rate = 100
            max_iterations = 200
            tolerance = 1e-6

            # Execute the Gradient Descent Method
            result = gradient_descent(initial_params, learning_rate, ma

            print("Optimal parameters:", result)
            print("Optimal function values:", T_fun(result))
```

A.2 problem1-picture.py

```
            import pandas as pd
            import matplotlib.pyplot as plt
            import numpy as np


            file_path = '/Users/guyue  //Mathematical_modeling/2024 _  /
            data = pd.read_csv(file_path)

            target_column_name = ['server','point_victor','p1_sets','p1_
            Parameters = {}
            num_data=[]

            for i in range(1000):
            row_index = i
            if row_index < len(data):
            for j in target_column_name:
            col_name = j
            element = data.at[i, j]
            if(element=='0'):
            element=0
            elif(element=='15'):
```

```
element=1
elif(element=='30'):
element=2
elif(element=='40'):
element=3
elif(element=='AD'):
element=4
num_data.append(element)
Parameters[row_index]=num_data
num_data=[]

#The Parameters dictionary contains the required data for ea
player1_target = {}
player2_target = {}

# Target Data: Possession (1, 2) x_r Points Scored (1, 2) N
for i in range(1000):
value_num = Parameters.get(i)
if value_num is not None:

player1_num = [0, 0, 0, 0, 0, 0, 0]  # Initialize player1_nu
player2_num = [0, 0, 0, 0, 0, 0, 0]  # Initialize player2_nu

con_num = 1

if value_num[1] == 1:
player1_num[0] = 1
player2_num[0] = 0
else:
player1_num[0] = 0
player2_num[0] = 1

if value_num[1] == 1:
for t in range(i - 1, 0, -1):
value_before = Parameters.get(t)
if value_before[1] == 1:
con_num += 1
else:
break
else:
for t in range(i - 1, 0, -1):
value_before = Parameters.get(t)
if value_before[1] == 2:
con_num += 1
else:
```

```
                    break
                    player1_num[1] = player2_num[1] = con_num

                    if value_num[10] == 1:
                    player1_num[2] = player2_num[2] = 0
                    elif value_num[10] == 2 and value_num[1] == 1:
                    player1_num[2] = 1
                    else:
                    player2_num[2] = 1

                    if value_num[8] == 1 and value_num[9] == 0:
                    player1_num[3] = 1
                    player2_num[3] = 0
                    elif value_num[8] == 0 and value_num[9] == 1:
                    player1_num[3] = 0
                    player2_num[3] = 1
                    else:
                    player1_num[3] = 0
                    player2_num[3] = 0

                    if value_num[0] == 1:
                    player1_num[4] = 1
                    player2_num[4] = 0
                    else:
                    player1_num[4] = 0
                    player2_num[4] = 1

                    if (int(value_num[4]) == 3 and int(value_num[7]) < 3) or (i
                    player1_num[5] = player2_num[5] = 1
                    else:
                    player1_num[5] = player2_num[5] = 0

                    if ((int(value_num[3]) == 6 and (int(value_num[6]) == 6 or
                    (int(value_num[3]) == 5 and int(value_num[6]) == 4)) or ((i
                    player1_num[6] = player2_num[6] = 1
                    else:
                    player1_num[6] = player2_num[6] = 0

                    player1_target[i] = player1_num.copy()
                    player2_target[i] = player2_num.copy()

                    def F_fun(initial_parameters,n,t):
                    q=0.6731191652937946
                    if t==1:
                    player=player1_target[n]
```

```
else:
player=player2_target[n]
k1, k2, k3, k4, k5, k6 = initial_parameters
F=((1-player[4])*(q/(1-q))+player[4])*player[0]*(1+k1*(play
return F

y=[]
yy=[]

for i in range(len(Parameters)):
y.append(F_fun([56.46144093   ,-26.96010033   ,195.48414152  ,2
yy.append(F_fun([56.46144093   ,-26.96010033   ,195.48414152

y1=[]
y2=[]
for i in range(len(y)):
a = 0
b = 0
for j in range(i+1):
a+=y[j]
b+=yy[j]
y1.append(a)
y2.append(b)
# The horizontal coordinates are the sequence of integers 1
x = list(range(1, len(y) + 1))

dy=[]
for i in range(len(y)):
dy.append(y1[i]-y2[i])
'''
degree = 10
coefficients_y = np.polyfit(x, y, degree)
coefficients_yy = np.polyfit(x, yy, degree)

fit_y = np.polyval(coefficients_y, x)
fit_yy = np.polyval(coefficients_yy, x)
'''


plt.plot(x, y, label='Player 1',linewidth=1)
plt.plot(x, yy, label='Player 2',linewidth=1)

#plt.plot(x, fit_y, label='Player 1')
#plt.plot(x, fit_yy, label='Player 2')
```

```
# Plot line graphs
#plt.plot(x, y, label='Player 1')
#plt.plot(x, yy, label='Player 2')
#plt.plot(x, dy, label='Momentum Trend')

# Set up chart titles and axis labels
#plt.title('Momentum Vision')
plt.xlabel('Points')
plt.ylabel('performance␣evaluation')

# Add legends
plt.legend()

# Display charts
plt.show()
```

B Code of problem 2

B.1 problem2.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os

# Calculate the winning percentage of the serving side
'''
q_num=0
for k in range(7284):
value_data=Parameters.get(k)
if(value_data[0]==value_data[1]):
q_num+=1
q_goal=q_num/7284
print(q_goal)
'''

# Consider Gradient Descent Method to form a BP parameter o

# Set server as the ball right, pointer_victor as the autho
# Whether serve is direct depends on p1_ace/p2_ace

# Specify the folder path
folder_path = '/Users/guyue  //Mathematical_modeling/2024 _

# Get all files in the folder
all_files = os.listdir(folder_path)

# Filter out CSV files
csv_files = [file for file in all_files if file.endswith('.

all_Parameters=[]
all_player1_target=[]
all_player2_target=[]

# Loop over CSV files
for csv_file in csv_files:
# Construct complete file paths
file_path = os.path.join(folder_path, csv_file)

# Use pandas to read CSV files
data = pd.read_csv(file_path)
```

```
target_column_name = ['server','point_victor','p1_sets','p1
Parameters = {}
num_data=[]

for i in range(1000):
row_index = i
if row_index < len(data):
for j in target_column_name:
col_name = j
element = data.at[i, j]
if(element=='0'):
element=0
elif(element=='15'):
element=1
elif(element=='30'):
element=2
elif(element=='40'):
element=3
elif(element=='AD'):
element=4
num_data.append(element)
Parameters[row_index]=num_data
num_data=[]

# The Parameters dictionary contains the required data for
player1_target = {}
player2_target = {}

# Target Data: Possession (1, 2) x_r Points Scored (1, 2) N

for i in range(1000):
value_num = Parameters.get(i)
if value_num is not None:

player1_num = [0, 0, 0, 0, 0, 0, 0]  # Initialize player1_nu
player2_num = [0, 0, 0, 0, 0, 0, 0]  # Initialize player2_nu

con_num = 1

if value_num[1] == 1:
player1_num[0] = 1
player2_num[0] = 0
else:
player1_num[0] = 0
```

```
player2_num[0] = 1

if value_num[1] == 1:
for t in range(i - 1, 0, -1):
value_before = Parameters.get(t)
if value_before[1] == 1:
con_num += 1
else:
break
else:
for t in range(i - 1, 0, -1):
value_before = Parameters.get(t)
if value_before[1] == 2:
con_num += 1
else:
break
player1_num[1] = player2_num[1] = con_num

if value_num[10] == 1:
player1_num[2] = player2_num[2] = 0
elif value_num[10] == 2 and value_num[1] == 1:
player1_num[2] = 1
else:
player2_num[2] = 1

if value_num[8] == 1 and value_num[9] == 0:
player1_num[3] = 1
player2_num[3] = 0
elif value_num[8] == 0 and value_num[9] == 1:
player1_num[3] = 0
player2_num[3] = 1
else:
player1_num[3] = 0
player2_num[3] = 0

if value_num[0] == 1:
player1_num[4] = 1
player2_num[4] = 0
else:
player1_num[4] = 0
player2_num[4] = 1

if (int(value_num[4]) == 3 and int(value_num[7]) < 3) or (i
player1_num[5] = player2_num[5] = 1
else:
```

```
player1_num[5] = player2_num[5] = 0

if ((int(value_num[3]) == 6 and (int(value_num[6]) == 6 or
(int(value_num[3]) == 5 and int(value_num[6]) == 4)) or ((i
player1_num[6] = player2_num[6] = 1
else:
player1_num[6] = player2_num[6] = 0

player1_target[i] = player1_num.copy()
player2_target[i] = player2_num.copy()
all_Parameters.append(Parameters)
all_player1_target.append(player1_target)
all_player2_target.append(player2_target)

def T_fun(initial_parameters):
all_sum=0
for p in range(len(all_Parameters)):
player1_target=all_player1_target[p]
player2_target=all_player1_target[p]
Parameters=all_Parameters[p]

# The objective function F
def F_fun(initial_parameters,n,t):
q=0.6731191652937946
if t==1:
player=player1_target[n]
else:
player=player2_target[n]
k1, k2, k3, k4, k5, k6 = initial_parameters
F=((1-player[4])*(q/(1-q))+player[4])*player[0]*(1+k1*(play
return F

sum=0
for l in range(len(player1_target)):
if l in Parameters:
value_list=Parameters[l]
if value_list[1]==1:
N=1
else:
N=0
sum+=((F_fun(initial_parameters, l, 1) - F_fun(initial_param
else:
break
all_sum+=sum
return all_sum
```

```
def gradient(initial_parameters):
del_initial_parameters_1=[0,0,0,0,0,0]
del_initial_parameters_2=[0,0,0,0,0,0]
del_initial_parameters_3=[0,0,0,0,0,0]
del_initial_parameters_4=[0,0,0,0,0,0]
del_initial_parameters_5=[0,0,0,0,0,0]
del_initial_parameters_6=[0,0,0,0,0,0]
i=1
for a in initial_parameters:
if i ==1:
del_initial_parameters_1[0]=a+0.001

del_initial_parameters_2[0]=a

del_initial_parameters_3[0]=a

del_initial_parameters_4[0]=a

del_initial_parameters_5[0]=a

del_initial_parameters_6[0]=a

elif i==2:
del_initial_parameters_1[1]=a

del_initial_parameters_2[1]=a+0.001

del_initial_parameters_3[1]=a

del_initial_parameters_4[1]=a

del_initial_parameters_5[1]=a

del_initial_parameters_6[1]=a

elif i==3:
del_initial_parameters_1[2]=a

del_initial_parameters_2[2]=a

del_initial_parameters_3[2]=a+0.001

del_initial_parameters_4[2]=a
```

```
del_initial_parameters_5[2]=a

del_initial_parameters_6[2]=a

elif i==4:
del_initial_parameters_1[3]=a

del_initial_parameters_2[3]=a

del_initial_parameters_3[3]=a

del_initial_parameters_4[3]=a+0.001

del_initial_parameters_5[3]=a

del_initial_parameters_6[3]=a

elif i==5:
del_initial_parameters_1[4]=a

del_initial_parameters_2[4]=a

del_initial_parameters_3[4]=a

del_initial_parameters_4[4]=a

del_initial_parameters_5[4]=a+0.001

del_initial_parameters_6[4]=a

elif i==6:
del_initial_parameters_1[5]=a

del_initial_parameters_2[5]=a

del_initial_parameters_3[5]=a

del_initial_parameters_4[5]=a

del_initial_parameters_5[5]=a

del_initial_parameters_6[5]=a+0.001
i+=1
```

```
df_k1=(T_fun(del_initial_parameters_1)-T_fun(initial_parame
df_k2=(T_fun(del_initial_parameters_2)-T_fun(initial_parame
df_k3=(T_fun(del_initial_parameters_3)-T_fun(initial_parame
df_k4=(T_fun(del_initial_parameters_4)-T_fun(initial_parame
df_k5=(T_fun(del_initial_parameters_5)-T_fun(initial_parame
df_k6=(T_fun(del_initial_parameters_6)-T_fun(initial_parame
gradients = []
gradients.append(df_k1)
gradients.append(df_k2)
gradients.append(df_k3)
gradients.append(df_k4)
gradients.append(df_k5)
gradients.append(df_k6)
return np.array(gradients)


# Gradient Descent Method
def gradient_descent(initial_params, learning_rate, max_ite
params = np.array(initial_params)
iteration = 0

while iteration < max_iterations:
grad = gradient(params)

params = params - learning_rate * grad

# Calculate the van of the gradient as a stopping condition
gradient_norm = np.linalg.norm(grad)

print(f"Iteration␣{iteration␣+␣1},␣Parameters:␣{params},␣Gr

if gradient_norm < tolerance:
break

iteration += 1

print(T_fun(params))

return params

# Initial parameters, learning rate, maximum number of itera
initial_params = [ 10,  26,  23, 73,   20,   20]
learning_rate = 100
max_iterations = 200
tolerance = 1e-6
```

```
# Execute the Gradient Descent Method
result = gradient_descent(initial_params, learning_rate, ma

print("Optimal parameters:", result)
print("Optimal function values:", T_fun(result))
```