电 子 科 技 大 学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

模式识别第二次作业

实 验 报 告

专　　业	控制工程
学　　号	**202422280516**
姓　　名	陈劭杰
承担内容	报告撰写

专　　业	控制工程
学　　号	**202422280540**
姓　　名	郭　昊
承担内容	代码编写、报告修改

目 录

第一章 实验内容

1.1 作业内容

①采用 BP 神经网络设计男女生分类器。采用的特征包括身高、体重、鞋码、50m 成绩共 4 个特征，BP 神经网络包含一个隐层，隐层结点数为 4。要求：自行编写代码完成后向传播算法，采用交叉验证的方式实现对于性能指标的评判（包含 SE,SP,ACC 和 AUC，AUC 的计算可以基于平台的软件包）。

②采用 SVM 设计男女生分类器。采用的特征包含身高、体重、鞋码、50m 成绩共 4 个特征。要求：采用平台提供的软件包进行分类器的设计以及测试，尝试不同的核函数设计分类器，采用交叉验证的方式实现对于性能指标的评判（包含 SE,SP,ACC 和 AUC，AUC 的计算基于平台的软件包）。

③对比分析二种方法的分类结果。

1.2 实验目的

①学习 BP 神经网络的原理、结构；

②学习 SVM 分类器的设计（包含采用不同的三种核函数）；

③学习交叉验证的应用和各项性能指标的具体含义。

1.3 实验原理

1.3.1 BP 神经网络

Ⅰ.原理介绍

生物网络和人工网络是神经网络的两大基本组成单元。生物神经网络一般指帮助生物进行思考和行动，用于产生生物意识的大脑神经元，细胞，突触等组成的网络。人工神经网络是一种通过模仿动物的行为活动特征，进行分布式并行信息处理的数学算法模型，由大量处理单元互联组成。

BP 神经网络（Back Propagation Neural Network, 反向传播神经网络）是一种多层前馈神经网络，使用反向传播算法进行权重和阈值的调整，是人工神经网络中最常用的一种。它广泛应用于模式识别、分类、预测和数据分析等领域。BP 神经网络的核心思想是通过不断调整网络参数，使得网络的输出更接近于期望的目标输出。

Ⅱ.BP 算法的具体步骤

① **输入样例**：从训练样例集中取一样例，把输入信息输入到网络中；

② **前向传播**：由网络分别计算各层节点的输出；

③ **误差计算**：计算网络的实际输出与期望输出的误差。

④ **反向传播**：从输出层反向计算到第一个隐层，按一定原则向减小误差方向调整网络的各个连接权值，即反向传播。

⑤ **重复训练**：对训练样例集中的每一个样例重复以上步骤，直到对整个训练样例集的误差达到要求时为止；

⑥ **求解应用问题**：在训练时，需要反向传播，而一旦训练结束，求解实阿问题时，则只需正向传播。

Ⅲ.具体算法

①参数介绍

O_i：节点 i 的输出；

$\mathbf{net}_j$：节点 j 的输入；

ω_{ij}：从节点 i 到节点 j 的连接权值。

$$\mathrm{net}_j = \sum_i \omega_{ij} O_i \tag{1-1}$$

②误差函数

$$e = \frac{1}{2}\sum_k (\hat{y}_k - y_k)^2 \tag{1-2}$$

分别表示输出层上第 k 个节点的期望输出与实际输出 $\hat{y}$， y

③连接权值的修正

$$\omega_{\mathrm{jk}}(t+1) = \omega_{\mathrm{jk}}(t) + \Delta\omega_{\mathrm{jk}} \tag{1-3}$$

$\omega_{\mathrm{jk}}(t+1)$和$\omega_{\mathrm{jk}}(t)$分别表示 t+1 和 t 时刻上从节点 j 到节点 k 的连接权值，$\Delta\omega_{\mathrm{jk}}$为修正量。

为了使连接权值沿着e的梯度变化方向得以改善网络逐渐收敛，取

$$\Delta\omega_{jk} = -\eta\frac{\partial e}{\partial \omega_{jk}} \tag{1-4}$$

其中，η为增益因子。

$$\frac{\partial e}{\partial \omega_{jk}} = \frac{\partial e}{\partial net_k}\cdot\frac{\partial net_k}{\partial \omega_{jk}} \tag{1-5}$$

$$\because net_k = \sum_j \omega_{jk} O_j \rightarrow \frac{\partial net_k}{\partial \omega_{jk}} = \frac{\partial}{\partial \omega_{jk}} \sum_j \omega_{jk} O_j = O_j \tag{1-6}$$

令$\delta_k = \frac{\partial e}{\partial net_k}$

$$\therefore \Delta\omega_{jk} = -\eta \frac{\partial e}{\partial \omega_{jk}} = -\eta \cdot \delta_k \cdot O_j \tag{1-7}$$

当节点 k 是输出层上的节点时：

$$\delta_k = -(\hat{y}_k - y_k) \cdot f'(net_k) \tag{1-8}$$

当节点 k 不是输出层上的节点时：

$$\delta_k = f'(net_k) \cdot \sum_m \delta_m w_{km} \tag{1-9}$$

对于输出层节点：

$$f'(net_k) = f(net_k)[1 - f(net_k)] = y_k(1 - y_k) \tag{1-10}$$

对于非输出层节点：

$$f'(net_k) = O_k(1 - O_k) \tag{1-11}$$

1.3.2 支持向量机（SVM）

给定训练样本集$D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}, y_i \in \{-1, +1\}$，分类学习最基本的想法就是基于训练集 D 在样本空间中找到一个划分超平面，将不同类别的样本分开。但能将训练样本分开的划分超平面可能有很多，如何寻找划分超平面是个需要思考的问题。

线性分类器的分类性能毕竟有限，而对于非线性问题一味放宽约束只能导致大量样本的错分。

这时可以通过非线性变换将其转化为某个高维空间中的线性问题，在变换空间求得最佳分类超平面。

对不同的核函数，对应不同的 SVM，常用的几种有：

Ⅰ.线性 SVM

$$\mathrm{K}(\mathrm{x_i}, \mathrm{x_j}) = \mathrm{x_i}, \mathrm{x_j} = \mathrm{x_i^T x_j} \tag{1-12}$$

Ⅱ.多项式 SVM

$$\mathrm{K}(\mathrm{x_i}, \mathrm{x_j}) = (\mathrm{x_i^T x_j} + 1)^{\mathrm{d}} \quad (\mathrm{d}\text{ 为多项式的阶数}) \tag{1-13}$$

Ⅲ.高斯核函数 SVM

$$K(x_i, x_j) = e^{-\|x_i - x_j\|^2/2\sigma^2} \quad (\sigma^2\text{为方差}) \tag{1-14}$$

1.3.3 性能指标

对于二分类问题，可以将样例根据其真实类别与分类器预测类别的组合划分为以下四种情形：

真正例(true positive)：预测为正，真实也为正；

假正例(false positive)：预测为正，但真实为反；

真反例(true negative)：预测为反，真实也为反；

假反例(true negative)：预测为反，但真实为正。

令 TP、FP、TN、FN 分别表示其对应的样例数，分类结果的 “混淆矩阵”(confusion matrix)如下所示：

表 1-1：混淆矩阵

真实情况	预测结果	
	正例	反例
正例	TP（真正例）	FN（假正例）
反例	FP（假正例）	TN（真反例）

常用性能指标如下：

表 1-2：常用性能指标表

指标类型	计算公式
准确率 (Accuracy)	$Accurancy = \frac{\text{SUM(正确预测的样本总数)}}{\text{SUM(样本总数)}} = \frac{(TP+TN)}{T+F} = \frac{(TP+TN)}{(P+N)}$
精确率 (Precision)	$Precision = \frac{\text{SUM(正确预测的正样本总数)}}{\text{SUM(预测的正样本总数)}} = \frac{TP}{P_{预测}} = \frac{TP}{TP+FP}$
灵敏性/召回率 (Sensitivity)	$Sensitivity = \frac{\text{SUM(正确预测的正样本总数)}}{\text{SUM(实际的正样本总数)}} = \frac{TP}{T} = \frac{TP}{TP+FN}$
特异性 (Specificity)	$Specificity = \frac{\text{SUM(正确预测的负样本总数)}}{\text{SUM(实际的负样本总数)}} = \frac{TN}{F} = \frac{TN}{TN+FP}$

ROC（Receiver Operating Characteristic）曲线是一种用于评估分类模型性能的图形工具。它通常用于二元分类问题，其中有两个类别：正类别和负类别。

ROC 曲线以不同的阈值设置下，绘制了 True Positive Rate（真正例率，也称为灵敏度，TPR）和 False Positive Rate（假正例率，FPR）之间的关系。

以 FPR 为横坐标、TPR 为纵坐标的曲线，它展示了不同阈值下模型的性能。理想情况下，ROC 曲线越接近左上角（0, 1），模型性能越好，因为它意味着更高的 TPR（灵敏度）和更低的 FPR。

ROC 曲线下的面积，即 AUC（Area Under the Curve），也是一个用于度量模型性能的指标。AUC 的取值范围在 0 到 1 之间，通常用于比较不同模型的性能。AUC 值越接近 1，表示模型性能越好。

第二章 数据预处理

2.1 实验语言

本次实验主要采用 C++语言进行，而 C++处理 xlsx 文件较为不便，因此使用 Python 将原数据集转为 csv 文件做进一步处理。

具体实验语言环境为：数据集预处理：python 3.8.5；BP 训练验证：C++ 20；BP 绘图：python 3.8.5；SVM：python 3.8.5。

2.2 数据过滤

实验过程中，主要使用身高、体重、鞋码、50m 成绩共 4 个特征。样本数据可能存在偏差或者错误导致不可信，因此首先要对数据进行筛选。本次实验中，性别设置为男 1 女 0，但是样本集中可能存在非 0/1 数据，需要进行数据筛选。此外，身高体重数据比例与正常值相差过大同样需要进行筛选，将异常的样本作为干扰项剔除。

性别 男1女0	身高(cm)	体重(kg)	鞋码	50米成绩
1	175	95	43	
1	177	68	42	
0.5	169	60	39	9.3
1	200	200	38	10
0	160	52	38	
0	158	57	38	×
1 1	170	60	41	7.7

图 2-1：部分无效数据展示（invalid_data.csv）

2.3 数据扩充

原样本数据集中，男女生占比严重不均匀，始终不能得到一个令人满意的训练结果。后经问题排查，利用 Python 对样本集中女生的数据进行复制扩充，使得男女生的性别比例保持在大约 1:1，再进行训练，此时会得到一个满意的结果。

第三章 BP 神经网络

3.1 BP 神经网络架构

3.1.1 网络结构

在本次实验中，采用了 BP 神经网络结构，其结构如图 2-2 所示，该神经网络包含一个节点数为 4 的输入层，一个节点数为 4 的隐藏层和一个节点数为 1 的输出层，输出时判断输出层节点的概率大小来判断预测结果是女生还是男生。

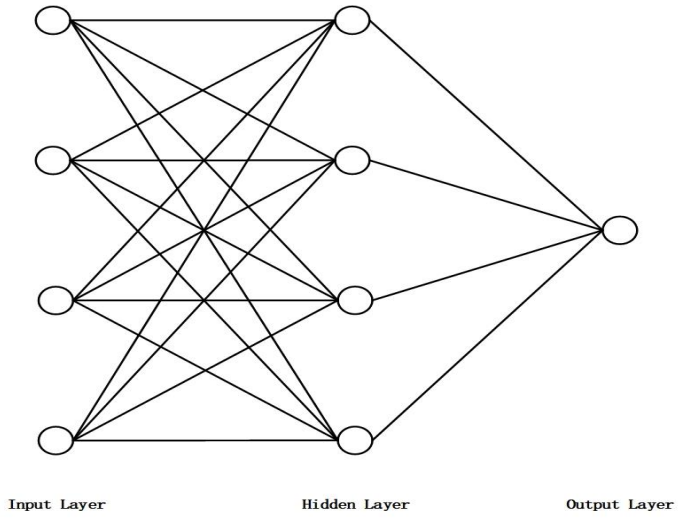


图 3-1：BP 神经网络示意图

3.1.2 参数设定

在初始化部分，设定 BP 神经网络的参数配置，输入层节点数为 4，隐藏层节点数为 4，输出层节点为 1，设定学习率为 0.001，训练轮次为 2000 轮。BP 神经网络采用交叉验证对数据集进行划分训练，并完成对性能指标的评判。在训练开始之前将样本集进行划分，划分为 30%的测试集和 70%的训练集。

网络中，采用 0-1 的均值分布赋初值会导致训练结果始终不能有效分离，修正后采用正态分布对权值进行赋初值。同时加入-1 到 1 的随机数作为偏置。

3.2 模型性能分析

对 BP 神经网络模型经过多次修正，将先后 200 多个不同的模型分别进行训练，记录每一次性能指标变化绘制曲线图，以此反映模型的鲁棒性。得到 BP 神经网络性能指标折线图如图 3-2 所示：

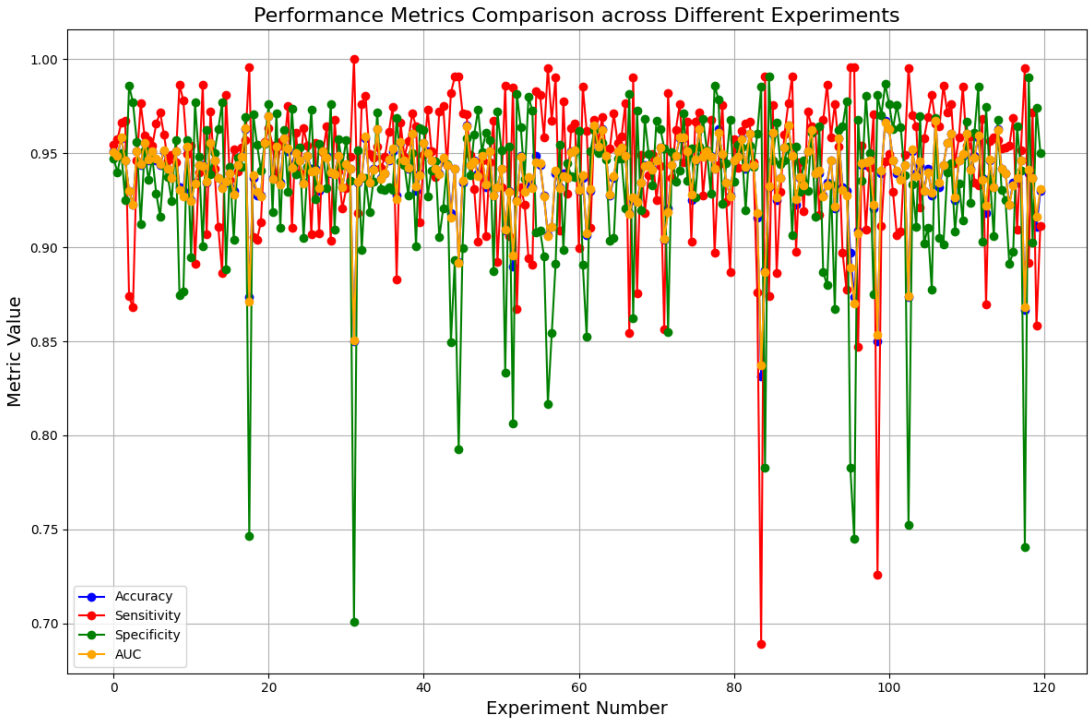


图 3-2：多轮测试中 BP 神经网络性能指标折线图(120 个指标)

可以看到，多次训练过程中出现了特异性（Specificity）这一项指标的不稳定，怀疑可能存在由以下几种因素：

①实验中，正负样本的比例不均衡，特异性 Specificity）容易受到负类样本数量的影响。如果负类样本较少，则即使分类错误的样本数不多，也可能导致特异性值大幅波动。

②实验中对负类样本拟合过度或不足时，会导致特异性表现出不稳定的趋势。这可能是因为模型在不同的训练过程中对负类样本的区分能力不一致。

③如果数据集中的负类样本存在较多异常值或错误标注，模型可能会在不同训练过程中产生显著不同的分类结果，从而影响特异性。

计算 276 轮次训练结果的均值和方差，得到结果如下表所示：

表 3-1：性能分析表

参数	Accuracy	Sensitivity	Specificity	AUC
均值	0.934	0.938	0.929	0.934
标准差	0.028	0.048	0.049	0.028

在对 276 个 BP 神经网络模型的训练中，结果显示各项性能指标（准确率、敏感性、特异性和 AUC）整体表现良好，但特异性波动较大，表现出较高的不稳定性。

从折线图和表格的标准差可以看到，特异性的标准差为 0.049，高于其他指标。这种不稳定性主要归因于样本扩充引起的负类样本特征多样性降低，使得模型难以在不同实验中对负样本保持一致的判断能力，进而导致特异性显著波动。为了提高特异性的稳定性，需要优化样本扩充策略。

经过分析，由于实验前期处理数据时进行了数据集的重复性扩充，导致模型会对部分样本的记忆性特别好，忽略对其他样本的学习，导致出现这一情况。

对训练过程中的误差进行记录，得到损失曲线如图 2-3 所示：

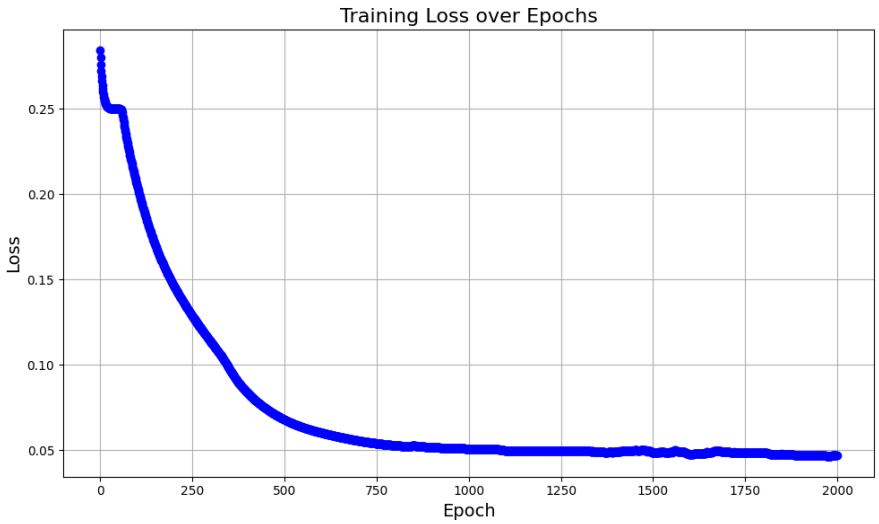


图 3-3：训练中的损失曲线

可以看到在训练初始，损失曲线出现了较为明显的停滞。可能存在几个方面的问题：

①模型参数的初始化对训练初期的收敛速度影响很大。如果参数初始化较差（如过于接近零或太大），模型在开始时需要较长时间来找到一个合适的方向进行优化。这会导致早期的损失值下降缓慢。

②学习率设置不当，早期阶段损失的下降速度较慢，可能是因为学习率设置得过低。在这种情况下，模型更新的步伐较小，参数调整得比较缓慢，导致损失值下降较慢。

为了突出问题，将学习率设定为 0.01 再次进行训练，得到图 3-4 如图所示：

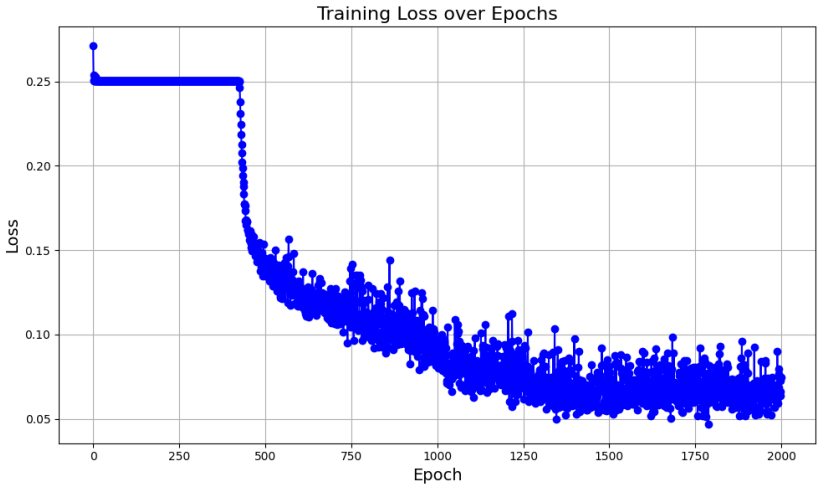


图 3-4：学习率为 0.01 的损失曲线

由图 3-4 可以看出，当学习率修改为 0.01 时，曲线停滞变得十分突出，怀疑存在局部最优解的情况。

绘制 BP 神经网络的 ROC 曲线如图 3-6 所示：

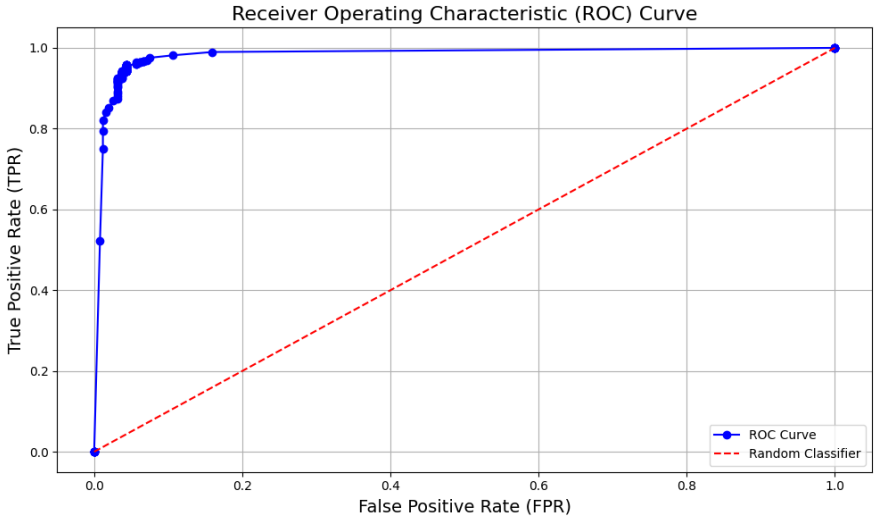


图 3-5：BP 神经网络的 ROC 曲线

图中的 ROC 曲线（蓝色曲线）紧贴左上角，显示出模型具有非常高的 True Positive Rate (TPR)（即灵敏度）和较低的 False Positive Rate (FPR)。这是理想的情况，说明模型能够很好地区分正类和负类样本。

对 ROC 曲线而言，AUC 值越接近 1，模型的性能就越好。从曲线形状来看，AUC 接近 1，说明该模型的分类能力很强。

BP 神经网络训练的混淆矩阵如图 3-7 所示：

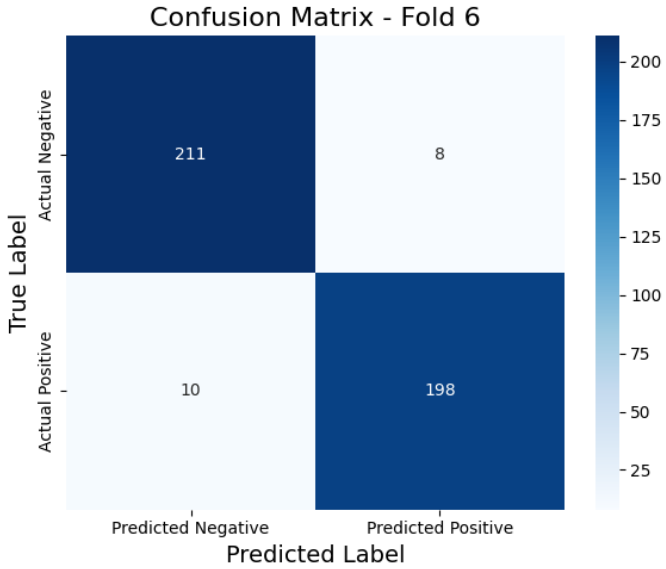


图 3-6：BP 神经网络的混淆矩阵

通过图 3-7 可以看出，这个模型在预测上表现出较高的准确性，精确率、召回率和 F1 值都维持在 95%以上，表明它对正负类别都有较好的预测能力。在这一折中的主要误差类型是 8 个假阳性（False Positive）和 10 个假阴性（False Negative），但这些错误率相对较低，说明模型总体表现良好。

综上所述，通过上述数据和曲线，发现 BP 算法可以较好地对男女进行分类，误差可以较快地收敛。

3.3 模型改进

3.3.1 早停机制

在训练后期，损失曲线已经逐渐趋于平稳，此时再反复进行训练已经没有意义，在此基础上，加入早停机制，只要检测到损失保持在一个不再变化的区间内，就结束训练，防止过拟合。输出情况如图 3-7 所示：

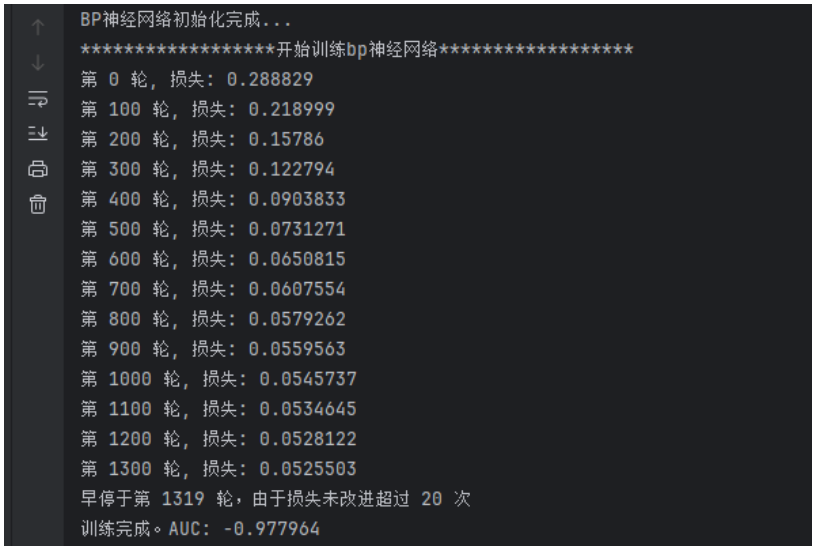


图 3-7：早停机制的训练结果

重新绘制损失曲线得到图 3-8：

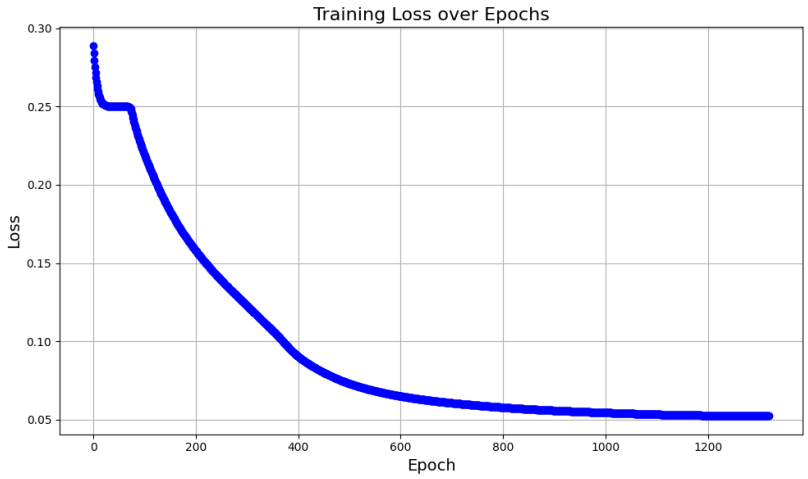


图 3-8：加入早停机制的损失曲线图

选择 20 轮作为早停机制的耐心值（最大允许无进步轮数）。加入后，在训练结果逐渐趋于平稳后，模型主动停止训练。有效防止了过拟合，使得模型整体性能（如准确率、敏感性和 AUC）表现较为稳定。

3.3.2 自适应学习率

自适应学习率（Adaptive Learning Rate）是一种动态调整模型学习率的方法，它在训练过程中根据模型的梯度变化情况自动调整学习率大小。与固定学习率的方法相比，自适应学习率可以更灵活地适应不同的训练阶段，使得模型在训练初期和后期都能更好地进行优化。

由图 3-3 可知，训练中的损失曲线在开始阶段出现了明显的停滞。针对此问题，可以考虑增加隐藏层，也可以考虑加入自适应学习率，有助于增加训练效率，或者提高训练的稳定性。

加入自适应学习率后，绘制训练损失曲线如图 3-9 所示：

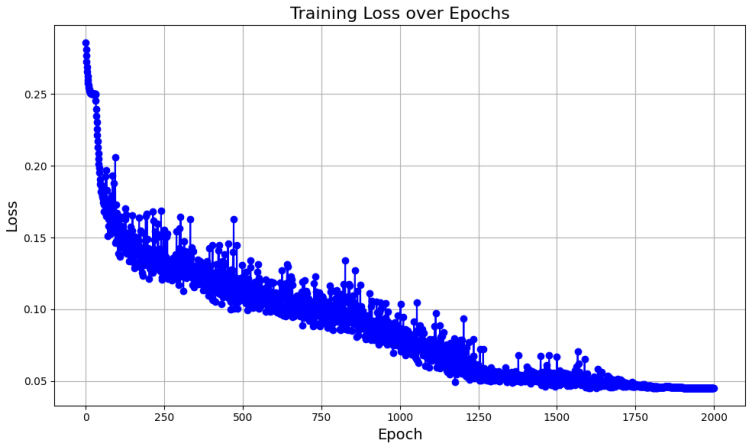


图 3-9：加入自适应学习率的损失曲线图

学习率的动态调整通过一系列参数来实现，以平衡模型的收敛速度和稳定性。初始学习率为 0.001，为训练提供了一个稳定的开始，防止步长过大导致的震荡问题。学习率衰减系数设为 0.9，在误差长期未见改善时逐步减小学习率，从而使模型在训练后期能够精细地调整权重，避免跳过最优解。最小学习率下限设置为 1e-5，以确保学习率不会无限下降，导致训练停滞。学习率增加系数为 1.05，在模型取得进展时适当提高学习率，加快训练收敛速度。

结合增长和衰减机制，以提升模型训练的效率。在训练初期，模型通过较高的初始学习率快速收敛；当训练进展到一定阶段后，如果连续多个轮次未见误差下降，则通过学习率衰减（即按一定比例降低学习率）防止跳过局部最优解。当模型表现出改善时，学习率略微增加，加速进一步收敛。该方法通过动态调整学习率，平衡了训练中的稳定性和收敛速度，最终帮助模型更有效地达到全局最优。

损失曲线下降率对比加入之前有了明显的增长，解决了一部分损失率停滞的情况，怀疑存在初始学习率过高、参数设置不当等问题，整体损失曲线出现了不稳定的情况。

结合以上改进，重新调整学习率参数和早停指标后，同时结合自适应学习率和早停机制，对模型再次进行训练，得到如图 3-10 所示的损失曲线图：

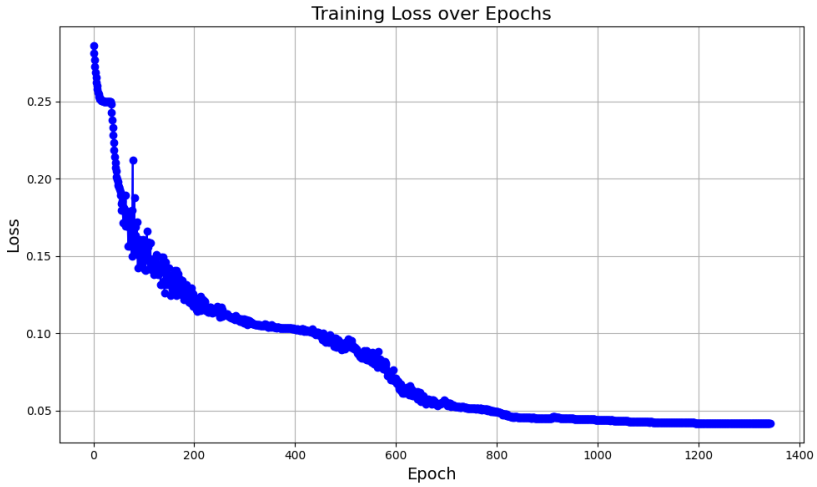


图 3-10：加入自适应学习率和早停机制的损失曲线图

通过图3-10可以看出，在加入早停机制和动态学习率调整后的训练过程中，模型展现出更加高效且稳健的收敛表现。动态学习率通过在训练过程中根据模型的改进情况调整学习率，使得模型在误差较大时能够更快收敛，而在误差趋于平稳时逐渐降低学习率，防止过大步长跳过最优解。结合早停机制，当损失值在多个 epoch 内未能显著改善时，训练过程自动终止，避免了模型过拟合的风险。从图像中可以看到，训练初期损失值下降迅速，随后随着学习率的调整，损失曲线趋于平稳，并最终在大约 1200 个 epoch 时停止，表明模型已达到较好的优化效果。这种组合策略不仅有效加速了训练过程，还提高了模型的泛化能力，同时节省了计算资源。

第四章 支持向量机（SVM）

使用 SVM，尝试不同的核函数设计分类器。核函数主要有线性核(linear)、多项式核(poly) 、高斯核(rbf)。

4.1 模型预处理

4.1.1 数据标准化处理

数据标准化（Data Standardization）是数据预处理的一个重要步骤，旨在调整数据尺度，使其具有均值为 0 和标准差为 1 的分布特性。在对数据进行标准化处理后，可以提高模型的收敛速度，避免数值不稳定性并且更加有利于特征之间的比较，尤其在特征值范围差异较大的情况下，可以有效提高机器学习模型的性能、稳定性和训练效率。应用标准化公式为：

$$X_{std} = \frac{X-\mu}{\sigma} \tag{4-1}$$

4.1.2 5 折交叉验证

当没有足够的数据用于训练模型时，划分数据的一部分进行验证会导致得到模型欠拟合，减少训练集，会使模型丧失部分数据集中重要的特征或趋势，这会增加偏差导致的误差。因此，我们需要一种方法来提供样本集训练模型并且留一部分数据集用于验证模型，k 折交叉验证（K-Flod）因此被提出。具体来说，先将数据集打乱，然后再将打乱后的数据集均匀分成 k 份，轮流选择其中的 k-1 份作为训练集，剩下的一份作验证，计算模型的误差平方和。迭代进行 k 次后将 k 次的误差平法和做平均作为选择最优模型的依据。

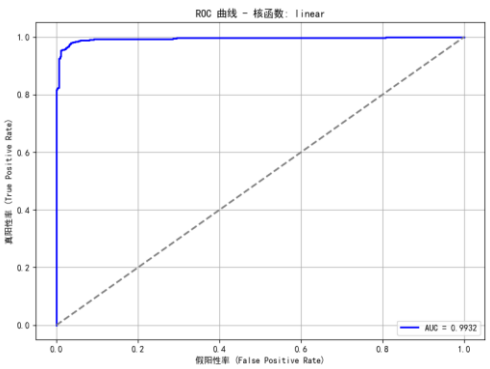
在进行 k 次交叉验证之后，使用 k 次平均成绩来作为整个模型的得分。每个数据在验证集中出现一次，并且在训练中出现 k-1 次。这将显著减少欠拟合，因为使用了数据集中的大多数的数据进行训练，同时也降低了过拟合的可能，因为也使用了大多数的数据进行模型的验证。

本实验采用交叉验证对数据集进行划分训练，并完成对性能指标的评判。交叉验证的 k 值一般取 10，但由于原始数据较少，分成 10 份会导致每份数据样本太少，导致混淆矩阵表中某些值可能为 0，在计算某些性能指标时可能会出现分母为 0 的情况，导致计算失败，于是 k 值选取的是 5。

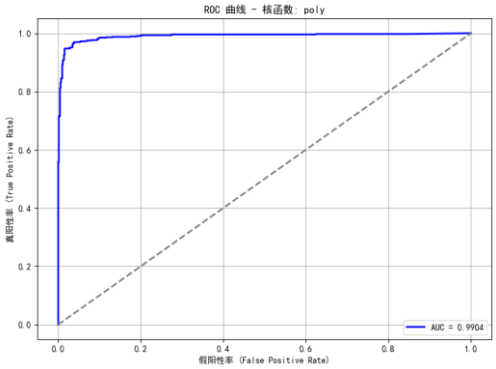
4.2 SVM 的性能指标

主要通过混淆矩阵和 ROC 来评判模型的性能。

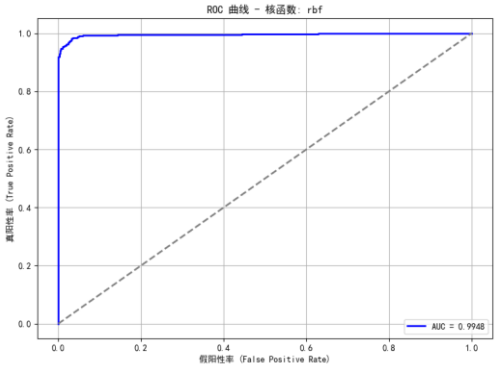
分别绘制不同核函数的 ROC 曲线如图 4-1 所示：



(a)linear 核函数



(b)poly 核函数

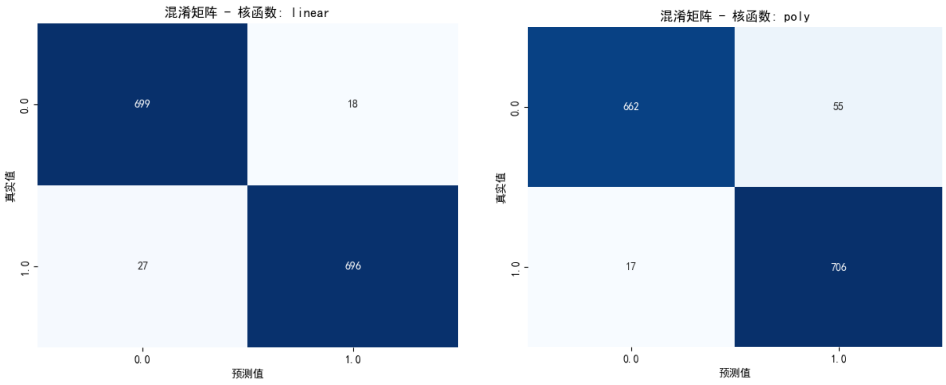


(c)rbf 核函数

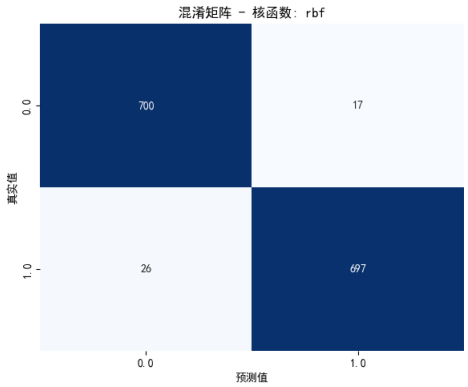
图 4-1：SVM 采用不同核函数的 ROC 曲线对比

通过 ROC 曲线以及混淆矩阵对比，可以看出三种核函数的分类模型表现都很出色。在 ROC 曲线中，Poly 核函数在这三者中表现略优于其他两者，但是负样本的误分类较多。整体表现来看，采用 rbf 核函数的分类器表现最佳，具有最高的准确率和较少的误分类。

绘制不同核函数的混淆矩阵如图 4-2 所示：



(a)linear 核函数　　(b)poly 核函数



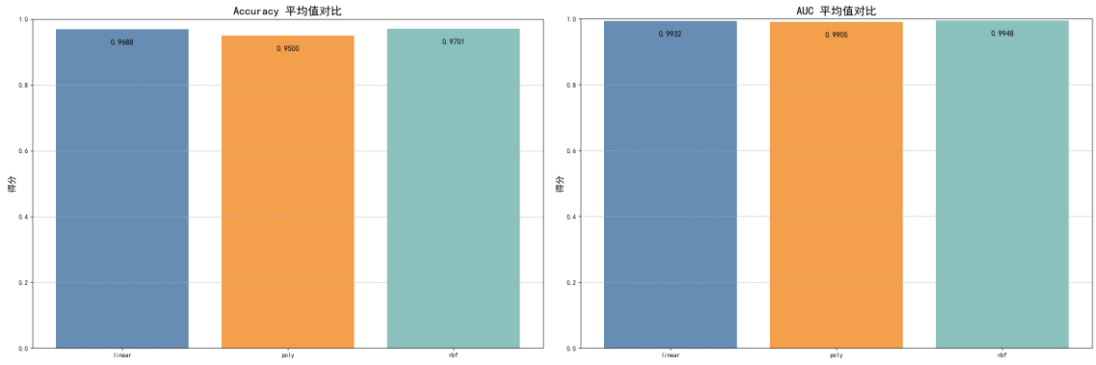
(c)rbf 核函数

图 4-2：SVM 采用不同核函数的混淆矩阵对比

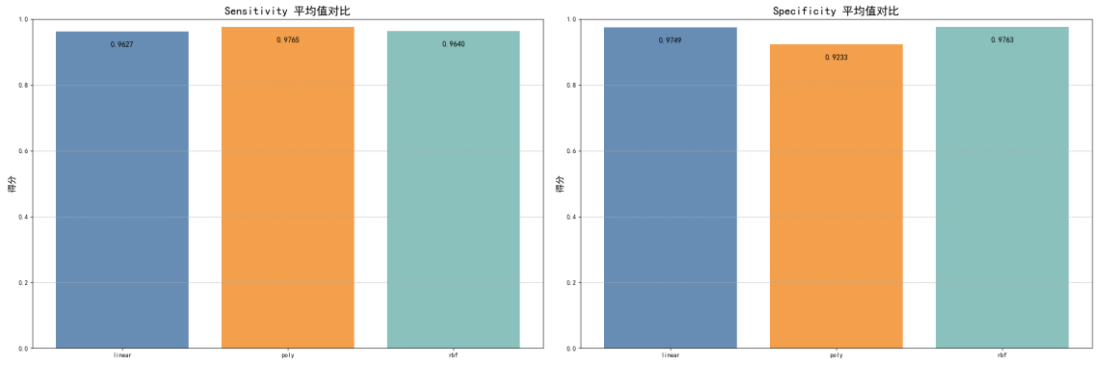
从混淆矩阵的分析来看，SVM 采用不同的核函数（linear、poly、rbf）在分类任务中的表现各有不同。其中，Linear 核函数在处理线性问题时表现较为均衡，但误分类率略高，尤其对类别 1 有一定的误判。Poly 核函数对类别 0 的表现较差，误分类数量较多，说明在处理复杂边界时多项式核的效果不稳定。相比之下，RBF 核函数表现最为出色，分类误差最小，对类别 0 和类别 1 都有较好的识别能力。总体来看，RBF 核函数能够更好地处理复杂的非线性数据分布，因此对于追求高分类准确率的任务，优先选择 RBF 核函数更为合理。

4.3 不同核函数的 SVM 对比分析

对于不同的核函数，将各项性能指标绘制柱状图进行对比，如图 4-3 所示：



(a) Accuracy 平均值对比　　(b) AUC 平均值对比



(c) Sensitivity 平均值对比　　(d) Specificity 平均值对比

图 4-3：SVM 采用不同核函数的性能指标对比

由各项性能指标得，各个核函数的分类效果区别不是特别大，但是采用 rbf 核函数的 SVM 的分类效果最佳，各项指标都略优于其他核函数。

4.4 特征组合分析

为了研究不同特征对分类结果的影响，选用特征投影法，对不同特征按照特定权值进行两两组合，对数据进行降维并可视化处理，得到不同核函数的特征组合投影如图 4-4 到图 4-6 所示（以身高体重为例）。通过特征组合投影图，可以清晰直观地看出不同特征对分类结果的影响。可以看出采用 rbf 核函数的特征组合投影的分类结果更加清晰。

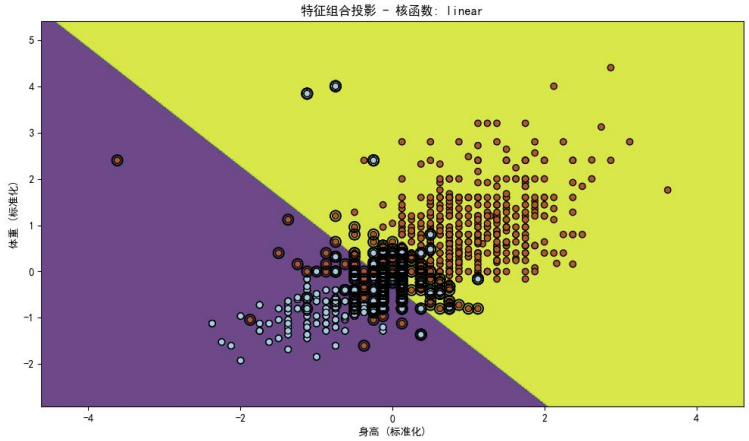


图 4-4：SVM 采用 linear 核函数的特征组合投影

图 4-4 显示了使用 Linear 核函数时的特征投影。该投影结果表现出一个线性分割边界，分割线几乎为一条直线，将两类样本分隔开。这表明 Linear 核函数对于线性可分的数据表现良好，但分类效果相对较为简单，对复杂的数据边界不足以描述其真实的分布情况。

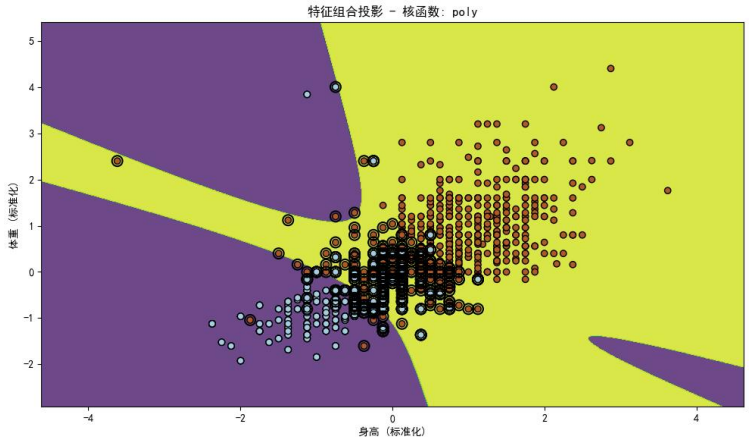


图 4-5：SVM 采用 poly 核函数的特征组合投影

图 4-5 中展示了 Poly 核函数的特征投影。分类边界较为复杂，呈现出多重弯曲形状。这反映出多项式核函数能够对特征空间进行高阶变换，以处理更复杂的样本关系，但同时也可能在一些区域出现过度拟合的情况，使得决策边界不够平滑且容易受到噪声影响。

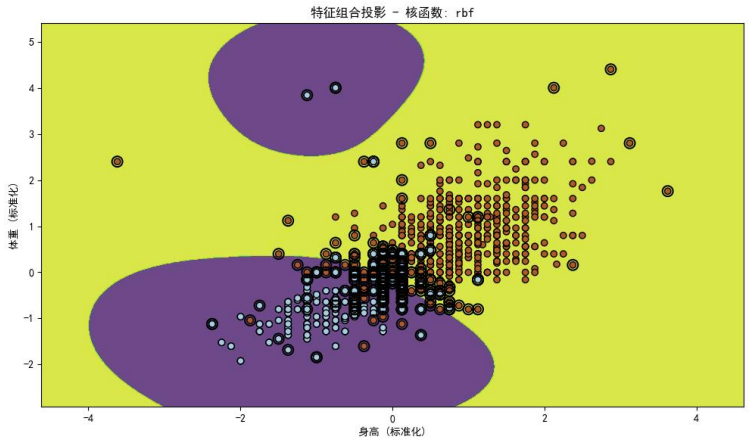


图 4-6：SVM 采用 rbf 核函数的特征组合投影

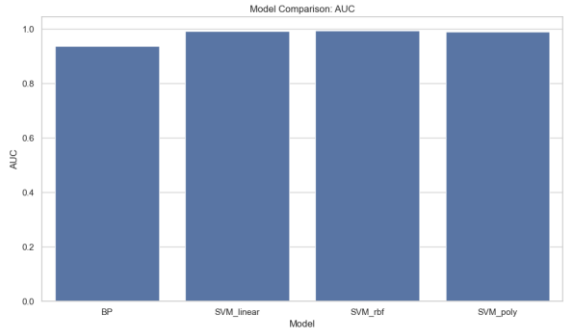
在图 4-6 中，使用 RBF 核函数的投影结果显示出更加复杂而平滑的决策边界。不同类别的区域由弯曲的边界精确分隔，特别是在高密度的数据交叉区域表现出良好的分离效果。RBF 核函数通过将数据映射到更高的维度来更好地捕捉数据之间的非线性关系，因此它对复杂分布的数据有更好的适应性。

综上，通过对三种核函数的降维投影结果的分析，Linear 核函数适用于线性可分数据，分割边界较为简单；Poly 核函数能够处理更复杂的数据，但可能存在过拟合的风险；而 RBF 核函数的投影效果显示了其在复杂数据集上的优秀性能，能够形成较平滑的边界，有效区分不同类别。因此，在处理复杂的非线性数据时，RBF 核函数往往是最优的选择。

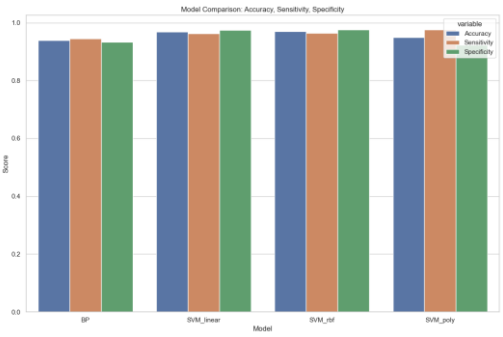
第五章 结果分析

5.1 BP 与 SVM 对比分析

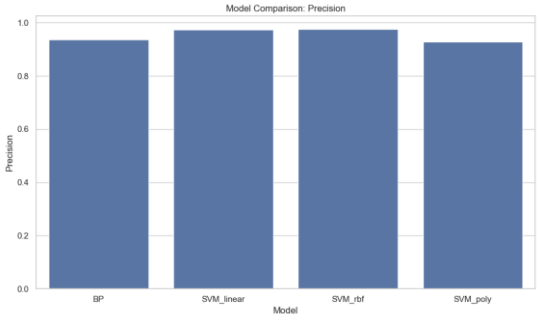
在以上实验的基础上，进一步将 BP 算法和 SVM 结果进行对比分析，绘制各项性能指标的柱状对比图如图 5-1 所示：



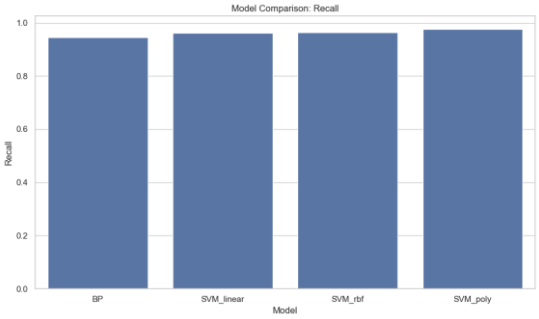
(a) AUC 对比



(b) 性能指标对比



(c) precision 对比



(d) recall 对比

图 5-1：BP 算法和 SVM 对比分析

从 BP 算法与 SVM 各种核函数的对比来看，BP 和 SVM (rbf) 核函数在各项性能指标上表现出色，尤其在敏感性和召回率方面表现优异，说明它们在识别正类样本上更为有效。相比之下，SVM 的 poly 核函数在特异性上稍逊一筹，表明其在识别负类样本时存在误判的风险，而 SVM 的 linear 核函数在敏感性和召回率方面略低于其他模型。总体而言，面对复杂的非线性数据时，BP 算法和 SVM (rbf) 核函数是更为理想的选择，能够提供更好的分类性能和更高的稳定性。

绘制不同模型的 ROC 曲线如图 5-2 所示：

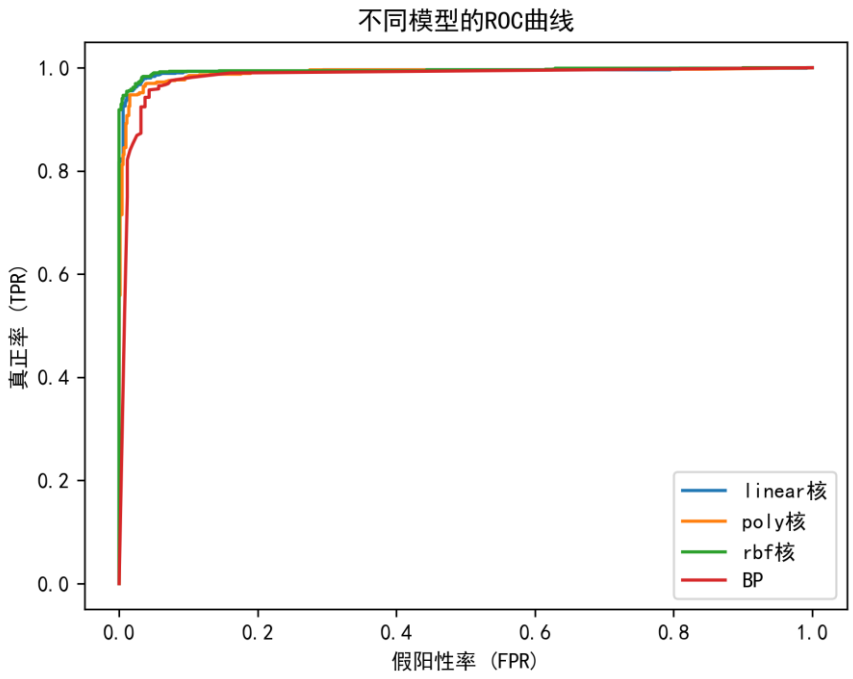


图 5-2：BP、SVM 的 ROC 曲线对比图

通过各项性能指标对比，可以看出 SVM 整体上都要优于 BP 算法得的结果。这可能是因为样本整体数量偏少，BP 算法不能得出一个较好的计算结果。同时，在所有模型里，采用 rbf 核函数的 SVM 的结果是最优的。

5.2 调整与改进

由于自身水平限制，时间原因，本实验仍有以下问题未解决：

①直接复制数据集会导致模型的泛化能力变差，增加过拟合风险，因为重复样本会让模型过度学习特定模式而缺乏多样性，从而难以在新数据上表现良好。为解决这些问题，应避免简单地复制数据，而应采用数据扩充技术（如旋转、缩放、同义词替换等）来增加样本多样性，还可以使用欠采样或过采样策略（如 SMOTE）来平衡数据。此外，确保训练集和验证集的分布尽可能贴近实际数据，以提高模型的稳定性和在新数据上的泛化能力。

②为了解决模型中的过拟合问题，可以采用多种策略。正则化（如 L1、L2 正则化）和 Dropout 技术能够有效降低模型的复杂度，防止其对训练数据的过度学习。简化模型结构、减少网络层数和神经元数量也有助于降低过拟合风险。

③为了优化当前臃肿的 C++ 代码，可以将重复的部分提取成独立的函数，减少代码冗余。利用类和对象的封装来模块化功能，从而提高代码的可维护性和复用性。

附录：代码

BP 神经网络

①xtoc.py（数据预处理）

```
import pandas as pd
def xlsx_to_csv(xlsx_file, csv_file):
    # 读取xlsx文件
    df = pd.read_excel(xlsx_file)

    # 统计男生和女生的数量
    male_count = df[df['性别 男1女0'] == 1].shape[0]
    female_count = df[df['性别 男1女0'] == 0].shape[0]

    # 如果女生数量少于男生，通过复制扩充女生数量与男生一样多
    if female_count < male_count:
        female_df = df[df['性别 男1女0'] == 0]
        copies_needed = (male_count - female_count) // female_count + 1
        expanded_female_df = pd.concat([female_df] * copies_needed,
ignore_index=True).iloc[:(male_count - female_count)]
        df = pd.concat([df, expanded_female_df], ignore_index=True)

    # 将数据保存到csv文件，处理中文编码问题
    df.to_csv(csv_file, index=False, encoding='utf-8-sig')

# 示例用法
xlsx_file = 'data.xlsx'  # 你的xlsx文件路径
csv_file = 'data.csv'    # 输出的csv文件路径

xlsx_to_csv(xlsx_file, csv_file)
```

②main.cpp

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include <sstream>
#include <algorithm>
#include <filesystem>
#include <locale>
#include <codecvt>
#include <random>
#include "bp.h"

namespace fs = std::filesystem;
```

```
/*****************************数据过滤范围*********************************/
const double MIN_HEIGHT = 140.0;//身高
const double MAX_HEIGHT = 220.0;
const double MIN_WEIGHT = 30.0;//体重
const double MAX_WEIGHT = 150.0;
const double MIN_SHOE_SIZE = 30.0;//鞋码
const double MAX_SHOE_SIZE = 50.0;
const double MIN_50M_TIME = 5.0;//50m
const double MAX_50M_TIME = 15.0;
const double MIN_VITAL_CAPACITY = 1000.0;//肺活量
const double MAX_VITAL_CAPACITY = 8000.0;

/*****************************数据预处理*********************************/
// 检查数据是否在合理范围内
bool isValidData(const std::vector<std::string> &data) {
    try {
        // 检查是否有空白数据
        for (const auto &item : data) {
            if (item.empty()) {
                return false;
            }
        }

        double gender = std::stod(data[1]);
        double height = std::stod(data[3]);
        double weight = std::stod(data[4]);
        double shoeSize = std::stod(data[5]);
        double fiftyMeter = std::stod(data[6]);
        double vitalCapacity = std::stod(data[7]);

        return (gender == 0 || gender == 1) &&
               (height >= MIN_HEIGHT && height <= MAX_HEIGHT) &&
               (weight >= MIN_WEIGHT && weight <= MAX_WEIGHT) &&
               (shoeSize >= MIN_SHOE_SIZE && shoeSize <= MAX_SHOE_SIZE) &&
               (fiftyMeter >= MIN_50M_TIME && fiftyMeter <= MAX_50M_TIME)
&&
               (vitalCapacity >= MIN_VITAL_CAPACITY && vitalCapacity <=
MAX_VITAL_CAPACITY);
    } catch (const std::exception &e) {
        // 如果转换失败，数据无效
        return false;
    }
}

// 解析CSV行
std::vector<std::string> parseCSVLine(const std::string &line) {
    std::vector<std::string> result;
    std::stringstream ss(line);
    std::string item;

    while (std::getline(ss, item, ',')) {
        result.push_back(item);
    }
```

```
    return result;
}

// 打开文件并处理数据
bool processFile(const std::string& inputFilePath, const std::string&
outputFilePath, const std::string& invalidFilePath) {
    if (!fs::exists(inputFilePath)) {
        std::cerr << "错误：输入文件不存在：" << inputFilePath << std::endl;
        return false;
    }

    std::wifstream inputFile(inputFilePath);
    inputFile.imbue(std::locale(inputFile.getloc(), new
std::codecvt_utf8<wchar_t>));
    std::wofstream outputFile(outputFilePath, std::ios::out |
std::ios::binary);
    outputFile.imbue(std::locale(outputFile.getloc(), new
std::codecvt_utf8<wchar_t>));
    std::wofstream invalidFile(invalidFilePath, std::ios::out |
std::ios::binary);
    invalidFile.imbue(std::locale(invalidFile.getloc(), new
std::codecvt_utf8<wchar_t>));

    if (!inputFile.is_open()) {
        std::cerr << "输入文件正被使用" << inputFilePath << std::endl;
        return false;
    }

    if (!outputFile.is_open()) {
        std::cerr << "无法创建输出文件" << outputFilePath << std::endl;
        return false;
    }

    if (!invalidFile.is_open()) {
        std::cerr << "无法创建无效数据文件" << invalidFilePath << std::endl;
        return false;
    }

    std::wstring line;
    bool isFirstLine = true;

    while (std::getline(inputFile, line)) {
        // 写入标题行
        if (isFirstLine) {
            outputFile << line << std::endl;
            invalidFile << line << std::endl;
            isFirstLine = false;
            continue;
        }

        // 解析行数据
        std::wstring_convert<std::codecvt_utf8<wchar_t>> converter;
        std::string utf8Line = converter.to_bytes(line);
```

```
        std::vector<std::string> data = parseCSVLine(utf8Line);

        // 检查数据完整性和有效性
        if (data.size() < 8) {
            invalidFile << line << std::endl; // 跳过不完整的行并记录
            continue;
        }

        // 检查是否是有效的数据
        if (isValidData(data)) {
            outputFile << line << std::endl;
        } else {
            invalidFile << line << std::endl; // 记录无效的数据
        }
    }

    inputFile.close();
    outputFile.close();
    invalidFile.close();

    std::cout << "数据过滤结果保存在 " << outputFilePath << std::endl;
    std::cout << "无效数据保存在 " << invalidFilePath << std::endl;
    return true;
}

/*****************************数据集划分*********************************/
// 函数：随机划分 CSV 文件为训练集和测试集，并存到文件中
void splitCSVFile(const std::string& inputFilePath, const std::string&
trainFilePath, const std::string& testFilePath, double train_ratio = 0.7)
{
    std::ifstream inputFile(inputFilePath);
    if (!inputFile.is_open()) {
        std::cerr << "无法打开输入文件: " << inputFilePath << std::endl;
        return;
    }

    std::vector<std::string> lines;
    std::string line;
    bool isFirstLine = true;
    std::string header;

    // 读取 CSV 所有行
    while (std::getline(inputFile, line)) {
        if (isFirstLine) {
            header = line;
            isFirstLine = false;
            continue;
        }
        lines.push_back(line);
    }
    inputFile.close();

    // 随机打乱数据
```

```
    std::shuffle(lines.begin(), lines.end(),
std::default_random_engine(std::random_device{}()));

    // 划分训练集和测试集
    size_t train_size = static_cast<size_t>(lines.size() * train_ratio);
    std::vector<std::string> train_lines(lines.begin(), lines.begin() +
train_size);
    std::vector<std::string> test_lines(lines.begin() + train_size,
lines.end());

    // 保存训练集到文件
    std::ofstream trainFile(trainFilePath);
    if (trainFile.is_open()) {
        trainFile << header << std::endl;
        for (const auto& train_line : train_lines) {
            trainFile << train_line << std::endl;
        }
        trainFile.close();
    } else {
        std::cerr << "无法创建训练集文件: " << trainFilePath << std::endl;
    }

    // 保存测试集到文件
    std::ofstream testFile(testFilePath);
    if (testFile.is_open()) {
        testFile << header << std::endl;
        for (const auto& test_line : test_lines) {
            testFile << test_line << std::endl;
        }
        testFile.close();
    } else {
        std::cerr << "无法创建测试集文件: " << testFilePath << std::endl;
    }
}

// 从 CSV 文件中读取训练数据集
bool loadData(const std::string& filePath,
std::vector<std::vector<double>>& X, std::vector<std::vector<double>>& y)
{
    std::ifstream inputFile(filePath);
    if (!inputFile.is_open()) {
        std::cerr << "无法打开文件 " << filePath << std::endl;
        return false;
    }

    std::string line;
    bool isFirstLine = true;

    // 读取文件内容并解析
    while (std::getline(inputFile, line)) {
        if (isFirstLine) {
            // 跳过标题行
            isFirstLine = false;
```

```
            continue;
        }

        std::vector<std::string> data = parseCSVLine(line);
        if (data.size() >= 8) {
            // 提取特征：身高、体重、鞋码、50m 成绩
            std::vector<double> features = {
                    std::stod(data[3]),  // 身高
                    std::stod(data[4]),  // 体重
                    std::stod(data[5]),  // 鞋码
                    std::stod(data[6])   // 50m 成绩
            };
            X.push_back(features);

            // 提取目标输出：性别 (0 或 1)
            y.push_back({std::stod(data[1])});
        }
    }
    inputFile.close();
    return true;
}

/*****************************训练*********************************/
// 训练 BP 神经网络的函数，并保存 ROC 和 AUC 数据
void trainBPNeuralNetwork(BPNeuralNetwork& bpnn, const
std::vector<std::vector<double>>& X, const
std::vector<std::vector<double>>& y,
                          int epochs, const std::string& lossFilePath,
std::vector<double>& loss_values,
                          const std::string& rocFilePath, const std::string&
aucFilePath) {
    // 打开文件保存损失
    std::ofstream lossFile(lossFilePath);
    if (!lossFile.is_open()) {
        std::cerr << "无法创建损失值文件" << std::endl;
        return;
    }
    lossFile << "Epoch,Loss" << std::endl;

    //参数
    int patience = 15; // 设置耐心值（最大允许无进步轮数）
    double best_loss = std::numeric_limits<double>::max();
    int no_improve_epochs = 0;
    double learning_rate = 0.001; // 初始化学习率
    double lr_decay = 0.9;  // 学习率衰减系数
    double min_learning_rate = 1e-5; // 最小学习率下限
    double increase_factor = 1.05; // 学习率增加系数

    for (int epoch = 0; epoch < epochs; ++epoch) {
        double total_loss = 0.0;

        for (size_t i = 0; i < X.size(); ++i) {
```

```
            // 前向传播
            std::vector<double> prediction = bpnn.forward(X[i]);
            // 计算损失
            double error = y[i][0] - prediction[0];
            total_loss += error * error;
            // 设置类别权重
            double class_weight = (y[i][0] == 1) ? 0.5 : 0.5;
            // 反向传播，并根据动态学习率调整
            //bpnn.setLearningRate(learning_rate); // 动态设置学习率
            bpnn.backward(y[i], class_weight);
        }

        total_loss /= X.size();
        loss_values.push_back(total_loss);
        if (epoch % 100 == 0) {
            std::cout << "第 " << epoch << " 轮，损失: " << total_loss <<
std::endl;
        }
        lossFile << epoch << "," << total_loss << std::endl;

        // Early stopping 检查逻辑
        if (total_loss < best_loss) {
            best_loss = total_loss;  // 更新最佳损失
            no_improve_epochs = 0;   // 重置无改进轮数
            // 增加学习率以加速收敛
            learning_rate = std::min(learning_rate * increase_factor,
1.0); // 限制学习率增长上限为 1.0
        } else {
            no_improve_epochs++;     // 增加无改进轮数
        }

        // 如果无改进轮数达到耐心值，降低学习率
        if (no_improve_epochs >= patience) {
            learning_rate *= lr_decay;  // 学习率衰减
            if (learning_rate < min_learning_rate) {
                learning_rate = min_learning_rate;  // 限制学习率不低于最小值
                std::cout << "学习率已降至最小值，早停于第 " << epoch << " 轮"
<< std::endl;
                break;
            }
            std::cout << "损失未改进，降低学习率至: " << learning_rate <<
std::endl;
            no_improve_epochs = 0;  // 重置无改进轮数，继续训练
        }
    }
    lossFile.close();

    // 保存 ROC 数据
    std::ofstream rocFile(rocFilePath);
    if (!rocFile.is_open()) {
        std::cerr << "无法创建 ROC 数据文件" << std::endl;
        return;
```

```
    }
    rocFile << "Threshold,TPR,FPR\n";

    // 使用更密集的阈值评估模型表现
    for (double threshold = 0.0; threshold <= 1.0; threshold += 0.01) { //
更小的步长，更密集的数据
        int tp = 0, tn = 0, fp = 0, fn = 0;

        for (size_t i = 0; i < X.size(); ++i) {
            std::vector<double> prediction = bpnn.forward(X[i]);
            int predicted_label = (prediction[0] >= threshold) ? 1 : 0;
            int actual_label = static_cast<int>(y[i][0]);

            if (predicted_label == 1 && actual_label == 1) {
                tp++;
            } else if (predicted_label == 0 && actual_label == 0) {
                tn++;
            } else if (predicted_label == 1 && actual_label == 0) {
                fp++;
            } else if (predicted_label == 0 && actual_label == 1) {
                fn++;
            }
        }

        // 计算 TPR 和 FPR
        double tpr = (tp + fn) ? static_cast<double>(tp) / (tp + fn) :
0.0; // Sensitivity (TPR)
        double fpr = (fp + tn) ? static_cast<double>(fp) / (fp + tn) :
0.0; // FPR

        // 保存到 ROC 文件
        rocFile << threshold << "," << tpr << "," << fpr << "\n";
    }
    rocFile.close();

    // 计算 AUC
    double auc = 0.0;
    std::ifstream rocData(rocFilePath);
    if (!rocData.is_open()) {
        std::cerr << "无法读取 ROC 数据文件以计算 AUC" << std::endl;
        return;
    }

    std::vector<std::pair<double, double>> roc_points;
    std::string line;
    std::getline(rocData, line); // 跳过标题行

    while (std::getline(rocData, line)) {
        std::stringstream ss(line);
        std::string value;
        double tpr, fpr;

        std::getline(ss, value, ','); // 跳过阈值字段
```

```
        std::getline(ss, value, ',');
        tpr = std::stod(value);
        std::getline(ss, value, ',');
        fpr = std::stod(value);

        roc_points.emplace_back(fpr, tpr);
    }
    rocData.close();

    // 计算 AUC 值（使用梯形法则）
    for (size_t i = 1; i < roc_points.size(); ++i) {
        double x_diff = roc_points[i].first - roc_points[i - 1].first;
        double y_avg = (roc_points[i].second + roc_points[i - 1].second) /
2.0;
        auc += x_diff * y_avg;
    }

    // 保存 AUC 到文件
    std::ofstream aucFile(aucFilePath);
    if (!aucFile.is_open()) {
        std::cerr << "无法创建 AUC 文件" << std::endl;
        return;
    }
    aucFile << "AUC\n";
    aucFile << auc << "\n";
    aucFile.close();

    std::cout << "训练完成。AUC: " << auc << std::endl;
}


void printConfusionMatrix(int tp, int tn, int fp, int fn) {
    std::cout << "混淆矩阵: " << std::endl;
    std::cout << "TP: " << tp << ", FN: " << fn << std::endl; // 真阳性与假
阴性
    std::cout << "FP: " << fp << ", TN: " << tn << std::endl; // 假阳性与真
阴性
}

/*****************************模型评估*********************************/
// 使用测试集评估 BP 神经网络，并保存错分样本
double evaluateBPNeuralNetwork(BPNeuralNetwork& bpnn, const
std::vector<std::vector<double>>& X_test, const
std::vector<std::vector<double>>& y_test, const std::string&
misclassifiedFilePath) {
    int correct_predictions = 0;
    std::ofstream misclassifiedFile(misclassifiedFilePath);
    if (!misclassifiedFile.is_open()) {
        std::cerr << "无法创建错分样本文件" << std::endl;
        return 0.0;
    }
    misclassifiedFile << "Features,Actual,Predicted" << std::endl;
```

```
    for (size_t i = 0; i < X_test.size(); ++i) {
        std::vector<double> prediction = bpnn.forward(X_test[i]);
        int predicted_label = (prediction[0] >= 0.5) ? 1 : 0;
        int actual_label = static_cast<int>(y_test[i][0]);
        if (predicted_label == actual_label) {
            ++correct_predictions;
        } else {
            // 保存错分样本
            misclassifiedFile << "\"";
            for (size_t j = 0; j < X_test[i].size(); ++j) {
                misclassifiedFile << X_test[i][j];
                if (j < X_test[i].size() - 1) {
                    misclassifiedFile << ",";
                }
            }
            misclassifiedFile << "\"," << actual_label << "," <<
predicted_label << "\n";
        }
    }
    misclassifiedFile.close();

    double accuracy = static_cast<double>(correct_predictions) /
X_test.size();
    std::cout << "模型在测试集上的准确率: " << accuracy * 100 << "%" <<
std::endl;
    return accuracy;
}

// 交叉验证（不包含训练过程）
void crossValidation(const std::vector<std::vector<double>>& X, const
std::vector<std::vector<double>>& y,BPNeuralNetwork& bpnn,const
std::string& metricsFilePath) {
    double total_se = 0.0;
    double total_sp = 0.0;
    double total_acc = 0.0;
    double total_auc = 0.0;

    // 打开 CSV 文件以保存性能指标
    std::ofstream metricsFile(metricsFilePath, std::ios::app);
    if (!metricsFile.is_open()) {
        std::cerr << "无法打开性能指标文件: " << metricsFilePath <<
std::endl;
        return;
    }
   // metricsFile << "Accuracy,Sensitivity,Specificity,AUC,TP,FP,TN,FN"
<< std::endl;


    // 评估模型性能
        int tp = 0, tn = 0, fp = 0, fn = 0;
        for (size_t i = 0; i < X.size(); ++i) {
            std::vector<double> prediction = bpnn.forward(X[i]);
            int predicted_label = (prediction[0] >= 0.5) ? 1 : 0;
```

```
            int actual_label = static_cast<int>(y[i][0]);

            if (predicted_label == 1 && actual_label == 1) {
                tp++;
            } else if (predicted_label == 0 && actual_label == 0) {
                tn++;
            } else if (predicted_label == 1 && actual_label == 0) {
                fp++;
            } else if (predicted_label == 0 && actual_label == 1) {
                fn++;
            }
        }

        printConfusionMatrix(tp, tn, fp, fn);
        // 计算性能指标
        double se = static_cast<double>(tp) / (tp + fn); // Sensitivity
        double sp = static_cast<double>(tn) / (tn + fp); // Specificity
        double acc = static_cast<double>(tp + tn) / (tp + tn + fp + fn);
// Accuracy
        double auc = (se + sp) / 2.0; // 近似计算 AUC

        total_se += se;
        total_sp += sp;
        total_acc += acc;
        total_auc += auc;

    // 输出平均性能指标
    std::cout << "SE: " << total_se << std::endl;
    std::cout << "SP: " << total_sp << std::endl;
    std::cout << "ACC: " << total_acc  << std::endl;
    std::cout << "AUC: " << total_auc  << std::endl;

    // 保存每个性能指标到文件
    metricsFile << acc << "," << se << "," << sp << "," << auc << "," <<
tp << "," << fp << "," << tn << "," << fn << std::endl;
}

/*****************************调试用*********************************/
// 打印 X 和 Y 向量
void printData(const std::vector<std::vector<double>>& X, const
std::vector<std::vector<double>>& y) {
    std::cout << "X Data:" << std::endl;
    for (const auto& row : X) {
        for (const auto& val : row) {
            std::cout << val << " ";
        }
        std::cout << std::endl;
    }

    std::cout << "Y Data:" << std::endl;
    for (const auto& val : y) {
        std::cout << val[0] << std::endl; // 假设 y 是一个二元向量
    }
```

```
}

int main() {
    /*****************************路径*********************************/
    std::string inputFilePath =
"E:/GitHub/AI_VI/2.BPSVM/BP/data/data.csv";//源数据集路径
    std::string outputFilePath =
"E:/GitHub/AI_VI/2.BPSVM/BP/data/filtered_data.csv";//过滤后的数据集路径
    std::string invalidFilePath =
"E:/GitHub/AI_VI/2.BPSVM/BP/data/invalid_data.csv";//无效数据路径

    std::string trainFilePath =
"E:/GitHub/AI_VI/2.BPSVM/BP/data/train.csv";//训练集路径
    std::string testFilePath =
"E:/GitHub/AI_VI/2.BPSVM/BP/data/test.csv";//测试集路径

    std::string lossFilePath =
"E:/GitHub/AI_VI/2.BPSVM/BP/data/loss_values.csv";//保存的损失值
    std::string ROCFilePath =
"E:/GitHub/AI_VI/2.BPSVM/BP/data/roc_values.csv";//保存的损失值
    std::string AUCFilePath =
"E:/GitHub/AI_VI/2.BPSVM/BP/data/auc_values.csv";//保存的损失值

    std::string misclassifiedFilePath =
"E:/GitHub/AI_VI/2.BPSVM/BP/data/result/misclassified_samples.csv";//模型
的错分样本

    std::string modelFilePath =
"E:/GitHub/AI_VI/2.BPSVM/BP/data/result/models.txt";//模型参数
    std::string matFilePath = "E:/GitHub/AI_VI/2.BPSVM/BP/data/mat.csv";//
评估结果的输出矩阵

    // 循环次数
    int n = 1;  // 定义需要执行多少次整个流程

    for (int iteration = 1; iteration <= n; ++iteration) {
        std::cout << "================= 第 " << iteration << " 次循环开始
==================" << std::endl;

        /*****************************BP 神经网络的参数配置
*********************************/
        int input_size = 4;       // 输入层节点数为 4
        int hidden_size = 4;      // 隐藏层节点数为 4
        int output_size = 1;      // 输出层节点数为 1 (性别分类：0 或 1)
        double learning_rate = 0.01;//学习率
        int epochs = 2000;//训练轮次

        //1.调用函数预处理文件
        std::cout << "*******************数据集预处理****************** "
<< std::endl;
        if (!processFile(inputFilePath, outputFilePath, invalidFilePath))
{
```

```
            std::cerr << "文件处理失败。" << std::endl;
            return 1;
        }
        else{
            std::cout << "数据集预处理成功！" << std::endl;
        }

        //2.划分数据集
        // 将输入文件随机划分为训练集和测试集，比例为7:3
        splitCSVFile(outputFilePath, trainFilePath, testFilePath);
        std::cout << "按照 7:3 比例随机划分为训练集和测试集保存到文件中。" <<
std::endl;

        //2.加载数据集
        // 构建训练数据集，读取 train.csv 文件
        std::vector<std::vector<double>> X;
        std::vector<std::vector<double>> Y;
        if (!loadData(trainFilePath, X, Y)) {
            return 1;
        }
        //printData(X,Y);
        // 构造测试数据集，读取 test.csv 文件
        std::vector<std::vector<double>> X_test;
        std::vector<std::vector<double>> Y_test;
        if (!loadData(testFilePath, X_test, Y_test)) {
            return 1;
        }
        std::cout << "数据集加载成功！" << std::endl;
        //printData(X_test,Y_test);

        //3.初始化 BP 神经网络
        BPNeuralNetwork bpnn(input_size, hidden_size, output_size,
learning_rate);
        std::cout << "BP 神经网络初始化完成..." << std::endl;

        //4.训练 BP 神经网络，并记录损失值到文件中,保存模型参数
        std::vector<double> loss_values;
        std::cout << "******************开始训练 bp 神经网络
****************** " << std::endl;
        trainBPNeuralNetwork(bpnn, X, Y, epochs, lossFilePath,
loss_values,ROCFilePath,AUCFilePath);
        // 保存模型参数
        bpnn.saveModel(modelFilePath);

        //5.根据测试集计算模型准确率
        std::cout << "******************模型性能交叉验证评估
******************" << std::endl;
        evaluateBPNeuralNetwork(bpnn, X_test,
Y_test,misclassifiedFilePath);

        //6 使用交叉验证评估模型
        crossValidation(X_test, Y_test, bpnn,matFilePath);
```

```
        std::cout << "================== 第 " << iteration << " 次循环结束
==================" << std::endl << std::endl;
    }

    return 0;
}
```

③bp.cpp

```
    // bp.cpp
#include "bp.h"
#include <cmath>
#include <random>
#include <iostream>
#include <fstream>

// BP 神经网络构造函数，初始化权重和偏置
BPNeuralNetwork::BPNeuralNetwork(int input_size, int hidden_size, int
output_size, double learning_rate)
        : learning_rate(learning_rate) {
    // 使用随机数生成器初始化权重和偏置，范围为 -1 到 1
    std::default_random_engine generator;
    std::uniform_real_distribution<double> distribution(-1.0, 1.0);

    weights_input_hidden.resize(input_size,
std::vector<double>(hidden_size));
    weights_hidden_output.resize(hidden_size,
std::vector<double>(output_size));
    bias_hidden.resize(hidden_size);
    bias_output.resize(output_size);

    std::normal_distribution<double> xavier_dist(0.0, sqrt(1.0 /
input_size));
    for (auto& row : weights_input_hidden) {
        for (auto& weight : row) {
            weight = xavier_dist(generator);
        }
    }

    for (auto& bias : bias_hidden) {
        bias = distribution(generator);
    }

    for (auto& bias : bias_output) {
        bias = distribution(generator);
    }
}

// 前向传播函数：计算网络的输出
std::vector<double> BPNeuralNetwork::forward(const std::vector<double>&
input) {
    this->input = input;

    // 计算隐藏层输出
```

```
    hidden_output.resize(bias_hidden.size());
    for (size_t i = 0; i < hidden_output.size(); ++i) {
        double sum = bias_hidden[i];
        for (size_t j = 0; j < input.size(); ++j) {
            sum += input[j] * weights_input_hidden[j][i];
        }
       hidden_output[i] = sigmoid(sum);
       // hidden_output[i] = relu(sum);  // 使用 ReLU 代替 Sigmoid

    }

    // 计算最终输出
    final_output.resize(bias_output.size());
    for (size_t i = 0; i < final_output.size(); ++i) {
        double sum = bias_output[i];
        for (size_t j = 0; j < hidden_output.size(); ++j) {
            sum += hidden_output[j] * weights_hidden_output[j][i];
        }
        final_output[i] = sigmoid(sum);
    }

    return final_output;
}

// 反向传播函数：根据目标值计算误差并更新权重
void BPNeuralNetwork::backward(const std::vector<double>& target, double
class_weight) {
    // 计算输出层误差
    std::vector<double> output_error(final_output.size());
    for (size_t i = 0; i < final_output.size(); ++i) {
        output_error[i] = (target[i] - final_output[i]) *
sigmoid_derivative(final_output[i]) * class_weight; // 应用类别权重
    }

    // 计算隐藏层误差
    std::vector<double> hidden_error(hidden_output.size(), 0.0);
    for (size_t i = 0; i < hidden_output.size(); ++i) {
        for (size_t j = 0; j < output_error.size(); ++j) {
            hidden_error[i] += output_error[j] *
weights_hidden_output[i][j];
        }
        hidden_error[i] *= sigmoid_derivative(hidden_output[i]);
    }

    // 更新权重和偏置（隐藏层到输出层）
    for (size_t i = 0; i < weights_hidden_output.size(); ++i) {
        for (size_t j = 0; j < weights_hidden_output[i].size(); ++j) {
            weights_hidden_output[i][j] += learning_rate *
hidden_output[i] * output_error[j];
        }
    }
    for (size_t i = 0; i < bias_output.size(); ++i) {
        bias_output[i] += learning_rate * output_error[i];
```

```
    }

    // 更新权重和偏置（输入层到隐藏层）
    for (size_t i = 0; i < weights_input_hidden.size(); ++i) {
        for (size_t j = 0; j < weights_input_hidden[i].size(); ++j) {
            weights_input_hidden[i][j] += learning_rate * input[i] *
hidden_error[j];
        }
    }
    for (size_t i = 0; i < bias_hidden.size(); ++i) {
        bias_hidden[i] += learning_rate * hidden_error[i];
    }
}


// Sigmoid 激活函数
double BPNeuralNetwork::sigmoid(double x) {
    return 1.0 / (1.0 + std::exp(-x));
}

// Sigmoid 激活函数的导数
double BPNeuralNetwork::sigmoid_derivative(double x) {
    return x * (1.0 - x);
}

double BPNeuralNetwork::relu(double x) {
    return (x > 0) ? x : 0;
}

double BPNeuralNetwork::relu_derivative(double x) {
    return (x > 0) ? 1 : 0;
}

// 保存模型参数到文件
void BPNeuralNetwork::saveModel(const std::string& filename) {
    std::ofstream ofs(filename);
    if (!ofs.is_open()) {
        std::cerr << "无法打开文件保存模型参数: " << filename << std::endl;
        return;
    }

    // 保存输入层到隐藏层的权重
    ofs << weights_input_hidden.size() << " " <<
weights_input_hidden[0].size() << std::endl;
    for (const auto& row : weights_input_hidden) {
        for (const auto& weight : row) {
            ofs << weight << " ";
        }
        ofs << std::endl;
    }

    // 保存隐藏层到输出层的权重
```

```
    ofs << weights_hidden_output.size() << " " <<
weights_hidden_output[0].size() << std::endl;
    for (const auto& row : weights_hidden_output) {
        for (const auto& weight : row) {
            ofs << weight << " ";
        }
        ofs << std::endl;
    }

    // 保存偏置
    ofs << bias_hidden.size() << std::endl;
    for (const auto& bias : bias_hidden) {
        ofs << bias << " ";
    }
    ofs << std::endl;

    ofs << bias_output.size() << std::endl;
    for (const auto& bias : bias_output) {
        ofs << bias << " ";
    }
    ofs << std::endl;

    ofs.close();
    std::cout << "models save to:" << filename << std::endl;
}

// 从文件加载模型参数
void BPNeuralNetwork::loadModel(const std::string& filename) {
    std::ifstream ifs(filename);
    if (!ifs.is_open()) {
        std::cerr << "无法打开文件加载模型参数: " << filename << std::endl;
        return;
    }

    // 加载输入层到隐藏层的权重
    int hidden_size, input_size;
    ifs >> input_size >> hidden_size;
    weights_input_hidden.resize(input_size,
std::vector<double>(hidden_size));
    for (auto& row : weights_input_hidden) {
        for (auto& weight : row) {
            ifs >> weight;
        }
    }

    // 加载隐藏层到输出层的权重
    int output_size;
    ifs >> hidden_size >> output_size;
    weights_hidden_output.resize(hidden_size,
std::vector<double>(output_size));
    for (auto& row : weights_hidden_output) {
        for (auto& weight : row) {
            ifs >> weight;
        }
```

```
    }

    // 加载偏置
    int hidden_bias_size;
    ifs >> hidden_bias_size;
    bias_hidden.resize(hidden_bias_size);
    for (auto& bias : bias_hidden) {
        ifs >> bias;
    }

    int output_bias_size;
    ifs >> output_bias_size;
    bias_output.resize(output_bias_size);
    for (auto& bias : bias_output) {
        ifs >> bias;
    }

    ifs.close();
    std::cout << "模型参数已从 " << filename << " 加载" << std::endl;
}

// 新增的设置学习率的函数实现
void BPNeuralNetwork::setLearningRate(double new_learning_rate) {
    learning_rate = new_learning_rate;
}
```

SVM

①main.cpp

```
    #include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include <sstream>
#include <algorithm>
#include <filesystem>
#include <locale>
#include <codecvt>
#include <random>

namespace fs = std::filesystem;

/*****************************数据过滤范围*********************************/
const double MIN_HEIGHT = 140.0;//身高
const double MAX_HEIGHT = 220.0;
const double MIN_WEIGHT = 30.0;//体重
const double MAX_WEIGHT = 150.0;
const double MIN_SHOE_SIZE = 30.0;//鞋码
const double MAX_SHOE_SIZE = 50.0;
const double MIN_50M_TIME = 5.0;//50m
const double MAX_50M_TIME = 15.0;
```

```
const double MIN_VITAL_CAPACITY = 1000.0;//肺活量
const double MAX_VITAL_CAPACITY = 8000.0;

/*****************************数据预处理*********************************/
// 检查数据是否在合理范围内
bool isValidData(const std::vector<std::string> &data) {
    try {
        // 检查是否有空白数据
        for (const auto &item : data) {
            if (item.empty()) {
                return false;
            }
        }

        double gender = std::stod(data[1]);
        double height = std::stod(data[3]);
        double weight = std::stod(data[4]);
        double shoeSize = std::stod(data[5]);
        double fiftyMeter = std::stod(data[6]);
        double vitalCapacity = std::stod(data[7]);

        return (gender == 0 || gender == 1) &&
               (height >= MIN_HEIGHT && height <= MAX_HEIGHT) &&
               (weight >= MIN_WEIGHT && weight <= MAX_WEIGHT) &&
               (shoeSize >= MIN_SHOE_SIZE && shoeSize <= MAX_SHOE_SIZE) &&
               (fiftyMeter >= MIN_50M_TIME && fiftyMeter <= MAX_50M_TIME)
&&
               (vitalCapacity >= MIN_VITAL_CAPACITY && vitalCapacity <=
MAX_VITAL_CAPACITY);
    } catch (const std::exception &e) {
        // 如果转换失败，数据无效
        return false;
    }
}

// 解析 CSV 行
std::vector<std::string> parseCSVLine(const std::string &line) {
    std::vector<std::string> result;
    std::stringstream ss(line);
    std::string item;

    while (std::getline(ss, item, ',')) {
        result.push_back(item);
    }
    return result;
}

// 打开文件并处理数据
bool processFile(const std::string& inputFilePath, const std::string&
outputFilePath, const std::string& invalidFilePath) {
    if (!fs::exists(inputFilePath)) {
        std::cerr << "错误：输入文件不存在：" << inputFilePath << std::endl;
        return false;
```

```
    }

    std::wifstream inputFile(inputFilePath);
    inputFile.imbue(std::locale(inputFile.getloc(), new
std::codecvt_utf8<wchar_t>));
    std::wofstream outputFile(outputFilePath, std::ios::out |
std::ios::binary);
    outputFile.imbue(std::locale(outputFile.getloc(), new
std::codecvt_utf8<wchar_t>));
    std::wofstream invalidFile(invalidFilePath, std::ios::out |
std::ios::binary);
    invalidFile.imbue(std::locale(invalidFile.getloc(), new
std::codecvt_utf8<wchar_t>));

    if (!inputFile.is_open()) {
        std::cerr << "输入文件正被使用" << inputFilePath << std::endl;
        return false;
    }

    if (!outputFile.is_open()) {
        std::cerr << "无法创建输出文件" << outputFilePath << std::endl;
        return false;
    }

    if (!invalidFile.is_open()) {
        std::cerr << "无法创建无效数据文件" << invalidFilePath << std::endl;
        return false;
    }

    std::wstring line;
    bool isFirstLine = true;

    while (std::getline(inputFile, line)) {
        // 写入标题行
        if (isFirstLine) {
            outputFile << line << std::endl;
            invalidFile << line << std::endl;
            isFirstLine = false;
            continue;
        }

        // 解析行数据
        std::wstring_convert<std::codecvt_utf8<wchar_t>> converter;
        std::string utf8Line = converter.to_bytes(line);
        std::vector<std::string> data = parseCSVLine(utf8Line);

        // 检查数据完整性和有效性
        if (data.size() < 8) {
            invalidFile << line << std::endl; // 跳过不完整的行并记录
            continue;
        }

        // 检查是否是有效的数据
```

```
        if (isValidData(data)) {
            outputFile << line << std::endl;
        } else {
            invalidFile << line << std::endl; // 记录无效的数据
        }
    }

    inputFile.close();
    outputFile.close();
    invalidFile.close();

    std::cout << "数据过滤结果保存在 " << outputFilePath << std::endl;
    std::cout << "无效数据保存在 " << invalidFilePath << std::endl;
    return true;
}

/*****************************数据集划分*********************************/
// 函数：随机划分 CSV 文件为训练集和测试集，并存到文件中
void splitCSVFile(const std::string& inputFilePath, const std::string&
trainFilePath, const std::string& testFilePath, double train_ratio = 0.7)
{
    std::ifstream inputFile(inputFilePath);
    if (!inputFile.is_open()) {
        std::cerr << "无法打开输入文件: " << inputFilePath << std::endl;
        return;
    }

    std::vector<std::string> lines;
    std::string line;
    bool isFirstLine = true;
    std::string header;

    // 读取 CSV 所有行
    while (std::getline(inputFile, line)) {
        if (isFirstLine) {
            header = line;
            isFirstLine = false;
            continue;
        }
        lines.push_back(line);
    }
    inputFile.close();

    // 随机打乱数据
    std::shuffle(lines.begin(), lines.end(),
std::default_random_engine(std::random_device{}()));

    // 划分训练集和测试集
    size_t train_size = static_cast<size_t>(lines.size() * train_ratio);
    std::vector<std::string> train_lines(lines.begin(), lines.begin() +
train_size);
    std::vector<std::string> test_lines(lines.begin() + train_size,
lines.end());
```

```
    // 保存训练集到文件
    std::ofstream trainFile(trainFilePath);
    if (trainFile.is_open()) {
        trainFile << header << std::endl;
        for (const auto& train_line : train_lines) {
            trainFile << train_line << std::endl;
        }
        trainFile.close();
    } else {
        std::cerr << "无法创建训练集文件: " << trainFilePath << std::endl;
    }

    // 保存测试集到文件
    std::ofstream testFile(testFilePath);
    if (testFile.is_open()) {
        testFile << header << std::endl;
        for (const auto& test_line : test_lines) {
            testFile << test_line << std::endl;
        }
        testFile.close();
    } else {
        std::cerr << "无法创建测试集文件: " << testFilePath << std::endl;
    }
}

// 从 CSV 文件中读取训练数据集
bool loadData(const std::string& filePath,
std::vector<std::vector<double>>& X, std::vector<std::vector<double>>& y)
{
    std::ifstream inputFile(filePath);
    if (!inputFile.is_open()) {
        std::cerr << "无法打开文件 " << filePath << std::endl;
        return false;
    }

    std::string line;
    bool isFirstLine = true;

    // 读取文件内容并解析
    while (std::getline(inputFile, line)) {
        if (isFirstLine) {
            // 跳过标题行
            isFirstLine = false;
            continue;
        }

        std::vector<std::string> data = parseCSVLine(line);
        if (data.size() >= 8) {
            // 提取特征：身高、体重、鞋码、50m 成绩
            std::vector<double> features = {
                    std::stod(data[3]),  // 身高
                    std::stod(data[4]),  // 体重
```

```
                std::stod(data[5]),  // 鞋码
                std::stod(data[6])   // 50m 成绩
            };
            X.push_back(features);

            // 提取目标输出：性别 (0 或 1)
            y.push_back({std::stod(data[1])});
        }
    }
    inputFile.close();
    return true;
}

void printConfusionMatrix(int tp, int tn, int fp, int fn) {
    std::cout << "混淆矩阵: " << std::endl;
    std::cout << "TP: " << tp << ", FN: " << fn << std::endl; // 真阳性与假阴性
    std::cout << "FP: " << fp << ", TN: " << tn << std::endl; // 假阳性与真阴性
}

/*****************************调试用*********************************/
// 打印 X 和 Y 向量
void printData(const std::vector<std::vector<double>>& X, const
std::vector<std::vector<double>>& y) {
    std::cout << "X Data:" << std::endl;
    for (const auto& row : X) {
        for (const auto& val : row) {
            std::cout << val << " ";
        }
        std::cout << std::endl;
    }

    std::cout << "Y Data:" << std::endl;
    for (const auto& val : y) {
        std::cout << val[0] << std::endl; // 假设 y 是一个二元向量
    }
}

int main() {
    /*****************************路径*********************************/
    std::string inputFilePath =
"E:/GitHub/AI_VI/2.BPSVM/SVM/data/data.csv";//源数据集路径
    std::string outputFilePath =
"E:/GitHub/AI_VI/2.BPSVM/SVM/data/filtered_data.csv";//过滤后的数据集路径
    std::string invalidFilePath =
"E:/GitHub/AI_VI/2.BPSVM/SVM/data/invalid_data.csv";//无效数据路径

    std::string trainFilePath =
"E:/GitHub/AI_VI/2.BPSVM/SVM/data/train.csv";//训练集路径
    std::string testFilePath =
"E:/GitHub/AI_VI/2.BPSVM/SVM/data/test.csv";//测试集路径
```

```
    std::string misclassifiedFilePath =
"E:/GitHub/AI_VI/2.BPSVM/SVM/data/result/misclassified_samples.csv";//模型
的错分样本

    std::string modelFilePath =
"E:/GitHub/AI_VI/2.BPSVM/SVM/data/result/models.txt";//模型参数
    std::string matFilePath =
"E:/GitHub/AI_VI/2.BPSVM/SVM/data/mat.csv";//评估结果的输出矩阵

    //1.调用函数预处理文件
    std::cout << "******************数据集预处理****************** " <<
std::endl;
    if (!processFile(inputFilePath, outputFilePath, invalidFilePath)) {
        std::cerr << "文件处理失败。" << std::endl;
        return 1;
    }
    else{
        std::cout << "数据集预处理成功！" << std::endl;
    }

    //2.划分数据集
    // 将输入文件随机划分为训练集和测试集，比例为7:3
    splitCSVFile(outputFilePath, trainFilePath, testFilePath);
    std::cout << "按照 7:3 比例随机划分为训练集和测试集保存到文件中。" <<
std::endl;

    //2.加载数据集
    // 构建训练数据集，读取 train.csv 文件
    std::vector<std::vector<double>> X;
    std::vector<std::vector<double>> Y;
    if (!loadData(trainFilePath, X, Y)) {
        return 1;
    }
    //printData(X,Y);
    // 构造测试数据集，读取 test.csv 文件
    std::vector<std::vector<double>> X_test;
    std::vector<std::vector<double>> Y_test;
    if (!loadData(testFilePath, X_test, Y_test)) {
        return 1;
    }
    std::cout << "数据集加载成功！" << std::endl;
    //printData(X_test,Y_test);


    return 0;
}
```

②SVM.py(数据预处理)

```
    import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, cross_val_predict
from sklearn.svm import SVC
```

```
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve
from sklearn.preprocessing import LabelEncoder, StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
import os

# 设置字体以支持中文
plt.rcParams['font.sans-serif'] = ['SimHei']  # 设置字体为黑体
plt.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

# 数据过滤范围
MIN_HEIGHT = 140.0  # cm
MAX_HEIGHT = 220.0
MIN_WEIGHT = 30.0   # kg
MAX_WEIGHT = 150.0
MIN_SHOE_SIZE = 30.0
MAX_SHOE_SIZE = 50.0
MIN_50M_TIME = 5.0  # seconds
MAX_50M_TIME = 15.0

# 训练编号的全局变量
training_id = 1
num = 1  # 训练总次数

# 1. 数据预处理函数
def preprocess_data(input_file, output_file, invalid_file):
    # 读取 CSV 文件，指定中文列名
    df = pd.read_csv(input_file)

    # 尝试将相关列转换为 float 类型，处理异常值
    try:
        df['身高(cm)'] = pd.to_numeric(df['身高(cm)'], errors='coerce')
        df['体重(kg)'] = pd.to_numeric(df['体重(kg)'], errors='coerce')
        df['鞋码'] = pd.to_numeric(df['鞋码'], errors='coerce')
        df['50 米成绩'] = pd.to_numeric(df['50 米成绩'], errors='coerce')
        df['性别 男 1 女 0'] = pd.to_numeric(df['性别 男 1 女 0'],
errors='coerce')
    except KeyError:
        print("确保 CSV 文件中包含正确的列名：身高(cm)，体重(kg)，鞋码，50 米成
绩，性别 男 1 女 0")
        return

    # 只保留性别列中值为 0 或 1 的样本，确保标签是二分类
    df = df[df['性别 男 1 女 0'].isin([0, 1])]

    # 检查并过滤掉无效数据
    valid_data = df[
        (df['身高(cm)'] >= MIN_HEIGHT) & (df['身高(cm)'] <= MAX_HEIGHT) &
        (df['体重(kg)'] >= MIN_WEIGHT) & (df['体重(kg)'] <= MAX_WEIGHT) &
        (df['鞋码'] >= MIN_SHOE_SIZE) & (df['鞋码'] <= MAX_SHOE_SIZE) &
        (df['50 米成绩'] >= MIN_50M_TIME) & (df['50 米成绩'] <=
MAX_50M_TIME)
```

```
    ]

    # 无效数据：使用`isna()`检查哪些数据被转换为NaN
    invalid_data = df[~df.index.isin(valid_data.index)]

    # 保存数据到对应文件
    valid_data.to_csv(output_file, index=False)
    invalid_data.to_csv(invalid_file, index=False)

    print(f"数据过滤结果保存在 {output_file}")
    print(f"无效数据保存在 {invalid_file}")


# 2. 使用交叉验证评估SVM分类器性能
def train_and_evaluate_svm_with_cv(filtered_data_file):
    global training_id  # 声明全局变量，以便更新训练编号

    # 读取数据
    df = pd.read_csv(filtered_data_file)

    # 提取特征和标签
    X = df[['身高(cm)', '体重(kg)', '鞋码', '50米成绩']]
    y = df['性别 男1女0']

    # 使用LabelEncoder进行编码（确保y只包含0和1）
    label_encoder = LabelEncoder()
    y = label_encoder.fit_transform(y)

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

    # 使用StratifiedKFold进行分层交叉验证
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

    # 初始化不同核函数的SVM模型并进行评估
    kernels = ['linear', 'poly', 'rbf']
    results = []  # 用于存储所有核函数的性能指标

    # 创建vs目录用于保存汇总评估结果
    vs_dir = os.path.join("result", "vs")
    if not os.path.exists(vs_dir):
        os.makedirs(vs_dir)

    # 汇总评估结果的CSV文件路径
    summary_results_csv_path = os.path.join(vs_dir, "summary_results.csv")

    # 如果CSV文件不存在，创建文件并添加表头
    if not os.path.exists(summary_results_csv_path):
        df_headers = pd.DataFrame(
            columns=["Training_ID", "Kernel", "Accuracy", "Sensitivity",
"Specificity", "AUC", "TP", "FP", "TN", "FN"])
```

```
        df_headers.to_csv(summary_results_csv_path, index=False)

    for kernel in kernels:
        print(f"正在训练和评估核函数: {kernel}")

        # 为当前核函数创建单独的子目录
        kernel_result_dir = os.path.join("result", kernel)
        if not os.path.exists(kernel_result_dir):
            os.makedirs(kernel_result_dir)

        # 初始化 SVM 分类器
        clf = SVC(kernel=kernel, C=1.0, gamma='scale', probability=True)

        # 交叉验证预测
        y_pred = cross_val_predict(clf, X, y, cv=skf, method='predict')
        y_pred_proba = cross_val_predict(clf, X, y, cv=skf,
method='predict_proba')[:, 1]

        # 计算性能指标
        cm = confusion_matrix(y, y_pred)
        tn, fp, fn, tp = cm.ravel()

        # 灵敏度（Sensitivity, SE）
        sensitivity = tp / (tp + fn)
        # 特异度（Specificity, SP）
        specificity = tn / (tn + fp)
        # 准确率（Accuracy, ACC）
        accuracy = (tp + tn) / (tp + tn + fp + fn)
        # 曲线下面积（AUC）
        auc_score = roc_auc_score(y, y_pred_proba)

        # 保存当前核函数的指标到results 中
        results.append({
            'kernel': kernel,
            'sensitivity': sensitivity,
            'specificity': specificity,
            'accuracy': accuracy,
            'auc': auc_score
        })

        # 输出性能指标
        print(f"***************{kernel}性能指标**********************")
        print("混淆矩阵:")
        print(cm)
        print(f"灵敏度（Sensitivity, SE）: {sensitivity:.4f}")
        print(f"特异度（Specificity, SP）: {specificity:.4f}")
        print(f"准确率（Accuracy, ACC）: {accuracy:.4f}")
        print(f"曲线下面积（AUC）: {auc_score:.4f}")

        # 保存各核函数的性能指标到CSV 文件中，增加 "训练编号" 字段
        results_dict = {
            "Training_ID": training_id,
```

```
            "Kernel": kernel,
            "Accuracy": accuracy,
            "Sensitivity": sensitivity,
            "Specificity": specificity,
            "AUC": auc_score,
            "TP": tp,
            "FP": fp,
            "TN": tn,
            "FN": fn,
        }
        results_df = pd.DataFrame([results_dict])
        results_df.to_csv(summary_results_csv_path, mode='a', index=False,
header=False)

        print(f"核函数 {kernel} 的性能指标汇总保存到:
{summary_results_csv_path}")

        # 如果是最后一次训练，保存可视化结果
        if training_id == num:
            print(f"保存第 {training_id} 次训练的可视化结果 - 核函数:
{kernel}")

            # 保存模型参数到文本文件（每个核函数独立）
            model_params_path = os.path.join(kernel_result_dir,
"model_params.txt")
            with open(model_params_path, "w") as file:
                file.write(f"kernel: {kernel}\n")
                file.write(f"params: {clf.get_params()}\n")

            print(f"模型参数保存到: {model_params_path}")

            # 可视化混淆矩阵并保存
            plt.figure(figsize=(6, 5))
            sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
                        xticklabels=label_encoder.classes_,
yticklabels=label_encoder.classes_)
            plt.xlabel('预测值')
            plt.ylabel('真实值')
            plt.title(f'混淆矩阵 - 核函数: {kernel}')
            plt.tight_layout()
            confusion_matrix_path = os.path.join(kernel_result_dir,
"confusion_matrix.png")
            plt.savefig(confusion_matrix_path)
            plt.close()  # 关闭图表，防止显示

            # 可视化 ROC 曲线并保存
            fpr, tpr, thresholds = roc_curve(y, y_pred_proba)  # 获取假阳性
率、真阳性率和阈值数据
            plt.figure(figsize=(8, 6))
            plt.plot(fpr, tpr, color='blue', lw=2, label=f'AUC =
{auc_score:.4f}')
            plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
            plt.xlabel('假阳性率 (False Positive Rate)')
```

```
            plt.ylabel('真阳性率 (True Positive Rate)')
            plt.title(f'ROC 曲线 - 核函数: {kernel}')
            plt.legend(loc="lower right")
            plt.grid()
            plt.tight_layout()
            roc_curve_path = os.path.join(kernel_result_dir,
"roc_curve.png")
            plt.savefig(roc_curve_path)
            plt.close()  # 关闭图表，防止显示

            # 保存ROC 曲线数据到CSV 文件
            roc_data_path = os.path.join(kernel_result_dir,
f"{kernel}_roc_curve_data.csv")
            roc_data = pd.DataFrame({
                'FPR': fpr,  # 假阳性率
                'TPR': tpr,  # 真阳性率
                'Thresholds': thresholds  # 阈值
            })
            roc_data.to_csv(roc_data_path, index=False)
            print(f"ROC 曲线数据保存到: {roc_data_path}")

            # 可视化灵敏度、特异度和准确率并保存
            metrics = {'灵敏度 (SE)': sensitivity, '特异度 (SP)':
specificity, '准确率 (ACC)': accuracy}
            plt.figure(figsize=(8, 5))
            plt.bar(metrics.keys(), metrics.values(), color=['skyblue',
'orange', 'green'])
            plt.ylabel('得分')
            plt.ylim(0, 1)
            plt.title(f'性能指标 - 核函数: {kernel}')
            plt.grid(axis='y', linestyle='--')
            plt.tight_layout()
            metrics_path = os.path.join(kernel_result_dir, "metrics.png")
            plt.savefig(metrics_path)
            plt.close()  # 关闭图表，防止显示

    # 完成一轮训练后，递增训练编号
    training_id += 1


# 3. 计算平均值并绘图
def calculate_average_and_plot():
    # 读取所有训练的汇总结果
    summary_results_csv_path = os.path.join("result", "vs",
"summary_results.csv")
    df = pd.read_csv(summary_results_csv_path)

    # 计算每个核函数的平均值
    avg_results = df.groupby('Kernel').mean().reset_index()

    # 保存平均值结果到一个新的 CSV 文件
```

```
    average_results_csv_path = os.path.join("result", "vs",
"average_results.csv")
    avg_results.to_csv(average_results_csv_path, index=False)
    print(f"正在保存前 {training_id - 1} 次训练的平均结果到:
{average_results_csv_path}")

    # 绘制平均性能指标图
    metrics_to_plot = ['Accuracy', 'Sensitivity', 'Specificity', 'AUC']
    for metric in metrics_to_plot:
        plt.figure(figsize=(12, 8))
        values = avg_results[metric].values
        kernels = avg_results['Kernel'].values
        bars = plt.bar(kernels, values, color=['#4E79A7', '#F28E2C',
'#76B7B2'], alpha=0.85)
        plt.ylabel('得分', fontsize=14)
        plt.ylim(0, 1)
        plt.title(f'{metric} 平均值对比', fontsize=18)
        plt.grid(axis='y', linestyle='--')

        # 在每个柱状图上显示具体数值
        for bar in bars:
            plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() -
0.05, f'{bar.get_height():.4f}',
                     ha='center', va='bottom', fontsize=12, color='black',
weight='bold')

        plt.tight_layout()
        comparison_path = os.path.join("result", "vs",
f"{metric}_average_comparison.png")
        plt.savefig(comparison_path)
        plt.close()

# 4. 特征组合投影函数，针对三种核函数进行可视化（实现特征两两组合）
def feature_combination_projection(filtered_data_file):
    # 读取数据
    df = pd.read_csv(filtered_data_file)

    # 提取特征和标签
    X = df[['身高(cm)', '体重(kg)', '鞋码', '50 米成绩']]
    y = df['性别 男 1 女 0']

    # 使用 LabelEncoder 进行编码（确保 y 只包含 0 和 1）
    label_encoder = LabelEncoder()
    y = label_encoder.fit_transform(y)

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

    # 核函数列表
    kernels = ['linear', 'poly', 'rbf']
    feature_names = ['身高', '体重', '鞋码', '50 米成绩']
```

```
    # 获取所有两两特征组合的索引
    feature_combinations = [(i, j) for i in range(len(feature_names)) for
j in range(i + 1, len(feature_names))]

    # 对每个核函数进行投影和可视化
    for kernel in kernels:
        for (i, j) in feature_combinations:
            # 使用两个特征进行组合投影
            X_selected = X[:, [i, j]]

            # 创建子文件夹保存每个核函数的结果
            touying_file = os.path.join("result/keshihua", kernel)
            if not os.path.exists(touying_file):
                os.makedirs(touying_file)

            # 使用 SVM 重新训练模型
            clf = SVC(kernel=kernel, C=1.0, gamma='scale',
probability=True)
            clf.fit(X_selected, y)

            # 创建网格以绘制决策边界
            x_min, x_max = X_selected[:, 0].min() - 1, X_selected[:,
0].max() + 1
            y_min, y_max = X_selected[:, 1].min() - 1, X_selected[:,
1].max() + 1
            xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                                 np.arange(y_min, y_max, 0.01))

            # 使用训练好的模型进行预测
            Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
            Z = Z.reshape(xx.shape)

            # 绘制决策边界和支持向量
            plt.figure(figsize=(10, 6))
            plt.contourf(xx, yy, Z, alpha=0.8)
            plt.scatter(X_selected[:, 0], X_selected[:, 1], c=y,
edgecolors='k', marker='o', cmap=plt.cm.Paired)
            plt.scatter(clf.support_vectors_[:, 0],
clf.support_vectors_[:, 1], s=100,
                        facecolors='none', edgecolors='k', linewidth=1.5)  #
支持向量
            plt.xlabel(f'{feature_names[i]} (标准化)')
            plt.ylabel(f'{feature_names[j]} (标准化)')
            plt.title(f'特征组合投影 - 核函数: {kernel}')
            plt.tight_layout()

            # 保存特征组合投影图到对应子文件夹
            projection_path = os.path.join(touying_file,
f"{kernel}_feature_combination_{feature_names[i]}_{feature_names[j]}.png"
)
            plt.savefig(projection_path)
            plt.close()  # 关闭图表，防止显示
            print(f"特征组合投影图保存到: {projection_path}")
```

```
            # 绘制性能指标图
            metrics = ['准确率', '灵敏度', '特异度', 'AUC']
            # 使用交叉验证得到的预测结果计算性能
            y_pred = cross_val_predict(clf, X_selected, y, cv=5)
            cm = confusion_matrix(y, y_pred)
            tn, fp, fn, tp = cm.ravel()

            accuracy = (tp + tn) / (tp + tn + fp + fn)
            sensitivity = tp / (tp + fn)
            specificity = tn / (tn + fp)
            auc_score = roc_auc_score(y, y_pred)

            performance_values = [accuracy, sensitivity, specificity,
auc_score]

            plt.figure(figsize=(8, 5))
            bars = plt.bar(metrics, performance_values, color=['#4E79A7',
'#F28E2C', '#76B7B2', '#E15759'], alpha=0.85)
            plt.ylabel('得分')
            plt.ylim(0, 1)
            plt.title(f'性能指标 - 核函数: {kernel} ({feature_names[i]} &
{feature_names[j]})')
            plt.grid(axis='y', linestyle='--')

            # 在每个柱状图上显示具体数值
            for bar in bars:
                plt.text(bar.get_x() + bar.get_width() / 2,
bar.get_height() - 0.05, f'{bar.get_height():.4f}',
                         ha='center', va='bottom', fontsize=12,
color='black', weight='bold')

            performance_path = os.path.join(touying_file,
f"{kernel}_performance_metrics_{feature_names[i]}_{feature_names[j]}.png"
)
            plt.savefig(performance_path)
            plt.close()
            print(f"性能指标图保存到: {performance_path}")



# 主程序
if __name__ == "__main__":
    # 数据文件路径
    input_file = "data/data.csv"
    filtered_data_file = "data/filtered_data.csv"
    invalid_data_file = "data/invalid_data.csv"

    # 如果输出目录不存在，创建它
    if not os.path.exists("data"):
        os.makedirs("data")

    # 1. 数据预处理
```

```
    print(f"********************数据预处理********************")
    preprocess_data(input_file, filtered_data_file, invalid_data_file)

    # 2. SVM 模型训练与交叉验证评估，执行 num 次
    print(f"********************模型训练与评估********************")
    for i in range(num):
        print(f"********************第 {i + 1} 次训练********************")
        train_and_evaluate_svm_with_cv(filtered_data_file)

    # 3. 计算平均值并绘图
    print(f"********************计算平均值并绘制图表********************")
    calculate_average_and_plot()

    # 4. 特征组合投影
    print(f"********************特征组合投影********************")
    #feature_combination_projection(filtered_data_file)
```

曲线绘制

①main.py（ROC 曲线绘制）

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

# 设置字体，确保支持中文
plt.rcParams['font.sans-serif'] = ['SimHei']  # 使用黑体字体，支持中文
plt.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

# 定义目录路径（相对路径）
data_dir = './data/'
result_dir = './result/'

# 如果 result 目录不存在，则创建它
if not os.path.exists(result_dir):
    os.makedirs(result_dir)

# 定义一个函数，用于绘制并保存所有 ROC 曲线到一张图
def plot_and_save_all_roc():
    # 加载 CSV 文件到 DataFrame
    linear_data = pd.read_csv(os.path.join(data_dir,
'linear_roc_curve_data.csv'))
    poly_data = pd.read_csv(os.path.join(data_dir,
'poly_roc_curve_data.csv'))
    rbf_data = pd.read_csv(os.path.join(data_dir,
'rbf_roc_curve_data.csv'))
    roc_values = pd.read_csv(os.path.join(data_dir, 'roc_values.csv'))

    # 绘制线性核的 ROC 曲线
    plt.plot(linear_data['FPR'], linear_data['TPR'], label='linear 核')
```

```
    # 绘制多项式核的 ROC 曲线
    plt.plot(poly_data['FPR'], poly_data['TPR'], label='poly 核')

    # 绘制 RBF 核的 ROC 曲线
    plt.plot(rbf_data['FPR'], rbf_data['TPR'], label='rbf 核')

    # 绘制额外的 ROC 数据（如 BP）
    plt.plot(roc_values['FPR'], roc_values['TPR'], label='BP')

    # 设置标签和标题
    plt.xlabel('假阳性率 (FPR)')
    plt.ylabel('真正率 (TPR)')
    plt.title('不同模型的 ROC 曲线')

    # 显示图例
    plt.legend()

    # 保存图片到 result 目录，格式可以是 'png', 'jpg', 'pdf' 等
    plt.savefig(os.path.join(result_dir, 'roc_curve.png'), dpi=300,
bbox_inches='tight')

    # 关闭当前图形，以便后续可以重新绘制
    plt.close()


# 定义函数来绘制和保存单独的 ROC 曲线
def plot_and_save_roc(data_file, label, save_path):
    """
    读取指定的 CSV 文件并绘制 ROC 曲线，然后保存图片。

    参数:
    - data_file: CSV 文件的路径
    - label: 曲线的标签（如线性核、多项式核等）
    - save_path: 保存图片的路径（包括文件名和格式）
    """
    # 读取 CSV 文件
    data = pd.read_csv(data_file)

    # 绘制 ROC 曲线
    plt.plot(data['FPR'], data['TPR'], label=label)

    # 设置标签和标题
    plt.xlabel('假阳性率 (FPR)')
    plt.ylabel('真正率 (TPR)')
    plt.title(f'{label}的 ROC 曲线')

    # 显示图例
    plt.legend()

    # 保存图片
```

```
    plt.savefig(save_path, dpi=300, bbox_inches='tight')

    # 关闭当前图形，以便后续绘图不被干扰
    plt.close()

# 调用函数进行绘制并保存所有曲线到一张图
plot_and_save_all_roc()

# 分别绘制并保存每个单独的曲线到 result 目录
plot_and_save_roc(os.path.join(data_dir, 'linear_roc_curve_data.csv'), '
线性核', os.path.join(result_dir, 'linear_roc_curve.png'))
plot_and_save_roc(os.path.join(data_dir, 'poly_roc_curve_data.csv'), '多项
式核', os.path.join(result_dir, 'poly_roc_curve.png'))
plot_and_save_roc(os.path.join(data_dir, 'rbf_roc_curve_data.csv'), 'RBF
核', os.path.join(result_dir, 'rbf_roc_curve.png'))
plot_and_save_roc(os.path.join(data_dir, 'roc_values.csv'), 'BP',
os.path.join(result_dir, 'bp_roc_curve.png'))
```

②main.py（数据可视化）

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

# 创建结果保存目录
result_dir = './result'
os.makedirs(result_dir, exist_ok=True)

result_M_dir = './result/Model_only'
os.makedirs(result_M_dir, exist_ok=True)

result_Compare_dir = './result/Model_Compare'
os.makedirs(result_Compare_dir, exist_ok=True)

# 读取表格文件（假设文件名为 'model_results.xlsx'）
# 如果是 CSV 文件，请将文件名和相应方法改为 read_csv
data_bp = pd.read_csv('./data/bp.csv')
data_linear = pd.read_csv('./data/linear.csv')
data_rbf = pd.read_csv('./data/rbf.csv')
data_poly = pd.read_csv('./data/poly.csv')

# 计算 Precision 和 Recall
data_bp['Precision'] = data_bp['TP'] / (data_bp['TP'] + data_bp['FP'])
data_bp['Recall'] = data_bp['TP'] / (data_bp['TP'] + data_bp['FN'])

data_linear['Precision'] = data_linear['TP'] / (data_linear['TP'] +
data_linear['FP'])
data_linear['Recall'] = data_linear['TP'] / (data_linear['TP'] +
data_linear['FN'])
```

```
data_rbf['Precision'] = data_rbf['TP'] / (data_rbf['TP'] +
data_rbf['FP'])
data_rbf['Recall'] = data_rbf['TP'] / (data_rbf['TP'] + data_rbf['FN'])

data_poly['Precision'] = data_poly['TP'] / (data_poly['TP'] +
data_poly['FP'])
data_poly['Recall'] = data_poly['TP'] / (data_poly['TP'] +
data_poly['FN'])

# 查看数据是否读取正确
print(data_bp.head())
print(data_linear.head())
print(data_rbf.head())
print(data_poly.head())

# 设置 Seaborn 的样式
sns.set(style="whitegrid")


# 通用可视化函数
def visualize_model(data, model_name):
    plt.figure(figsize=(16, 10))

    # 子图 1: Accuracy, Sensitivity, Specificity 对比
    plt.subplot(2, 2, 1)
    sns.boxplot(data=data[['Accuracy', 'Sensitivity', 'Specificity']])
    plt.title(f'{model_name} Model Performance Metrics')
    plt.ylabel('Score')

    # 子图 2: AUC
    plt.subplot(2, 2, 2)
    sns.histplot(data['AUC'], kde=True, color='blue')
    plt.title(f'{model_name} Model AUC Distribution')
    plt.xlabel('AUC')
    plt.ylabel('Frequency')

    # 子图 3: 混淆矩阵中的 TP, FP, TN, FN
    plt.subplot(2, 2, 3)
    sample_data = data[['TP', 'FP', 'TN', 'FN']].iloc[::max(1, len(data)
// 20)]  # 仅绘制部分数据以减少拥挤
    sample_data.plot(kind='bar', stacked=True, ax=plt.gca(),
color=['skyblue', 'orange', 'green', 'red'])
    plt.title(f'{model_name} Model Confusion Matrix Values (Sampled)')
    plt.xlabel('Sampled Training Iterations')
    plt.ylabel('Count')
    plt.xticks(rotation=45, ha='right')

    # 子图 4: Precision and Recall
    plt.subplot(2, 2, 4)
    sns.boxplot(data=data[['Precision', 'Recall']])
    plt.title(f'{model_name} Model Precision and Recall')
    plt.ylabel('Score')
```

```
    plt.tight_layout()
    plt.savefig(os.path.join(result_M_dir,
f'{model_name}_performance.png'))
    plt.close()


# 可视化多个模型对比
def visualize_model_comparison(models_data, model_names):
    metrics = ['Accuracy', 'Sensitivity', 'Specificity', 'AUC',
'Precision', 'Recall']
    comparison_data = pd.DataFrame()

    for data, name in zip(models_data, model_names):
        model_metrics = data[metrics].mean()
        model_metrics['Model'] = name
        comparison_data = pd.concat([comparison_data,
pd.DataFrame([model_metrics])], ignore_index=True)

    plt.figure(figsize=(16, 10))

    # 子图 1: Accuracy, Sensitivity, Specificity 对比
    plt.subplot(2, 2, 1)
    sns.barplot(x='Model', y='value', hue='variable',
                data=pd.melt(comparison_data, id_vars=['Model'],
value_vars=['Accuracy', 'Sensitivity', 'Specificity']))
    plt.title('Model Comparison: Accuracy, Sensitivity, Specificity')
    plt.ylabel('Score')

    # 子图 2: AUC
    plt.subplot(2, 2, 2)
    sns.barplot(x='Model', y='AUC', data=comparison_data)
    plt.title('Model Comparison: AUC')
    plt.ylabel('AUC')

    # 子图 3: Precision
    plt.subplot(2, 2, 3)
    sns.barplot(x='Model', y='Precision', data=comparison_data)
    plt.title('Model Comparison: Precision')
    plt.ylabel('Precision')

    # 子图 4: Recall
    plt.subplot(2, 2, 4)
    sns.barplot(x='Model', y='Recall', data=comparison_data)
    plt.title('Model Comparison: Recall')
    plt.ylabel('Recall')

    plt.tight_layout()
    plt.savefig(os.path.join(result_dir, 'model_comparison.png'))
    plt.close()


# 可视化多个模型对比
def visualize_model_comparison1(models_data, model_names):
```

```
    metrics = ['Accuracy', 'Sensitivity', 'Specificity', 'AUC',
'Precision', 'Recall']
    comparison_data = pd.DataFrame()

    for data, name in zip(models_data, model_names):
        model_metrics = data[metrics].mean()
        model_metrics['Model'] = name
        comparison_data = pd.concat([comparison_data,
pd.DataFrame([model_metrics])], ignore_index=True)

    # 图 1: Accuracy, Sensitivity, Specificity 对比
    plt.figure(figsize=(12, 8))
    sns.barplot(x='Model', y='value', hue='variable',
                data=pd.melt(comparison_data, id_vars=['Model'],
value_vars=['Accuracy', 'Sensitivity', 'Specificity']))
    plt.title('Model Comparison: Accuracy, Sensitivity, Specificity')
    plt.ylabel('Score')
    plt.tight_layout()
    plt.savefig(os.path.join(result_Compare_dir,
'performance_metrics.png'))
    plt.close()

    # 图 2: AUC 对比
    plt.figure(figsize=(10, 6))
    sns.barplot(x='Model', y='AUC', data=comparison_data)
    plt.title('Model Comparison: AUC')
    plt.ylabel('AUC')
    plt.tight_layout()
    plt.savefig(os.path.join(result_Compare_dir, 'auc.png'))
    plt.close()

    # 图 3: Precision 对比
    plt.figure(figsize=(10, 6))
    sns.barplot(x='Model', y='Precision', data=comparison_data)
    plt.title('Model Comparison: Precision')
    plt.ylabel('Precision')
    plt.tight_layout()
    plt.savefig(os.path.join(result_Compare_dir, 'precision.png'))
    plt.close()

    # 图 4: Recall 对比
    plt.figure(figsize=(10, 6))
    sns.barplot(x='Model', y='Recall', data=comparison_data)
    plt.title('Model Comparison: Recall')
    plt.ylabel('Recall')
    plt.tight_layout()
    plt.savefig(os.path.join(result_Compare_dir, 'recall.png'))
    plt.close()

    # 图 1: 综合对比 (Accuracy, Sensitivity, Specificity, AUC, Precision,
Recall)
    plt.figure(figsize=(14, 8))
    sns.set_palette('Set2')
    sns.barplot(x='Model', y='value', hue='variable',
```

```
                data=pd.melt(comparison_data, id_vars=['Model'],
value_vars=metrics))
    plt.title('Model Comparison Across All Metrics')
    plt.ylabel('Score')
    plt.tight_layout()
    plt.savefig(os.path.join(result_Compare_dir, 'all.png'))
    plt.close()

# 调用可视化函数
visualize_model(data_bp, "BP")
visualize_model(data_linear, "SVM_linear")
visualize_model(data_rbf, "SVM_rbf")
visualize_model(data_poly, "SVM_poly")

# # 调用模型对比可视化函数
visualize_model_comparison([data_bp, data_linear, data_rbf, data_poly],
["BP", "SVM_linear", "SVM_rbf", "SVM_poly"])

# # 调用模型保存可视化函数
# visualize_model_comparison1([data_bp, data_linear, data_rbf,
data_poly], ["BP", "SVM_linear", "SVM_rbf", "SVM_poly"])
```