Chapter 5 Programming Assignments

王迦楠　数学科学学院　求数 2101　3210105175

2024 年 2 月 4 日

restatement of the problem

First Problem momentum is “strength or force gained by motion or by a series of events.” It directly impacts the player’s performance in subsequent points. To assess the players’ performance, it is crucial to have a clear understanding of ”momentum.” We will focus on the following tasks:

1. determine the influencing factors of ”momentum”
2. Quantify the variations in ”momentum” using fluctuating data.
3. Visualize the process of ”momentum” changes.

1 Model Review

In this section, we will primarily focus on establishing a model using the Analytic Hierarchy Process (AHP) to address problems 1 and 2. We will break down the problems into five parts:

1. Problem Analysis
2. Data Cleaning and Processing
3. Collinearity Detection
4. Analytic Hierarchy Process (AHP)

1.1 Problem Analysis

To investigate the reasons behind "momentum," we first need to provide a preliminary definition for "momentum." The magnitude of "momentum" is defined as

$$f_{ijk} = \boldsymbol{\omega} \cdot \boldsymbol{x_{ijk}}$$

where:

1. f_{ijk} represents the "momentum" of player k before the jth point number in the ith match (in the order given by the table).

2. $\boldsymbol{x_{ijk}}$ is an n-dimensional column vector representing some influencing factors at the corresponding moment. Specific details will be provided later.

3. $\boldsymbol{\omega}$ is an n-dimensional row vector indicating the specific weights of the influencing factors, which will be obtained through the Analytic Hierarchy Process (AHP).

4. In this formula, there are two different calculation methods, one representing rounds where the player serves and the other representing rounds where the opponent serves. We can express it as

$$\boldsymbol{\omega} = \boldsymbol{\omega_0} \circ \boldsymbol{\delta} = (\omega_0^{(0)}\delta^{(0)}, \omega_0^{(1)}\delta^{(1)}, \ldots, \omega_0^{(n)}\delta^{(n)})$$

   representing a vector formed by element-wise multiplication of two vectors of the same dimension. Here, $\boldsymbol{\delta}$ is a $0, 1$ vector indicating whether it is the player's serving round. In the specific calculation, we will consider two cases separately.

For the specific definition of $\boldsymbol{x_{ij}^n}$, we believe that, in addition to whether the player is serving, many other factors can have an impact, including the player's skills, fatigue level, and real-time mental state of the game (here, we mainly consider these three points). Based on these three main aspects, we have organized 12 factors as preliminary influencing factors, as follows:

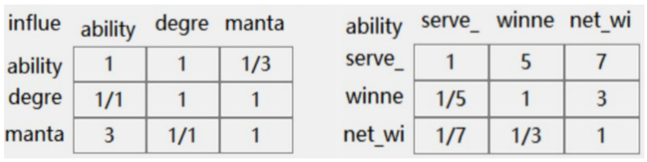
1.2 Data Processing and Normalization

1.3 Collinearity Detection

After processing the data, considering the potential collinearity among factors within the same category, such as serving aces, first-serve scoring rate, and whether the previous point was scored may be correlated, as well as running distance and the number of strokes possibly being related, we conducted collinearity detection using Stata. The results of the detection indicate a significant variance inflation factor between running distance and the number of strokes. Therefore, we decided to exclude one of them, choosing to retain the remaining 11 variables for the Analytic Hierarchy Process (AHP).

1.4 Analytic Hierarchy Process

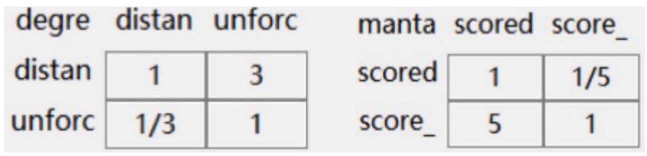
We have previously decomposed the included factors from top to bottom into several levels, where factors within the same level are subordinate to factors in the level above or influence factors in the level above. They also dominate factors in the next level or are influenced by factors in the next level. Starting from the second level of the hierarchy, we construct comparison matrices for each factor influencing the factor in the level above, until reaching the bottom level. Each element in the matrix indicates the preference level between factor i and factor j at the same level. It is essential to note that we have separately established a series of such comparison matrices for two different serving types (serving by oneself and serving by the opponent). Here, we illustrate the matrix using serving by oneself as an example:



influe	ability	degre	manta
ability	1	1	1/3
degre	1/1	1	1
manta	3	1/1	1

ability	serve_	winne	net_wi
serve_	1	5	7
winne	1/5	1	3
net_wi	1/7	1/3	1

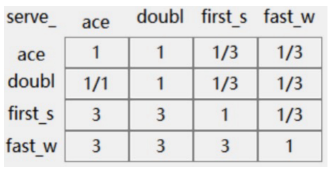
图 1: Comparison matrix for influencing factors and ability



degre	distan	unforc
distan	1	3
unforc	1/3	1

manta	scored	score_
scored	1	1/5
score_	5	1

图 2: Comparison matrix for degree of fatigue and mantality



serve_	ace	doubl	first_s	fast_w
ace	1	1	1/3	1/3
doubl	1/1	1	1/3	1/3
first_s	3	3	1	1/3
fast_w	3	3	3	1

图 3: Comparison matrix for serving

We obtain the weights for each component by calculating the maximum eigenvalue and normalizing its corresponding eigenvector. Certainly, for each matrix, we first need to test consistency using the Consistency Ratio (CR), where $CR = \frac{CI}{RI}$, $CI = \frac{\lambda_{max}-n}{n-1}$, $RI = 0.0, 0.58, 0.9$ (for matrices of size $2, 3, 4$). The computed Consistency Ratios for the matrices are 0.076, 0.037, 0.0, 0.0, 0.046. Since they are all less than 0.1, it confirms the consistency of the matrices.

Therefore, the weights for our model are as follows:

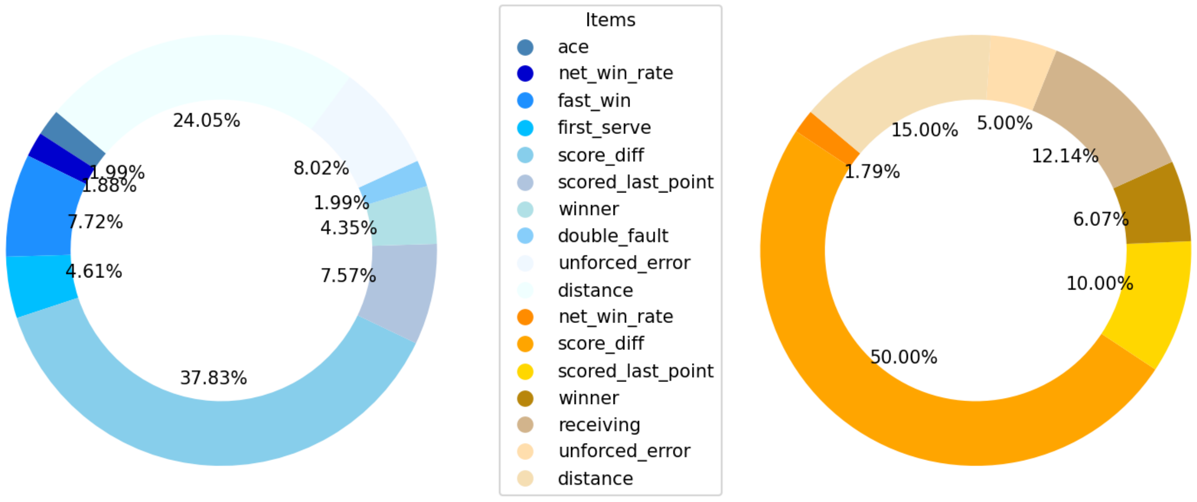


图 4: Weights in two different situations

Analyzing the various factors in the chart, it is evident that the most impactful factor is whether the previous point was scored. Following closely is the distance covered during the play, which aligns well with common intuition.

Thus, our final momentum is defined as:(need to change symbol)

$$momentum = \begin{cases} \sum_{n=1,n\neq 5}^{11} \omega_n x_n, & \text{if the player serves} \\ \sum_{n=5}^{11} \omega_n x_n, & \text{if the opponent serves} \end{cases}$$

Where $\omega_n (n = 1, \ldots, 11)$ represent the weight of the factors, which is listed in Figure 4.

Now, we illustrate the graph of the ”momentum” in the first match:

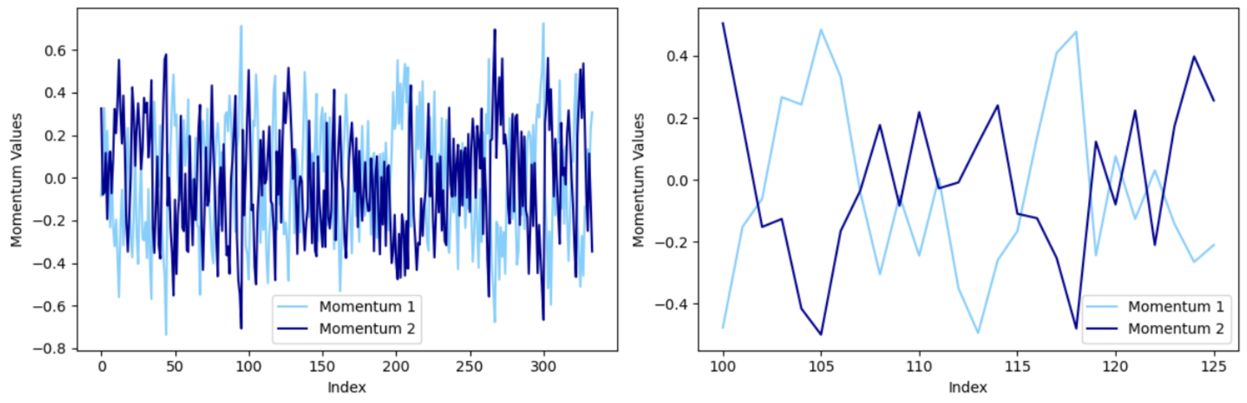


图 5: Momentum change in the first competition(global and local)

It can be observed that the variation in ”momentum” is a process of give and take.

Third Problem model overview In this section, we will primarily focus on establishing a model using the Gated Recurrent Unit(GRU) algorithm to address problem 3 . We will break down the problems into five parts:

1. definition of the swing of the play
2. Gated Recurrent Unit(GRU) algorithm
3. Permutation Feature Importance theory
4. code
5. second part analysis

1.5 definition of the swing of the play

We first give the definition of the swings of the play. Based on our previous definition of ”momentum”, the significant changes of the game largely depend on the ”momentum” of the two players. Therefore, we choose ”momentum” to represent the swings of the play. The specific definition is as follows:

We use $\Delta f(t)$ to represent difference in ”momentum” between the two players., so it can be easily seen that if $\Delta f(t)$ and $\Delta f(t+1)$ has different signs, it indicates a ”swing” in the game’s momentum. In this way, we can define four states of ”momentum” at time t:

$$states = \begin{cases} state1, & \text{if } \Delta f(t) < 0 \text{ and } \Delta f(t+1) > 0, \text{ which means rise from negative to positive} \\ state2, & \text{if } \Delta f(t) > 0 \text{ and } \Delta f(t+1) > 0, \text{ which means stay positive} \\ state3, & \text{if } \Delta f(t) < 0 \text{ and } \Delta f(t+1) < 0, \text{ which means stay negative} \\ state4, & \text{if } \Delta f(t) < 0 \text{ and } \Delta f(t+1) > 0, \text{ which means decrease from positive to negative} \end{cases}$$

（这里可以画一张图展示四种变化）Obviously, if state 1 or state 4 appears, we can determine that a swing has occurred.

1.6 Indroduction of Gated Recurrent Unit(GRU) algorithm

GRU (Gated Recurrent Unit) is a type of recurrent neural network (RNN). It is similar to the LSTM (Long Short-Term Memory). But compared to LSTM, GRU is more effective and easier to train.

We will focus on a single unit of recurrent neural network to interpret the hidden mathematical principles:

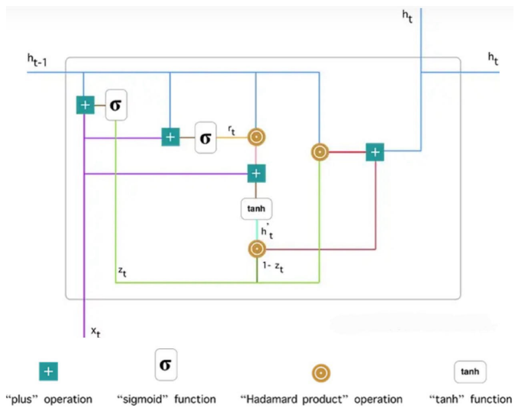


图 6: Structure of GRU

1. The update gate at time step t is computed using the following formula:

$$z_t = \sigma(W_z \cdot x_t + U_z \cdot h_{t-1})$$

   Here, when x_t is input to the network unit, it is multiplied by its own weight W_z. Similarly, h_{t-1}, which holds the information from the previous $t-1$ units, is multiplied by its own weight U_z. These two results are then summed together and passed through a sigmoid activation function to compress the result between 0 and 1.

2. The essence of the reset gate is for the model to decide how much past information to forget. To compute it, we use:

$$r_t = \sigma(W_r \cdot x_t + U_r \cdot h_{t-1})$$

3. The computation of the new memory content using the reset gate is as follows:

$$h'_t = tanh(Wx_t + r_t \odot Uh_{t-1})$$

4. The computation for the new memory content h_t using the update gate is as follows:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t$$

1.7 Permutation Feature Importance theory

We measure the importance of a feature by calculating the increase in the model's prediction error after permuting the feature. A feature is "important" if shuffling its values increases the model error, because in this case the model relied on the feature for the prediction. A feature is "unimportant" if shuffling its values leaves the model error unchanged, because in this case the model ignored the feature for the prediction.

Algorithm 1: Permutation Feature Importance

```
1 Input: Trained model $\hat{f}$, feature matrix $X$, target vector $y$, error measure $L(y, \hat{f})$.
2 Estimate the original model error $e_{orig} = L(y, \hat{f}(X))$ (e.g. mean squared error)
3 For each feature $j \in \{1, ..., p\}$ do:
4     ∘ Generate feature matrix $X_{perm}$ by permuting feature $j$ in the data $X$. This
      breaks the association between feature $j$ and true outcome $y$.
5     ∘ Estimate error $e_{perm} = L(Y, \hat{f}(X_{perm}))$ based on the predictions of the
      permuted data.
6     ∘ Calculate permutation feature importance as quotient $FI_j = e_{perm}/e_{orig}$ or
      difference $FI_j = e_{perm} - e_{orig}$
7 Sort features by descending FI.
```