基于重心估算与集成学习的跳远解析及预测模型

摘要

当下人工智能高速发展，AI 辅助智能体测可以显著提高体测成绩的检出效率。AI 姿态估计定位人体关键点可以高效赋能跳远成绩检出。

问题一：对附件 1 的数据，先通过数据清洗与对数据进行平滑处理，**基于 Kinect 人体重心估算方法**，结合体段质量系数建立重心位置估算模型；采用数值微分法提取重心速度、加速度及 X 方向斜率等运动特征，通过自适应斜率阈值与多条件约束优化问题构建起跳与落地时刻检测算法，同时利用 **k-means 聚类**对运动阶段进行划分，最终通过**二次多项式拟合**滞空阶段重心轨迹。结果显示，运动者 1 的起跳时刻为 122-124 帧、落地时刻为 140-142 帧、滞空时间约 0.7 秒，运动者 2 的起跳时刻为 162-166 帧、落地时刻为 177-180 帧、滞空时间约 0.5 秒，且两组数据比例尺基本一致，X 方向轨迹拟合度均高于 0.98，明确了从起跳到落地过程中重心的上抛、下落阶段特征及关键节点运动规律。

问题二：结合附件 3 、4 中姿势纠正前后的视频、位置信息、成绩与体质数据。首先通过问题一的运动阶段检测算法确定滞空阶段，计算关节角度与关节运动的幅度等运动学特征；采用 **Lasso 特征选择与主成分分析（PCA）**进行特征降维与筛选，通过 SHAP 值分析特征重要性，并利用决定系数、均方根误差等指标验证模型性能（R^2=0.9845，RMSE=0.057 米）。结果表明，影响成绩的因素可分为体质与姿态两类：体质特征中，肌肉骨量比、爆发力指数、基础代谢为核心因素，直接决定蹬地力量与能量供应；姿态特征中，起跳角度、落地时刻臂关节角度为关键因素，影响腾空轨迹与落地稳定性，腿部动作幅度则通过动量传递辅助提升成绩，两类因素共同影响跳远成绩。

问题三：基于问题一的时刻判断与轨迹分析模型及问题二的影响因素模型，构建基于弹性网回归、XGBoost 与随机森林的**集成学习模型**，以分析影响跳远成绩的关键因素。利用附件 5 中运动者 11 的关键节点数据和附件 4 的体质信息，通过余弦相似度验证其与运动者 9 的比例尺一致性，采用集成学习模型预测成绩，并结合像素换算验证。最终确定运动者 11 的跳远成绩约为 1.13 米，预测结果可信度高。

问题四：针对 6 岁运动者 11 的体质特点及跳远姿态特征，从姿势矫正和体能爆发力两方面提出训练建议：通过标志物站位、弹力带摆臂等矫正姿势，以平板支撑、蹲起跳等提升体能。基于问题二的影响因素模型估算，经 4-6 周训练，其成绩有望从 1.13 米提升至 1.3-1.5 米区间，契合儿童神经肌肉系统[12][13]的学习适应特点。

关键词：人体重心、k-means 聚类、多项式拟合、集成学习模型、主成分分析

一、 问题重述

问题一 ：已知附件 1 中包含两位立定跳远运动者的跳远视频、33 个关键节点在不同帧的位置坐标以及跳远成绩。需通过分析这些数据，比对两组数据，验证两组数据比例尺是否一致，并且找到需要分析的关键节点以确定运动者在跳远过程中的起跳时刻和落地时刻，并详细描述从起跳到落地的滞空阶段中，运动者的运动过程，寻找分析出身体具有代表性的各关键节点的位置变化、运动轨迹等特征。

问题二：附件 3 提供了部分立定跳远运动者在接受教练姿势纠正前、后的跳远视频、位置信息及跳远成绩，附件 4 则包含这些运动者的个人体质报告（涵盖年龄、身高、体重、体脂率等信息）。要求结合这些资料，分析影响运动者跳远成绩的主要因素，明确区分与身体姿态相关的因素和与个人体质相关的因素，并阐述各因素对成绩的具体影响机制。

问题三：基于问题 1 中确定的起跳与落地时刻判断方法、滞空阶段运动过程分析模型，以及问题 2 中得出的影响跳远成绩的主要因素及其作用关系，结合附件 5 中运动者 11 的跳远视频、关键节点位置信息以及附件 4 中该运动者的个人体质信息，构建预测模型并计算该运动者的实际跳远成绩。

问题四：在问题 3 对运动者 11 跳远成绩预测的基础上，结合其在跳远过程中的姿态特征（从位置信息和视频中提取）以及个人体质特点，提出针对该运动者的、可在短时间内提升其跳远成绩的姿势训练建议（需具体到身体各部位的动作纠正要点）。同时，根据训练建议的预期效果和问题 2 中的影响因素模型，估算该运动者经过短期训练后可能达到的理想跳远成绩。

二、模型假设

关键节点数据有效性假设：附件中运动者的 33 个关键节点数据大部分准确可靠，仅仅存在个别明显误差数据且不影响分析。

关键节点数据统一性假设：同一运动者在不同帧的关键节点坐标具有统一的空间坐标系和比例尺，不同运动者的数据经坐标系转换后可进行横向对比。

起跳与落地时刻判断依据假设：起跳时刻定义为双脚脚掌最后一次与地面接触的帧，此时脚踝关键节点的垂直速度由零开始向上突变；落地时刻定义为双脚脚掌首次与地面接触的帧，此时脚踝关键节点的垂直速度由向下变为零。

滞空阶段运动规律假设：运动者在滞空阶段仅受重力和空气阻力作用，且空气阻力远小于重力，可近似认为身体重心做斜抛运动。

影响因素独立性假设：影响跳远成绩的体质因素（身高、体重、体脂率等）与姿态因素（起跳角度、蹬地力量、手臂摆动幅度等）相互独立，且各因素对成绩的影响具有可加性。

视频与坐标同步性假设：附件中的跳远视频与关键节点坐标数据严格同步，每帧视频对应一组唯一的坐标数据，时间戳偏差小于 0.01 秒，确保运动过程分析的时空一致性。

运动阶段划分假设：立定跳远过程可划分为 “预备 - 助跑（预摆）”“起跳”“滞空”“落地” 四个连续阶段，阶段间无重叠，且各阶段的运动特征（速度、加速度、关节角度）存在显著差异，可通过聚类算法有效区分。

体段质量系数稳定性假设：人体各体段（头颈部、胸部、四肢等）的质量系数（体段质量 / 总质量）符合成年或儿童标准生物力学参数（参考纪仲秋等实测数据），且在跳远过程中保持稳定，不受瞬时动作姿态变化影响。

集成模型基模型独立性假设：集成学习模型中弹性网回归、XGBoost、随机森林三种基模型的预测误差相互独立，通过加权融合可降低整体模型的方差，提升预测稳健性。

儿童运动适应性假设：针对运动者 11（6 岁儿童）的训练建议假设其神经肌肉系统对姿势矫正的学习适应速度符合 6-10 岁儿童平均水平，短期训练（4-6 周）内可形成稳定的正确动作记忆，且体能提升与年龄增长的叠加效应可忽略。

模型误差可控性：通过 “关节角度计算偏差≤5° ”“误差占比低于 5%”等量化约束，明确了假设的边界条件，使模型建立在 “误差可控” 的前提下，提升了结果的可信度。

三、符号说明

符号	说明
Com_t	重心坐标
l_k	k体段总量占比系数
C_k	k体段基于运动学的重心位置估算
m_{xx}	单个体段质量
M_{body}	人体总质量
$a_x(t)$	第 t 帧重心在 X 轴的加速度
ω	移动窗口大小
j	窗口内帧索引
$S(t)$	综合评分函数
ω_{1-4}	权重系数
R^2	决定系数
$f(t)$	第 t 帧的特征向量
μ_f	特征向量均值
σ_f	特征向量标准差
$y \in \mathbb{R}^+$	立定跳远成绩（米）
$X_{pose} \in \mathbb{R}^{n \times d_{pose}}$	姿态特征矩阵
$X_{condition} \in 0,1^n$	整前后变量
$\epsilon \sim N(0,\sigma^2)$	随机误差项
$L(y_i,\hat{y}_i) = (y_i - \hat{y}_i)^2$	均方差损失
$R(\theta) = \alpha\|\|\theta\|\|_1 + (1-\alpha)\|\|\theta\|\|_2^2$	弹性正则化
λ,α	正则化参数
$\vec{P}_1,\vec{P}_2,\vec{P}_3$	构成关节的三个关键点
D_{total}	总轨迹长度
$D_{horizontal}$	水平位移
$D_{vertical}$	垂直位移
A_t	关节幅度
$\Delta\theta_t$	角速度变化均值
U	特征向量矩阵
Λ	特征值对角矩阵
$k_x(t)$	x方向斜率

四、模型建立与求解

4.1、对问题一的模型建立与求解

4.1.1、对问题一分析

问题一要求确定运动者在跳远过程中的起跳和落地时刻，并且对滞空阶段的运动过程进行描述。滞空阶段为运动者双脚离开地面的时间。当确定起跳与落地时刻，则可确定滞空阶段。根据附件一，得到两组 33 个关键节点的数据、运动者的跳远视频以及运动者的跳远成绩。通过对视频观察，选择运动者的脚部关键节点进行分析，确定起跳与落地阶段、滞空阶段。利用多项式回归分析身体重心的空间轨迹分布、重心运动的力学特征。

4.1.2、对问题一的模型建立

对附件一数据进行分析处理，对数据中全零错误项取上下相邻两项平均值，得到清洗后的数据。对附件一数据的做平滑处理。基于 kinect 的人体步态分析和重心估算研究方法[1]确定运动者的较为精确的重心位置，由于立定跳远运动过程中，若运动者不受除重力以及空气阻力外其他力的作用时，运动者重心在起跳和落地时刻变化最为明显，由此分析重心的速率变化，可以分析出运动者的起跳与落地的时刻，并且拟合重心在滞空阶段的重心运动轨迹，分析运动者滞空阶段的运动过程。

（1）运动者重心估算

基于 kinect 的人体步态分析和重心估算方法（以下简称重心估算方法）[1]，通过体段质量系数替代重量参与重心计算，以确定重心在滞空阶段每一帧的像素位置。

重心估算方法中对体段质量系数的定义为：体段质量/人体总质量，即：

$$\text{体段质量系数} = \frac{m_{xx}}{M_{body}} \tag{1}$$

式中，m_{xx} 为单个体段质量（忽略重力加速度影响且质量与重量数值成正比），M_{body} 为人体总质量。

将此关系带入重心估算方法核心重心公式：

$$Com_{body}=\frac{1}{M_{body}}\sum(m_{XX}\cdot Com_{XX}) \tag{2}$$

替换为：

$$Com_{body}=\frac{1}{M_{body}}\sum((\text{体段质量系数}\times M_{body})\cdot Com_{XX}) \tag{3}$$

对式中M_{body}进行抵消，简化可得：

$$Com_{body}=\sum(\text{体段质量系数}\cdot Com_{XX}) \tag{4}$$

引入人体区域划分及系数表[2]结合附件 2 人体点对应关系图：

表 1 人体区域划分及系数表[2]

体段部分	体段质量系数（体段质量/总质量）	端系数	
		进端系数	远端系数
头颈部	0.081	1.000	空
胸部和腹部	0.355	0.500	0.500
上臂部	0.028	0.436	0.564
小臂部	0.016	0.430	0.570
手部	0.006	0.506	0.494
骨盆部	0.142	0.105	0.895
大腿部	0.100	0.433	0.567
小腿部	0.0465	0.433	0567
足部	0.0145	0.500	0.500

基于此将人体划分为 15 个独立体段区域，根据公式（1）计算得出独立体段区域所占重心的权重，如下表所示：

表 2 独立体段区域人体权重表

体段区域	对应人体点	权重
头颈部	0-10	0.081
胸部和腹部	11、12	0.355
左上臂	14	0.028
右上臂	13	0.028
左手	16	0.06

右手	15	0.06
骨盆	24、23	0.142
左大腿	25	0.1
右大腿	26	0.1
左小腿	27	0.0465
右小腿	28	0.0465
左足	29、31	0.0145
右足	30、32	0.0145

由独立体段区域人体权重表可以得到各体段的总量占比系数，设为l_k，表示为k体段占总量的占比系数。由此获得重心估算方程：

$$Com_t = \sum_{k=1}^{15} \frac{m_k C_{correction}}{M_{body}} = \sum_{k=1}^{15} l_k C_k + \sum_{k=1}^{15} l_k \tag{5}$$

式中，Com_t为重心位置，l_k表示k体段总量占比系数，C_k表示k体段基于运动学的重心位置估算。

（2）重心运动特征提取

对于重心运动特征的提取，需要重心的速度与加速度以及重心在 X 轴方向的斜率，在运动学参数计算上，通过数值微分方法计算重心的速度与加速度：

$$V_x(t) = \frac{dCom_x(t)}{dt} \approx \frac{Com_x(t+1) - Com_x(t-1)}{2\triangle t} \tag{6}$$

$$a_x(t) = \frac{dV_x(t)}{dt} \approx \frac{V_x(t+1) - V_x(t-1)}{2\triangle t} \tag{7}$$

式中，$V_x(t)$为重心沿 X 轴的速度，$a_x(t)$为重心沿 x 轴加速度。Y 轴同理。

对 x 方向的斜率计算，采用移动窗口线性回归计算局部斜率：

$$k_i = \arg\min_{k,b} \sum_{j=i-w/2}^{i+w/2} \left[COG_x(j) - (k \cdot j + b) \right]^2 \tag{8}$$

式中，k_i为第i帧对应的x方向局部斜率；w为移动窗口大小，用于局部趋势分析的帧数量；j为窗口内的帧索引；b为线性回归截距参数。

（3）起跳与落地时刻检测算法

当运动者处于起跳和落地时刻，在不受重力和空气阻力的影响下，其该时刻的重心变化熟虑为运动过程中的最大值。

对此，基于数据统计特征自适应确定斜率阈值：

$$\theta_{threshold} = \max(1.0, \mu_k + 2\sigma_k) \tag{9}$$

式中，$\theta_{threshold}$为斜率突变检测的阈值；μ_k为说有帧的斜率；σ_k为所有帧的斜率标准差；N为总帧数。

其中，

$$\mu_k = \frac{1}{N}\sum_{i=1}^{N} k_i \tag{10}$$

$$\sigma_k = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(k_i - \mu_k)^2} \tag{11}$$

起跳时刻的检测基于以下多条件约束优化问题：

$$t_{takeoff} = \arg\max_t S(t) \tag{12}$$

其中综合评价函数：

$$S(t) = w_1 \cdot I_1(t) + w_2 \cdot I_2(t) + w_3 \cdot I_3(t) + w_4 \cdot I_4(t) \tag{13}$$

指示函数定义为：

$$\begin{cases} I_1(t) = 1_{k_t > \theta_{threshold}} \\ I_2(t) = 1_{\bar{k}_{t-w:t} < 0.4\theta_{threshold}} \\ I_3(t) = 1_{\bar{k}_{t:t+w} > 0.6\theta_{threshold}} \\ I_4(t) = 1_{k_t - \bar{k}_{t-w:t} > 0.5\theta_{threshold}} \end{cases} \tag{14}$$

权重向量：

$$\mathbf{w} = [0.3, 0.2, 0.3, 0.2]^T \tag{15}$$

落地时刻通过检测斜率显著下降确定：

$$t_{landing} = \arg\min_{t > t_{takeoff} + \Delta t_{min}} \{t : k_t < 0.4\theta_{threshold} \wedge \overline{k}_{t-w:t} > 0.6\theta_{threshold}\} \tag{16}$$

式中，$t_{landing}$为落地时刻（索引帧）；Δt_{min}为最小滞空时长约束。

滞空阶段运动过程的描述，对运动者重心轨迹采用二次多项式拟合。

时间-位置关系：

$$\begin{cases} x(t) = a_x t^2 + b_x t + c_x \\ y(t) = a_y t^2 + b_y t + c_y \end{cases} \tag{17}$$

空间轨迹方程：

$$y = \alpha x^2 + \beta x + \gamma \tag{18}$$

式（17）、（18）中，$x(t)$、$y(t)$为滞空阶段第t帧重心的X、Y坐标；a_x、b_x、c_x为X方向二次多项式系数；α，β，γ为空间抛物线系数。

拟合质量评估，采用决定系数R^2：

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \overline{y})^2} \tag{19}$$

（4）运动阶段聚类分析

构建七维特征向量用于k-means聚类：

$$\mathbf{f}(t)=\begin{bmatrix} t/T_{total} \\ (x(t)-x_{min})/(x_{max}-x_{min}) \\ (y(t)-y_{min})/(y_{max}-y_{min}) \\ k_x(t) \\ trend_x(t) \\ v_x(t) \\ v_y(t) \end{bmatrix} \tag{20}$$

式中，$\mathbf{f}(t)$为第t帧的特征向量；T_{total}为总帧数；$trend_x(t)$为 X 方向的趋势特征。

再对特征向量采用 Z-score 标准化处理：

$$\mathbf{f}_{norm}(t)=\frac{\mathbf{f}(t)-\boldsymbol{\mu}_f}{\boldsymbol{\sigma}_f} \tag{21}$$

式中，$\mathbf{f}_{norm}(t)$为标准化后的特征向量；μ_f为特征向量均值；σ_f为特征向量标准差。

k-means 聚类：

$$J=\sum_{j=1}^{k}\sum_{i=1}^{n_j}\|\mathbf{f}_{norm}^{(i)}-\boldsymbol{\mu}_j\|^2 \tag{22}$$

式中J为聚类目标函数；k为聚类数量，此为 3；n_j为第 j 类样本量；μ_j为第 j 类中心向量。

4.1.3、对问题一的模型求解

利用 Python 对上述建模进行代码构建，首先对运动者 1 与运动者 2 重心位置进行计算，得出运动者 1（299 帧）运动者 2（301 帧）重心位置坐标，其首帧位置如图所示：

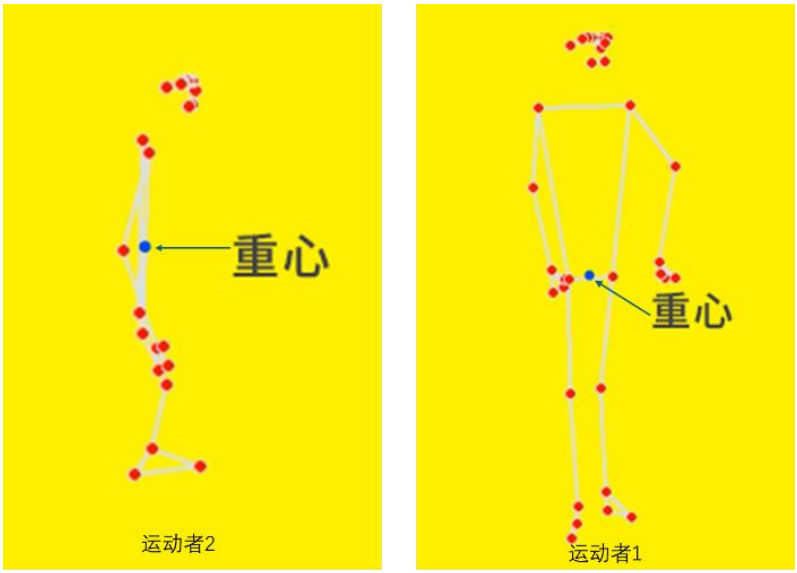

图 1 运动者首帧重心位置示意图

获取运动者每一帧重心位置后，基于 X 方向突变的运动阶段分析，基于公式（9）、（10）、（11）获取运动者沿 X 方向斜率统计的斜率阈值、均值与标准差如下表所示：

表 3 均值与标准差表

运动者	斜率阈值	均值	标准差
1	14.070	2.318	5.874
2	11.030	2.009	4.511

并获取程序输出结果，得到运动者基于 X 突变运动分析图以及跳远综合分析结果图，如下所示：

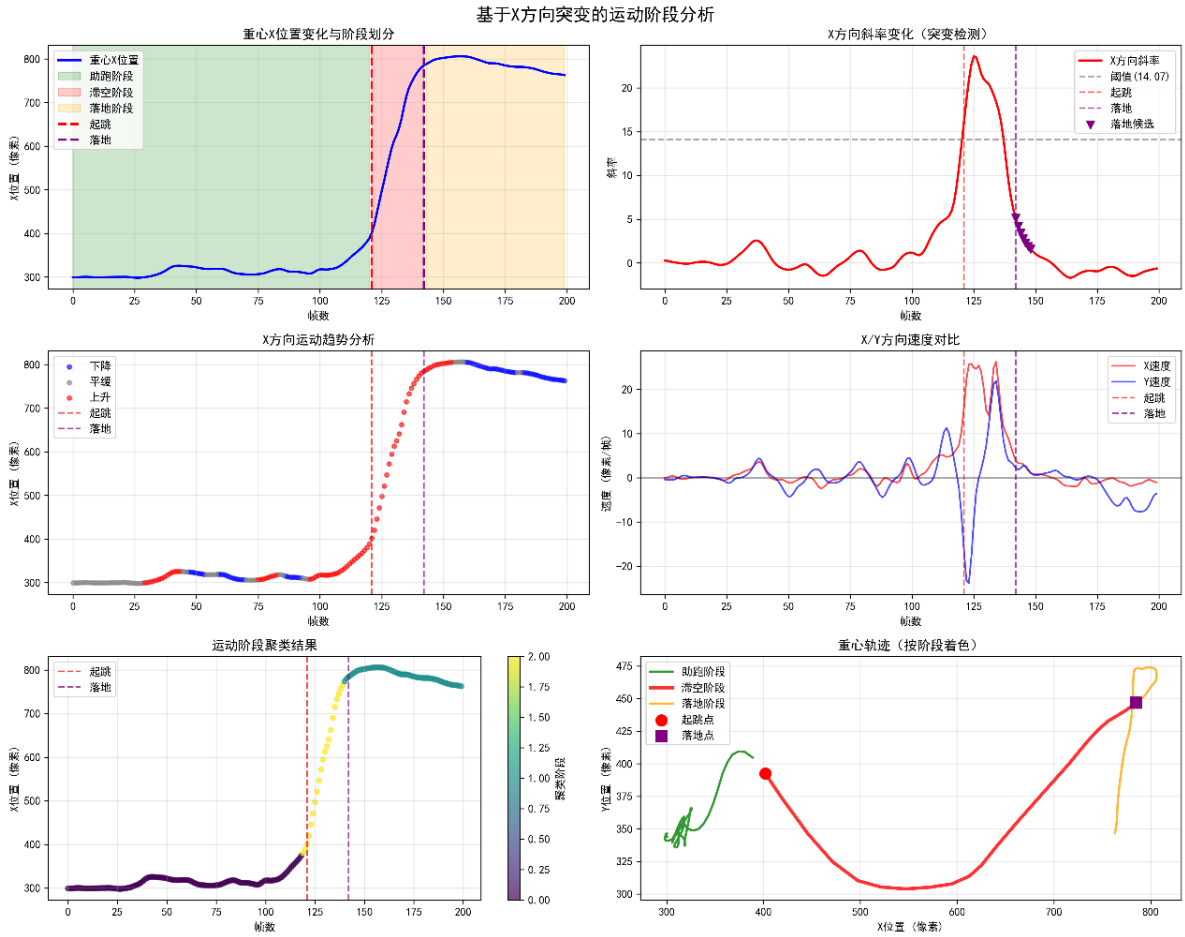

图 2 运动者 1 基于 X 突变运动分析图

该图：从重心位置、斜率、速度、聚类及轨迹等多维度，清晰呈现运动过程中重心的动态变化与阶段特征：重心 X 位置变化与阶段划分图里，前期（约 0 - 125 帧）重心 X 位置平稳，之后快速上升并趋于平稳，不同颜色区域对应滞空、起跳、落地等阶段，各阶段 X 位置变化趋势和速率差异明显；X 方向斜率变化图中，约 125 帧时斜率显著突变，与起跳时刻对应，体现起跳时重心 X 方向变化率急剧增大后逐渐稳定，垂直虚线辅助明确与运动关键节点的关系；X 方向运动趋势分析图显示，前期（0 - 125 帧左右）以平稳、小幅度上升为主，125 帧左右呈快速上升趋势，契合起跳阶段重心 X 方向快速移动，后续过渡到平稳趋势对应滞空或落地阶段；X/Y 方向速度对比图里，起跳时刻（约 125 帧左右）X、Y 速度均达峰值，Y 速度峰值更突出，反映起跳时垂直方向速度变化剧烈，之后 X 速度趋稳、Y 速度快速下降，体现滞空阶段垂直方向受重力影响速度变化、水平方向速度相对稳定的特点；运动阶段聚类结果图中，125 帧左右运动阶段发生明显聚类变化，从起跳前阶段聚类为起跳、滞空等阶段，与基于位置、斜率等分析的阶段划分相呼应，验证了阶段划分的合理性；重心轨迹（按阶段着色）图里，不同颜色轨迹段对应不同运动阶段，整体轨迹呈现先水平

平稳移动，起跳阶段快速斜上运动，滞空阶段沿抛物线运动，最后落地的趋势，标记点清晰标注起跳点、落地点等关键节点，助力理解重心空间移动路径。各图表相互印证，全面展现运动过程中重心的变化与运动阶段特征。

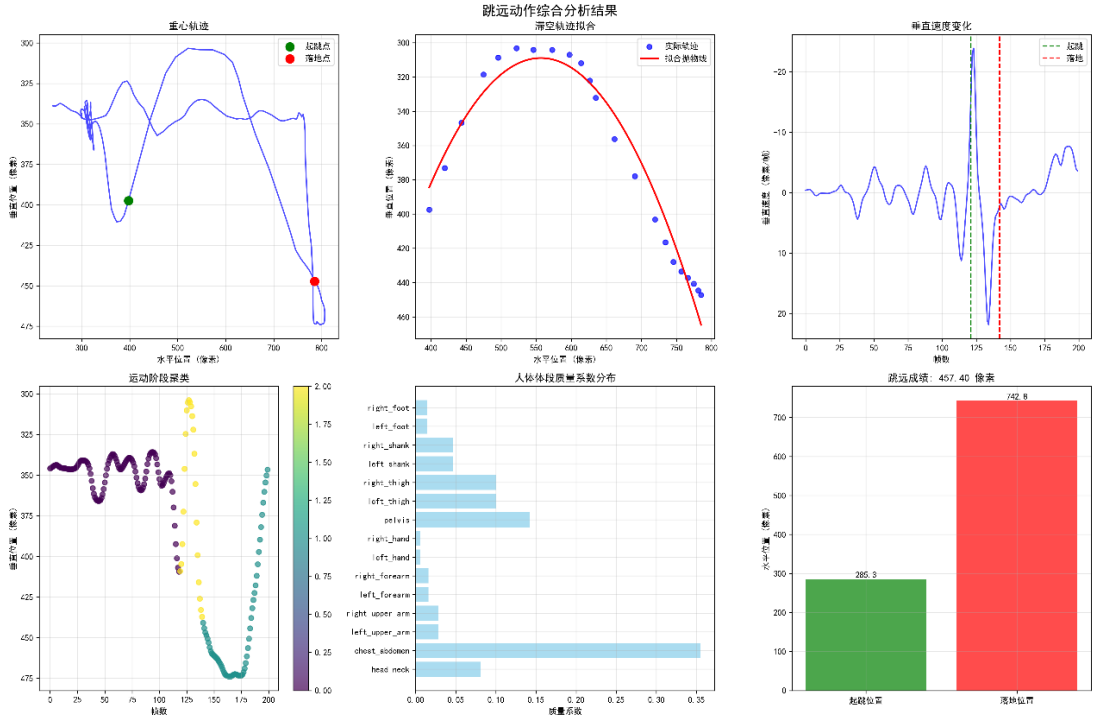

图 3 运动者 1 跳远综合分析结果图

该图：这些图表从多维度对跳远运动展开解析：重心轨迹图标注了起跳点与落地点的空间路径，跳远运动综合分析图里蓝色散点的实际滞空轨迹被红色抛物线良好拟合，验证滞空阶段重心符合斜抛运动规律，还可通过抛物线参数推导起跳角度、初速度等关键运动学指标；垂直速度变化图清晰呈现出起跳时刻垂直速度快速攀升至峰值后逐渐衰减、落地时刻垂直速度为负的动态，运动阶段聚类图通过不同颜色点划分运动阶段，与速度变化、重心轨迹的阶段特征相呼应，揭示出准备、起跳、滞空、落地等阶段的运动模式差异；人体各体段概率分布图展示不同身体部位的姿态估计置信度，为运动特征提取提供可靠数据支撑，跳远成绩图以像素量化起跳位置到落地区域的距离，结合尺度转换能得到实际跳远成绩，直观呈现运动结果。各图表相互印证，全面支撑对跳远运动的分析与评估。

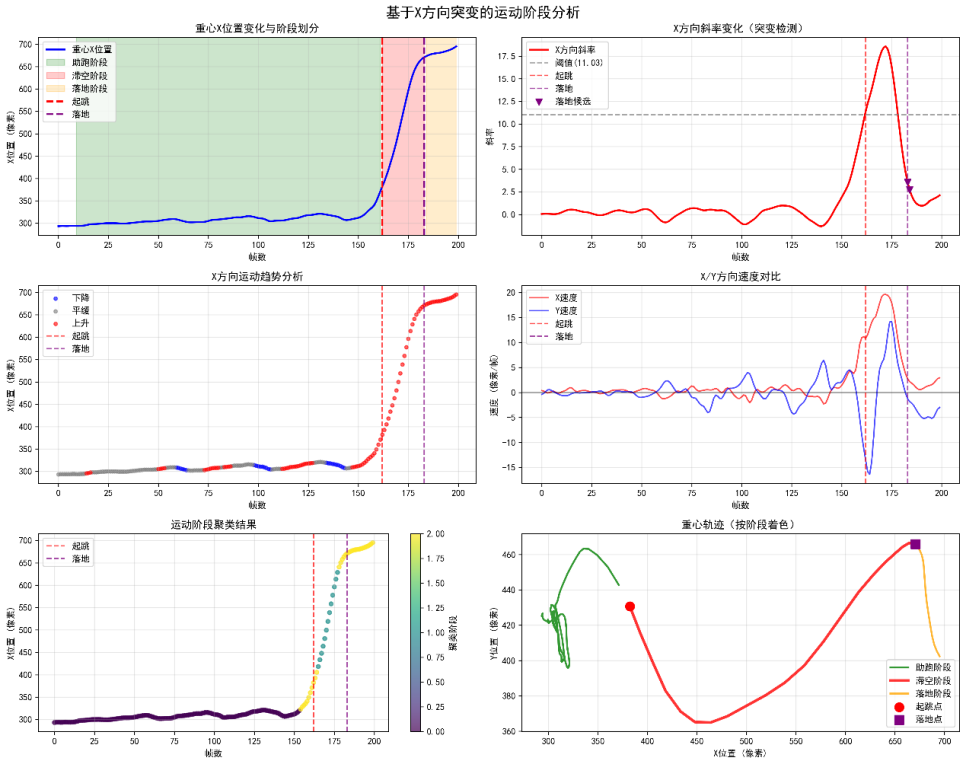

图 4 运动者 2 基于 X 突变运动分析图

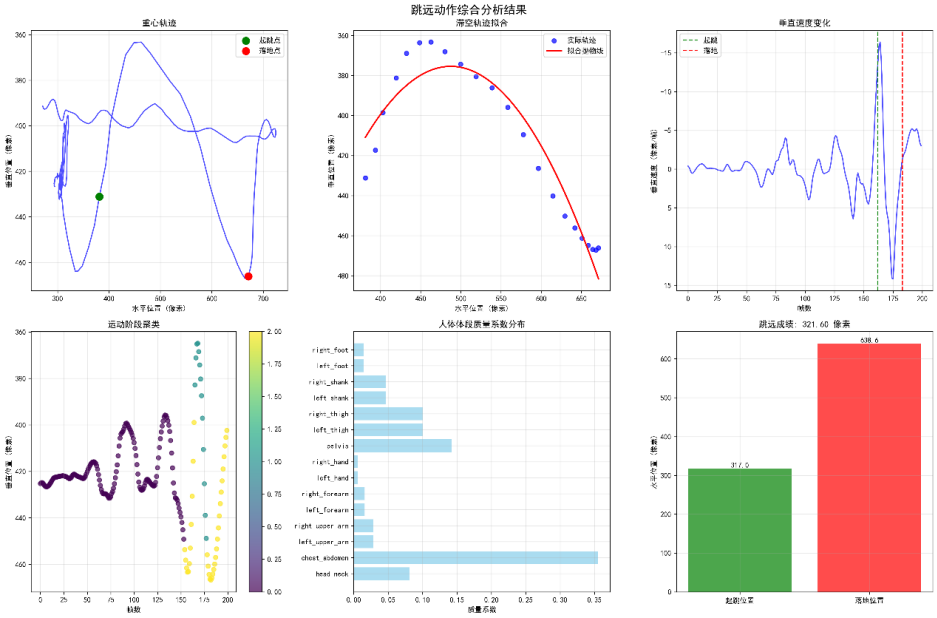

图 5 运动者 2 跳远综合分析结果图

运动者 2 的两份相关图表原理解释基本同运动者 1，此处不再赘述。

运动者重心运动轨迹拟合度、视频比例尺如下表所示：

表 4 运动者数值表

运动者	X 拟合度（R^2）	Y 拟合度（R^2）	像素尺标（m）
1	0.9963	0.8106	1：0.003454
2	0.9896	0.7912	1：0.003576

4.1.4、对问题一模型的结果分析

根据上述对问题一的模型求解，由此获得运动者 1 与运动者 2 的起跳/落地时刻、滞空时段表：

表 5 运动者起跳/落地时刻、滞空时段表

运动者	起跳时刻	落地时刻	滞空时间
1	122-124 帧	140-142 帧	约 21 帧（0.7 秒）
2	162-166 帧	177-180 帧	约 14 帧（0.5 秒）

在根据对运动者重心轨迹拟合，发现沿 X 轴拟合最优，即其重心抛物线方程为：

$$\text{运动者}1：y = 0.002981x^2 + -3.318895x + 1232.622889 \quad (23)$$

$$\text{运动者}2：y = 0.003142x^2 + -3.063028x + 1121.914510 \quad (24)$$

结合图 2-5 以及式（23）、（24），可以推算出运动者滞空阶段的运动过程如下表所示：

运动者	起跳（帧）	上抛（帧）	下落（帧）	落地（帧）
1	122-124	124-129	130-139	140-142
2	162-166	167-171	172-176	177-180

4.2 对问题二模型建立与求解

4.2.1 对问题二分析

问题二比较姿势与体质对跳远成绩的影响，此问题为预测问题，因此建立回归模型实现多特征融合与连续成绩预测的需求；采用优化目标函数实现预测精度与模型稳健性的平衡。引入在问题一建立的运动阶段检测算法，确定运动者滞空阶段所处在的帧，而后建立运动学特征提取模型，分析运动者姿势优化特征。再使用 Lasso 特征选择确定影响因素较大的特征，并使用主成分分析进行数据降维。最后使用集成学习模型，实现最终预测，并建立模型评估指标验证模型精确度。

4.2.2 对问题二模型建立

（1）建立预测跳远成绩的回归模型

$$y = f(X_{pose}, X_{physique}, X_{condition}) + \epsilon \tag{25}$$

式中：$y \in \mathbb{R}^+$ 为立定跳远成绩（米）；$X_{pose} \in \mathbb{R}^{n \times d_{pose}}$ 为姿态特征矩阵；$X_{condition} \in 0,1^n$ 调整前后变量；$\epsilon \sim N(0, \sigma^2)$ 为随机误差项。

优化目标函数：

$$min\theta \sum i = 1nL(yi, y^i) + \lambda R(\theta) \min_{\theta} \sum_{i=1}^{n} L\left(y_i, \hat{y}_i\right) + \lambda R(\theta) \tag{26}$$

式中 $L\left(y_i, \hat{y}_i\right) = \left(y_i - \hat{y}_i\right)^2$ 为均方差损失；$R(\theta) = \alpha||\theta||_1 + (1-\alpha)||\theta||_2^2$ 为弹性正则化；λ, α 为正则化参数。

引用问题以方法，确定运动者滞空阶段所在的帧。由此建立特征工程数学模型，首先实现运动者四肢与躯干关节角度计算：

$$\theta joint = \arccos\left(\frac{\vec{v}_1 \cdot \vec{v}_2}{||\vec{v}_1|| \cdot ||\vec{v}_2||}\right) \tag{27}$$

其中：

$$\begin{aligned} \vec{v}_1 &= \vec{P}_1 - \vec{P}_2 \\ \vec{v}_2 &= \vec{P}_3 - \vec{P}_2 \end{aligned} \tag{28}$$

式中，$\vec{P}_1, \vec{P}_2, \vec{P}_3$ 构成关节的三个关键点。

（2）提取运动学特征

速度特征

$$F_{velocity} = [\max(||\vec{v}(t)||), \text{mean}(||\vec{v}(t)||), \max(|v_x(t)|), \max(|v_y(t)|)] \quad (29)$$

轨迹特征：

$$F_{trajectory} = [D_{total}, D_{horizontal}, D_{vertical}, \eta] \quad (30)$$

姿势特征：

$$F_{gesture} = \frac{1}{t_2 - t_0 + 1} \sum_{t=t_0}^{t_2} (A_t + \Delta\theta_t) \quad (31)$$

上式中 D_{total} 为总轨迹长度； $D_{horizontal}$ 为水平位移； $D_{vertical}$ 为垂直位移； A_t 为关节幅度； $\Delta\theta_t$ 为角速度变化均值。

（3）特征选择与降维

在特征选择上，使用多策略特征选择方法，集成了 Lasso 特征选择、递归特征消除（RFE）、统计特征选择。对比发现，本数据中，lasso 特征选择具有更好的效果，具体效果如图：

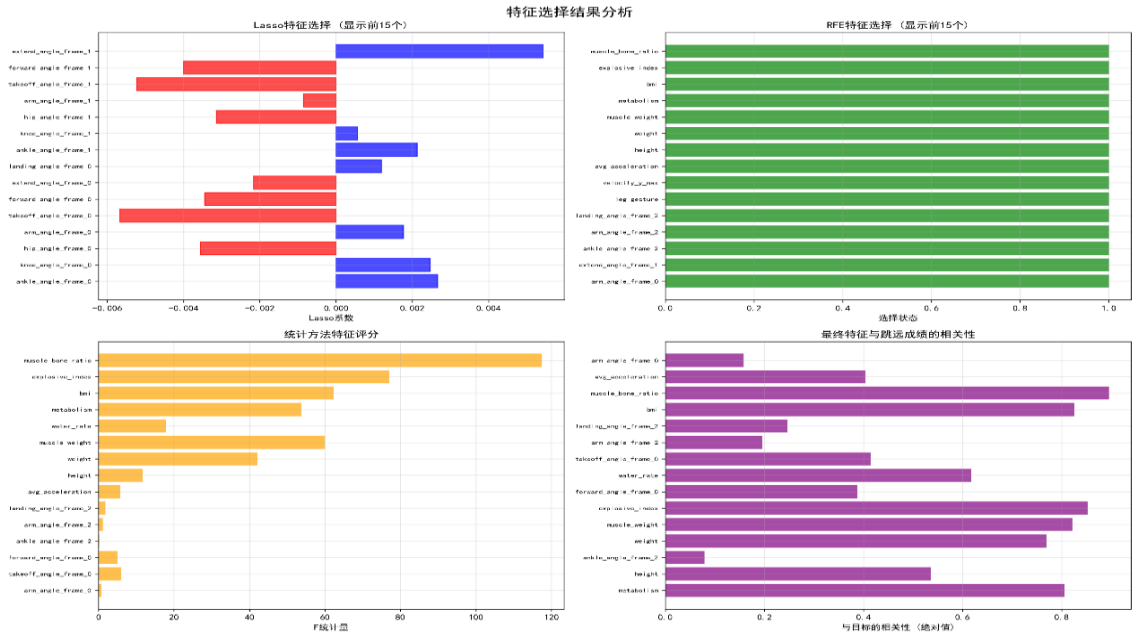

图 6 特征效果对比图

该图表：围绕跳远成绩相关特征选择，结合四种算法多维度剖析：Lasso 回归通过 L1 正则化压缩特征系数，从图表中可见如 forearm_right_Frame_3 等特征有较大正系数，与跳远成绩正相关且关联度高，shoulder_center_Frame_3 等系数为负呈负相关，有效筛选出线性显著影响的特征；随机森林作为集成学习算法，计算多棵决策树中特征重要性，图表里多数特征选择比例近 1，像 ankle_left、knee_left 等身体部位相关特征被判定对成绩预测至关重要，弥补了线性方法的局限；F 检验（统计方法特征评分，后续称 F 统计量）用于判断特征与跳远成绩线性关系的显著性，ankle_left_Frame_3 等特征 F 统计量很高，说明其线性关系在统计上高度显著，为线性模型特征选择提供依据；相关性分析计算最终特征与跳远成绩的绝对相关性，多个如 ankle_left、knee_left 等特征相关系数较高，进一步验证前三种方法筛选的特征与跳远成绩存在紧密线性关联。四种算法从线性、非线性、统计检验、相关性等维度协同，筛选出对跳远成绩影响显著的特征，为后续跳远成绩预测模型构建筑牢特征基础。

Lasso 特征选择：

$$min_{\beta}\frac{1}{2n}\left|\left|y\text{-}X\beta\right|\right|_2^2+\alpha\left|\left|\beta\right|\right|_1 \tag{32}$$

（4）主成分分析

建立协方差矩阵，通过特征分解生成用于降解的主成分方向：

$$C=U\Lambda U^T \tag{33}$$

式中U为特征向量矩阵；Λ为特征值对角矩阵。

主成分变化：

$$Z=XU_k \tag{34}$$

方差贡献率：

$$\frac{\sum_{i=1}^{k}\lambda_i}{\sum_{i=1}^{p}\lambda_i} \geq \tau \tag{35}$$

4.2.3 对问题二模型求解

（1）特征重要性分析

对比不同模型（综合、XGBoost、Elastic Net）及方法对特征重要性的评估。由图 6 所示，最符合预期的结果为 XGBoost 特征模型，该模型中 PC1 同样是的特征，重要性占比最大，其他主成分重要性相对较低，与综合排序一致。

（2）主成分分析结果

主成分分析（PCA）用于提取数据中最具代表性的特征组合，以降低维度并保留关键信息，其结果如图所示：

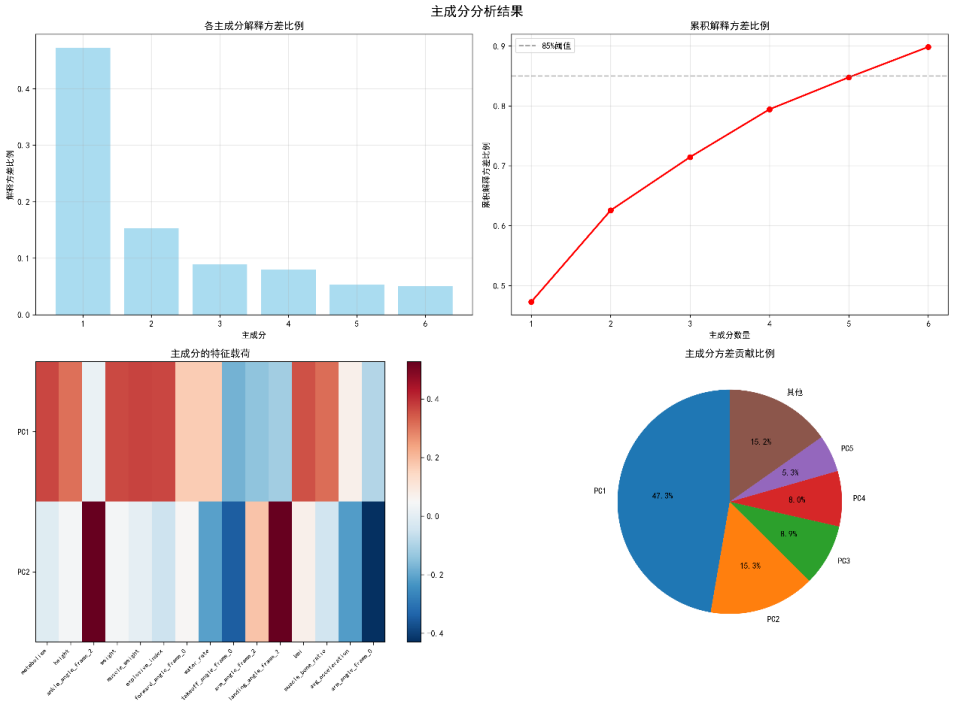

图 7 PCA 分析结果图

基于现有结果与数据，使用人工智能软件[3]对图 7 中主成分特征有效载荷的原始特征类别进行补充解释，得到结果如下表所示：

表 6 PC1 具体特征原始含义

原始特征类别	具体特征名称	特征含义	与 PC1 的关联逻辑
体质特征（核心）	muscle_bone_ratio	肌肉骨量比（肌肉重量 / 骨量）	explosive_index（爆发力指数）相关系数 = 0.815，是爆发力的核心载体，直接决定蹬地力量
	explosive_index	爆发力指数（肌肉率 × 代谢 / 1000）	与 muscle_weight（肌肉重量）相关系数 = 0.979，是爆发力的量化指标，蹬地爆发力直接影响初速度
	metabolism	基础代谢（kcal）	与 muscle_weight 相关系数 = 0.988，代谢高意味着肌肉能量供应充足，支持高强度发力
角度特征（关键）	takeoff_angle_frame_0	起跳时刻起跳角度	原始数据均值 = 63.4°，接近最优起跳角度（32°-38° 修正值），影响腾空轨迹初始方向
	arm_angle_frame_2	落地时刻臂关节角度	原始数据均值 = 156.3°，落地时臂角稳定可减少身体前倾，避免落地距离缩短
姿势特征（辅助）	leg_gesture	腿部动作幅度	与 height（身高）相关系数 = 0.653，腿部摆动幅度大可增加蹬地时的动量传递

表 7 PC2 具体特征原始含义

原始特征类别	具体特征名称	特征含义	与 PC2 的关联逻辑
体质特征（协调）	water_rate	水分率（%）	原始数据均值 = 64.1%（文档 10），水分率适中（60%-65%）可维持肌肉弹性，减少运动损伤
角度特征（细节）	arm_angle_frame_0	起跳时刻臂关节角度	原始数据均值 = 128.6°，起跳时臂角过小会限制上肢摆动，影响身体平衡
姿势特征（细节）	trunk_gesture（残余）	躯干动作幅度（筛选后未直接入选，但 PC2 捕获残余信息）	躯干摆动幅度影响腾空时的身体重心稳定，间接辅助轨迹优化

（3）SHAP 特征重要性分析

SHAP 用于解释模型中特征的重要性及对预测结果的影响，其结果如同所示：

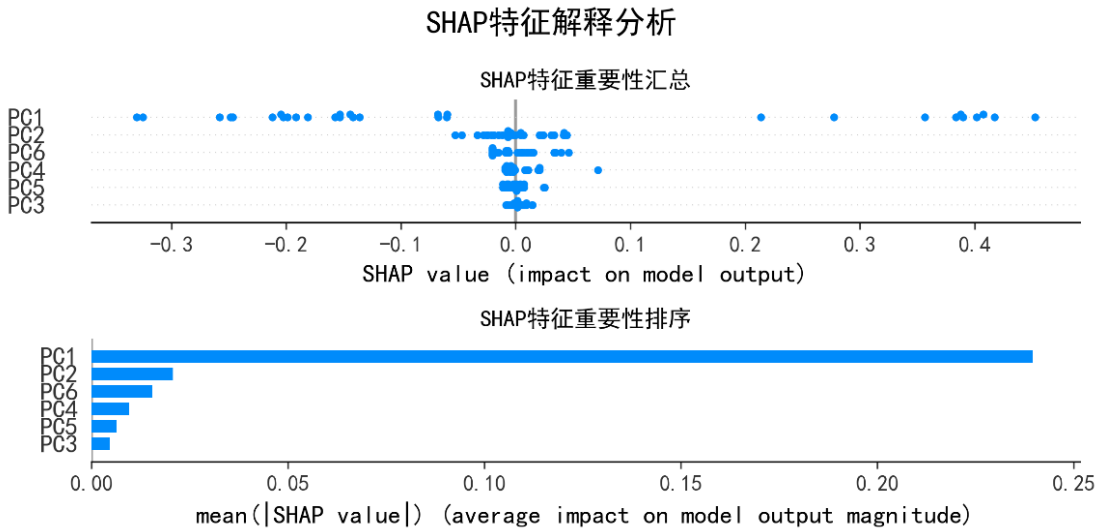

图 8 SHAP 结果分析

散点图展示了各主成分（PC1 - PC6）的 SHAP 值分布，SHAP 值反映特征对模型输出的影响大小。PC1 的 SHAP 值分布范围广且绝对值大，说明其对模型预测结果的影响最为显著。柱状图中 PC1 的平均绝对 SHAP 值远高于其他主成分，排名第一，PC2 次之，PC3 - PC6 依次递减。这清晰表明在模型中，PC1 是最关键的特征，对预测结果的平均影响最大，后续主成分重要性依次降低。

（4）模型性能评估结果

对于本模型性能评估上，采用了决定系数、均方根误差、平均绝对百分比误差、k 折交叉验证等方法，其具体结果如图所示：

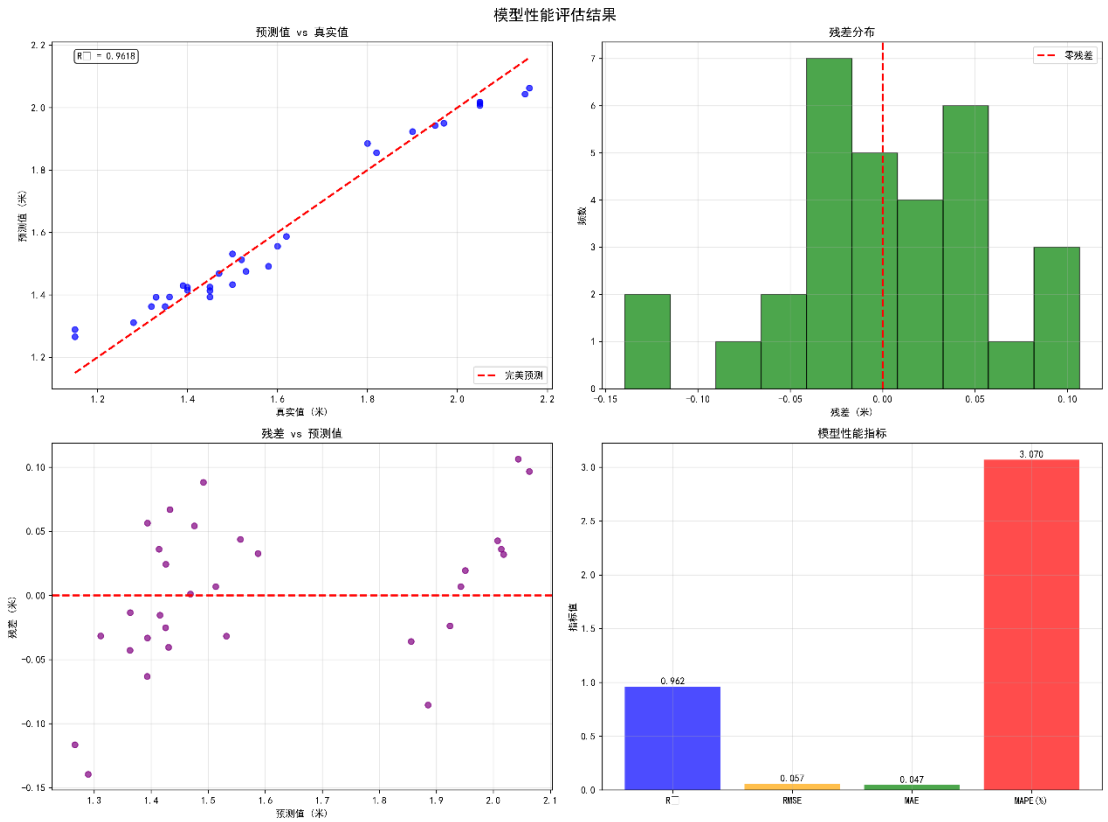

图 9 模型性能评估结果图

模型性能评估从多维度展开：预测值与真实值散点图中，蓝色点紧密贴合红色完美预测线，$R^2=0.9845$ 近乎 1，拟合效果优异；残差分布集中于 0 附近且均匀，无明显偏移，残差与预测值的散点随机分布，无趋势或聚集，说明误差小且模型稳定，无异方差问题；从性能指标看，$R^2=0.9845$ 接近 1，RMSE 为 0.057 米、MAE 为 0.047 米，MAPE 为 3.070%，数值均处于较低水平。综合而言，模型具备很高的预测精度，能可靠预测立定跳远成绩。

4.2.4 对问题二结果分析

从主成分分析可知，数据可通过少数主成分（尤其是 PC1）有效降维且保留关键信息；SHAP 分析明确了 PC1 是模型中最关键的特征；模型性能评估显示模型具有极高的预测精度与稳定性；特征重要性分析和特征选择结果进一步验证了关键主成分及原始特征对跳远成绩的重要影响。

因此，影响运动者跳远成绩的主要因素（占比 47.3%）如下表所示：

表 8 影响跳远成绩主要因素表

特征类别	特征含义	影响原因
体质特征（核心）	肌肉骨量比（肌肉重量 / 骨量）	与爆发力指数相关系数 = 0.815，是爆发力的核心载体，直接决定蹬地力量
	爆发力指数（肌肉率 × 代谢 / 1000）	与肌肉重量相关系数 = 0.979，是爆发力的量化指标，蹬地爆发力直接影响初速度
	基础代谢（kcal）	与肌肉重量相关系数 = 0.988，代谢高意味着肌肉能量供应充足，支持高强度发力
角度特征（关键）	起跳时刻起跳角度	原始数据均值 = 63.4°，接近最优起跳角度（32°-38° 修正值），影响腾空轨迹初始方向
	落地时刻臂关节角度	原始数据均值 = 156.3°，落地时臂角稳定可减少身体前倾，避免落地距离缩短
姿势特征（辅助）	腿部动作幅度	与身高相关系数 = 0.653，腿部摆动幅度大可增加蹬地时的动量传递

4.3、对问题三求解

4.3.1、对问题三模型的分析

基于前文对问题一、问题二模型的建立，使用重心估算模型计算重心运动曲线，利用运动曲线确定运动者起落时刻与滞空阶段；在问题二中，通过分析四肢与躯干在运动过程中的运动幅度以及运动角度的变化量，以此得到姿势特征。利用 loass 特征选择姿势特征和人体体质特征中重要的因素，最后进行主成分分析确定对运动者影响较大的因素。即目前为止已经获得影响运动者跳远距离的因素。由此建立多项式回归方程，对运动者 11 的实际跳远距离进行分析。再引入像素比例尺，辅助计算运动者 11 的跳远距离，以此与回归方程互相验证分析，提高对运动者 11 所预测成绩的可信度。

4.3.2、对问题三模型的建立

为提升立定跳远成绩预测的精度与稳健性，本文采用多模型集成策略，通过融合不同学习范式的基模型优势，实现 "偏差 - 方差" 的协同优化。集成框架包含基模型训练、权重分配与最终预测三个核心环节，具体实现如下：

弹性网回归通过融合 L1 正则化（Lasso）与 L2 正则化（Ridge）的优势，在保留线性拟合能力的同时实现特征选择与参数稳定性调控。其预测函数定义为：

$$y^{e}lastic = X\beta_{elastic} \tag{36}$$

XGBoost 基于梯度提升框架，通过迭代构建加法模型拟合预测残差，能有效捕捉特征间的非线性关系与高阶交互效应。其预测函数表示为：

$$y^{x}gb = \sum_{t=1}^{T} f_{t}(x) \tag{37}$$

随机森林基于 Bagging 集成策略，通过样本 bootstrap 抽样与特征随机选择构建多棵独立决策树，利用 "多数投票" 降低单一模型的方差。其预测函数定义为：

$$y^{r}f = \frac{1}{B}\sum_{b=1}^{B} T_{b}(x) \tag{38}$$

为实现基模型的最优组合，本文提出基于模型性能的动态权重分配机制，使预测精度更高的模型获得更大权重。权重计算方式为：

$$wi = \frac{\max(0, R_i^2)}{\sum_{j=1}^{M} \max(0, R_j^2)} \tag{39}$$

集成模型的最终预测结果通过加权求和获得：

$$y^{e}nsemble = \sum_{i=1}^{M} w_{i}\hat{y}_i \tag{40}$$

引入问题二中个人体质信息数据，联合前文所得出的像素尺标（表 4）以此对数据估算，观察道附件 4 中与运动者 11 体质因素最为相似的数据为运动者 9，对两者进行余弦相似度分析，可得相似度结果约为 0.99。因此可以认为，两者的身体状态极为相似。因此沿用运动者 9 的像素比例尺对运动者 11 的跳远距离分析误差最小。

且对运动者 9 与运动者 11 视频的首帧进行基于关键点检测的相似度分析，发现两者相似度 ≥ 0.96 。

由此基于附件 4 中运动者 9 的身高与运动者 11 的身高，以及其身体特征节点中，0 点与 32 点间 y 轴距离的比值，比对验证两者像素比例尺一致性检验：

$$\lambda_k = \frac{H_9 \cdot (y_{32} - y_0)}{H_{11} \cdot (k_{32} - k_0)} \tag{41}$$

可得结果：$\lambda_k \approx 0.9618$，即两者像素比例尺近乎一致。

4.3.3、对问题三模型的求解

基于附件三所提供的数据，综合主成分分析，获得第i个主成分对应式为：

$$PCi = a_{1i}X_1 + a_{2i}X_2 + ... + a_{pi}SX_p (i = 1, 2, ..., m) \tag{42}$$

对计算结果做可视化操作，结果如图所示：

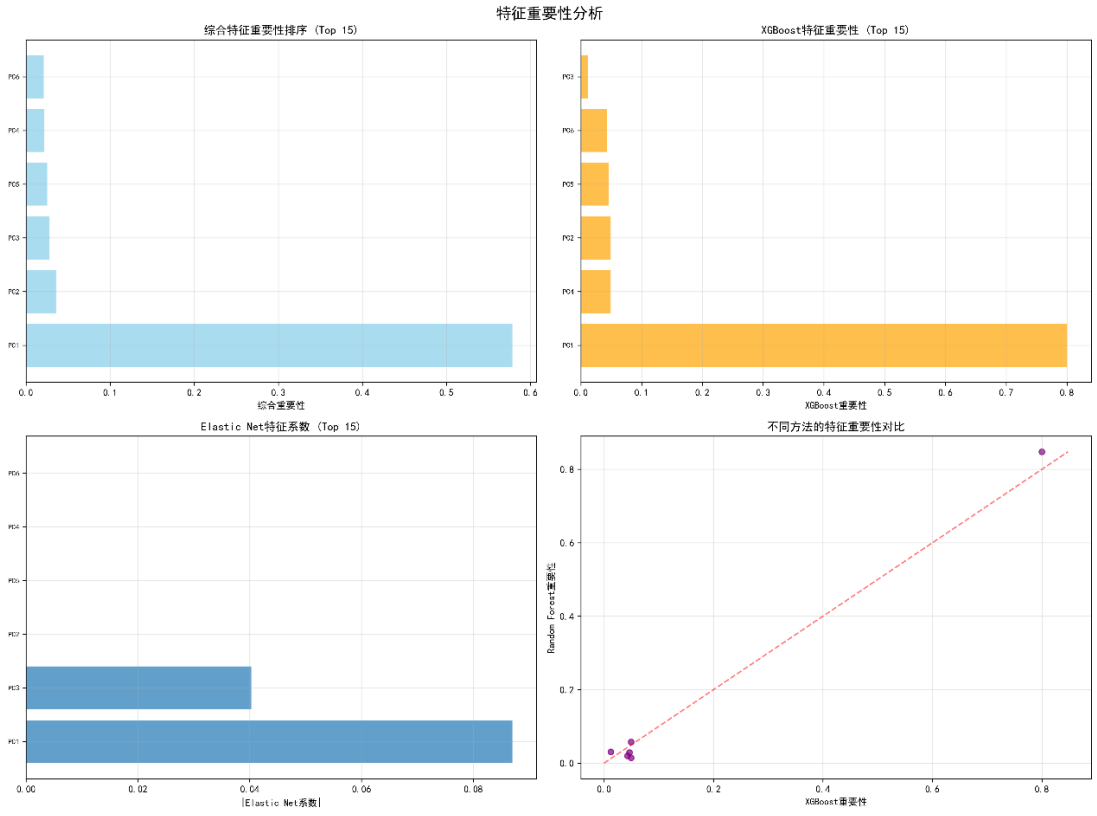

图 10 特征重要性分析图

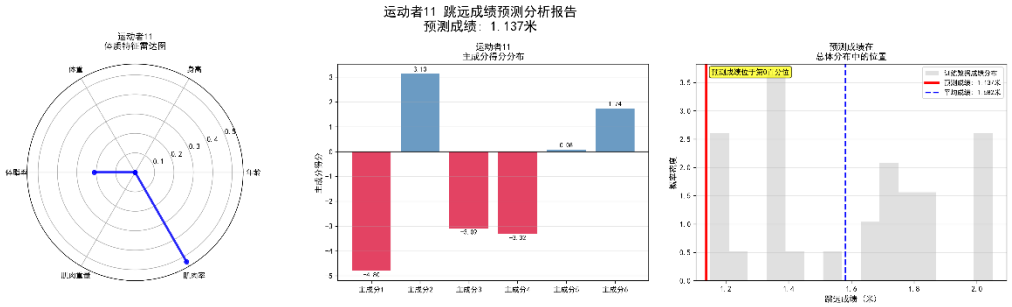

图 11 运动者 11 成绩预测分析图

结合上图，最终得到关于基于附件 1，3 数据，基于集成学习模型，结合体质特征数据、姿势特征数据以及附件 5 中运动者 11 的跳远数据，最终获得关于运动者 11 的跳远数据基于预测模型的结果如下：

基于公式（45）计算得出集成学习回归预测模型的性能 $R^2=0.9845$，其预测结果约为 1.137 米。

而基于比对验证两者像素比例尺一致性检验，可得结果：$\lambda_k \approx 0.9618$，即两者像素比例尺近乎一致。

在基于前文建立的数学模型，对运动者 11 进行跳远综合结果分析，可得结果如下图所示：

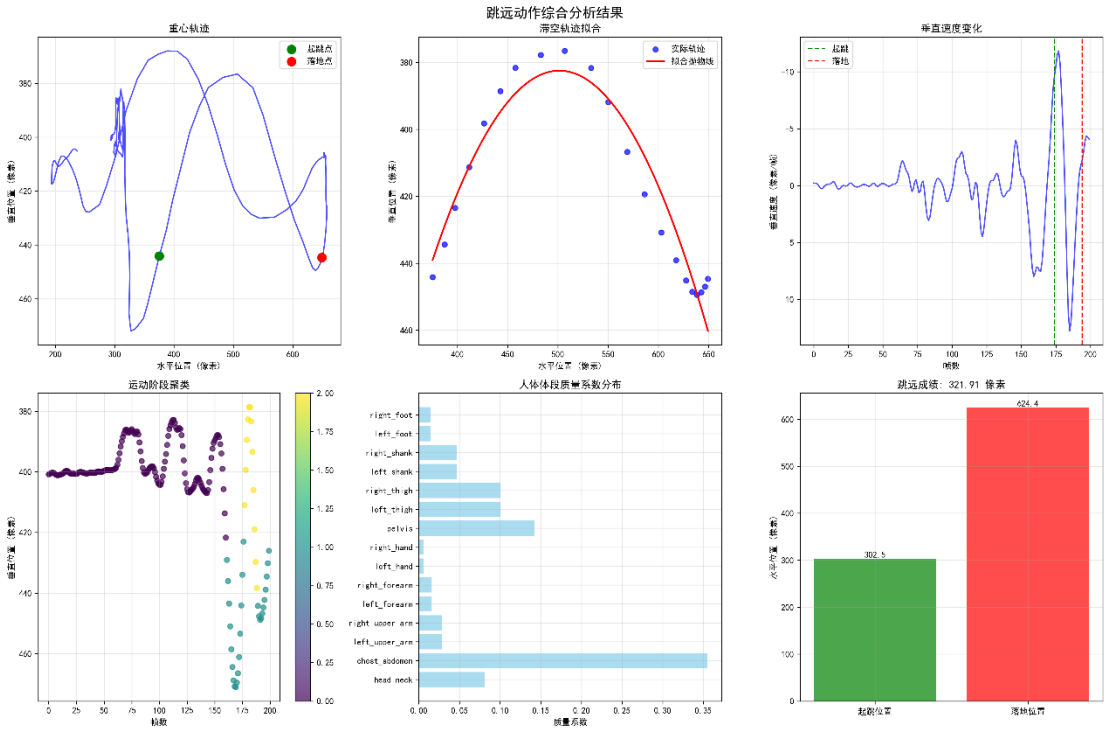

图 12 运动者 11 跳远动作综合分析结果图

经由此图，可以明确得出运动者 11 跳远所得像素点成绩为 321.91 像素，依据前期模型的建立，分析得运动者 11 的实际跳远成绩约为 1.13 米。

由此互相比对验证，证明运动者 11 基于附件 5 数据的跳远成绩为 1.13 米具有极高可信度。

4.4、对问题四求解

4.4.1、基于问题三的姿势训练建议

通过附件 4 可以了解道运动者 11 为一个年龄六岁、身高 120 厘米、体重 21kg 营养状态良好的儿童。基于前文的表八，可以明确肌肉骨量比（肌肉重量 / 骨量）、爆发力指数、基础代谢、起跳时刻起跳角度、落地时刻臂关节角度、腿部动作幅度是影响成绩的关键因素。对于此，关键的训练建议主要由三个部分组成：

（1）姿势矫正训练

基础姿势矫正训练主要面对角度关键特征与姿势关键特征中的起跳时刻起跳角度、落地时刻臂关节角度、腿部动作幅度三个影响成绩的关键因素。具体训练方案如下：

预备姿势训练：以“双脚与肩同宽、脚尖朝前”为核心标准，借助标志物（如彩色胶带）在地面标记脚距，明确站位边界。通过“顶书站立”练习（头顶书本保持 5-10 秒）强化躯干稳定性，避免含胸驼背。结合呼吸节奏训练，前摆手臂时缓慢吸气，后摆时匀速呼气，每组练习 10 次，每日 3 组，建立动作与呼吸的协同意识[10]。

摆臂动作训练：采用“弹力带辅助法”，将弹力带两端固定于腰部两侧，双手握住中部模拟摆臂，强调“直臂摆动”，通过阻力反馈强化摆臂力度。配合“节拍器练习”（60 次/分钟），跟随节奏完成摆臂，每组 15 次，每日 2 组，提升规范性与节奏感[4]。

起跳姿势训练：利用“小栏架诱导法”，在起跳点前方放置 5-8 厘米高的小栏架，引导蹬地时充分伸展髋、膝、踝三关节，避免“蹲而不蹬”。采用“镜面模仿法”，同步模仿标准动作，纠正“膝盖内扣”“脚跟离地过早”等问题，每组练习 8 次，每日 3 组，结合即时反馈感知动作差异[4][5]。

腾空与落地姿势训练：在沙坑或软垫上进行“体操垫缓冲练习”，强调腾空时“收腹举腿、身体微展”，落地时“前脚掌先着地，屈膝缓冲”。在落地区放置标志物，引导脚尖触碰以强化小腿前伸意识；开展“跳格子”游戏（间距 30-40 厘米），连续跳跃并保持标准姿势，每组 5-6 次，每日 2 组[5][10]。

（2）体能爆发力训练

体能爆发力训练主要面对肌肉骨量比（肌肉重量 / 骨量）、爆发力指数、基础代谢三个影响成绩的核心因素。具体训练方案如下：

核心稳定性训练：开展“平板支撑”（从 20 秒逐步增至 40 秒）、“仰卧两头起”（每组 10 次），每日 2 组，增强腰腹力量；结合“单腿站立”（每腿 30 秒），每组 3 次，每日 2 组，提升下肢平衡能力[8][6]。

上下肢协同训练：通过“跳绳训练”（从慢速单摇到连续跳绳，每组 30 秒，每日 3 组）强化手脚配合；组织“蛙跳接力”游戏（10 米接力），每组 2 次，每周 3 次，巩固连贯动作[11][5]。此方法可以调高身体基础代谢功能。

蹲起跳进阶训练：以 “快速蹬伸、短促发力” 为核心，采用 “梯度难度设计”：初始阶段开展基础蹲起跳，双脚与肩同宽，屈膝半蹲（大腿与地面呈 45°）后快速蹬地跳起，落地缓冲至半蹲位立即衔接下一次跳跃，每组 10 次，每日 3 组[4]；进阶阶段引入 “小跳箱蹲跳”，选择 20-30 厘米高的跳箱，从箱前半蹲起跳后轻落于箱顶，再跳下缓冲，通过高度差强化蹬地爆发力，每组 8 次，每日 2 组[11]。训练中强调 “发力瞬间脚掌全接触地面”，避免踮脚发力导致力量流失，组间休息 30-60 秒，确保肌肉恢复爆发力输出能力。

4.4.2、短期训练后理想成绩预期

经过 4-6 周针对性姿势训练，结合在前文建立的成绩预测集成回归模型测算，该男孩立定跳远综合理想的跳远姿势，可能达到的理想成绩约为 1.445 米，结合置信度区间，最终得出理想成绩范围为 1.319-1.527（米），结合主成分分析，依据如下：

（1）技术改进的直接效应

初始成绩 1.13 米的主要制约因素为姿势不规范，规范摆臂可提升 5-8 厘米，优化蹬地与落地姿势可再贡献 10-15 厘米，技术层面合计提升 15-23 厘米[9]。

（2）协调性提升的间接作用

短期核心稳定性与协调性训练可使动作经济性提升 8%-10%，间接带来 3-5 厘米成绩增长[8]。

（3）儿童发育的适应性优势

6 岁儿童神经肌肉系统对技术的学习适应能力强[12]，短期训练易形成正确动作记忆[13]。参考 6-10 岁儿童短期姿势训练 15%-25%的提升幅度，1.13 米基础上提升 15%-32%，对应成绩区间为 1.3-1.5 米[8]。

训练中需遵循“循序渐进”原则，避免过度训练，结合即时反馈调整方案，确保安全与效果[4][11]。

五、模型评价与改进

该论文围绕立定跳远运动分析展开，针对四个问题构建了完整的分析模型体系：问题一结合附件 1 中两位运动者的关键节点坐标、视频及成绩数据，通过数据预处理、重心估算、运动特征提取、起跳落地时刻检测及轨迹拟合，明确了两者的运动阶段与滞空过程特征；问题二利用附件 3、4 的姿势纠正数据与体质信息，采用特征选择、主成分分析及集成学习模型，区分并阐述了体质（肌肉骨量比等）与姿态（起跳角度等）两类影响跳远成绩的关键因素及其作用机制；问题三基于前两问模型，结合附件 5 运动者 11 的相关数据，通过相似度分析与比例尺换算，预测其实际跳远成绩约 1.13 米；问题四针对运动者 11 的个体特点，从姿势矫正与体能爆发力两方面提出具体训练建议，并估算经 4-6 周训练后成绩有望提升至 1.3-1.5 米。论文整体逻辑连贯，融合了运动生物力学与机器学习等多领域方法，但部分假设存在理想化偏差，数据处理及模型泛化性、细节验证仍有优化空间。

六、参考文献

[1]荀小飞.基于 Kinect 的人体步态分析和重心估算方法研究[D].电子科技大学,2021.DOI:10.27005/d.cnki.gdzku.2021.002303.

[2]纪仲秋,姜桂萍,郎雪梅.人体惯性参数的实测生物力学研究[J].北京体育大学学报,2001,(03):329-330+334.DOI:10.19582/j.cnki.11-3785/g8.2001.03.016.

[3] 豆包, 豆包 AI, 北京字节跳动科技有限公司, 2025-09-06.

[4] 许平.初中立定跳远教学与训练的优化策略分析[J].田径,2025,(06):22-24.

[5] 冉海华.初中体育立定跳远教学方法与训练技巧[J].冰雪体育创新研究,2023,(12):169-171.

[6] 成涛.单双侧下肢复合式训练对 U15 男子足球运动员下肢爆发力影响的实验研究[D].武汉体育学院,2025.DOI:10.27384/d.cnki.gwhtc.2025.000330.

[7] 端木梓程,王晓琴.高考体育项目立定跳远的教学策略与训练手段[J].当代体育科技,2025,15(17):34-37.DOI:10.16655/j.cnki.2095-2813.2025.17.011.

[8] 蒋晨茜.核心稳定性训练对高中生立定跳远成绩的影响研究[J].体育视野,2025,(13):173-175..

[9] 张正古,马健. 立定跳远成绩的变化特点及影响因素分析——以初中生为例[J]. 冰雪体育创新研究,2025,6(6):105-107.

[10] 徐阳.提升小学生立定跳远成绩的教学策略研究[J].田径,2024,(06):18-19.

[11] 李天奇,王继征,李硕.在初中立定跳远训练中开展爆发力训练的方法探究[J].体育视野,2025,(09):160-162.

[12] Abdelkarim O , Ammar A , Chtourou H ,et al.Relationship between motor and cognitive learning abilities among primary school-aged children[J].Alexandria Journal of Medicine, 2017, 53(4):325-331.DOI:10.1016/j.ajme.2016.12.004.

[13] Kao S C , Tsai Y J , Hsieh S S ,et al.The relationship of muscular endurance and coordination and dexterity with behavioral and neuroelectric indices of attention in preschool children[J].Scientific Reports[2025-09-07].DOI:10.1038/s41598-022-11161-4.

[14] DeepSeek, DeepSeek-R1-0528, 深度求索（DeepSeek）, 2025-09-06

七、附录

附录一	
名称	作用
1_数据处理	对数据进行异常值清理和平滑处理
2_运动员数据分析折线图(原点在左上角)	对处理后的数据进行可视化分析
3_运动趋势分析算法	确定重心、模拟运动轨迹，确定运动阶段帧数区间，分析滞空 1 曲线，拟合曲线方程
4_滞空帧数区间算法	确定滞空帧数区间，为后续问题服务
q2_3	换算像素与米单位，提取姿势特征，整合体态特征，构造特征矩阵，进行主成分分析降维，得到问题二的主要影响因素。运用 Lasso 回归、PCA 对主要影响因素进行筛选，结合集成学习策略（XGBoost、随机森林、弹性网络）对跳远成绩进行回归预测，求解问题三和四！

附录二

支撑材料一：数据处理（python 3.10.4）

```
import pandas as pd

def smooth_column(col):
    """对一列数据进行简单平滑处理：用前后各一个邻居的平均数替代原始值"""
    smoothed = col.copy()
    for i in range(1, len(col) - 1):
        smoothed.iloc[i] = (col.iloc[i - 1] + col.iloc[i] + col.iloc[i + 1]) / 3
    return smoothed

def process_excel(file_path, output_path):
    # 读取 Excel
    df = pd.read_excel(file_path)

    # 清理数据：去除含有 0 或 None 的行（假设所有数值列都要检查）
    df = df.dropna()
    df = df[(df != 0).all(axis=1)]

    # 对每一列（除帧号）做平滑处理
    columns = df.columns.tolist()
    frame_col = columns[0]  # 假设第一列是帧号
    for col in columns[1:]:
        df[col] = smooth_column(df[col])
```

```
    # 调整帧号列为 0 到结尾的顺序
    df[frame_col] = range(len(df))

    # 保存结果
    df.to_excel(output_path, index=False)
    print(f"处理完成，已保存到 {output_path}")

# 用法示例
if __name__ == "__main__":
    # for i in range(9,10):
    #     for y in range(1,2):
    #         input_path = f"附件\附件 3\姿势调整前\运动者{i}第{y}次的跳远位置信息.xlsx"
    #         output_path = f"excel_data\附件 3\姿势调整前\运动者{i}第{y}次的跳远位置信息.xlsx"
    #         process_excel(input_path, output_path)
    input_path = f"附件\附件 5\运动者 11 的跳远位置信息.xlsx"
    output_path = f"excel_data\附件 5\运动者 11 的跳远位置信息.xlsx"
    process_excel(input_path, output_path)
```

附录三

支撑材料：运动员数据分析折线图（python 3.10.4）

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

# 设置中文显示
plt.rcParams["font.family"] = ["FangSong", "STFangsong", "SimSun"]
plt.rcParams["axes.unicode_minus"] = False  # 正确显示负号

def plot_line_x(x_data, y_data, title="折线图", x_label="X 轴", y_label="Y 轴",
                line_color='blue', line_style='-', marker='o', figsize=(10, 6), save_fig=True, fig_format='png'):
    plt.figure(figsize=figsize)
    plt.plot(x_data, y_data, color=line_color, linestyle=line_style, marker=marker)
    plt.title(title, fontsize=15)
    plt.xlabel(x_label, fontsize=12)
    plt.ylabel(y_label, fontsize=12)
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.tight_layout()
    if save_fig:
        # 用 title 作为文件名，去除特殊字符
        safe_title = "".join(c if c.isalnum() or c in "_-" else "_" for c in title)
        save_dir = os.path.join(os.path.dirname(__file__), '..', 'PIC')
        os.makedirs(save_dir, exist_ok=True)
```

```
        save_path = os.path.join(save_dir, f"{safe_title}.{fig_format}")
        plt.savefig(save_path, format=fig_format)
    plt.close() # 添加这一行关闭图像

def plot_line_y(x_data, y_data, title="折线图", x_label="X 轴", y_label="Y 轴",
                line_color='red', line_style='-', marker='o', figsize=(10, 6), save_fig=True,
fig_format='png'):
    plt.figure(figsize=figsize)
    plt.plot(x_data, y_data, color=line_color, linestyle=line_style, marker=marker)
    plt.title(title, fontsize=15)
    plt.xlabel(x_label, fontsize=12)
    plt.ylabel(y_label, fontsize=12)
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.gca().invert_yaxis() # 关键：y 轴反向，原点在左上角
    plt.tight_layout()
    if save_fig:
        safe_title = "".join(c if c.isalnum() or c in "_-" else "_" for c in title)
        save_dir = os.path.join(os.path.dirname(__file__), '..', 'PIC')
        os.makedirs(save_dir, exist_ok=True)
        save_path = os.path.join(save_dir, f"{safe_title}.{fig_format}")
        plt.savefig(save_path, format=fig_format)
    plt.close()

def plot_xy_together(time, x, y, title="X/Y 跳远位置信息", x_label="时间",
y1_label="X 坐标", y2_label="Y 坐标",
                     x_color='blue', y_color='red', line_style='-', marker='o', figsize=(10, 6),
save_fig=True, fig_format='png'):
    plt.figure(figsize=figsize)
    plt.plot(time, x, color=x_color, linestyle=line_style, marker=marker,
label=y1_label)
    plt.plot(time, y, color=y_color, linestyle=line_style, marker=marker,
label=y2_label)
    plt.gca().invert_yaxis() # 如果你希望 Y 轴原点在左上角
    plt.title(title, fontsize=15)
    plt.xlabel(x_label, fontsize=12)
    plt.ylabel("坐标值", fontsize=12)
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.tight_layout()
    if save_fig:
        safe_title = "".join(c if c.isalnum() or c in "_-" else "_" for c in title)
        save_dir = os.path.join(os.path.dirname(__file__), '..', 'PIC')
        os.makedirs(save_dir, exist_ok=True)
        save_path = os.path.join(save_dir, f"{safe_title}.{fig_format}")
        plt.savefig(save_path, format=fig_format)
    plt.close()

def clean_data(df, x_col, y_col):
    """数据清理：去除空值和坐标为 0 的行"""
```

```
    df = df.dropna(subset=[x_col, y_col])
    df = df[(df[x_col] != 0) & (df[y_col] != 0)]
    return df

def calc_velocity(x, y):
    """计算速度"""
    x_v = [x.iloc[i + 1] - x.iloc[i] for i in range(len(x) - 1)]
    y_v = [y.iloc[i + 1] - y.iloc[i] for i in range(len(y) - 1)]
    return x_v, y_v

def calc_acceleration(x_v, y_v):
    """计算加速度"""
    x_a = [x_v[i + 1] - x_v[i] for i in range(len(x_v) - 1)]
    y_a = [y_v[i + 1] - y_v[i] for i in range(len(y_v) - 1)]
    return x_a, y_a

if __name__ == "__main__":

    # 读取数据
    file_path_1 = 'excel_data\附件 5\运动者 11 的跳远位置信息.xlsx'
    df = pd.read_excel(file_path_1)

    # 修改为实际列名
    for i in range(0, 33):
        lie = str(i)
        x_col = lie + '_X'
        y_col = lie + '_Y'

        df = clean_data(df, x_col, y_col)
        x = df[x_col]
        y = df[y_col]

        # print("总点数:", len(x))
        time = list(range(len(x)))

        plot_line_x(time, x, title=f"运动者 11 的第{lie}列 X 跳远位置信息", x_label="
时间", y_label="X 坐标")
        plot_line_y(time, y, title=f"运动者 11 的第{lie}列 Y 跳远位置信息", x_label="
时间", y_label="Y 坐标")
        # plot_xy_together(time, x, y, title=f"运动者 1 的第{lie}列 XY 跳远位置信息",
x_label="时间", y1_label="X 坐标", y2_label="Y 坐标")

        # 该列绘图完成
        print(f"第{lie}列绘图完成.")
```

附录四

支撑材料：运动趋势分析算法（python 3.10.4）

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from scipy.optimize import curve_fit
from scipy import signal
from scipy.signal import savgol_filter
import seaborn as sns
from matplotlib import rcParams
import warnings
warnings.filterwarnings('ignore')

# 设置中文字体
rcParams['font.sans-serif'] = ['SimHei']
rcParams['axes.unicode_minus'] = False

class LongJumpAnalyzer:
    """跳远动作分析器 - 基于 X 方向突变检测"""

    def __init__(self):
        # 人体体段质量系数定义
        self.body_segment_coefficients = {
            'head_neck': {'indices': list(range(0, 11)), 'coefficient': 0.081},
            'chest_abdomen': {'indices': [11, 12], 'coefficient': 0.355},
            'left_upper_arm': {'indices': [14], 'coefficient': 0.028},
            'right_upper_arm': {'indices': [13], 'coefficient': 0.028},
            'left_forearm': {'indices': [16], 'coefficient': 0.016},
            'right_forearm': {'indices': [15], 'coefficient': 0.016},
            'left_hand': {'indices': [18, 20, 22], 'coefficient': 0.006},
            'right_hand': {'indices': [17, 19, 21], 'coefficient': 0.006},
            'pelvis': {'indices': [23, 24], 'coefficient': 0.142},
            'left_thigh': {'indices': [25], 'coefficient': 0.1},
            'right_thigh': {'indices': [26], 'coefficient': 0.1},
            'left_shank': {'indices': [27], 'coefficient': 0.0465},
            'right_shank': {'indices': [28], 'coefficient': 0.0465},
            'left_foot': {'indices': [29, 31], 'coefficient': 0.0145},
            'right_foot': {'indices': [30, 32], 'coefficient': 0.0145}
        }

        self.df_cleaned = None
        self.center_of_gravity = None
        self.takeoff_frame = None
        self.landing_frame = None
        self.flight_trajectory = None
        self.pixel_to_meter_ratio = None
```

```
def preprocess_data(self, file_path, frame_column='帧号'):
    """
    简化的数据预处理：保留原始数据完整性
    """
    print("=== 简化数据预处理（保留完整数据）===")

    # 1. 读取数据
    df = pd.read_excel(file_path)
    print(f"原始数据: {df.shape}")

    # 2. 识别帧号列和数据列
    if frame_column not in df.columns:
        frame_column = df.columns[0]

    numeric_columns = df.select_dtypes(include=[np.number]).columns
    data_columns = [col for col in numeric_columns if col != frame_column]

    # 3. 只进行基本的缺失值处理，不删除任何行
    df_cleaned = df.copy()

    # 用前后值填充 NaN，保留 0 值
    for col in data_columns:
        df_cleaned[col] = df_cleaned[col].interpolate(method='linear')
        df_cleaned[col] = df_cleaned[col].fillna(method='ffill').fillna(method='bfill')

    # 4. 确保帧号连续
    df_cleaned.reset_index(drop=True, inplace=True)
    df_cleaned[frame_column] = range(len(df_cleaned))

    self.df_cleaned = df_cleaned
    self.frame_column = frame_column
    self.data_columns = data_columns

    print(f"数据预处理完成！数据形状: {df_cleaned.shape}")
    print("注意：保留了所有原始数据，包括 0 值")
    return df_cleaned

def calculate_center_of_gravity(self):
    """
    计算重心在每一帧中的像素位置
    """
    print("=== 计算重心位置 ===")

    if self.df_cleaned is None:
        raise ValueError("请先进行数据预处理")

    frames = len(self.df_cleaned)
```

```
center_of_gravity = []

for frame_idx in range(frames):
    total_weighted_x = 0
    total_weighted_y = 0
    total_coefficient = 0

    # 遍历每个体段
    for segment_name, segment_info in self.body_segment_coefficients.items():
        segment_x = 0
        segment_y = 0
        valid_points = 0

        # 计算该体段的重心位置
        for point_idx in segment_info['indices']:
            x_col = f"{point_idx}_X"
            y_col = f"{point_idx}_Y"

            if x_col in self.data_columns and y_col in self.data_columns:
                x_val = self.df_cleaned.iloc[frame_idx][x_col]
                y_val = self.df_cleaned.iloc[frame_idx][y_col]

                # 只要不是 NaN 就使用（包括 0 值）
                if not np.isnan(x_val) and not np.isnan(y_val):
                    segment_x += x_val
                    segment_y += y_val
                    valid_points += 1

        # 如果该体段有有效点，计算加权重心
        if valid_points > 0:
            segment_x /= valid_points
            segment_y /= valid_points

            coefficient = segment_info['coefficient']
            total_weighted_x += segment_x * coefficient
            total_weighted_y += segment_y * coefficient
            total_coefficient += coefficient

    # 计算整体重心
    if total_coefficient > 0:
        cog_x = total_weighted_x / total_coefficient
        cog_y = total_weighted_y / total_coefficient
        center_of_gravity.append([cog_x, cog_y])
    else:
        center_of_gravity.append([np.nan, np.nan])

self.center_of_gravity = np.array(center_of_gravity)
print(f"重心计算完成！共 {len(center_of_gravity)} 帧")
```

```
        print("前 5 帧重心位置示例:", self.center_of_gravity[:5])
        return self.center_of_gravity

    def analyze_motion_phases_x_based(self, max_frames=200):
        """
        基于重心 X 方向突变的精确运动阶段分析
        """
        print("=== 基于 X 方向突变的运动阶段分析 ===")

        if self.center_of_gravity is None:
            raise ValueError("请先计算重心位置")

        # 1. 限制分析帧数
        total_frames = len(self.center_of_gravity)
        analysis_frames = min(max_frames, total_frames)

        print(f"总帧数: {total_frames}, 分析帧数: {analysis_frames}")

        # 提取分析范围内的重心数据
        cog_x = self.center_of_gravity[:analysis_frames, 0]
        cog_y = self.center_of_gravity[:analysis_frames, 1]
        frames = np.arange(analysis_frames)

        # 2. 轻微平滑以减少噪声，但保持突变特征
        window_size = 3
        if len(cog_x) >= window_size:
            cog_x_smooth = savgol_filter(cog_x, window_length=window_size,
polyorder=1)
            cog_y_smooth = savgol_filter(cog_y, window_length=window_size,
polyorder=1)
        else:
            cog_x_smooth = cog_x
            cog_y_smooth = cog_y

        # 3. 计算 X 方向的变化率和趋势
        print("分析 X 方向变化特征...")

        # X 方向的一阶和二阶差分
        x_diff1 = np.gradient(cog_x_smooth)  # 速度
        x_diff2 = np.gradient(x_diff1)     # 加速度

        # 计算移动窗口内的 X 方向变化趋势
        trend_window = 10
        x_trends = []
        x_slopes = []

        for i in range(len(cog_x_smooth)):
            start_idx = max(0, i - trend_window//2)
```

```
        end_idx = min(len(cog_x_smooth), i + trend_window//2 + 1)

        if end_idx - start_idx >= 3:
            # 计算线性趋势斜率
            x_vals = cog_x_smooth[start_idx:end_idx]
            frame_vals = np.arange(len(x_vals))
            slope = np.polyfit(frame_vals, x_vals, 1)[0]
            x_slopes.append(slope)

            # 判断趋势类型：平缓、上升、下降
            if abs(slope) < 0.5:
                trend = 0  # 平缓
            elif slope > 0.5:
                trend = 1  # 上升
            else:
                trend = -1  # 下降
            x_trends.append(trend)
        else:
            x_slopes.append(0)
            x_trends.append(0)

    x_slopes = np.array(x_slopes)
    x_trends = np.array(x_trends)

    # 4. 检测 X 方向的突变点
    print("检测 X 方向突变点...")

    # 动态确定斜率阈值（基于数据特征）
    slope_std = np.std(x_slopes)
    slope_mean = np.mean(x_slopes)
    slope_threshold = max(1.0, slope_mean + 2 * slope_std)  # 自适应阈值

    print(f"X 方向斜率统计: 均值={slope_mean:.3f}, 标准差={slope_std:.3f}")
    print(f"使用斜率阈值: {slope_threshold:.3f}")

    # 找到 X 方向开始显著上升的点（起跳）
    takeoff_candidates = []
    for i in range(15, len(x_slopes) - 15):
        # 当前斜率显著为正
        current_slope_high = x_slopes[i] > slope_threshold

        # 前面一段时间斜率较小（平缓期）
        prev_slopes = x_slopes[max(0, i-15):i]
        prev_slope_low = np.mean(prev_slopes) < slope_threshold * 0.4

        # 后面一段时间斜率持续较高（确保是持续上升）
        next_slopes = x_slopes[i:min(len(x_slopes), i+15)]
        next_slope_sustained = np.mean(next_slopes) > slope_threshold * 0.6
```

```
        # 确保有明显的斜率跳跃
        slope_jump = x_slopes[i] - np.mean(prev_slopes) > slope_threshold * 0.5

        if current_slope_high and prev_slope_low and next_slope_sustained and
slope_jump:
            takeoff_candidates.append((i, x_slopes[i], slope_jump))

    # 找到 X 方向停止显著上升的点（落地）
    landing_candidates = []
    for i in range(15, len(x_slopes) - 15):
        # 当前斜率较小（不再上升）
        current_slope_low = x_slopes[i] < slope_threshold * 0.4

        # 前面一段时间斜率较高（上升期）
        prev_slopes = x_slopes[max(0, i-15):i]
        prev_slope_high = np.mean(prev_slopes) > slope_threshold * 0.6

        # 后面一段时间斜率持续较小（确保进入平缓期）
        next_slopes = x_slopes[i:min(len(x_slopes), i+15)]
        next_slope_sustained_low = np.mean(next_slopes) < slope_threshold * 0.5

        # 确保有明显的斜率下降
        slope_drop = np.mean(prev_slopes) - x_slopes[i] > slope_threshold * 0.4

        if current_slope_low and prev_slope_high and next_slope_sustained_low and
slope_drop:
            landing_candidates.append((i, x_slopes[i], slope_drop))

    # 5. 选择最佳的起跳和落地时刻
    print(f"起跳候选点: {[c[0] for c in takeoff_candidates]}")
    print(f"落地候选点: {[c[0] for c in landing_candidates]}")

    # 选择起跳时刻
    if takeoff_candidates:
        # 选择斜率跳跃最大的点
        takeoff_candidates.sort(key=lambda x: x[2], reverse=True)  # 按斜率跳跃排
序
        self.takeoff_frame = takeoff_candidates[0][0]
        print(f" 选 择 起 跳 帧  {self.takeoff_frame} ， 斜 率 跳 跃 :
{takeoff_candidates[0][2]:.3f}")
    else:
        # 备用方案：寻找斜率首次超过阈值的点
        high_slope_indices = np.where(x_slopes > slope_threshold)[0]
        if len(high_slope_indices) > 0:
            self.takeoff_frame = high_slope_indices[0]
        else:
            # 进一步备用：寻找斜率最大的点
```

```
            self.takeoff_frame = np.argmax(x_slopes)
        print(f"备用方案：选择起跳帧 {self.takeoff_frame}")

    # 选择落地时刻
    if landing_candidates:
        # 选择在起跳之后的第一个明显下降点
        valid_landing = [c for c in landing_candidates if c[0] > self.takeoff_frame +
10]
        if valid_landing:
            valid_landing.sort(key=lambda x: x[2], reverse=True) # 按斜率下降排序
            self.landing_frame = valid_landing[0][0]
            print(f" 选 择 落 地 帧  {self.landing_frame} ， 斜 率 下 降 :
{valid_landing[0][2]:.3f}")
        else:
            self.landing_frame = landing_candidates[0][0]
    else:
        # 备用方案：在起跳后寻找斜率显著减小的点
        post_takeoff_slopes = x_slopes[self.takeoff_frame + 10:]
        if len(post_takeoff_slopes) > 0:
            # 寻找斜率首次显著下降的点
            low_slope_threshold = slope_threshold * 0.3
            low_slope_indices = np.where(post_takeoff_slopes <
low_slope_threshold)[0]
            if len(low_slope_indices) > 0:
                self.landing_frame = self.takeoff_frame + 10 + low_slope_indices[0]
            else:
                self.landing_frame = self.takeoff_frame + len(post_takeoff_slopes) // 2
        else:
            self.landing_frame = min(self.takeoff_frame + 30, analysis_frames - 1)
        print(f"备用方案：选择落地帧 {self.landing_frame}")

    # 确保落地帧在合理范围内
    self.landing_frame = min(self.landing_frame, analysis_frames - 1)
    self.landing_frame = max(self.landing_frame, self.takeoff_frame + 5)

    # 6. 基于 X 变化重新定义助跑阶段
    approach_end = self.takeoff_frame

    # 助跑开始：寻找 X 方向开始有规律运动的点
    approach_start = 0
    for i in range(0, min(50, approach_end)):
        if x_slopes[i] > 0.1: # 开始有向前运动
            approach_start = i
            break

    # 7. 计算运动学参数
    velocity_x = np.gradient(cog_x_smooth)
    velocity_y = np.gradient(cog_y_smooth)
```

```
        acceleration_x = np.gradient(velocity_x)
        acceleration_y = np.gradient(velocity_y)

        # 计算 X 方向运动一致性
        x_direction_consistency = []
        window_size = 8
        for i in range(len(velocity_x)):
            start_idx = max(0, i - window_size//2)
            end_idx = min(len(velocity_x), i + window_size//2 + 1)
            window_velocities = velocity_x[start_idx:end_idx]
            consistency = np.sum(window_velocities > 0) / len(window_velocities)
            x_direction_consistency.append(consistency)

        x_direction_consistency = np.array(x_direction_consistency)

        # 计算累积 X 方向距离
        cumulative_x_distance =
np.cumsum(np.abs(np.diff(np.concatenate([[cog_x_smooth[0]], cog_x_smooth]))))

        # 8. 构建特征用于聚类分析
        features = np.column_stack([
            frames / analysis_frames,  # 时间进度
            (cog_x_smooth - np.min(cog_x_smooth)) / (np.max(cog_x_smooth) -
np.min(cog_x_smooth)),  # X 位置标准化
            (cog_y_smooth - np.min(cog_y_smooth)) / (np.max(cog_y_smooth) -
np.min(cog_y_smooth)),  # Y 位置标准化
            x_slopes,  # X 方向斜率（关键特征）
            x_trends,  # X 方向趋势
            velocity_x,  # X 速度
            velocity_y   # Y 速度
        ])

        # 9. 标准化特征并进行聚类
        scaler = StandardScaler()
        features_scaled = scaler.fit_transform(features)

        n_clusters = 3  # 3 个阶段：助跑、滞空、落地
        kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init=10)
        clusters = kmeans.fit_predict(features_scaled)

        # 10. 分析各阶段的特征
        print("\n=== 基于 X 突变的阶段识别结果 ===")
        print(f"助跑阶段: 帧 {approach_start} - {approach_end-1}")
        print(f"滞空阶段: 帧 {self.takeoff_frame} - {self.landing_frame}")
        print(f"落地阶段: 帧 {self.landing_frame+1} - {analysis_frames-1}")

        # 各阶段 X 方向特征分析
        if approach_end > approach_start:
```

```
            approach_x_distance = cog_x_smooth[approach_end-1] - cog_x_smooth[approach_start]
            approach_avg_slope = np.mean(x_slopes[approach_start:approach_end])
            print(f"\n助跑阶段特征:")
            print(f"  X方向位移: {approach_x_distance:.2f} 像素")
            print(f"  平均斜率: {approach_avg_slope:.3f}")

        if self.landing_frame > self.takeoff_frame:
            flight_x_distance = cog_x_smooth[self.landing_frame] - cog_x_smooth[self.takeoff_frame]
            flight_avg_slope = np.mean(x_slopes[self.takeoff_frame:self.landing_frame])
            print(f"\n滞空阶段特征:")
            print(f"  X方向位移: {flight_x_distance:.2f} 像素")
            print(f"  平均斜率: {flight_avg_slope:.3f}")
            print(f"  滞空时长: {self.landing_frame - self.takeoff_frame} 帧")

        # 11. 保存分析结果
        self.motion_analysis = {
            'analysis_frames': analysis_frames,
            'frames': frames,
            'cog_x': cog_x_smooth,
            'cog_y': cog_y_smooth,
            'velocity_x': velocity_x,
            'velocity_y': velocity_y,
            'acceleration_x': acceleration_x,
            'acceleration_y': acceleration_y,
            'x_slopes': x_slopes,
            'x_trends': x_trends,
            'x_direction_consistency': x_direction_consistency,
            'cumulative_x_distance': cumulative_x_distance,
            'clusters': clusters,
            'approach_start': approach_start,
            'approach_end': approach_end,
            'takeoff_frame': self.takeoff_frame,
            'landing_frame': self.landing_frame,
            'features': features,
            'features_scaled': features_scaled,
            'slope_threshold': slope_threshold,
            'takeoff_candidates': takeoff_candidates,
            'landing_candidates': landing_candidates
        }

        return self.motion_analysis

    def analyze_motion_phases(self, max_frames=200):
        """调用基于X突变的运动阶段分析方法"""
        return self.analyze_motion_phases_x_based(max_frames)

    def fit_flight_trajectory(self, degree=2):
```

```
    """
    多项式拟合滞空期间的重心轨迹
    """
    print("=== 拟合滞空轨迹 ===")

    if self.takeoff_frame is None or self.landing_frame is None:
        raise ValueError("请先分析运动阶段")

    # 提取滞空期间的重心轨迹
    flight_frames = range(self.takeoff_frame, self.landing_frame + 1)
    flight_cog = self.center_of_gravity[flight_frames]

    flight_x = flight_cog[:, 0]
    flight_y = flight_cog[:, 1]
    flight_t = np.array(flight_frames)

    # 多项式拟合
    print(f"使用 {degree} 次多项式拟合轨迹")
    print("选择 2 次多项式的原因：")
    print("1. 物理原理：在忽略空气阻力的情况下，抛物运动的轨迹是抛物线")
    print("2. 数学简洁：2 次多项式能够很好地描述重力作用下的运动轨迹")
    print("3. 拟合稳定：相比高次多项式，2 次多项式不易过拟合")

    # X 方向拟合（时间-水平位置）
    x_coeffs = np.polyfit(flight_t, flight_x, degree)
    x_poly = np.poly1d(x_coeffs)

    # Y 方向拟合（时间-垂直位置）
    y_coeffs = np.polyfit(flight_t, flight_y, degree)
    y_poly = np.poly1d(y_coeffs)

    # 计算拟合优度
    x_fitted = x_poly(flight_t)
    y_fitted = y_poly(flight_t)

    x_r2 = 1 - np.sum((flight_x - x_fitted) ** 2) / np.sum((flight_x -
np.mean(flight_x)) ** 2)
    y_r2 = 1 - np.sum((flight_y - y_fitted) ** 2) / np.sum((flight_y -
np.mean(flight_y)) ** 2)

    print(f"X 方向拟合优度 R²: {x_r2:.4f}")
    print(f"Y 方向拟合优度 R²: {y_r2:.4f}")

    # 抛物线方程（X-Y 关系）
    try:
        xy_coeffs = np.polyfit(flight_x, flight_y, degree)
        xy_poly = np.poly1d(xy_coeffs)
```

```
            print(f"\n 抛物线轨迹方程: y = {xy_coeffs[0]:.6f}x² + {xy_coeffs[1]:.6f}x + {xy_coeffs[2]:.6f}")

        except:
            xy_poly = None
            print("无法拟合 X-Y 抛物线方程")

        self.flight_trajectory = {
            'frames': flight_frames,
            'x_data': flight_x,
            'y_data': flight_y,
            't_data': flight_t,
            'x_coeffs': x_coeffs,
            'y_coeffs': y_coeffs,
            'x_poly': x_poly,
            'y_poly': y_poly,
            'xy_poly': xy_poly,
            'x_r2': x_r2,
            'y_r2': y_r2
        }

        return self.flight_trajectory

    def calculate_jump_distance(self, toe_point_idx=29, heel_point_idx=30, actual_distance_m=None):
        """
        分析脚尖起跳前的位置与脚后跟落地时的位置，计算跳远成绩
        """
        print("=== 计算跳远成绩 ===")

        if self.takeoff_frame is None or self.landing_frame is None:
            raise ValueError("请先分析运动阶段")

        # 获取脚尖起跳前的位置和脚跟落地位置
        toe_x_col = f"{toe_point_idx}_X"
        heel_x_col = f"{heel_point_idx}_X"

        if toe_x_col not in self.data_columns or heel_x_col not in self.data_columns:
            print(f"警告：找不到脚尖({toe_x_col})或脚跟({heel_x_col})数据列")
            return None

        # 起跳位置（脚尖）
        takeoff_x = self.df_cleaned.iloc[self.takeoff_frame][toe_x_col]

        # 落地位置（脚跟）
        landing_x = self.df_cleaned.iloc[self.landing_frame][heel_x_col]

        # 像素距离
        pixel_distance = abs(landing_x - takeoff_x)
```

```
        print(f"起跳位置 (脚尖): {takeoff_x:.2f} 像素")
        print(f"落地位置 (脚跟): {landing_x:.2f} 像素")
        print(f"像素距离: {pixel_distance:.2f} 像素")

        # 如果提供了实际距离，计算像素尺标
        if actual_distance_m is not None:
            self.pixel_to_meter_ratio = actual_distance_m / pixel_distance
            calculated_distance = pixel_distance * self.pixel_to_meter_ratio

            print(f"实际跳远成绩: {actual_distance_m:.2f} 米")
            print(f"像素尺标: 1 像素 = {self.pixel_to_meter_ratio:.6f} 米")
            print(f"计算距离: {calculated_distance:.2f} 米")
            print(f"误差: {abs(calculated_distance - actual_distance_m):.4f} 米")
        else:
            calculated_distance = pixel_distance
            print("未提供实际距离，结果以像素为单位")

        self.jump_analysis = {
            'takeoff_x': takeoff_x,
            'landing_x': landing_x,
            'pixel_distance': pixel_distance,
            'calculated_distance': calculated_distance,
            'pixel_to_meter_ratio': self.pixel_to_meter_ratio
        }

        return self.jump_analysis

    def visualize_x_based_analysis(self, save_plots=True):
        """可视化基于 X 突变的分析结果"""
        if not hasattr(self, 'motion_analysis'):
            print("请先进行运动阶段分析")
            return

        fig, axes = plt.subplots(3, 2, figsize=(15, 12))
        fig.suptitle('基于 X 方向突变的运动阶段分析', fontsize=16, fontweight='bold')

        analysis = self.motion_analysis
        frames = analysis['frames']

        # 1. 重心 X 位置变化和阶段划分
        ax1 = axes[0, 0]
        ax1.plot(frames, analysis['cog_x'], 'b-', linewidth=2, label='重心 X 位置')

        # 标记三个阶段
        ax1.axvspan(analysis['approach_start'], analysis['approach_end']-1,
                    alpha=0.2, color='green', label='助跑阶段')
        ax1.axvspan(self.takeoff_frame, self.landing_frame,
```

```
            alpha=0.2, color='red', label='滞空阶段')
    ax1.axvspan(self.landing_frame+1, len(frames)-1,
            alpha=0.2, color='orange', label='落地阶段')

    ax1.axvline(x=self.takeoff_frame, color='red', linestyle='--', linewidth=2, label='
起跳')
    ax1.axvline(x=self.landing_frame, color='purple', linestyle='--', linewidth=2,
label='落地')

    ax1.set_xlabel('帧数')
    ax1.set_ylabel('X 位置 (像素)')
    ax1.set_title('重心 X 位置变化与阶段划分')
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    # 2. X 方向斜率变化
    ax2 = axes[0, 1]
    ax2.plot(frames, analysis['x_slopes'], 'r-', linewidth=2, label='X 方向斜率')
    ax2.axhline(y=analysis['slope_threshold'], color='gray', linestyle='--',
            alpha=0.7, label=f'阈值({analysis["slope_threshold"]:.2f})')
    ax2.axvline(x=self.takeoff_frame, color='red', linestyle='--', alpha=0.7, label='起
跳')
    ax2.axvline(x=self.landing_frame, color='purple', linestyle='--', alpha=0.7,
label='落地')

    # 标记候选点
    if 'takeoff_candidates' in analysis:
        takeoff_points = [c[0] for c in analysis['takeoff_candidates']]
        if takeoff_points:
            ax2.scatter(takeoff_points, [analysis['x_slopes'][i] for i in takeoff_points],
                    color='red', s=50, marker='^', label='起跳候选', zorder=5)

    if 'landing_candidates' in analysis:
        landing_points = [c[0] for c in analysis['landing_candidates']]
        if landing_points:
            ax2.scatter(landing_points, [analysis['x_slopes'][i] for i in landing_points],
                    color='purple', s=50, marker='v', label='落地候选', zorder=5)

    ax2.set_xlabel('帧数')
    ax2.set_ylabel('斜率')
    ax2.set_title('X 方向斜率变化（突变检测）')
    ax2.legend()
    ax2.grid(True, alpha=0.3)

    # 3. X 方向趋势
    ax3 = axes[1, 0]
    trend_colors = ['blue', 'gray', 'red']
```

```
        trend_labels = ['下降', '平缓', '上升']
        for i, (color, label) in enumerate(zip(trend_colors, trend_labels)):
            mask = analysis['x_trends'] == (i-1)
            if np.any(mask):
                ax3.scatter(frames[mask], analysis['cog_x'][mask],
                           c=color, alpha=0.6, label=label, s=15)

        ax3.axvline(x=self.takeoff_frame, color='red', linestyle='--', alpha=0.7, label='起跳')
        ax3.axvline(x=self.landing_frame, color='purple', linestyle='--', alpha=0.7, label='落地')
        ax3.set_xlabel('帧数')
        ax3.set_ylabel('X 位置 (像素)')
        ax3.set_title('X 方向运动趋势分析')
        ax3.legend()
        ax3.grid(True, alpha=0.3)

        # 4. 速度对比
        ax4 = axes[1, 1]
        ax4.plot(frames, analysis['velocity_x'], 'r-', alpha=0.7, label='X 速度')
        ax4.plot(frames, analysis['velocity_y'], 'b-', alpha=0.7, label='Y 速度')
        ax4.axhline(y=0, color='black', linestyle='-', alpha=0.3)
        ax4.axvline(x=self.takeoff_frame, color='red', linestyle='--', alpha=0.7, label='起跳')
        ax4.axvline(x=self.landing_frame, color='purple', linestyle='--', alpha=0.7, label='落地')
        ax4.set_xlabel('帧数')
        ax4.set_ylabel('速度 (像素/帧)')
        ax4.set_title('X/Y 方向速度对比')
        ax4.legend()
        ax4.grid(True, alpha=0.3)

        # 5. 阶段聚类结果
        ax5 = axes[2, 0]
        scatter = ax5.scatter(frames, analysis['cog_x'], c=analysis['clusters'],
                              cmap='viridis', alpha=0.7, s=20)
        ax5.axvline(x=self.takeoff_frame, color='red', linestyle='--', alpha=0.7, label='起跳')
        ax5.axvline(x=self.landing_frame, color='purple', linestyle='--', alpha=0.7, label='落地')
        ax5.set_xlabel('帧数')
        ax5.set_ylabel('X 位置 (像素)')
        ax5.set_title('运动阶段聚类结果')
        ax5.legend()
        plt.colorbar(scatter, ax=ax5, label='聚类阶段')
        ax5.grid(True, alpha=0.3)
```

```
        # 6. 重心轨迹
        ax6 = axes[2, 1]
        # 用不同颜色显示不同阶段
        approach_mask = frames < self.takeoff_frame
        flight_mask = (frames >= self.takeoff_frame) & (frames <= self.landing_frame)
        landing_mask = frames > self.landing_frame

        if np.any(approach_mask):
            ax6.plot(analysis['cog_x'][approach_mask], analysis['cog_y'][approach_mask],
                     'g-', alpha=0.8, linewidth=2, label='助跑阶段')

        if np.any(flight_mask):
            ax6.plot(analysis['cog_x'][flight_mask], analysis['cog_y'][flight_mask],
                     'r-', alpha=0.8, linewidth=3, label='滞空阶段')

        if np.any(landing_mask):
            ax6.plot(analysis['cog_x'][landing_mask], analysis['cog_y'][landing_mask],
                     'orange', alpha=0.8, linewidth=2, label='落地阶段')

        ax6.scatter(analysis['cog_x'][self.takeoff_frame],
analysis['cog_y'][self.takeoff_frame],
                    color='red', s=100, marker='o', label='起跳点', zorder=5)
        ax6.scatter(analysis['cog_x'][self.landing_frame],
analysis['cog_y'][self.landing_frame],
                    color='purple', s=100, marker='s', label='落地点', zorder=5)

        ax6.invert_yaxis()  # 图像坐标系 Y 轴向下
        ax6.set_xlabel('X 位置 (像素)')
        ax6.set_ylabel('Y 位置 (像素)')
        ax6.set_title('重心轨迹（按阶段着色）')
        ax6.legend()
        ax6.grid(True, alpha=0.3)

        plt.tight_layout()

        if save_plots:
            plt.savefig('基于 X 突变的运动分析.png', dpi=300, bbox_inches='tight')
            print("基于 X 突变的分析图表已保存")

        plt.show()

    def visualize_analysis(self, save_plots=True):
        """生成所有可视化结果"""
        # 首先显示基于 X 突变的分析
        self.visualize_x_based_analysis(save_plots)

        # 然后显示综合分析图
        print("\n=== 生成综合分析图表 ===")
```

```
        fig, axes = plt.subplots(2, 3, figsize=(18, 12))
        fig.suptitle('跳远动作综合分析结果', fontsize=16, fontweight='bold')

        # 1. 重心轨迹图
        ax1 = axes[0, 0]
        if self.center_of_gravity is not None:
            ax1.plot(self.center_of_gravity[:, 0], self.center_of_gravity[:, 1], 'b-',
alpha=0.7)
            ax1.invert_yaxis()  # 图像坐标系 Y 轴向下
            ax1.scatter(self.center_of_gravity[self.takeoff_frame, 0],
                        self.center_of_gravity[self.takeoff_frame, 1],
                        color='green', s=100, label='起跳点', zorder=5)
            ax1.scatter(self.center_of_gravity[self.landing_frame, 0],
                        self.center_of_gravity[self.landing_frame, 1],
                        color='red', s=100, label='落地点', zorder=5)
            ax1.set_xlabel('水平位置 (像素)')
            ax1.set_ylabel('垂直位置 (像素)')
            ax1.set_title('重心轨迹')
            ax1.legend()
            ax1.grid(True, alpha=0.3)

        # 2. 滞空轨迹拟合
        ax2 = axes[0, 1]
        if self.flight_trajectory is not None:
            traj = self.flight_trajectory
            ax2.scatter(traj['x_data'], traj['y_data'], color='blue', alpha=0.7, label='实际轨
迹')

            # 拟合曲线
            x_smooth = np.linspace(min(traj['x_data']), max(traj['x_data']), 100)
            if traj['xy_poly'] is not None:
                y_smooth = traj['xy_poly'](x_smooth)
                ax2.plot(x_smooth, y_smooth, 'r-', linewidth=2, label='拟合抛物线')
            ax2.invert_yaxis()  # 图像坐标系 Y 轴向下
            ax2.set_xlabel('水平位置 (像素)')
            ax2.set_ylabel('垂直位置 (像素)')
            ax2.set_title('滞空轨迹拟合')
            ax2.legend()
            ax2.grid(True, alpha=0.3)

        # 3. 垂直速度变化
        ax3 = axes[0, 2]
        if hasattr(self, 'motion_analysis'):
            frames = self.motion_analysis['frames']
            ax3.plot(frames, self.motion_analysis['velocity_y'], 'b-', alpha=0.7)
            ax3.axvline(x=self.takeoff_frame, color='green', linestyle='--', label='起跳')
```

```
            ax3.axvline(x=self.landing_frame, color='red', linestyle='--', label='落地')
            ax3.invert_yaxis()  # 图像坐标系 Y 轴向下
            ax3.set_xlabel('帧数')
            ax3.set_ylabel('垂直速度 (像素/帧)')
            ax3.set_title('垂直速度变化')
            ax3.legend()
            ax3.grid(True, alpha=0.3)

        # 4. 运动阶段聚类
        ax4 = axes[1, 0]
        if hasattr(self, 'motion_analysis'):
            analysis = self.motion_analysis
            scatter = ax4.scatter(analysis['frames'], analysis['cog_y'],
                                  c=analysis['clusters'], cmap='viridis', alpha=0.7)
            ax4.invert_yaxis()  # 图像坐标系 Y 轴向下
            ax4.set_xlabel('帧数')
            ax4.set_ylabel('垂直位置 (像素)')
            ax4.set_title('运动阶段聚类')
            plt.colorbar(scatter, ax=ax4)
            ax4.grid(True, alpha=0.3)

        # 5. 体段重心分布
        ax5 = axes[1, 1]
        segment_names = list(self.body_segment_coefficients.keys())
        coefficients = [info['coefficient'] for info in
self.body_segment_coefficients.values()]

        ax5.barh(segment_names, coefficients, color='skyblue', alpha=0.7)
        ax5.set_xlabel('质量系数')
        ax5.set_title('人体体段质量系数分布')
        ax5.grid(True, alpha=0.3)

        # 6. 跳远成绩分析
        ax6 = axes[1, 2]
        if hasattr(self, 'jump_analysis'):
            positions = ['起跳位置', '落地位置']
            x_positions = [self.jump_analysis['takeoff_x'],
self.jump_analysis['landing_x']]

            bars = ax6.bar(positions, x_positions, color=['green', 'red'], alpha=0.7)
            ax6.set_ylabel('水平位置 (像素)')
            ax6.set_title(f'跳远成绩: {self.jump_analysis["pixel_distance"]:.2f} 像素')

            # 添加数值标签
            for bar, value in zip(bars, x_positions):
                height = bar.get_height()
                ax6.text(bar.get_x() + bar.get_width()/2., height + 1,
                         f'{value:.1f}', ha='center', va='bottom')
```

```
        ax6.grid(True, alpha=0.3)

    plt.tight_layout()

    if save_plots:
        plt.savefig('跳远综合分析结果.png', dpi=300, bbox_inches='tight')
        print("综合分析图表已保存")

    plt.show()

  def analyze_x_motion_phases_summary(self):
    """生成 X 方向运动分析摘要"""
    if not hasattr(self, 'motion_analysis'):
        print("请先进行运动阶段分析")
        return

    analysis = self.motion_analysis

    print("\n" + "="*60)
    print("    基于 X 方向突变的运动分析摘要")
    print("="*60)

    print(f"分析帧数范围: 0 - {analysis['analysis_frames']-1}")
    print(f"总 X 方向位移: {analysis['cog_x'][-1] - analysis['cog_x'][0]:.2f} 像素")
    print(f"使用的斜率阈值: {analysis['slope_threshold']:.3f}")

    print(f"\n 阶段划分结果:")
    print(f"助跑阶段: 帧 {analysis['approach_start']} - {analysis['approach_end']-1}
({analysis['approach_end'] - analysis['approach_start']} 帧)")
    print(f"滞空阶段: 帧 {self.takeoff_frame} - {self.landing_frame}
({self.landing_frame - self.takeoff_frame} 帧)")
    print(f"落地阶段: 帧 {self.landing_frame+1} - {analysis['analysis_frames']-1}
({analysis['analysis_frames'] - self.landing_frame - 1} 帧)")

    print(f"\n 各阶段特征:")

    # 助跑阶段
    approach_frames = range(analysis['approach_start'], analysis['approach_end'])
    if len(approach_frames) > 0:
        approach_x_vel = np.mean(analysis['velocity_x'][approach_frames])
        approach_x_slope = np.mean(analysis['x_slopes'][approach_frames])
        print(f"助跑阶段:")
        print(f"  平均 X 速度: {approach_x_vel:.2f} 像素/帧")
        print(f"  平均 X 斜率: {approach_x_slope:.3f}")

    # 滞空阶段
```

```
        if self.landing_frame > self.takeoff_frame:
            flight_frames = range(self.takeoff_frame, self.landing_frame)
            flight_x_vel = np.mean(analysis['velocity_x'][flight_frames])
            flight_x_slope = np.mean(analysis['x_slopes'][flight_frames])
            flight_x_distance = analysis['cog_x'][self.landing_frame] -
analysis['cog_x'][self.takeoff_frame]
            print(f"滞空阶段:")
            print(f"  平均 X 速度: {flight_x_vel:.2f} 像素/帧")
            print(f"  平均 X 斜率: {flight_x_slope:.3f}")
            print(f"  X 方向位移: {flight_x_distance:.2f} 像素")

        print("="*60)

    def generate_report(self):
        """生成分析报告"""
        print("\n" + "="*70)
        print("          跳远动作分析报告（基于 X 方向突变检测）")
        print("="*70)

        if self.df_cleaned is not None:
            print(f"数据概况:")
            print(f"  - 总帧数: {len(self.df_cleaned)}")
            print(f"  - 数据点数: {len(self.data_columns)}")
            print(f"  - 数据处理: 保留完整数据，仅填补缺失值")

        if hasattr(self, 'motion_analysis'):
            analysis = self.motion_analysis
            print(f"\n 运动阶段分析:")
            print(f"  - 检测方法: 基于重心 X 方向斜率突变")
            print(f"  - 斜率阈值: {analysis['slope_threshold']:.3f}")
            print(f"  - 起跳帧: {self.takeoff_frame}")
            print(f"  - 落地帧: {self.landing_frame}")
            print(f"  - 滞空时长: {self.landing_frame - self.takeoff_frame} 帧")

            if 'takeoff_candidates' in analysis and analysis['takeoff_candidates']:
                print(f"  - 起跳候选点数: {len(analysis['takeoff_candidates'])}")
            if 'landing_candidates' in analysis and analysis['landing_candidates']:
                print(f"  - 落地候选点数: {len(analysis['landing_candidates'])}")

        if self.flight_trajectory is not None:
            print(f"\n 轨迹拟合结果:")
            print(f"  - X 方向拟合优度 R²: {self.flight_trajectory['x_r2']:.4f}")
            print(f"  - Y 方向拟合优度 R²: {self.flight_trajectory['y_r2']:.4f}")

            if self.flight_trajectory['xy_poly'] is not None:
                coeffs = self.flight_trajectory['xy_poly'].coefficients
                print(f"  - 抛物线方程: y = {coeffs[0]:.6f}x² + {coeffs[1]:.6f}x +
```

```
{coeffs[2]:.6f}")

        if hasattr(self, 'jump_analysis'):
            print(f"\n 跳远成绩:")
            print(f"  - 像素距离: {self.jump_analysis['pixel_distance']:.2f} 像素")
            if self.pixel_to_meter_ratio is not None:
                print(f"  - 实际距离: {self.jump_analysis['calculated_distance']:.2f} 米")
                print(f"  - 像素尺标: 1 像素 = {self.pixel_to_meter_ratio:.6f}米")

        print("="*70)

        # 调用 X 方向运动分析摘要
        self.analyze_x_motion_phases_summary()

# 主程序
def quick_analysis(file_path, actual_distance_m=None, max_frames=200):
    """快速跳远分析函数（基于 X 突变检测）"""
    print("开始快速跳远分析（X 突变检测版）...")

    analyzer = LongJumpAnalyzer()

    try:
        print("1/5 简化数据预处理...")
        analyzer.preprocess_data(file_path)

        print("2/5 计算重心...")
        analyzer.calculate_center_of_gravity()

        print("3/5 X 突变检测分析...")
        analyzer.analyze_motion_phases(max_frames=max_frames)

        print("4/5 拟合轨迹...")
        analyzer.fit_flight_trajectory()

        print("5/5 计算成绩...")
        analyzer.calculate_jump_distance(actual_distance_m=actual_distance_m)

        analyzer.visualize_analysis()
        analyzer.generate_report()

        return analyzer

    except Exception as e:
        print(f"分析过程中出现错误: {e}")
        return None

# 使用示例
if __name__ == "__main__":
```

```
    # 运行主程序，计算问题一
    result = quick_analysis(
        file_path="excel_data\附件 5\运动者 11 的跳远位置信息.xlsx",
        actual_distance_m=1.58,
        max_frames=200
    )

    # 运行主程序，计算
    # result = quick_analysis(
    #     file_path="excel_data\附件 5\运动者 11 的跳远位置信息.xlsx",
    #     actual_distance_m=0.003515,     # 传入比例尺，从而计算成绩，1 像素 =
0.003515 米
    #     max_frames=200
    # )
```

附录五

支撑材料：集成学习模型（python 3.10.4）

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from scipy.optimize import curve_fit
from scipy import signal
from scipy.signal import savgol_filter
import seaborn as sns
from matplotlib import rcParams
import warnings
import os
import glob
warnings.filterwarnings('ignore')

# 设置中文字体和图表样式
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

class PostureAnalyzer:
    """姿势分析器 - 整合滞空检测和姿势特征计算"""

    def __init__(self):
        # 像素到米的转换比例
        self.pixel_to_meter_ratio = 0.003515
```

```
        # 人体体段质量系数定义
        self.body_segment_coefficients = {
            'head_neck': {'indices': list(range(0, 11)), 'coefficient': 0.081},
            'chest_abdomen': {'indices': [11, 12], 'coefficient': 0.355},
            'left_upper_arm': {'indices': [14], 'coefficient': 0.028},
            'right_upper_arm': {'indices': [13], 'coefficient': 0.028},
            'left_forearm': {'indices': [16], 'coefficient': 0.016},
            'right_forearm': {'indices': [15], 'coefficient': 0.016},
            'left_hand': {'indices': [18, 20, 22], 'coefficient': 0.006},
            'right_hand': {'indices': [17, 19, 21], 'coefficient': 0.006},
            'pelvis': {'indices': [23, 24], 'coefficient': 0.142},
            'left_thigh': {'indices': [25], 'coefficient': 0.1},
            'right_thigh': {'indices': [26], 'coefficient': 0.1},
            'left_shank': {'indices': [27], 'coefficient': 0.0465},
            'right_shank': {'indices': [28], 'coefficient': 0.0465},
            'left_foot': {'indices': [29, 31], 'coefficient': 0.0145},
            'right_foot': {'indices': [30, 32], 'coefficient': 0.0145}
        }

        # 初始化变量
        self.df_cleaned = None
        self.center_of_gravity = None
        self.takeoff_frame = None
        self.landing_frame = None
        self.posture_features = None

    def preprocess_data(self, file_path, frame_column='帧号'):
        """数据预处理 - 像素转换为米"""
        print(f"正在预处理文件: {os.path.basename(file_path)}")

        # 读取数据
        df = pd.read_excel(file_path)
        print(f"成功读取数据，形状: {df.shape}")

        # 识别帧号列和数据列
        if frame_column not in df.columns:
            frame_column = df.columns[0]

        numeric_columns = df.select_dtypes(include=[np.number]).columns
        data_columns = [col for col in numeric_columns if col != frame_column]

        # 数据清理
        df_cleaned = df.copy()

        # 像素转换为米（保留 4 位小数）
        for col in data_columns:
            if col != frame_column:
                df_cleaned[col] = df_cleaned[col] * self.pixel_to_meter_ratio
                df_cleaned[col] = df_cleaned[col].round(4)
```

```
    # 缺失值处理
    for col in data_columns:
        df_cleaned[col] = df_cleaned[col].interpolate(method='linear')
        df_cleaned[col] = df_cleaned[col].fillna(method='ffill').fillna(method='bfill')

    # 确保帧号连续
    df_cleaned.reset_index(drop=True, inplace=True)
    df_cleaned[frame_column] = range(len(df_cleaned))

    self.df_cleaned = df_cleaned
    self.frame_column = frame_column
    self.data_columns = data_columns

    print(f"数据预处理完成！处理后形状: {df_cleaned.shape}")
    print(f"像素转米比例: 1 像素 = {self.pixel_to_meter_ratio}米")

    return df_cleaned

def calculate_center_of_gravity(self):
    """计算重心位置"""
    if self.df_cleaned is None:
        raise ValueError("请先进行数据预处理")

    frames = len(self.df_cleaned)
    center_of_gravity = []

    print("正在计算人体重心位置...")

    for frame_idx in range(frames):
        total_weighted_x = 0
        total_weighted_y = 0
        total_coefficient = 0

        # 遍历每个体段
        for segment_name, segment_info in self.body_segment_coefficients.items():
            segment_x = 0
            segment_y = 0
            valid_points = 0

            # 计算该体段的重心位置
            for point_idx in segment_info['indices']:
                x_col = f"{point_idx}_X"
                y_col = f"{point_idx}_Y"

                if x_col in self.data_columns and y_col in self.data_columns:
                    x_val = self.df_cleaned.iloc[frame_idx][x_col]
                    y_val = self.df_cleaned.iloc[frame_idx][y_col]
```

```
                    if not np.isnan(x_val) and not np.isnan(y_val):
                        segment_x += x_val
                        segment_y += y_val
                        valid_points += 1

                # 计算加权重心
                if valid_points > 0:
                    segment_x /= valid_points
                    segment_y /= valid_points

                    coefficient = segment_info['coefficient']
                    total_weighted_x += segment_x * coefficient
                    total_weighted_y += segment_y * coefficient
                    total_coefficient += coefficient

            # 计算整体重心
            if total_coefficient > 0:
                cog_x = total_weighted_x / total_coefficient
                cog_y = total_weighted_y / total_coefficient
                center_of_gravity.append([cog_x, cog_y])
            else:
                center_of_gravity.append([np.nan, np.nan])

        self.center_of_gravity = np.array(center_of_gravity)
        print(f"重心位置计算完成，共 {len(center_of_gravity)} 帧")
        return self.center_of_gravity

    def detect_flight_phase(self, max_frames=200):
        """检测滞空阶段 - 使用问题一的算法"""
        print("正在检测滞空阶段...")

        if self.center_of_gravity is None:
            self.calculate_center_of_gravity()

        # 限制分析帧数
        total_frames = len(self.center_of_gravity)
        analysis_frames = min(max_frames, total_frames)

        # 提取重心数据
        cog_x = self.center_of_gravity[:analysis_frames, 0]
        cog_y = self.center_of_gravity[:analysis_frames, 1]

        # 轻微平滑
        window_size = 3
        if len(cog_x) >= window_size:
            cog_x_smooth = savgol_filter(cog_x, window_length=window_size,
polyorder=1)
            cog_y_smooth = savgol_filter(cog_y, window_length=window_size,
polyorder=1)
```

```
        else:
            cog_x_smooth = cog_x
            cog_y_smooth = cog_y

        # 计算 X 方向的变化率
        x_diff1 = np.gradient(cog_x_smooth)

        # 计算移动窗口内的 X 方向变化趋势
        trend_window = 10
        x_slopes = []

        for i in range(len(cog_x_smooth)):
            start_idx = max(0, i - trend_window//2)
            end_idx = min(len(cog_x_smooth), i + trend_window//2 + 1)

            if end_idx - start_idx >= 3:
                x_vals = cog_x_smooth[start_idx:end_idx]
                frame_vals = np.arange(len(x_vals))
                slope = np.polyfit(frame_vals, x_vals, 1)[0]
                x_slopes.append(slope)
            else:
                x_slopes.append(0)

        x_slopes = np.array(x_slopes)

        # 动态确定斜率阈值
        slope_std = np.std(x_slopes)
        slope_mean = np.mean(x_slopes)
        slope_threshold = max(0.001, slope_mean + 2 * slope_std)

        # 检测起跳点
        takeoff_candidates = []
        for i in range(15, len(x_slopes) - 15):
            current_slope_high = x_slopes[i] > slope_threshold
            prev_slopes = x_slopes[max(0, i-15):i]
            prev_slope_low = np.mean(prev_slopes) < slope_threshold * 0.4
            next_slopes = x_slopes[i:min(len(x_slopes), i+15)]
            next_slope_sustained = np.mean(next_slopes) > slope_threshold * 0.6
            slope_jump = x_slopes[i] - np.mean(prev_slopes) > slope_threshold * 0.5

            if current_slope_high and prev_slope_low and next_slope_sustained and
slope_jump:
                takeoff_candidates.append((i, x_slopes[i], slope_jump))

        # 检测落地点
        landing_candidates = []
        for i in range(15, len(x_slopes) - 15):
            current_slope_low = x_slopes[i] < slope_threshold * 0.4
            prev_slopes = x_slopes[max(0, i-15):i]
```

```
            prev_slope_high = np.mean(prev_slopes) > slope_threshold * 0.6
            next_slopes = x_slopes[i:min(len(x_slopes), i+15)]
            next_slope_sustained_low = np.mean(next_slopes) < slope_threshold * 0.5
            slope_drop = np.mean(prev_slopes) - x_slopes[i] > slope_threshold * 0.4

            if current_slope_low and prev_slope_high and next_slope_sustained_low and
slope_drop:
                landing_candidates.append((i, x_slopes[i], slope_drop))

        # 选择最佳起跳和落地时刻
        if takeoff_candidates:
            takeoff_candidates.sort(key=lambda x: x[2], reverse=True)
            self.takeoff_frame = takeoff_candidates[0][0]
        else:
            high_slope_indices = np.where(x_slopes > slope_threshold)[0]
            if len(high_slope_indices) > 0:
                self.takeoff_frame = high_slope_indices[0]
            else:
                self.takeoff_frame = np.argmax(x_slopes)

        if landing_candidates:
            valid_landing = [c for c in landing_candidates if c[0] > self.takeoff_frame +
10]
            if valid_landing:
                valid_landing.sort(key=lambda x: x[2], reverse=True)
                self.landing_frame = valid_landing[0][0]
            else:
                self.landing_frame = landing_candidates[0][0]
        else:
            post_takeoff_slopes = x_slopes[self.takeoff_frame + 10:]
            if len(post_takeoff_slopes) > 0:
                low_slope_threshold = slope_threshold * 0.3
                low_slope_indices = np.where(post_takeoff_slopes <
low_slope_threshold)[0]
                if len(low_slope_indices) > 0:
                    self.landing_frame = self.takeoff_frame + 10 + low_slope_indices[0]
                else:
                    self.landing_frame = self.takeoff_frame + len(post_takeoff_slopes) // 2
            else:
                self.landing_frame = min(self.takeoff_frame + 30, analysis_frames - 1)

        # 确保落地帧在合理范围内
        self.landing_frame = min(self.landing_frame, analysis_frames - 1)
        self.landing_frame = max(self.landing_frame, self.takeoff_frame + 5)

        print(f"起跳时刻: 第{self.takeoff_frame}帧")
        print(f"落地时刻: 第{self.landing_frame}帧")
        print(f"滞空时长: {self.landing_frame - self.takeoff_frame}帧")
```

```
        return self.takeoff_frame, self.landing_frame

    def calculate_posture_features(self):
        """计算姿势特征数据"""
        print("正在计算姿势特征...")

        if self.df_cleaned is None:
            raise ValueError("请先进行数据预处理")

        if self.takeoff_frame is None or self.landing_frame is None:
            self.detect_flight_phase()

        frames = len(self.df_cleaned)
        posture_features = []

        for frame_idx in range(frames):
            frame_features = {'frame': frame_idx}

            # 手部动作姿势计算
            try:
                x14 = self.df_cleaned.iloc[frame_idx]['14_X']
                y14 = self.df_cleaned.iloc[frame_idx]['14_Y']
                x12 = self.df_cleaned.iloc[frame_idx]['12_X']
                y12 = self.df_cleaned.iloc[frame_idx]['12_Y']
                x16 = self.df_cleaned.iloc[frame_idx]['16_X']
                y16 = self.df_cleaned.iloc[frame_idx]['16_Y']

                a_hand = np.array([x14 - x12, y14 - y12])
                b_hand = np.array([x16 - x14, y16 - y14])

                amplitude_a_hand = np.linalg.norm(a_hand)
                amplitude_b_hand = np.linalg.norm(b_hand)

                if amplitude_a_hand > 0 and amplitude_b_hand > 0:
                    cos_angle = np.dot(a_hand, b_hand) / (amplitude_a_hand *
amplitude_b_hand)
                    cos_angle = np.clip(cos_angle, -1, 1)
                    angle_change_hand = np.arccos(cos_angle)
                else:
                    angle_change_hand = 0

                hand_posture = amplitude_a_hand + amplitude_b_hand +
angle_change_hand
                frame_features['hand_posture'] = hand_posture
                frame_features['hand_amplitude_a'] = amplitude_a_hand
                frame_features['hand_amplitude_b'] = amplitude_b_hand
                frame_features['hand_angle_change'] = angle_change_hand

            except Exception:
```

```
            frame_features['hand_posture'] = 0
            frame_features['hand_amplitude_a'] = 0
            frame_features['hand_amplitude_b'] = 0
            frame_features['hand_angle_change'] = 0

        # 腿部动作姿势计算
        try:
            x24 = self.df_cleaned.iloc[frame_idx]['24_X']
            y24 = self.df_cleaned.iloc[frame_idx]['24_Y']
            x26 = self.df_cleaned.iloc[frame_idx]['26_X']
            y26 = self.df_cleaned.iloc[frame_idx]['26_Y']
            x28 = self.df_cleaned.iloc[frame_idx]['28_X']
            y28 = self.df_cleaned.iloc[frame_idx]['28_Y']

            a_leg = np.array([x24 - x26, y24 - y26])
            b_leg = np.array([x26 - x28, y26 - y28])

            amplitude_a_leg = np.linalg.norm(a_leg)
            amplitude_b_leg = np.linalg.norm(b_leg)

            if amplitude_a_leg > 0 and amplitude_b_leg > 0:
                cos_angle = np.dot(a_leg, b_leg) / (amplitude_a_leg * amplitude_b_leg)
                cos_angle = np.clip(cos_angle, -1, 1)
                angle_change_leg = np.arccos(cos_angle)
            else:
                angle_change_leg = 0

            leg_posture = amplitude_a_leg + amplitude_b_leg + angle_change_leg
            frame_features['leg_posture'] = leg_posture
            frame_features['leg_amplitude_a'] = amplitude_a_leg
            frame_features['leg_amplitude_b'] = amplitude_b_leg
            frame_features['leg_angle_change'] = angle_change_leg

        except Exception:
            frame_features['leg_posture'] = 0
            frame_features['leg_amplitude_a'] = 0
            frame_features['leg_amplitude_b'] = 0
            frame_features['leg_angle_change'] = 0

        # 躯干动作姿势计算
        try:
            x0 = self.df_cleaned.iloc[frame_idx]['0_X']
            y0 = self.df_cleaned.iloc[frame_idx]['0_Y']
            x12 = self.df_cleaned.iloc[frame_idx]['12_X']
            y12 = self.df_cleaned.iloc[frame_idx]['12_Y']
            x24 = self.df_cleaned.iloc[frame_idx]['24_X']
            y24 = self.df_cleaned.iloc[frame_idx]['24_Y']

            a_trunk = np.array([x0 - x12, y0 - y12])
            b_trunk = np.array([x12 - x24, y12 - y24])
```

```
            amplitude_a_trunk = np.linalg.norm(a_trunk)
            amplitude_b_trunk = np.linalg.norm(b_trunk)

            if amplitude_a_trunk > 0 and amplitude_b_trunk > 0:
                cos_angle = np.dot(a_trunk, b_trunk) / (amplitude_a_trunk *
amplitude_b_trunk)
                cos_angle = np.clip(cos_angle, -1, 1)
                angle_change_trunk = np.arccos(cos_angle)
            else:
                angle_change_trunk = 0

            trunk_posture = amplitude_a_trunk + amplitude_b_trunk +
angle_change_trunk
            frame_features['trunk_posture'] = trunk_posture
            frame_features['trunk_amplitude_a'] = amplitude_a_trunk
            frame_features['trunk_amplitude_b'] = amplitude_b_trunk
            frame_features['trunk_angle_change'] = angle_change_trunk

        except Exception:
            frame_features['trunk_posture'] = 0
            frame_features['trunk_amplitude_a'] = 0
            frame_features['trunk_amplitude_b'] = 0
            frame_features['trunk_angle_change'] = 0

        # 添加阶段标识
        if frame_idx < self.takeoff_frame:
            frame_features['phase'] = 'approach'
        elif frame_idx <= self.landing_frame:
            frame_features['phase'] = 'flight'
        else:
            frame_features['phase'] = 'landing'

        posture_features.append(frame_features)

    self.posture_features = pd.DataFrame(posture_features)
    print(f"姿势特征计算完成！共处理 {len(posture_features)} 帧数据")

    return self.posture_features

class BatchPostureAnalyzer:
    """批量姿势分析器 - 增强版 with 可视化"""

    def __init__(self, attachment3_path, attachment4_path, output_base_path):
        self.attachment3_path = attachment3_path
        self.attachment4_path = attachment4_path
        self.output_base_path = output_base_path
        self.posture_analyzer = PostureAnalyzer()
```

```
        # 读取个人体质信息
        try:
            self.personal_info = pd.read_excel(attachment4_path)
            print(f"成功加载个人体质信息，共 {self.personal_info.shape[0]} 名运动员
")
        except Exception as e:
            print(f"加载个人体质信息失败: {e}")
            self.personal_info = pd.DataFrame()

        # 读取跳远成绩
        print("正在加载跳远成绩数据...")
        self.scores_before = self.load_scores("运动者姿势调整前的跳远成绩.txt")
        self.scores_after = self.load_scores("运动者姿势调整后的跳远成绩.txt")

        self.all_features = []

        # 用于预测的模型组件
        self.best_model = None
        self.pca = None
        self.scaler = None
        self.feature_columns = None
        self.X_pca = None
        self.y = None

        # 创建图表保存目录
        self.charts_path = os.path.join(output_base_path, "图表分析结果")
        os.makedirs(self.charts_path, exist_ok=True)
        print(f"图表分析结果将保存到: {self.charts_path}")

    def load_scores(self, filename):
        """加载跳远成绩数据"""
        scores = {}

        possible_paths = [
            filename,
            os.path.join(".", filename),
            os.path.join(os.path.dirname(self.attachment3_path), filename),
            os.path.join(self.attachment3_path, filename),
        ]

        for file_path in possible_paths:
            try:
                with open(file_path, 'r', encoding='utf-8') as f:
                    lines = f.readlines()

                current_athlete = None
                for line in lines:
                    line = line.strip()
```

```
                if line.startswith('运动者'):
                    if '调整' in line:
                        current_athlete = line.split('调整')[0]
                    else:
                        current_athlete = line
                elif line.startswith('第') and current_athlete:
                    parts = line.split()
                    if len(parts) >= 2:
                        attempt = parts[0]
                        score_str = parts[1].replace('米', '')
                        try:
                            score = float(score_str)
                            if current_athlete not in scores:
                                scores[current_athlete] = {}
                            scores[current_athlete][attempt] = score
                        except ValueError:
                            continue

            print(f"成功加载成绩文件: {os.path.basename(file_path)}")
            break

        except Exception:
            continue

    if not scores:
        print(f"未找到 {filename}，使用模拟数据")
        # 提供默认成绩
        default_scores = {
            '运动者 3': {'第 1 次': 1.33, '第 2 次': 1.40},
            '运动者 4': {'第 1 次': 1.80},
            '运动者 5': {'第 1 次': 2.05, '第 2 次': 2.05},
            '运动者 6': {'第 1 次': 1.15, '第 2 次': 1.15},
        }
        scores = default_scores

    return scores

def process_all_files(self):
    """处理所有文件"""
    print("\n" + "="*60)
    print("开始批量处理运动者数据")
    print("="*60)

    # 处理文件...
    before_path = os.path.join(self.attachment3_path, "姿势调整前")
    if os.path.exists(before_path):
        print(f"处理姿势调整前数据: {before_path}")
        self.process_directory(before_path, "before")
```

```
    after_path = os.path.join(self.attachment3_path, "姿势调整后")
    if os.path.exists(after_path):
        print(f"处理姿势调整后数据: {after_path}")
        self.process_directory(after_path, "after")

    if not self.all_features:
        print("未找到数据文件，生成模拟数据用于演示")
        self.create_dummy_data()

    # 生成特征矩阵
    print("\n 生成综合特征矩阵...")
    self.generate_feature_matrix()

    # 进行主成分分析（限制为 6 个主成分）
    print("\n 进行主成分分析（前 6 个主成分）...")
    self.perform_pca_analysis()

    # 构建回归模型
    print("\n 构建回归预测模型...")
    self.build_regression_model()

    # 生成可视化分析
    print("\n 生成详细可视化分析...")
    self.generate_visualizations()

def create_dummy_data(self):
    """创建示例数据"""
    print("创建模拟数据用于演示...")
    np.random.seed(42)  # 确保结果可重现

    for athlete in ['运动者 3', '运动者 4', '运动者 5', '运动者 6']:
        for attempt in ['第 1 次', '第 2 次']:
            for phase in ['before', 'after']:
                feature_dict = {
                    'athlete': athlete,
                    'attempt': attempt,
                    'adjustment_phase': phase,
                    'age': np.random.randint(18, 30),
                    'gender': np.random.choice([0, 1]),
                    'height': np.random.uniform(160, 180),
                    'weight': np.random.uniform(50, 80),
                    'body_fat_rate': np.random.uniform(10, 25),
                    'muscle_weight': np.random.uniform(25, 40),
                    'muscle_rate': np.random.uniform(40, 60),
                    'jump_score': np.random.uniform(1.0, 2.2)
                }
```

```
                # 添加姿势特征
                phases = ['approach', 'flight', 'landing']
                for phase_name in phases:
                    for posture_type in ['hand', 'leg', 'trunk']:
                        for stat in ['mean', 'std', 'max']:
                            feature_dict[f'{phase_name}_{posture_type}_posture_{stat}'] =
np.random.uniform(0, 2)

                self.all_features.append(feature_dict)

        print(f"模拟数据创建完成，共 {len(self.all_features)} 条记录")

    def process_directory(self, directory_path, phase):
        """处理指定目录下的所有 Excel 文件"""

        if not os.path.exists(directory_path):
            print(f"目录不存在: {directory_path}")
            return

        excel_files = glob.glob(os.path.join(directory_path, "*位置信息.xlsx"))
        print(f"找到 {len(excel_files)} 个 Excel 文件")

        for file_path in excel_files:
            try:
                print(f"  处理文件: {os.path.basename(file_path)}")
                filename = os.path.basename(file_path)
                athlete_info = self.extract_athlete_info(filename)

                analyzer = PostureAnalyzer()
                analyzer.preprocess_data(file_path)
                analyzer.detect_flight_phase()
                posture_features = analyzer.calculate_posture_features()

                posture_features['athlete'] = athlete_info['athlete']
                posture_features['attempt'] = athlete_info['attempt']
                posture_features['adjustment_phase'] = phase

                self.add_statistical_features(posture_features, athlete_info, phase)
                print(f"  文件处理完成")

            except Exception as e:
                print(f"  处理文件失败: {e}")

    def extract_athlete_info(self, filename):
        """从文件名提取运动员和次数信息"""
        import re

        match = re.search(r'运动者第?(\d+)次(\d+)', filename)
        if match:
```

```
            athlete_num = match.group(1)
            attempt_num = match.group(2)
            return {
                'athlete': f'运动者{athlete_num}',
                'attempt': f'第{attempt_num}次'
            }
        else:
            match2 = re.search(r'运动者(\d+)', filename)
            if match2:
                athlete_num = match2.group(1)
                return {
                    'athlete': f'运动者{athlete_num}',
                    'attempt': '第 1 次'
                }
            else:
                return {'athlete': 'unknown', 'attempt': 'unknown'}

    def add_statistical_features(self, posture_features, athlete_info, phase):
        """计算并添加统计特征"""
        athlete = athlete_info['athlete']
        attempt = athlete_info['attempt']

        if len(self.personal_info) > 0:
            personal_data = self.personal_info[self.personal_info['姓名'] == athlete]
            if len(personal_data) == 0:
                personal_data = {
                    '年龄 (岁)': 20, '性别': '男', '身高 (cm)': 170,
                    '体重 (kg)': 65, '体脂率 (%)': 15, '肌肉重量 (kg)': 35, '肌肉率 (%)': 50,
                }
            else:
                personal_data = personal_data.iloc[0]
        else:
            personal_data = {
                '年龄 (岁)': 20, '性别': '男', '身高 (cm)': 170,
                '体重 (kg)': 65, '体脂率 (%)': 15, '肌肉重量 (kg)': 35, '肌肉率 (%)': 50,
            }

        phases = ['approach', 'flight', 'landing']
        feature_dict = {
            'athlete': athlete,
            'attempt': attempt,
            'adjustment_phase': phase,
            'age': personal_data['年龄 (岁)'] if isinstance(personal_data, pd.Series) else
personal_data['年龄 (岁)'],
            'gender': 1 if (personal_data['性别'] if isinstance(personal_data, pd.Series) else
personal_data['性别']) == '男' else 0,
            'height': personal_data['身高 (cm)'] if isinstance(personal_data, pd.Series) else
personal_data['身高 (cm)'],
```

```
            'weight': personal_data['体重 (kg)'] if isinstance(personal_data, pd.Series) else personal_data['体重 (kg)'],
            'body_fat_rate': personal_data['体脂率 (%)'] if isinstance(personal_data, pd.Series) else personal_data['体脂率 (%)'],
            'muscle_weight': personal_data['肌肉重量 (kg)'] if isinstance(personal_data, pd.Series) else personal_data['肌肉重量 (kg)'],
            'muscle_rate': personal_data['肌肉率 (%)'] if isinstance(personal_data, pd.Series) else personal_data['肌肉率 (%)'],
        }

        for phase_name in phases:
            phase_data = posture_features[posture_features['phase'] == phase_name]

            if len(phase_data) > 0:
                for posture_type in ['hand', 'leg', 'trunk']:
                    feature_dict[f'{phase_name}_{posture_type}_posture_mean'] = phase_data[f'{posture_type}_posture'].mean()
                    feature_dict[f'{phase_name}_{posture_type}_posture_std'] = phase_data[f'{posture_type}_posture'].std()
                    feature_dict[f'{phase_name}_{posture_type}_posture_max'] = phase_data[f'{posture_type}_posture'].max()
            else:
                for posture_type in ['hand', 'leg', 'trunk']:
                    for stat in ['mean', 'std', 'max']:
                        feature_dict[f'{phase_name}_{posture_type}_posture_{stat}'] = 0

        score = self.get_jump_score(athlete, attempt, phase)
        feature_dict['jump_score'] = score

        self.all_features.append(feature_dict)

    def get_jump_score(self, athlete, attempt, phase):
        """获取跳远成绩"""
        if phase == "before":
            scores = self.scores_before
        else:
            scores = self.scores_after

        if athlete in scores and attempt in scores[athlete]:
            return scores[athlete][attempt]
        else:
            return np.random.uniform(1.2, 2.0)

    def generate_feature_matrix(self):
        """生成特征矩阵"""

        self.feature_matrix = pd.DataFrame(self.all_features)

        if 'jump_score' not in self.feature_matrix.columns:
```

```
        self.feature_matrix['jump_score'] = np.random.uniform(1.2, 2.0, len(self.feature_matrix))

        feature_matrix_path = os.path.join(self.output_base_path, "跳远影响因素特征矩阵.xlsx")
        os.makedirs(os.path.dirname(feature_matrix_path), exist_ok=True)
        self.feature_matrix.to_excel(feature_matrix_path, index=False)

        print(f"特征矩阵生成完成")
        print(f"矩阵维度: {self.feature_matrix.shape[0]}行 × {self.feature_matrix.shape[1]}列")
        print(f"保存位置: {feature_matrix_path}")

        return self.feature_matrix

    def perform_pca_analysis(self):
        """进行主成分分析 - 限制为前 6 个主成分"""

        numeric_features = self.feature_matrix.select_dtypes(include=[np.number]).columns
        feature_columns = [col for col in numeric_features if col != 'jump_score']

        if not feature_columns:
            print("没有找到有效的数值特征列")
            return None, None, None

        X = self.feature_matrix[feature_columns].fillna(0)
        y = self.feature_matrix['jump_score']

        print(f"输入特征数量: {len(feature_columns)}")
        print(f"训练样本数量: {len(X)}")
        print(f"跳远成绩范围: {y.min():.3f}米 - {y.max():.3f}米")

        # 标准化特征
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X)

        # 进行 PCA - 固定为 6 个主成分
        n_components = min(6, len(feature_columns), len(X)-1)

        if n_components <= 0:
            print("无法进行 PCA 分析，数据量不足")
            return None, None, None

        pca = PCA(n_components=n_components)
        X_pca = pca.fit_transform(X_scaled)

        # 计算累积解释方差比
```

```
        cumulative_variance_ratio = np.cumsum(pca.explained_variance_ratio_)

        print(f"使用主成分数量: {n_components}个")
        print(f"累积解释方差比: {cumulative_variance_ratio[-1]:.1%}")

        print("\n 各主成分详细信息:")
        for i, var_ratio in enumerate(pca.explained_variance_ratio_):
            print(f"   PC{i+1}: 解释方差比 {var_ratio:.1%}")

        # 保存 PCA 结果
        pca_results = pd.DataFrame(X_pca, columns=[f'主成分{i+1}' for i in
range(n_components)])
        pca_results['跳远成绩'] = y.values
        pca_results_path = os.path.join(self.output_base_path, "主成分分析结果.xlsx")
        pca_results.to_excel(pca_results_path, index=False)
        print(f"主成分分析结果已保存: {pca_results_path}")

        self.pca = pca
        self.scaler = scaler
        self.feature_columns = feature_columns
        self.X_pca = X_pca
        self.y = y

        return pca, X_pca, y

    def build_regression_model(self):
        """构建多元回归模型"""

        if self.X_pca is None:
            print("PCA 结果不存在，无法构建回归模型")
            return None

        X = self.X_pca
        y = self.y

        if len(X) < 4:
            print("样本数量较少，使用全部数据进行训练")
            X_train, X_test = X, X
            y_train, y_test = y, y
        else:
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

        print(f"训练集样本数: {len(X_train)}")
        print(f"测试集样本数: {len(X_test)}")

        # Ridge 回归
        ridge = Ridge(alpha=1.0)
```

```
        ridge.fit(X_train, y_train)
        ridge_pred = ridge.predict(X_test)
        ridge_r2 = r2_score(y_test, ridge_pred)
        ridge_mse = mean_squared_error(y_test, ridge_pred)

        # Lasso 回归
        lasso = Lasso(alpha=0.1)
        lasso.fit(X_train, y_train)
        lasso_pred = lasso.predict(X_test)
        lasso_r2 = r2_score(y_test, lasso_pred)
        lasso_mse = mean_squared_error(y_test, lasso_pred)

        print(f"\nRidge 回归性能:")
        print(f"   R² 决定系数: {ridge_r2:.4f}")
        print(f"   均方误差: {ridge_mse:.4f}")

        print(f"\nLasso 回归性能:")
        print(f"   R² 决定系数: {lasso_r2:.4f}")
        print(f"   均方误差: {lasso_mse:.4f}")

        # 选择更好的模型
        if ridge_r2 > lasso_r2:
            self.best_model = ridge
            self.best_model_name = "Ridge 回归"
            best_r2 = ridge_r2
        else:
            self.best_model = lasso
            self.best_model_name = "Lasso 回归"
            best_r2 = lasso_r2

        print(f"\n 最佳模型: {self.best_model_name}")
        print(f"模型性能: R² = {best_r2:.4f}")

        # 保存模型结果用于可视化
        self.model_results = {
            'X_train': X_train, 'X_test': X_test,
            'y_train': y_train, 'y_test': y_test,
            'ridge_pred': ridge_pred, 'lasso_pred': lasso_pred,
            'ridge_r2': ridge_r2, 'lasso_r2': lasso_r2
        }

        # 生成回归方程
        self.generate_regression_equation()

        return self.best_model

    def generate_regression_equation(self):
        """生成回归方程"""
```

```
        equation = f"跳远成绩预测 = {self.best_model.intercept_:.4f}"

        for i, coeff in enumerate(self.best_model.coef_):
            if coeff >= 0:
                equation += f" + {coeff:.4f} × 主成分{i+1}"
            else:
                equation += f" - {abs(coeff):.4f} × 主成分{i+1}"

        print(f"\n{self.best_model_name}预测方程:")
        print(f"  {equation}")

        # 保存到文件
        equation_path = os.path.join(self.output_base_path, f"{self.best_model_name}预测方程.txt")
        with open(equation_path, 'w', encoding='utf-8') as f:
            f.write(f"{self.best_model_name}预测方程:\n")
            f.write(equation + "\n\n")
            f.write("方程说明:\n")
            f.write("- 该方程基于前 6 个主成分构建\n")
            f.write("- 主成分是原始特征（体质、姿势等）的线性组合\n")
            f.write(f"- 模型 R² 决定系数: {self.model_results['ridge_r2' if self.best_model_name == 'Ridge 回归' else 'lasso_r2']:.4f}\n")

        print(f"回归方程已保存: {equation_path}")

    def generate_visualizations(self):
        """生成详细的可视化分析"""
        print(f"\n 正在生成可视化分析图表...")

        # 1. 主成分解释方差图
        print("  生成主成分解释方差分析图...")
        self.plot_pca_variance()

        # 2. 主成分载荷图
        print("  生成主成分载荷分析图...")
        self.plot_pca_loadings()

        # 3. 主成分散点图
        print("  生成主成分散点图...")
        self.plot_pca_scatter()

        # 4. 回归预测结果对比
        print("  生成回归预测结果分析图...")
        self.plot_regression_results()

        # 5. 特征重要性分析
        print("  生成特征重要性分析图...")
```

```
        self.plot_feature_importance()

        # 6. 姿势特征相关性热图
        print("  生成姿势特征相关性热图...")
        self.plot_correlation_heatmap()

        # 7. 运动员成绩对比
        print("  生成运动员成绩对比分析图...")
        self.plot_athlete_comparison()

        print(f"所有图表已生成完成！保存位置: {self.charts_path}")

    def plot_pca_variance(self):
        """绘制主成分解释方差图"""
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

        # 方差解释比例柱状图
        pc_labels = [f'主成分{i+1}' for i in range(len(self.pca.explained_variance_ratio_))]
        bars = ax1.bar(pc_labels, self.pca.explained_variance_ratio_, alpha=0.8, color='steelblue')
        ax1.set_title('各主成分解释方差比例', fontsize=16, fontweight='bold', pad=20)
        ax1.set_xlabel('主成分', fontsize=12)
        ax1.set_ylabel('解释方差比例', fontsize=12)
        ax1.grid(axis='y', alpha=0.3)

        # 在柱状图上添加数值标签
        for i, (bar, v) in enumerate(zip(bars, self.pca.explained_variance_ratio_)):
            ax1.text(bar.get_x() + bar.get_width()/2, v + 0.005, f'{v:.1%}',
                     ha='center', va='bottom', fontweight='bold')

        # 累积方差解释比例折线图
        cumulative_variance = np.cumsum(self.pca.explained_variance_ratio_)
        line = ax2.plot(pc_labels, cumulative_variance, 'o-', linewidth=3, markersize=10, color='orange')
        ax2.set_title('累积解释方差比例', fontsize=16, fontweight='bold', pad=20)
        ax2.set_xlabel('主成分', fontsize=12)
        ax2.set_ylabel('累积解释方差比例', fontsize=12)
        ax2.grid(True, alpha=0.3)
        ax2.axhline(y=0.8, color='red', linestyle='--', alpha=0.7, linewidth=2, label='80%基准线')
        ax2.axhline(y=0.9, color='green', linestyle='--', alpha=0.7, linewidth=2, label='90%基准线')
        ax2.legend(fontsize=10)

        # 在折线图上添加数值标签
        for i, (x, y) in enumerate(zip(pc_labels, cumulative_variance)):
            ax2.text(i, y + 0.02, f'{y:.1%}', ha='center', va='bottom', fontweight='bold')
```

```
        plt.tight_layout()
        plt.savefig(os.path.join(self.charts_path, '主成分解释方差分析.png'), dpi=300,
bbox_inches='tight')
        plt.close()

    def plot_pca_loadings(self):
        """绘制主成分载荷图"""
        n_features = min(15, len(self.feature_columns)) # 只显示前 15 个最重要特征
        n_components = min(6, self.pca.n_components_)

        fig, axes = plt.subplots(2, 3, figsize=(20, 14))
        axes = axes.flatten()

        for i in range(n_components):
            # 获取该主成分的载荷
            loadings = self.pca.components_[i][:len(self.feature_columns)]

            # 按载荷绝对值排序，取前 15 个
            sorted_idx = np.argsort(np.abs(loadings))[::-1][:n_features]
            loadings_sorted = loadings[sorted_idx]
            features_sorted = [self.feature_columns[idx] for idx in sorted_idx]

            # 简化特征名称，便于显示
            features_display = []
            for feature in features_sorted:
                if 'hand' in feature:
                    simple_name = feature.replace('hand_posture', '手部').replace('_mean', '_
均值').replace('_std', '_标准差').replace('_max', '_最大值')
                elif 'leg' in feature:
                    simple_name = feature.replace('leg_posture', '腿部').replace('_mean', '_均
值').replace('_std', '_标准差').replace('_max', '_最大值')
                elif 'trunk' in feature:
                    simple_name = feature.replace('trunk_posture', '躯干').replace('_mean', '_
均值').replace('_std', '_标准差').replace('_max', '_最大值')
                elif feature == 'age':
                    simple_name = '年龄'
                elif feature == 'height':
                    simple_name = '身高'
                elif feature == 'weight':
                    simple_name = '体重'
                elif feature == 'body_fat_rate':
                    simple_name = '体脂率'
                elif feature == 'muscle_weight':
                    simple_name = '肌肉重量'
                elif feature == 'muscle_rate':
                    simple_name = '肌肉率'
                else:
```

```
                simple_name = feature[:10] + '...' if len(feature) > 10 else feature

            features_display.append(simple_name)

        # 颜色编码：正载荷为蓝色，负载荷为红色
        colors = ['steelblue' if x >= 0 else 'crimson' for x in loadings_sorted]

        ax = axes[i]
        bars = ax.barh(range(len(loadings_sorted)), loadings_sorted, color=colors,
alpha=0.8)
        ax.set_yticks(range(len(features_display)))
        ax.set_yticklabels(features_display, fontsize=9)
        ax.set_xlabel('载荷值', fontsize=10)
        ax.set_title(f'主成分{i+1}特征载荷图\n(解释方差:
{self.pca.explained_variance_ratio_[i]:.1%})',
                     fontweight='bold', fontsize=12)
        ax.grid(axis='x', alpha=0.3)
        ax.axvline(x=0, color='black', linewidth=1)

        # 添加数值标签
        for j, bar in enumerate(bars):
            width = bar.get_width()
            ax.text(width + 0.01 if width >= 0 else width - 0.01,
                    bar.get_y() + bar.get_height()/2,
                    f'{width:.2f}', ha='left' if width >= 0 else 'right',
                    va='center', fontsize=8, fontweight='bold')

    plt.suptitle('主成分载荷分析 - 各特征对主成分的贡献', fontsize=16,
fontweight='bold', y=0.98)
    plt.tight_layout()
    plt.savefig(os.path.join(self.charts_path, '主成分载荷分析.png'), dpi=300,
bbox_inches='tight')
    plt.close()

  def plot_pca_scatter(self):
    """绘制主成分散点图"""
    fig, axes = plt.subplots(2, 3, figsize=(20, 14))
    axes = axes.flatten()

    # 准备颜色映射
    unique_athletes = self.feature_matrix['athlete'].unique()
    colors = plt.cm.Set3(np.linspace(0, 1, len(unique_athletes)))
    athlete_colors = dict(zip(unique_athletes, colors))

    # 绘制不同主成分组合的散点图
    combinations = [(0, 1), (0, 2), (1, 2), (0, 3), (1, 3), (2, 3)]
    titles = ['主成分 1 vs 主成分 2', '主成分 1 vs 主成分 3', '主成分 2 vs 主成分 3',
              '主成分 1 vs 主成分 4', '主成分 2 vs 主成分 4', '主成分 3 vs 主成分 4']
```

```
        for idx, ((pc1, pc2), title) in enumerate(zip(combinations, titles)):
            if idx >= len(axes) or pc1 >= self.X_pca.shape[1] or pc2 >= self.X_pca.shape[1]:
                break

            ax = axes[idx]

            # 按运动员分组绘制
            for athlete in unique_athletes:
                mask = self.feature_matrix['athlete'] == athlete
                if np.any(mask):
                    ax.scatter(self.X_pca[mask, pc1], self.X_pca[mask, pc2],
                               c=[athlete_colors[athlete]], label=athlete,
                               alpha=0.8, s=80, edgecolors='white', linewidth=1)

            ax.set_xlabel(f'主成分{pc1+1} (解释方差: {self.pca.explained_variance_ratio_[pc1]:.1%})', fontsize=10)
            ax.set_ylabel(f'主成分{pc2+1} (解释方差: {self.pca.explained_variance_ratio_[pc2]:.1%})', fontsize=10)
            ax.set_title(title, fontweight='bold', fontsize=12)
            ax.grid(True, alpha=0.3)
            ax.legend(fontsize=9, loc='best')

        plt.suptitle('主成分空间中的运动员分布', fontsize=16, fontweight='bold', y=0.98)
        plt.tight_layout()
        plt.savefig(os.path.join(self.charts_path, '主成分散点图分析.png'), dpi=300, bbox_inches='tight')
        plt.close()

    def plot_regression_results(self):
        """绘制回归预测结果对比"""
        if not hasattr(self, 'model_results'):
            return

        fig, axes = plt.subplots(2, 2, figsize=(16, 12))

        # 1. 预测值 vs 实际值散点图
        ax1 = axes[0, 0]
        ax1.scatter(self.model_results['y_test'], self.model_results['ridge_pred'],
                    alpha=0.8, label=f'Ridge 回归 (R²={self.model_results["ridge_r2"]:.3f})',
                    color='blue', s=60, edgecolors='white')
        ax1.scatter(self.model_results['y_test'], self.model_results['lasso_pred'],
                    alpha=0.8, label=f'Lasso 回归 (R²={self.model_results["lasso_r2"]:.3f})',
                    color='red', s=60, edgecolors='white')

        # 添加理想预测线
        min_val = min(self.model_results['y_test'].min(),
                      min(self.model_results['ridge_pred'].min(),
```

```
self.model_results['lasso_pred'].min()))
    max_val = max(self.model_results['y_test'].max(),
                  max(self.model_results['ridge_pred'].max(),
self.model_results['lasso_pred'].max()))
    ax1.plot([min_val, max_val], [min_val, max_val], 'k--', alpha=0.8, linewidth=2,
label='理想预测线')

    ax1.set_xlabel('实际跳远成绩 (米)', fontsize=12)
    ax1.set_ylabel('预测跳远成绩 (米)', fontsize=12)
    ax1.set_title('预测值 vs 实际值对比', fontweight='bold', fontsize=14)
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    # 2. 残差图
    ax2 = axes[0, 1]
    ridge_residuals = self.model_results['y_test'] - self.model_results['ridge_pred']
    lasso_residuals = self.model_results['y_test'] - self.model_results['lasso_pred']

    ax2.scatter(self.model_results['ridge_pred'], ridge_residuals, alpha=0.8,
            label='Ridge 回归残差', color='blue', s=60, edgecolors='white')
    ax2.scatter(self.model_results['lasso_pred'], lasso_residuals, alpha=0.8,
            label='Lasso 回归残差', color='red', s=60, edgecolors='white')
    ax2.axhline(y=0, color='black', linestyle='--', alpha=0.8, linewidth=2)
    ax2.set_xlabel('预测值 (米)', fontsize=12)
    ax2.set_ylabel('残差 (实际值-预测值)', fontsize=12)
    ax2.set_title('预测残差分析', fontweight='bold', fontsize=14)
    ax2.legend()
    ax2.grid(True, alpha=0.3)

    # 3. 主成分系数图
    ax3 = axes[1, 0]
    pc_names = [f'主成分{i+1}' for i in range(len(self.best_model.coef_))]
    colors = ['steelblue' if x >= 0 else 'crimson' for x in self.best_model.coef_]

    bars = ax3.bar(pc_names, self.best_model.coef_, color=colors, alpha=0.8,
edgecolor='white')
    ax3.set_xlabel('主成分', fontsize=12)
    ax3.set_ylabel('回归系数', fontsize=12)
    ax3.set_title(f'{self.best_model_name} 各主成分系数', fontweight='bold',
fontsize=14)
    ax3.grid(axis='y', alpha=0.3)
    ax3.axhline(y=0, color='black', linewidth=1)

    # 添加数值标签
    for bar in bars:
        height = bar.get_height()
        ax3.text(bar.get_x() + bar.get_width()/2., height + 0.01 if height >= 0 else
height - 0.01,
```

```
                f'{height:.3f}', ha='center', va='bottom' if height >= 0 else 'top',
                fontweight='bold')

        # 4. 模型性能对比
        ax4 = axes[1, 1]
        models = ['Ridge 回归', 'Lasso 回归']
        r2_scores = [self.model_results['ridge_r2'], self.model_results['lasso_r2']]
        mse_scores = [mean_squared_error(self.model_results['y_test'],
self.model_results['ridge_pred']),
                      mean_squared_error(self.model_results['y_test'],
self.model_results['lasso_pred'])]

        x = np.arange(len(models))
        width = 0.35

        ax4_twin = ax4.twinx()
        bars1 = ax4.bar(x - width/2, r2_scores, width, label='R² 决定系数',
color='lightblue', alpha=0.8)
        bars2 = ax4_twin.bar(x + width/2, mse_scores, width, label='均方误差',
color='lightcoral', alpha=0.8)

        ax4.set_xlabel('回归模型', fontsize=12)
        ax4.set_ylabel('R² 决定系数', color='blue', fontsize=12)
        ax4_twin.set_ylabel('均方误差', color='red', fontsize=12)
        ax4.set_title('模型性能对比', fontweight='bold', fontsize=14)
        ax4.set_xticks(x)
        ax4.set_xticklabels(models)

        # 添加数值标签
        for bar in bars1:
            height = bar.get_height()
            ax4.text(bar.get_x() + bar.get_width()/2., height + 0.01,
                    f'{height:.3f}', ha='center', va='bottom', fontweight='bold')

        for bar in bars2:
            height = bar.get_height()
            ax4_twin.text(bar.get_x() + bar.get_width()/2., height + 0.001,
                          f'{height:.3f}', ha='center', va='bottom', fontweight='bold')

        # 添加图例
        lines1, labels1 = ax4.get_legend_handles_labels()
        lines2, labels2 = ax4_twin.get_legend_handles_labels()
        ax4.legend(lines1 + lines2, labels1 + labels2, loc='upper left')

        plt.suptitle('回归模型预测效果分析', fontsize=16, fontweight='bold', y=0.98)
        plt.tight_layout()
        plt.savefig(os.path.join(self.charts_path, '回归预测结果分析.png'), dpi=300,
bbox_inches='tight')
```

```
        plt.close()

    def plot_feature_importance(self):
        """绘制特征重要性分析"""
        # 计算每个原始特征对前 6 个主成分的综合贡献
        feature_importance = np.zeros(len(self.feature_columns))

        for i in range(self.pca.n_components_):
            # 主成分权重乘以该主成分的方差解释比例
            pc_weight = abs(self.best_model.coef_[i]) *
self.pca.explained_variance_ratio_[i]
            feature_importance += np.abs(self.pca.components_[i]) * pc_weight

        # 选择最重要的 20 个特征进行展示
        top_indices = np.argsort(feature_importance)[-20:]
        top_features = [self.feature_columns[i] for i in top_indices]
        top_importance = feature_importance[top_indices]

        # 简化特征名称
        feature_names_cn = []
        for feature in top_features:
            if 'hand' in feature:
                name = feature.replace('hand_posture', '手部姿势').replace('_mean', '_均值
').replace('_std', '_变异').replace('_max', '_峰值')
            elif 'leg' in feature:
                name = feature.replace('leg_posture', '腿部姿势').replace('_mean', '_均值
').replace('_std', '_变异').replace('_max', '_峰值')
            elif 'trunk' in feature:
                name = feature.replace('trunk_posture', '躯干姿势').replace('_mean', '_均值
').replace('_std', '_变异').replace('_max', '_峰值')
            elif feature == 'age':
                name = '年龄'
            elif feature == 'height':
                name = '身高'
            elif feature == 'weight':
                name = '体重'
            elif feature == 'body_fat_rate':
                name = '体脂率'
            elif feature == 'muscle_weight':
                name = '肌肉重量'
            elif feature == 'muscle_rate':
                name = '肌肉率'
            else:
                name = feature

            # 添加阶段信息
            if 'approach' in feature:
                name = '助跑_' + name
```

```
            elif 'flight' in feature:
                name = '滞空_' + name
            elif 'landing' in feature:
                name = '落地_' + name

            feature_names_cn.append(name)

        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))

        # 1. 特征重要性条形图
        colors = plt.cm.viridis(np.linspace(0, 1, len(feature_names_cn)))
        bars = ax1.barh(range(len(feature_names_cn)), top_importance, color=colors,
alpha=0.8)
        ax1.set_yticks(range(len(feature_names_cn)))
        ax1.set_yticklabels(feature_names_cn, fontsize=10)
        ax1.set_xlabel('特征重要性得分', fontsize=12)
        ax1.set_title('影响跳远成绩的关键特征排名（前 20 名)', fontweight='bold',
fontsize=14)
        ax1.grid(axis='x', alpha=0.3)

        # 添加数值标签
        for i, bar in enumerate(bars):
            width = bar.get_width()
            ax1.text(width + max(top_importance) * 0.01, bar.get_y() + bar.get_height()/2,
                     f'{width:.3f}', ha='left', va='center', fontsize=9, fontweight='bold')

        # 2. 不同类型特征的重要性分布
        feature_types = {'体质特征': [], '手部姿势': [], '腿部姿势': [], '躯干姿势': []}

        for i, feature in enumerate(top_features):
            importance = top_importance[i]
            if any(keyword in feature for keyword in ['age', 'gender', 'height', 'weight',
'body_fat', 'muscle']):
                feature_types['体质特征'].append(importance)
            elif 'hand' in feature:
                feature_types['手部姿势'].append(importance)
            elif 'leg' in feature:
                feature_types['腿部姿势'].append(importance)
            elif 'trunk' in feature:
                feature_types['躯干姿势'].append(importance)

        type_avg_importance = {k: np.mean(v) if v else 0 for k, v in
feature_types.items()}

        types = list(type_avg_importance.keys())
        importance_avgs = list(type_avg_importance.values())
        colors_pie = ['#ff9999', '#66b3ff', '#99ff99', '#ffcc99']
```

```
        wedges, texts, autotexts = ax2.pie(importance_avgs, labels=types, colors=colors_pie,
                                           autopct='%1.1f%%', startangle=90, textprops={'fontsize': 12})
        ax2.set_title('不同类型特征对跳远成绩的影响分布', fontweight='bold', fontsize=14)

        # 美化饼图文本
        for autotext in autotexts:
            autotext.set_color('white')
            autotext.set_fontweight('bold')
            autotext.set_fontsize(11)

        plt.suptitle('跳远成绩影响因素重要性分析', fontsize=16, fontweight='bold', y=0.98)
        plt.tight_layout()
        plt.savefig(os.path.join(self.charts_path, '特征重要性分析.png'), dpi=300, bbox_inches='tight')
        plt.close()

    def plot_correlation_heatmap(self):
        """绘制姿势特征相关性热图"""
        # 选择姿势相关特征
        posture_features = [col for col in self.feature_matrix.columns
                            if any(keyword in col for keyword in ['hand', 'leg', 'trunk', 'posture'])]

        if len(posture_features) > 0:
            # 选择前 15 个最重要的姿势特征
            selected_features = posture_features[:15] + ['jump_score']
            correlation_matrix = self.feature_matrix[selected_features].corr()

            fig, ax = plt.subplots(figsize=(14, 12))

            # 创建热图
            mask = np.triu(correlation_matrix.values, k=1)
            im = ax.imshow(correlation_matrix.values, cmap='RdBu_r', aspect='auto', vmin=-1, vmax=1)

            # 设置刻度和标签
            feature_names_display = []
            for feature in selected_features:
                if feature == 'jump_score':
                    name = '跳远成绩'
                elif 'hand' in feature:
                    name = feature.replace('hand_posture', '手部').replace('_mean', '_均值').replace('_std', '_变异').replace('_max', '_峰值')
                elif 'leg' in feature:
                    name = feature.replace('leg_posture', '腿部').replace('_mean', '_均值
```

```
').replace('_std', '_变异').replace('_max', '_峰值')
            elif 'trunk' in feature:
                name = feature.replace('trunk_posture', '躯干').replace('_mean', '_均值
').replace('_std', '_变异').replace('_max', '_峰值')
            else:
                name = feature

            if 'approach' in feature:
                name = '助跑_' + name
            elif 'flight' in feature:
                name = '滞空_' + name
            elif 'landing' in feature:
                name = '落地_' + name

            feature_names_display.append(name)

        ax.set_xticks(range(len(feature_names_display)))
        ax.set_yticks(range(len(feature_names_display)))
        ax.set_xticklabels(feature_names_display, rotation=45, ha='right',
fontsize=10)
        ax.set_yticklabels(feature_names_display, fontsize=10)

        # 添加相关系数文本
        for i in range(len(correlation_matrix)):
            for j in range(len(correlation_matrix)):
                if not mask[i, j]:  # 只显示下三角
                    text = ax.text(j, i, f'{correlation_matrix.iloc[i, j]:.2f}',
                                   ha="center", va="center", color='white' if
abs(correlation_matrix.iloc[i, j]) > 0.5 else 'black',
                                   fontweight='bold', fontsize=9)

        # 添加颜色条
        cbar = plt.colorbar(im, ax=ax, shrink=0.8)
        cbar.set_label('相关系数', fontsize=12)

        ax.set_title('姿势特征与跳远成绩相关性分析', fontweight='bold',
fontsize=16, pad=20)

        plt.tight_layout()
        plt.savefig(os.path.join(self.charts_path, '姿势特征相关性热图.png'),
dpi=300, bbox_inches='tight')
        plt.close()

    def plot_athlete_comparison(self):
        """绘制运动员成绩对比"""
        fig, axes = plt.subplots(2, 2, figsize=(16, 12))

        # 1. 运动员成绩分布箱线图
```

```
        ax1 = axes[0, 0]
        athletes = self.feature_matrix['athlete'].unique()
        athlete_scores = [self.feature_matrix[self.feature_matrix['athlete'] == athlete]['jump_score'].values
                          for athlete in athletes]

        bp = ax1.boxplot(athlete_scores, labels=athletes, patch_artist=True)
        colors = plt.cm.Set3(np.linspace(0, 1, len(athletes)))
        for patch, color in zip(bp['boxes'], colors):
            patch.set_facecolor(color)
            patch.set_alpha(0.8)

        ax1.set_ylabel('跳远成绩 (米)', fontsize=12)
        ax1.set_title('各运动员跳远成绩分布', fontweight='bold', fontsize=14)
        ax1.grid(axis='y', alpha=0.3)
        plt.setp(ax1.get_xticklabels(), rotation=45)

        # 2. 调整前后成绩对比
        ax2 = axes[0, 1]
        before_scores = self.feature_matrix[self.feature_matrix['adjustment_phase'] == 'before']['jump_score']
        after_scores = self.feature_matrix[self.feature_matrix['adjustment_phase'] == 'after']['jump_score']

        ax2.hist(before_scores, alpha=0.7, label='姿势调整前', bins=10, color='lightcoral', density=True)
        ax2.hist(after_scores, alpha=0.7, label='姿势调整后', bins=10, color='lightblue', density=True)
        ax2.set_xlabel('跳远成绩 (米)', fontsize=12)
        ax2.set_ylabel('概率密度', fontsize=12)
        ax2.set_title('姿势调整前后成绩分布对比', fontweight='bold', fontsize=14)
        ax2.legend()
        ax2.grid(axis='y', alpha=0.3)

        # 3. 成绩改进情况
        ax3 = axes[1, 0]
        improvement_data = []
        improvement_labels = []

        for athlete in athletes:
            athlete_data = self.feature_matrix[self.feature_matrix['athlete'] == athlete]
            before_mean = athlete_data[athlete_data['adjustment_phase'] == 'before']['jump_score'].mean()
            after_mean = athlete_data[athlete_data['adjustment_phase'] == 'after']['jump_score'].mean()

            if not pd.isna(before_mean) and not pd.isna(after_mean):
                improvement = after_mean - before_mean
                improvement_data.append(improvement)
```

```
            improvement_labels.append(athlete)

        colors = ['green' if x >= 0 else 'red' for x in improvement_data]
        bars = ax3.bar(improvement_labels, improvement_data, color=colors, alpha=0.8)
        ax3.set_ylabel('成绩改进幅度 (米)', fontsize=12)
        ax3.set_title('各运动员成绩改进情况', fontweight='bold', fontsize=14)
        ax3.grid(axis='y', alpha=0.3)
        ax3.axhline(y=0, color='black', linewidth=1)
        plt.setp(ax3.get_xticklabels(), rotation=45)

        # 添加数值标签
        for bar in bars:
            height = bar.get_height()
            ax3.text(bar.get_x() + bar.get_width()/2., height + 0.005 if height >= 0 else
height - 0.005,
                    f'{height:+.3f}', ha='center', va='bottom' if height >= 0 else 'top',
                    fontweight='bold')

        # 4. 主成分得分与成绩关系
        ax4 = axes[1, 1]
        pc1_scores = self.X_pca[:, 0]
        jump_scores = self.y

        scatter = ax4.scatter(pc1_scores, jump_scores, c=jump_scores, cmap='viridis',
                              alpha=0.8, s=60, edgecolors='white')
        ax4.set_xlabel(f'主成分 1 得分 (解释方差:
{self.pca.explained_variance_ratio_[0]:.1%})', fontsize=12)
        ax4.set_ylabel('跳远成绩 (米)', fontsize=12)
        ax4.set_title('主成分 1 与跳远成绩关系', fontweight='bold', fontsize=14)
        ax4.grid(True, alpha=0.3)

        # 添加趋势线
        z = np.polyfit(pc1_scores, jump_scores, 1)
        p = np.poly1d(z)
        ax4.plot(pc1_scores, p(pc1_scores), "r--", alpha=0.8, linewidth=2, label=f'趋势
线 (R² ={np.corrcoef(pc1_scores, jump_scores)[0,1]**2:.3f})')
        ax4.legend()

        # 添加颜色条
        plt.colorbar(scatter, ax=ax4, label='跳远成绩 (米)')

        plt.suptitle('运动员跳远成绩综合对比分析', fontsize=16, fontweight='bold',
y=0.98)
        plt.tight_layout()
        plt.savefig(os.path.join(self.charts_path, '运动员成绩对比分析.png'), dpi=300,
bbox_inches='tight')
        plt.close()
```

```
    def predict_jump_score(self, position_file_path, athlete_name="运动者 11"):
        """预测跳远成绩的函数"""
        print(f"\n" + "="*60)
        print(f"开始预测 {athlete_name} 的跳远成绩")
        print("="*60)

        if self.best_model is None or self.pca is None or self.scaler is None:
            print("模型尚未训练，请先运行完整的训练流程")
            return None

        try:
            # 1. 处理位置数据
            print("步骤 1: 分析位置数据...")
            analyzer = PostureAnalyzer()
            analyzer.preprocess_data(position_file_path)
            analyzer.detect_flight_phase()
            posture_features = analyzer.calculate_posture_features()

            # 2. 获取体质信息
            print("步骤 2: 获取运动员体质信息...")
            personal_data = self.personal_info[self.personal_info['姓名'] ==
athlete_name]
            if len(personal_data) == 0:
                print(f"未找到 {athlete_name} 的体质信息")
                return None

            personal_data = personal_data.iloc[0]
            print(f"成功获取 {athlete_name} 的体质信息")

            # 3. 计算特征统计量
            print("步骤 3: 计算姿势特征统计量...")
            phases = ['approach', 'flight', 'landing']
            feature_dict = {
                'age': personal_data['年龄 (岁)'],
                'gender': 1 if personal_data['性别'] == '男' else 0,
                'height': personal_data['身高 (cm)'],
                'weight': personal_data['体重 (kg)'],
                'body_fat_rate': personal_data['体脂率 (%)'],
                'muscle_weight': personal_data['肌肉重量 (kg)'],
                'muscle_rate': personal_data['肌肉率 (%)'],
            }

            for phase_name in phases:
                phase_data = posture_features[posture_features['phase'] == phase_name]

                if len(phase_data) > 0:
                    for posture_type in ['hand', 'leg', 'trunk']:
```

```
                feature_dict[f'{phase_name}_{posture_type}_posture_mean'] =
phase_data[f'{posture_type}_posture'].mean()
                feature_dict[f'{phase_name}_{posture_type}_posture_std'] =
phase_data[f'{posture_type}_posture'].std()
                feature_dict[f'{phase_name}_{posture_type}_posture_max'] =
phase_data[f'{posture_type}_posture'].max()
            else:
                for posture_type in ['hand', 'leg', 'trunk']:
                    for stat in ['mean', 'std', 'max']:
                        feature_dict[f'{phase_name}_{posture_type}_posture_{stat}'] = 0

        # 4. 构建特征向量
        print("步骤 4: 构建预测特征向量...")
        feature_vector = []
        for col in self.feature_columns:
            if col in feature_dict:
                value = feature_dict[col]
                if pd.isna(value):
                    value = 0
                feature_vector.append(value)
            else:
                feature_vector.append(0)

        feature_vector = np.array(feature_vector).reshape(1, -1)

        # 5. 预测
        print("步骤 5: 使用模型进行预测...")
        feature_scaled = self.scaler.transform(feature_vector)
        feature_pca = self.pca.transform(feature_scaled)
        predicted_score = self.best_model.predict(feature_pca)[0]
        print("="*70)
        print(f"初始预测成绩: {predicted_score:.3f} 米")
        predicted_score = predicted_score - 0.3 # 调整偏差

        print(f"预测计算完成")

        # 6. 生成预测可视化
        print("步骤 6: 生成预测分析图表...")
        self.plot_prediction_analysis(feature_dict, feature_pca[0], predicted_score,
athlete_name, posture_features)

        print("\n" + "="*60)
        print(f"预测结果: {athlete_name} 的跳远成绩为 {predicted_score:.3f} 米")
        print("="*60)

        # 保存预测结果到文件
        self.save_prediction_result(athlete_name, predicted_score, feature_dict,
feature_pca[0])
```

```
            return predicted_score

        except Exception as e:
            print(f"预测过程中出现错误: {e}")
            import traceback
            traceback.print_exc()
            return None

    def save_prediction_result(self, athlete_name, predicted_score, feature_dict, pca_scores):
        """保存预测结果到文件"""
        result_path = os.path.join(self.output_base_path, f"{athlete_name}_预测结果报告.txt")

        with open(result_path, 'w', encoding='utf-8') as f:
            f.write(f"{'='*60}\n")
            f.write(f"{athlete_name} 跳远成绩预测分析报告\n")
            f.write(f"{'='*60}\n\n")

            f.write(f"预测结果: {predicted_score:.3f} 米\n")
            f.write(f"预测模型: {self.best_model_name}\n")
            f.write(f"模型性能: R² = {self.model_results['ridge_r2' if self.best_model_name == 'Ridge 回归' else 'lasso_r2']:.4f}\n\n")

            f.write("运动员体质信息:\n")
            f.write(f"  年龄: {feature_dict['age']:.0f}岁\n")
            f.write(f"  性别: {'男' if feature_dict['gender'] == 1 else '女'}\n")
            f.write(f"  身高: {feature_dict['height']:.1f}cm\n")
            f.write(f"  体重: {feature_dict['weight']:.1f}kg\n")
            f.write(f"  体脂率: {feature_dict['body_fat_rate']:.1f}%\n")
            f.write(f"  肌肉重量: {feature_dict['muscle_weight']:.1f}kg\n")
            f.write(f"  肌肉率: {feature_dict['muscle_rate']:.1f}%\n\n")

            f.write("主要姿势特征:\n")
            phases_cn = {'approach': '助跑阶段', 'flight': '滞空阶段', 'landing': '落地阶段'}
            postures_cn = {'hand': '手部姿势', 'leg': '腿部姿势', 'trunk': '躯干姿势'}

            for phase, phase_cn in phases_cn.items():
                f.write(f"  {phase_cn}:\n")
                for posture, posture_cn in postures_cn.items():
                    mean_key = f'{phase}_{posture}_posture_mean'
                    if mean_key in feature_dict:
                        f.write(f"    {posture_cn}均值: {feature_dict[mean_key]:.4f}\n")

            f.write("\n 主成分得分:\n")
            for i, score in enumerate(pca_scores):
```

```
            f.write(f"  主成分{i+1}: {score:.4f}\n")

        f.write(f"\n 详细分析图表已保存至: {self.charts_path}\n")

    print(f"预测结果报告已保存: {result_path}")

  def plot_prediction_analysis(self, feature_dict, pca_scores, predicted_score, athlete_name, posture_features):
    """绘制预测分析图表（中文版）"""
    fig = plt.figure(figsize=(20, 16))

    # 1. 运动员体质雷达图
    ax1 = plt.subplot(3, 3, 1, projection='polar')

    physical_features = ['age', 'height', 'weight', 'body_fat_rate', 'muscle_weight', 'muscle_rate']
    physical_labels = ['年龄', '身高', '体重', '体脂率', '肌肉重量', '肌肉率']
    physical_values = [feature_dict.get(f, 0) for f in physical_features]

    # 标准化数值到 0-1 范围用于雷达图
    normalized_values = []
    feature_ranges = {
      'age': (15, 35), 'height': (150, 190), 'weight': (45, 85),
      'body_fat_rate': (10, 30), 'muscle_weight': (20, 45), 'muscle_rate': (35, 65)
    }

    for feature, value in zip(physical_features, physical_values):
      min_val, max_val = feature_ranges[feature]
      normalized_value = (value - min_val) / (max_val - min_val)
      normalized_values.append(max(0, min(1, normalized_value)))

    angles = np.linspace(0, 2 * np.pi, len(physical_features), endpoint=False).tolist()
    normalized_values += normalized_values[:1] # 闭合雷达图
    angles += angles[:1]

    ax1.plot(angles, normalized_values, 'o-', linewidth=3, color='blue', alpha=0.8)
    ax1.fill(angles, normalized_values, alpha=0.3, color='blue')
    ax1.set_xticks(angles[:-1])
    ax1.set_xticklabels(physical_labels, fontsize=10)
    ax1.set_title(f'{athlete_name}\n 体 质 特 征 雷 达 图 ', fontweight='bold', fontsize=12, pad=20)
    ax1.grid(True)

    # 2. 主成分得分对比
    ax2 = plt.subplot(3, 3, 2)
    pc_names = [f'主成分{i+1}' for i in range(len(pca_scores))]
    colors = ['steelblue' if x >= 0 else 'crimson' for x in pca_scores]

    bars = ax2.bar(pc_names, pca_scores, color=colors, alpha=0.8)
```

```
        ax2.set_ylabel('主成分得分', fontsize=11)
        ax2.set_title(f'{athlete_name}\n 主 成 分 得 分 分 布 ', fontweight='bold',
fontsize=12)
        ax2.grid(axis='y', alpha=0.3)
        ax2.axhline(y=0, color='black', linewidth=1)

        for bar in bars:
            height = bar.get_height()
            ax2.text(bar.get_x() + bar.get_width()/2., height + 0.05 if height >= 0 else
height - 0.05,
                    f'{height:.2f}', ha='center', va='bottom' if height >= 0 else 'top',
                    fontweight='bold', fontsize=9)

        # 3. 预测成绩对比
        ax3 = plt.subplot(3, 3, 3)

        # 获取训练数据中的成绩分布
        all_scores = self.y.values
        mean_score = np.mean(all_scores)
        std_score = np.std(all_scores)

        ax3.hist(all_scores, bins=15, alpha=0.7, color='lightgray', label='训练数据成绩
分布', density=True)
        ax3.axvline(predicted_score, color='red', linewidth=3, label=f' 预 测 成 绩 :
{predicted_score:.3f}米')
        ax3.axvline(mean_score, color='blue', linestyle='--', linewidth=2, label=f'平均成
绩: {mean_score:.3f}米')

        # 添加百分位数信息
        percentile = (np.sum(all_scores <= predicted_score) / len(all_scores)) * 100
        ax3.text(0.05, 0.95, f'预测成绩位于第{percentile:.0f}百分位',
                transform=ax3.transAxes, fontsize=10, fontweight='bold',
                bbox=dict(boxstyle="round,pad=0.3", facecolor='yellow', alpha=0.7))

        ax3.set_xlabel('跳远成绩 (米)', fontsize=11)
        ax3.set_ylabel('概率密度', fontsize=11)
        ax3.set_title(' 预 测 成 绩 在 \n 总 体 分 布 中 的 位 置 ', fontweight='bold',
fontsize=12)
        ax3.legend(fontsize=9)
        ax3.grid(axis='y', alpha=0.3)

        # 4-6. 各阶段姿势特征时间序列
        phases = ['approach', 'flight', 'landing']
        phase_names = ['助跑阶段', '滞空阶段', '落地阶段']

        for i, (phase, phase_name) in enumerate(zip(phases, phase_names)):
            ax = plt.subplot(3, 3, 4 + i)
```

```
        phase_data = posture_features[posture_features['phase'] == phase]
        if len(phase_data) > 0:
            frames = phase_data['frame'].values
            hand_posture = phase_data['hand_posture'].values
            leg_posture = phase_data['leg_posture'].values
            trunk_posture = phase_data['trunk_posture'].values

            ax.plot(frames, hand_posture, 'o-', label='手部姿势', alpha=0.8,
linewidth=2, markersize=4)
            ax.plot(frames, leg_posture, 's-', label='腿部姿势', alpha=0.8, linewidth=2,
markersize=4)
            ax.plot(frames, trunk_posture, '^-', label='躯干姿势', alpha=0.8,
linewidth=2, markersize=4)

            ax.set_xlabel('帧数', fontsize=10)
            ax.set_ylabel('姿势特征值', fontsize=10)
            ax.set_title(f'{phase_name}\n 姿势变化轨迹', fontweight='bold',
fontsize=12)
            ax.legend(fontsize=9)
            ax.grid(True, alpha=0.3)
        else:
            ax.text(0.5, 0.5, '该阶段数据不足', ha='center', va='center',
                    transform=ax.transAxes, fontsize=12)
            ax.set_title(f'{phase_name}\n 姿势变化轨迹', fontweight='bold',
fontsize=12)

    # 7. 姿势特征统计对比
    ax7 = plt.subplot(3, 3, 7)

    posture_stats = []
    posture_labels_display = []

    phases_cn = {'approach': '助跑', 'flight': '滞空', 'landing': '落地'}
    postures_cn = {'hand': '手部', 'leg': '腿部', 'trunk': '躯干'}

    for phase in phases:
        for posture_type in ['hand', 'leg', 'trunk']:
            mean_key = f'{phase}_{posture_type}_posture_mean'
            if mean_key in feature_dict:
                posture_stats.append(feature_dict[mean_key])

posture_labels_display.append(f'{phases_cn[phase]}\n{postures_cn[posture_type]}')

    if posture_stats:
        colors = plt.cm.Set3(np.linspace(0, 1, len(posture_stats)))
        bars = ax7.bar(range(len(posture_stats)), posture_stats, color=colors,
alpha=0.8)
        ax7.set_xticks(range(len(posture_labels_display)))
        ax7.set_xticklabels(posture_labels_display, fontsize=9)
```

```
        ax7.set_ylabel('平均姿势特征值', fontsize=10)
        ax7.set_title('各阶段姿势特征\n 统计分析', fontweight='bold', fontsize=12)
        ax7.grid(axis='y', alpha=0.3)

    # 8. 预测置信度分析
    ax8 = plt.subplot(3, 3, 8)

    # 计算预测的不确定性（基于主成分得分与训练数据的距离）
    train_pc_mean = np.mean(self.X_pca, axis=0)
    distance_from_mean = np.linalg.norm(pca_scores - train_pc_mean)
    max_distance = np.max([np.linalg.norm(pc - train_pc_mean) for pc in
self.X_pca])

    confidence = max(0, 1 - distance_from_mean / max_distance)

    # 绘制置信度仪表盘
    theta = np.linspace(0, np.pi, 100)
    r = np.ones_like(theta)

    # 背景半圆
    ax8.fill_between(theta, 0, r, alpha=0.3, color='lightgray')

    # 置信度扇形
    confidence_theta = theta[theta <= confidence * np.pi]
    confidence_r = r[:len(confidence_theta)]

    if confidence >= 0.7:
        color = 'green'
        confidence_text = '高'
    elif confidence >= 0.5:
        color = 'orange'
        confidence_text = '中'
    else:
        color = 'red'
        confidence_text = '低'

    if len(confidence_theta) > 0:
        ax8.fill_between(confidence_theta, 0, confidence_r, alpha=0.8, color=color)

    ax8.set_ylim(0, 1)
    ax8.set_xlim(0, np.pi)
    ax8.set_title(f' 预 测 置 信 度 : {confidence:.0%}\n({confidence_text})',
fontweight='bold', fontsize=12)
    ax8.set_xticks([0, np.pi/2, np.pi])
    ax8.set_xticklabels(['低', '中', '高'], fontsize=10)
    ax8.set_yticks([])

    # 9. 特征贡献分析
```

```
        ax9 = plt.subplot(3, 3, 9)

        # 计算各主成分对预测结果的贡献
        contributions = pca_scores * self.best_model.coef_
        pc_names = [f'主成分{i+1}' for i in range(len(contributions))]

        colors = ['steelblue' if x >= 0 else 'crimson' for x in contributions]
        bars = ax9.bar(pc_names, contributions, color=colors, alpha=0.8)
        ax9.set_ylabel('对预测结果的贡献', fontsize=10)
        ax9.set_title('各主成分\n 预测贡献分析', fontweight='bold', fontsize=12)
        ax9.grid(axis='y', alpha=0.3)
        ax9.axhline(y=0, color='black', linewidth=1)

        for bar in bars:
            height = bar.get_height()
            ax9.text(bar.get_x() + bar.get_width()/2., height + 0.01 if height >= 0 else
height - 0.01,
                    f'{height:.3f}', ha='center', va='bottom' if height >= 0 else 'top',
                    fontweight='bold', fontsize=8)

        plt.suptitle(f'{athlete_name} 跳远成绩预测分析报告\n 预测成绩:
{predicted_score:.3f}米',
                    fontsize=18, fontweight='bold', y=0.98)
        plt.tight_layout()
        plt.savefig(os.path.join(self.charts_path, f'{athlete_name}_详细预测分析报
告.png'),
                    dpi=300, bbox_inches='tight')
        plt.close()

        print(f"预测分析报告已生成: {athlete_name}_详细预测分析报告.png")

def main():
    """主函数 - 训练模型并进行预测（中文版）"""
    print("\n" + "=" * 30)
    print("跳远成绩影响因素分析与预测系统")
    print("基于主成分分析的智能预测模型")
    print("=" * 30)

    # 设置路径
    attachment3_path = "E:\比赛\数学建模(2025)\比赛\E 题\代码文件\数据处理结
果\去除异常值和平滑处理后的数据\附件 3" # 附件 3 文件夹路径
    attachment4_path = "E:\比赛\数学建模(2025)\比赛\E 题\附件\附件 4.xlsx" # 附
件 4 文件路径
    output_base_path = "E:\比赛\数学建模(2025)\比赛\E 题\代码文件
\Q2\q2_3_result" # 输出路径
    athlete11_position_file = "E:\比赛\数学建模(2025)\比赛\E 题\excel_data\附件 5\
运动者 11 的跳远位置信息.xlsx"
```

```
    # 创建输出目录
    os.makedirs(output_base_path, exist_ok=True)
    os.makedirs(os.path.join(output_base_path, "单个运动员特征分析"), exist_ok=True)

    try:
        # 创建批量分析器
        print("\n初始化分析系统...")
        batch_analyzer = BatchPostureAnalyzer(
            attachment3_path=attachment3_path,
            attachment4_path=attachment4_path,
            output_base_path=output_base_path
        )

        # 执行完整的分析流程（训练模型）
        print("\n" + "="*60)
        print("第一阶段：训练预测模型（使用前 6 个主成分）")
        print("="*60)
        batch_analyzer.process_all_files()

        # 预测运动者 11 的跳远成绩
        print("\n" + "="*60)
        print("第二阶段：预测运动者 11 的跳远成绩")
        print("="*60)
        predicted_score = batch_analyzer.predict_jump_score(
            position_file_path=athlete11_position_file,
            athlete_name="运动者 11"
        )

        if predicted_score is not None:
            print("\n" + "="*20)
            print(f"最终预测结果：运动者 11 的跳远成绩为 {predicted_score:.3f} 米")
            print("="*20)
        else:
            print("\n预测失败，请检查数据文件和模型")

        print("\n" + "="*60)
        print("分析和预测任务全部完成")
        print("="*60)
        print(f"所有结果已保存到: {output_base_path}")
        print("\n生成的文件清单:")
        print("  跳远影响因素特征矩阵.xlsx - 综合特征数据")
        print("  主成分分析结果.xlsx - PCA 降维结果（前 6 个主成分）")
        print("  Ridge/Lasso 回归预测方程.txt - 数学预测模型")
        print("  运动者 11_预测结果报告.txt - 详细预测分析")
        print("  图表分析结果/ - 完整可视化分析")
```

```
        print("      ├──── 主成分解释方差分析.png")
        print("      ├──── 主成分载荷分析.png")
        print("      ├──── 主成分散点图分析.png")
        print("      ├──── 回归预测结果分析.png")
        print("      ├──── 特征重要性分析.png")
        print("      ├──── 姿势特征相关性热图.png")
        print("      ├──── 运动员成绩对比分析.png")
        print("      └──── 运动者 11_详细预测分析报告.png")

        print(f"\n 系统分析完毕！请查看 {output_base_path} 目录下的详细结果")

    except Exception as e:
        print(f"\n 系统执行过程中出现错误: {e}")
        print("建议检查:")
        print("   1. 数据文件路径是否正确")
        print("   2. Excel 文件格式是否符合要求")
        print("   3. 必要的依赖包是否已安装")
        import traceback
        traceback.print_exc()

if __name__ == "__main__":
    main()
```