

**本科课程实践报告**

**(2022级)**

题 目：基于Mnist数据集的分类器训练与模型评估

学 院:\_\_\_计算机与人工智能学院\_\_\_\_\_\_\_\_\_\_\_

专　　业:\_\_\_人工智能\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

班　　级:\_\_\_22人工智能1班、2班\_\_\_\_\_\_\_\_\_\_\_

姓 名:\_\_\_李凯荣、邓镇宏、张洪森\_\_\_\_\_\_\_\_\_

学 号: \_\_\_22211360121、22211357205 、20211880141\_

指导老师:\_\_\_潘志勇\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

完成日期:\_\_\_2024.12.21\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* **文档修改历史**

| **版本号** | **版本日期** | **修改总结** | **修订人** |
| --- | --- | --- | --- |
| 1.0 | 2024.11.09 | 文档整体框架、SVM算法设计内容 | 李凯荣 |
| 2.0 | 2024.11.16 | 随机森林算法设计相关 | 李凯荣 |
| 3.0 | 2024.11.23 | 神经网络算法设计相关 | 李凯荣 |
| 4.0 | 2024.11.30 | 参考文献、附录 | 李凯荣 |
| 5.0 | 2024.12.09 | 模型改进、摘要、整体修订 | 李凯荣 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

基于Mnist数据集的分类器训练与模型评估

摘 要

随着信息技术的快速发展，手写数字识别成为广泛应用于金融、教育等多个领域的关键技术，在实际生活中，手写识别的需求愈发强烈。手写识别的挑战主要在于个体书写习惯的多样性和随意性，使得很多传统算法难以达到较高的准确率要求，为解决这个问题，本文基于MNIST数据集，研究并实现**支持向量机、随机森林、卷积神经网络**三种常见的分类算法，并评估和改进模型性能，探讨其在手写数字识别应用中的适用性和优势。

在**支持向量机模型**中，针对线性核函数、高斯核函数、多项式核函数，通过网格搜索优化超参数，最终得出**最优核函数为线性核函数**，在其最优超参数条件下，实现**测试集准确率98.82%**的效果。在**随机森林模型**中，通过多个决策树集成，在默认情况下可以达到**97.04%**的测试集准确率。通过对数据集的**欠采样、过采样**操作结果对比评估，得到可以通过欠采样操作，实现**节约模型训练时间**，且模型测试集**准确率保持在96.68%的低损失效果**。在**神经网络模型**中，卷积神经网络强大的非线性建模能力在手写数字识别中表现突出，最终测试集**准确率高达99.14%**。**最终得出结论，卷积神经网络在手写数字识别任务中，具有显著的优势。**

在其他内容方面，本文首先探求如何实现对模型训练效率的提升。通过**分箱法**可以实现在**20分箱大小**下，模型依旧**保持高准确率，且训练时间明显降低**的效果。而**PCA降维**处理数据会导致**模型准确率降低、训练时间增加，因此不适用**。接着基于已训练的模型，开发了一个**手写数字识别系统**，用户可以通过界面书写，实时获得识别结果，该系统的成功实现进一步证明上述算法在实际应用中具有**可行性。**

基于未来的展望，手写数字识别技术发展方向聚焦于**算法流程的优化**和**应用场景的扩展**，如更复杂的手写风格和实时性要求。与此同时，结合**多模态数据或更先进的深度学习模型**，有望进一步提高识别准确率，推动手写识别技术在多领域的广泛应用。

**关键词**：MNIST数据集、手写数字识别、支持向量机、随机森林、卷积神经网络

**Classifier Training and Model Evaluation Based on the MNIST Dataset**

**ABSTRACT**

With the rapid development of information technology, handwritten digit recognition has become a key technology widely applied in various fields such as finance and education. The demand for handwritten recognition in daily life is growing increasingly strong. The main challenge of handwritten recognition lies in the diversity and randomness of individual writing habits, making it difficult for many traditional algorithms to meet high accuracy requirements. To address this issue, this paper, based on the MNIST dataset, studies and implements three common classification algorithms: Support Vector Machines (SVM), Random Forests (RF), and Convolutional Neural Networks (CNN). The performance of these models is evaluated and improved, and their applicability and advantages in handwritten digit recognition are explored.

For the SVM model, hyperparameters for linear kernels, Gaussian kernels, and polynomial kernels are optimized using grid search, with the optimal kernel determined to be the linear kernel. Under its optimal hyperparameters, the test accuracy achieves 98.82%. For the Random Forest model, the integration of multiple decision trees achieves a test accuracy of 97.04% under default settings. By comparing the effects of undersampling and oversampling on the dataset, it is found that undersampling can reduce training time while maintaining a low test accuracy loss of 96.68%. For neural network models, CNNs demonstrate outstanding performance in handwritten digit recognition due to their powerful nonlinear modeling capabilities, achieving a final test accuracy as high as 99.14%. The conclusion is drawn that CNNs exhibit significant advantages in handwritten digit recognition tasks.

In other aspects, this paper first investigates how to improve model training efficiency. Using the binning method, it is shown that with a bin size of 20, the model can still maintain high accuracy while significantly reducing training time. In contrast, applying PCA for dimensionality reduction decreases model accuracy and increases training time, making it unsuitable. Subsequently, based on the trained models, a handwritten digit recognition system is developed, enabling users to write via an interface and receive real-time recognition results. The successful implementation of this system further demonstrates the feasibility of the aforementioned algorithms in practical applications.

Looking ahead, the development of handwritten digit recognition technology focuses on optimizing algorithm workflows and expanding application scenarios, such as accommodating more complex handwriting styles and real-time requirements. Additionally, combining multimodal data or leveraging more advanced deep learning models is expected to further improve recognition accuracy and promote the widespread application of handwritten recognition technology across multiple domains.

**Keywords:** MNIST dataset, handwritten digit recognition, Support Vector Machines, Random Forests, Convolutional Neural Networks

**目 录**

[题 目：基于Mnist数据集的分类器训练与模型评估 I](#_Toc184479249)

[基于Mnist数据集的分类器训练与模型评估 i](#_Toc184479250)

[摘 要 i](#_Toc184479251)

[第1章 绪 论 - 5 -](#_Toc184479252)

[1.1 选题背景 - 5 -](#_Toc184479253)

[1.1.1 概述 - 5 -](#_Toc184479254)

[1.1.2 数据集介绍 - 5 -](#_Toc184479255)

[第2章 分类器训练的具体过程 - 6 -](#_Toc184479256)

[2.1 数据处理 - 6 -](#_Toc184479257)

[2.2 支持向量机 - 6 -](#_Toc184479258)

[2.2.1 支持向量机数据处理 - 6 -](#_Toc184479259)

[2.2.2 支持向量机模型训练的具体过程 - 6 -](#_Toc184479260)

[2.3 随机森林 - 7 -](#_Toc184479261)

[2.3.1 随机森林算法简介 - 7 -](#_Toc184479262)

[2.3.2 适用原因 - 8 -](#_Toc184479263)

[2.3.3 随机森林模型训练的具体过程 - 8 -](#_Toc184479264)

[2.4 神经网络 - 9 -](#_Toc184479265)

[2.4.1 卷积神经网络 - 9 -](#_Toc184479266)

[2.4.2 卷积神经网络的训练过程 - 9 -](#_Toc184479267)

[第3章 结果评估与优化 - 11 -](#_Toc184479268)

[3.1 支持向量机 - 11 -](#_Toc184479269)

[3.1.1 支持向量机结果 - 11 -](#_Toc184479270)

[3.1.2 调参评估与改进 - 12 -](#_Toc184479271)

[3.2 随机森林 - 14 -](#_Toc184479272)

[3.2.1 随机森林训练结果 - 14 -](#_Toc184479273)

[3.2.2 改进方法 - 16 -](#_Toc184479274)

[3.2.3 改进结果评估 - 17 -](#_Toc184479275)

[3.3 神经网络 - 18 -](#_Toc184479276)

[3.3.1 神经网络训练结果 - 18 -](#_Toc184479277)

[3.4 结果汇总 - 19 -](#_Toc184479278)

[3.5 附加内容 - 20 -](#_Toc184479279)

[3.5.1 提高模型训练方法 - 20 -](#_Toc184479280)

[3.5.2 手写功能 - 21 -](#_Toc184479281)

[第4章 结论与展望 - 23 -](#_Toc184479282)

[4.1 结论 - 23 -](#_Toc184479283)

[4.2 展望 - 23 -](#_Toc184479284)

[参 考 文 献 - 25 -](#_Toc184479285)

[附 录（小二号黑体居中） - 26 -](#_Toc184479286)

[SVM算法实现代码 - 26 -](#_Toc184479287)

[随机森林欠拟合过拟合可视化展现代码 - 30 -](#_Toc184479288)

[CNN卷积神经网络代码： - 33 -](#_Toc184479289)

[LSTM长短期记忆神经网络代码 - 34 -](#_Toc184479290)

[随机森林最大特征数和最小样本数调参 - 35 -](#_Toc184479291)

[随机森林树的数量调参 - 37 -](#_Toc184479292)

[随机森林树深度调参 - 39 -](#_Toc184479293)

[致 谢 - 42 -](#_Toc184479294)

第1章 绪 论

**1.1 选题背景**

1.1.1 概述

随着近年来信息化技术高速发展，各行各业正加速实现信息化改革。信息技术对整个人类社会的影响力与日俱增，不仅为人们的生产和生活带来了极大的便利，也带来了新的挑战。

在越来越多行业倡导“无纸化”和“信息化”的办公和生活的大背景下，一个重要的应用场景就是通过计算机系统准确识别机器字库数字和手写数字，实现数据的电子信息化。这种需求在金融、教育、科学等多个领域都有很广泛的应用场景，非常有实用价值。

如何准确的识别手写数字就成了当前国内外相关研究的难点和热点问题之一。机器字库的数字因具有明确的规范写法，相对容易识别，而手写数字因人的习惯不同和写字的随意性，更加难以准确识别。传统的方法难以准确的识别出手写数字多变的风格，因此识别准确率较低。

目前对于手写数字进行的机器学习识别方法多种多样，本文选取三种较为常用的分类识别算法，通过训练分类器，比对分类器性能，以达到评估各个分类算法效果的目的。

1.1.2 数据集介绍

本文采用的手写数字数据来源MNIST手写数据集(The MNIST database of handwritten digits)，其数据来自于美国国家标准与技术研究所(National Institute of Standards and Technology，简称 NIST)，由来自 250名不同人的手写数字构成，能较全面的反映出各种手写字体的风格。

该数据集含有60,000个训练样本和10,000个测试样本，每个样本为28x28像素的灰度图像。数据集分为训练集和测试集，用于训练和评估机器学习模型。

MNIST数据集最初设计用于研究手写数字识别算法，其设计初衷是为了提供一个通用的基准，以促进算法之间的比较。在机器学习领域，MNIST常被用来评估学习算法的性能，特别是那些用于分类的算法。

# 第2章 分类器训练的具体过程

2.1 数据处理

MNIST数据集提供的每个样本为28×28的灰度值矩阵，数值为0~255，通过将数值标准化，能够减少权重初始值对模型训练的影响，使得模型更加的稳定，在各维度数据都相似的情况下，利于加快模型收敛，优化训练耗时，提高效率。

进行训练数据归一化处理，将数值处于0~255间的数据，统一归一化到0~1之间，其代表的含义与原来不变。通过归一化，能够减少数据不同尺度引起的偏差，特别是在特征取值范围差异较大的情况下。对于MNIST数据集，通过归一化处理，有助于在各类模型中达到更快收敛效果。

2.2 支持向量机

2.2.1 支持向量机数据处理

支持向量机算法依赖于特征之间的的距离与计算内积的结果。由于SVM算法的计算消耗较大，这里进一步采用二值化处理，更便于SVM算法实现对于本数据集的分类器训练效果。

对于训练集与测试集，分别计算它们各自的均值，比较特征值与均值，将大于均值的设置为1，小于均值的设置为0，实现对整个MNIST数据集的二值化处理，有助于减少噪声影响和波动，使得数据更加干净，且进一步降低计算复杂度。因为手写数字的识别更关注于轮廓、区域的有无，二值化后更利于形状提取和类似于模糊判断处理。

接着需要进行样本的一维展开。SVM模型训练需要输入的特征为一维向量，因此需要将图像的灰度特征值矩阵展平。以便于使用每个像素点作为特征，利用SVM模型训练实现分类。

2.2.2 支持向量机模型训练的具体过程

支持向量机（Support Vector Machine，简称SVM）是一种用于分类和回归分析的机器学习模型，尤其适合处理线性不可分问题。SVM的核心思想是通过找到一个最优的超平面来将数据点进行分类，使得分类的间隔最大化。

SVM在高维数据下通常表现良好，并且在小样本、非线性和高维的情况下有很强的泛化能力。它广泛应用于文本分类、图像识别、生物信息学等领域。其对高维数据的适应能力，尤其在特征空间维度较高时表现良好。它通过最大化分类间隔来提高泛化能力，减少过拟合风险。此外，SVM利用核函数能够有效处理非线性问题，使其适合复杂的分类任务。

在本文的python代码中的实现采用了sklearn库中自带的SVM分类器直接训练实现。但我们小组在实践过程中也进一步尝试了自己实现支持向量机算法的具体流程，因此在下文会一并补充说明。

由于识别手写字符是一个较为宽容的情况，因此本文采取软间隔形式的支持向量机算法实现对MNIST数据集的分类器训练。

支持向量机算法的根本目的是寻找到区分不同类别的最优超平面。对于维度较低情况下无法找到分类超平面时，升高维度找到对应到分类超平面。由于升维导致计算复杂进一步提升，因此利用核函数的手段，对计算进行优化处理，使得计算超平面结果更易得取。

对于分类归属的具体判定，采用决策函数值大小进行对于某类别的归属得分判断。

其中K函数为使用的线性核函数，α和y是在样本k情况下的拉格朗日乘子和标签。

接着利用拉格朗日对偶问题，实现最大化SVM的目标函数求解。

其中的核函数K利用矩阵乘法即可实现计算。

对于求解SVM的二次规划，采取SMO算法实现，其核心思想是迭代选择两对拉格朗日乘子进行优化。

通过上述过程，即可利用支持向量机算法基于MNIST已有的训练集实现分类器的训练。

2.3 随机森林

2.3.1 随机森林算法简介

随机森林（Random Forest）是一种基于决策树的集成学习方法，用于分类和回归任务。它通过结合多个决策树的预测结果来提高模型的准确性和鲁棒性。随机森林的核心思想是集成多个“弱学习器”（即决策树），通过集体投票或平均结果来获得更好的性能。

随机森林通过集成多棵决策树，降低了单棵树的过拟合风险，对噪声和异常值有很好的容忍性，因此结果更加稳定且准确。随机森林还可以评估各特征的重要性，帮助进行特征选择，是一种强大且易于解释的集成模型。随机森林算法建立在多个决策树组成，最终的分类结果通过投票或回归结果平均来确定。

决策树是一种常用的监督学习算法，适用于分类和回归任务。其主要特点是通过一系列的决策规则将数据集分成不同的类别或数值范围。决策树算法对于数据的预处理要求较低，并且易于理解和解释，但对于深度较大的决策树，易出现过拟合问题。

随机森林算法通过将多个决策树进行集成组合，提高整个模型的综合稳定性和确定性。随机森林算法在模型训练过程中，使用从训练集有放回采样的bootstrap采样方法，并且在分裂节点的特征选择中，采用随机特征选择的方法实现，能够增加模型的多样性且利于寻找最佳分裂。

随机森林算法拥有较好的泛化能力，并且降低了决策树算法易于过拟合的风险，是一种强大的集成算法。

2.3.2 适用原因

Mnist数据集的每个样本为28×28的灰度矩阵，共784个像素特征，因此是一个高维数据集，并且特征之间为非线性关系。随机森林算法能够处理高维特征并学习复杂的非线性关系，利于捕捉Mnist数据集中像素特征间的复杂交互，提供一个高效的分类性能。

随机森林算法能够实现对每个像素特征的重要性排序，帮助更好地实现手写数字识别分类功能，并且其实现上相对简单，通过简单的超参数调节，便可快速适用于Mnist数据集进行分类器训练，对于超参数调节的敏感性较低，易于在默认参数下便拥有较好的性能。

对于Mnist规模数据集，随机森林算法可以实现每棵树单独训练、并行计算的特点能够更高效完成分类器训练。它能在保持高准确率的同时，减少过拟合的风险，适合在短时间内进行训练并获得不错的分类性能。因此，随机森林在处理 MNIST 数据集时是一个合理且高效的选择，尤其是对于入门级和中等复杂度的图像分类任务。

2.3.3 随机森林模型训练的具体过程

首先将sklearn库的随机森林模型算法相关库、matplotlib库的绘图相关库导入到python文件中。在此，选择决策树数量默认为100个，并且选择随机数种子为42，确保每次训练的可再现性。

将Mnist数据集自划分的训练数据集导入进行训练，在随机森林模型训练完毕后，用Mnist数据集自带的测试集进行模型准确率的计算。

为了方便结果分析，对该训练好的模型，进行结果数据可视化分析与评估。绘制混淆矩阵、ROC曲线以及模型学习曲线。

2.4 神经网络

2.4.1 卷积神经网络

神经网络（Neural Network）是一种受生物神经系统启发的机器学习模型，广泛用于处理复杂的非线性问题，如图像识别、自然语言处理和时间序列预测等。它通过模拟人脑神经元的连接来学习数据中的模式和关系。

神经网络具备强大的非线性建模能力，能够学习复杂的数据关系，因此特别适合复杂的任务。其灵活的网络结构使得模型能够根据不同类型的任务和复杂度进行调整。此外，神经网络在图像识别、语音处理、自然语言处理等领域表现尤为出色，被广泛应用于多种智能化任务中。

卷积神经网络（Convolutional Neural Networks，CNN）是一种深度学习算法，特别擅长于处理具有网格结构的数据，如图像、视频等，它通过卷积层、池化层、全连接层等组成部分实现图像的特征提取与分类任务。本文采用神经网络中的标准卷积神经网络，即LeNet卷积神经网络，进行模型训练。

卷积神经网络结构包含：卷积层、激活层、池化层、全连接层。卷积层是CNN的核心层，通过卷积核对输入图像进行卷积操作，提取出局部特征。利用不同的卷积核进行卷积操作，即可实现对图像不同特征的分别提取。激活层打破网络的线性结构，使得神经网络可以进行非线性映射，从而更适于处理复杂的识别任务。进入池化层后，进行对特征进一步采样，减少数据空间尺寸，降维减少计算量，提高训练效率。神经网络的最后阶段中，将高维特征图展平，通过一个或多个全连接层进行最终分类概率的输出。

2.4.2 卷积神经网络的训练过程

进行卷积操作实现对图像的特征提取。输入为图像的二维矩阵I，以二维矩阵K作为卷积核，实现卷积操作。I(x,y)为图像矩阵在坐标上的像素值，K(m,n)为卷积核在坐标上的值，得到卷积计算公式即公式2.4.2-1。

引入激活函数实现卷积神经网络的非线性映射。通常使用ReLU函数作为卷积神经网络中的激活函数。ReLU函数的定义即公式2.4.2-2。

通过ReLU函数，实现将负值转换为0，将正值保留，使得卷积神经网络能够有效的进行复杂模式的学习。

池化层操作采用平均池化操作。对于给定的池化窗口，计算所有值的平均值，接着池化窗口以一定步长继续滑动，直到最终得到降维后的特征矩阵。

平均池化在保留特征时更为平滑，同时利于减小过拟合风险，尽管作为一种降维手段不可避免会丢失一部分细节，但是图像像素点与其周围像素具有相似的特点使得这种降维方式更利于图像识别和提升模型训练的效率。

进入全连接层后，将提取的特征展平为一维向量，进行分类，使用softmax函数进行概率值转换，得到分到各类的概率值，最终取出最高概率的类别作为结果。

# 第3章 结果评估与优化

3.1 支持向量机

3.1.1 支持向量机结果

在采用sklearn库的支持向量机算法条件下，选择利用在默认参数条件下的线性核函数进行模型训练，最终用MNIST本身自带的测试集进行分类器性能结果评测，得到测试**正确率为97.92%**。

利用matplotlib库进行混淆矩阵热力图的绘制，可以得到最终的支持向量机分类结果混淆矩阵。

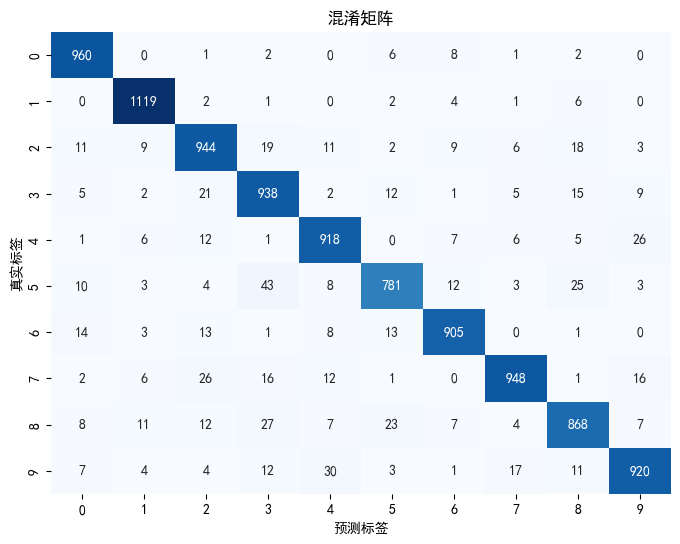


图 1 SVM分类结果混淆矩阵

从图1可以看出对于0~9的手写数字分类，大部分都基本能够准确的实现分类，但仍然有小部分的错误分类发生。

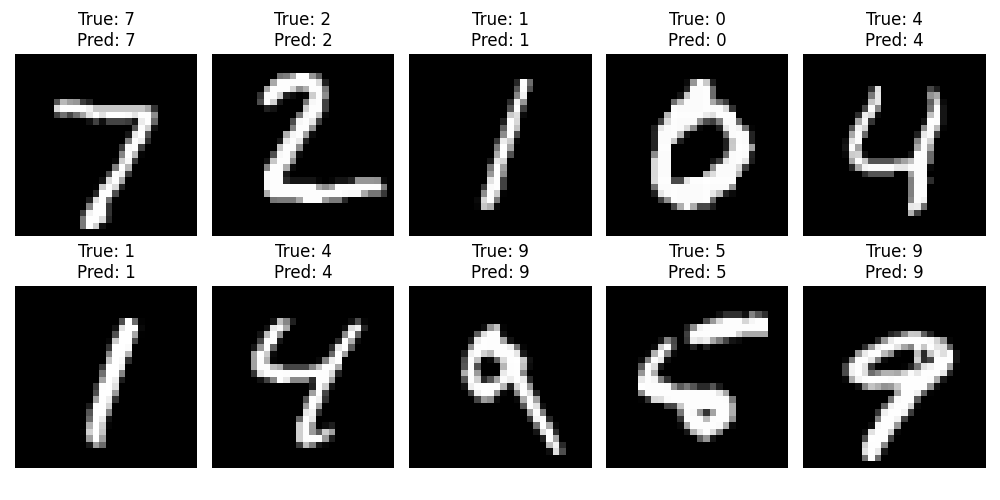


图 2 测试集中的几个分类示例

从图2可以看出，在测试集的随机10个测试样本分类中，SVM算法都能较好地实现分类效果。

绘制SVM模型的ROC曲线图。看得出支持向量机模型在分类Mnist数据集相关内容中有很好的效果。

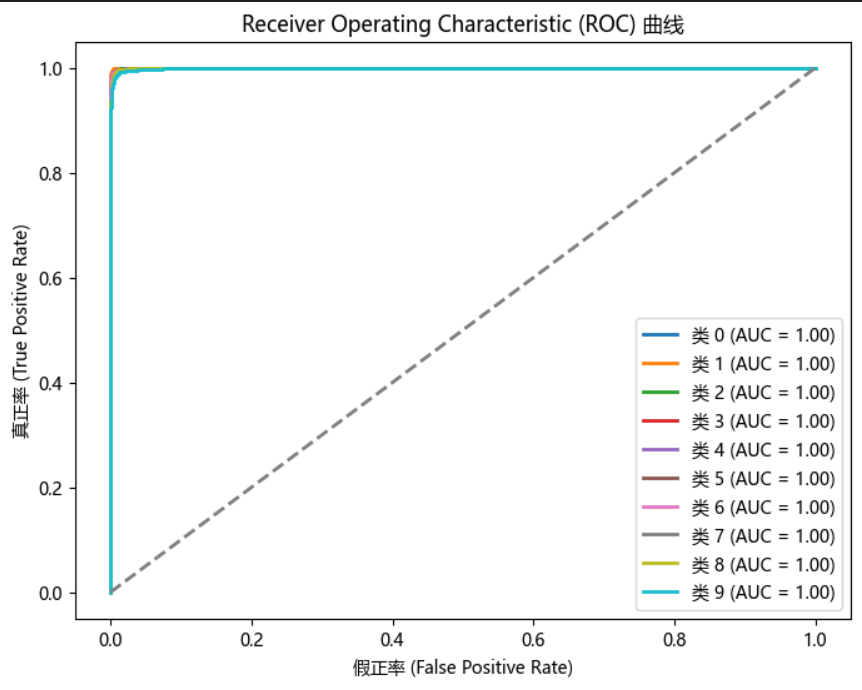


图 3 SVM模型的ROC曲线图

3.1.2 调参评估与改进

在SVM函数中，最重要的参数是c参函数（正则化参数）、kernel参数（核函数选择参数）和gamma参数，其他的可调整参数具体再作介绍。

正则化参数控制分类误差的惩罚力度，决定了目标函数中的损失项与正则化项的权重。核函数用于将原始低维特征映射到更高维的特征空间，以便实现线性可分，不同核函数选择影响最终的分类结果，本文选取最为常用的线性核函数、高斯核函数、多项式核函数进行比对。gamma参数与核函数的特性密切相关，其影响了高斯核函数和多项式核函数的行为，从而影响模型对局部和全局趋势的偏好。

参数获取方法中，本文采用了网格搜索方法。网格搜索方法通过对所有超参数的可能值进行穷举组合，将参数值进行网格排列，在每个网格点上训练模型进行交叉验证，最终选择保留最优的参数组合，实现最优超参数的获取。

通过网格搜索，我们最终得到在不同核函数条件下最优超参数组合。

在选择**线性核函数情况**下，需调节的参数为正则化参数。对比默认参数，最优参数对于准确率**提升0.9%效果。**

|  |  |  |
| --- | --- | --- |
| **正则化参数** | **交叉验证准确率** | **测试集准确率** |
| 0.1 | 0.9222 | 0.9882 |

表格 1 线性核函数下的最优参数

在选择**多项式核函数的情况**下，需要调节的参数为正则化参数、多项式核函数参数coef0、多项式核函数阶数degree、Gamma参数。从表格2可以发现选择多项式核函数，对于Mnist数据集模型训练效果并没有默认参数的线性核函数好，**效果提升为-1.51%**。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **正则化参数** | **核函数参数** | **核函数阶数** | **Gamma值** | **交叉验证准确率** | **测试集准确率** |
| 10 | 1 | 4 | 0.01 | 0.8930 | 0.9641 |

表格 2 多项式核函数下的最优参数

对多项式核函数超参数网格搜索数据可视化，得到超参数热力图。

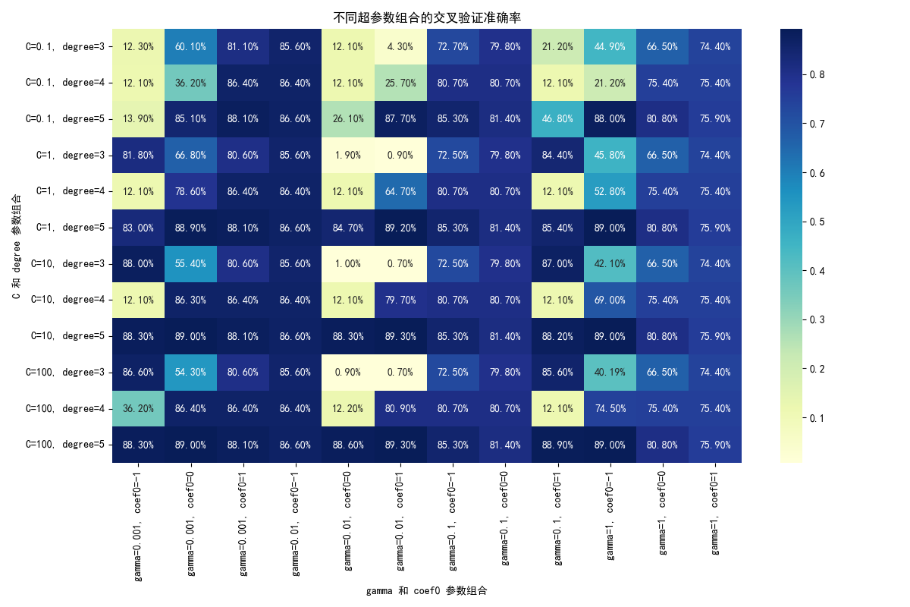


图 4 多项式核函数超参数热力图

选择高斯核函数的情况下，需要调整的参数为正则化参数、Gamma参数。根据表格3中的结果，最优参数下的高斯核函数，效果不如最优参数下的线性核函数，效果提升为**-0.91%。**

|  |  |  |  |
| --- | --- | --- | --- |
| **正则化参数** | **Gamma值** | **交叉验证准确率** | **测试集准确率** |
| 10 | 0.01 | 0.8990 | 0.9701 |

表格 3 高斯核函数下的最优参数

对高斯核函数超参数网格搜索进行数据可视化，得到超参数热力图。

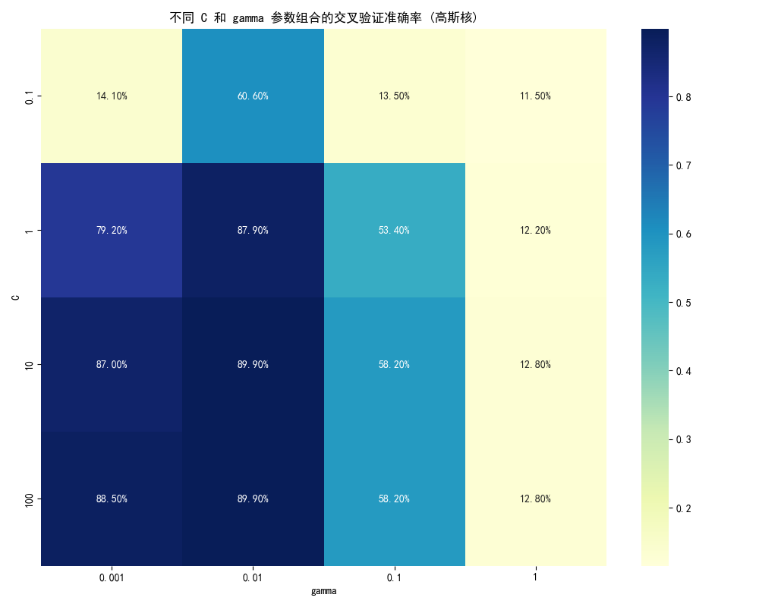


图 5 高斯核函数超参数热力图

综合以上对SVM算法的改进内容，我们最终得到在**最优参数条件下的线性核函数**对模型分类拥有最好的实现效果，测试集**准确率能够达到98.82%。**

|  |  |
| --- | --- |
| **核函数类型** | **测试集准确率** |
| 线性核函数linear | 98.82% |
| 多项式核函数poly | 96.41% |
| 高斯核函数rbf | 97.01% |

表格 4 核函数对比结果

3.2 随机森林

3.2.1 随机森林训练结果

通过对Mnist自带的测试集导入到训练好的随机森林分类器中进行准确度测试，得到最终模型的**正确率为97.04%。**

利用matplotlib库绘制得到随机森林识别模型的混淆矩阵。

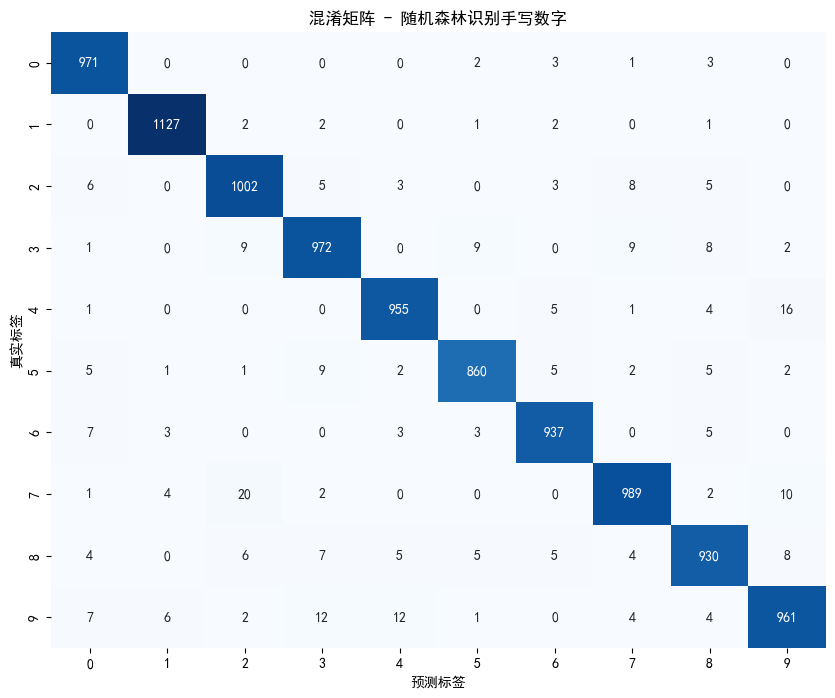


图 6 RF分类结果混淆矩阵

从图3中可见，随机森林对于手写数字的错误分类控制在20以下，拥有较好的分类效果。其弱势分类数字为7、4。4

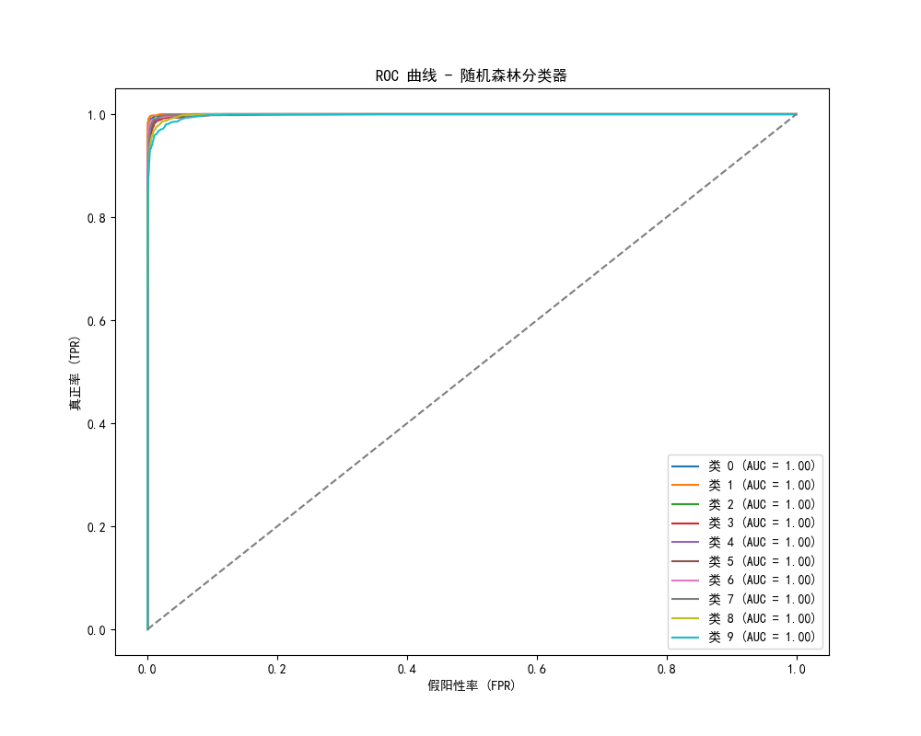


图 7 RF模型ROC曲线图

绘制随机森林ROC曲线图，即图4，可以观察得到，本随机森林分类器在该任务中具有极为优异的效果。每个类别的ROC曲线都几乎达到图左上角，表明分类器对于各类别预测具有高准确性。图例中每个类别的AUC值都为近似于1.00，能够较完美实现所有类别的区分，证明此随机森林分类器在Mnist数据集上表现非常理想，具有高准确率。

为了进一步评估随机森林模型在Mnist训练集的性能与训练数据之间的联系，进行学习曲线的绘制。

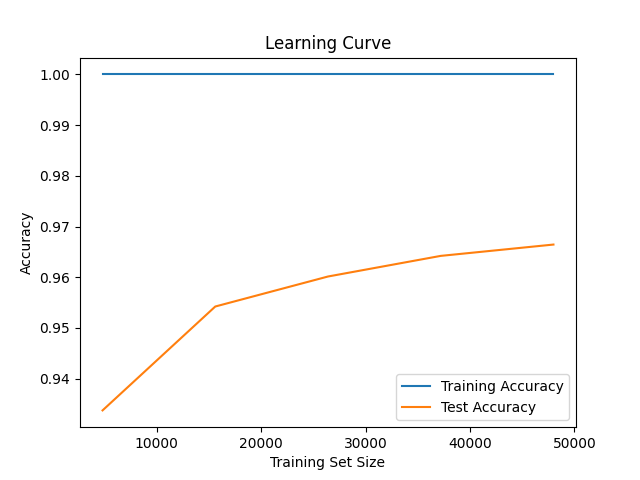


图 8 RF模型学习曲线图

观察图5可知，随机森林训练准确率始终保持在接近100%水平，说明模型在训练数据上表现极佳。随着训练集规模的增加，测试准确率逐渐提高并趋于平稳，近97%。测试准确率低于训练准确率，但仍然很高，这表明模型有很好的泛化性能。整体而言，模型在手写数字识别上表现优秀。

3.2.2 改进方法

对于随机森林过拟合情况的避免，一般可以采用：欠采样（下采样）、过采样（上采样）的方法来平衡样本的数量，达到防止某一类别过拟合同时避免样本数量少的类欠拟合的方法。

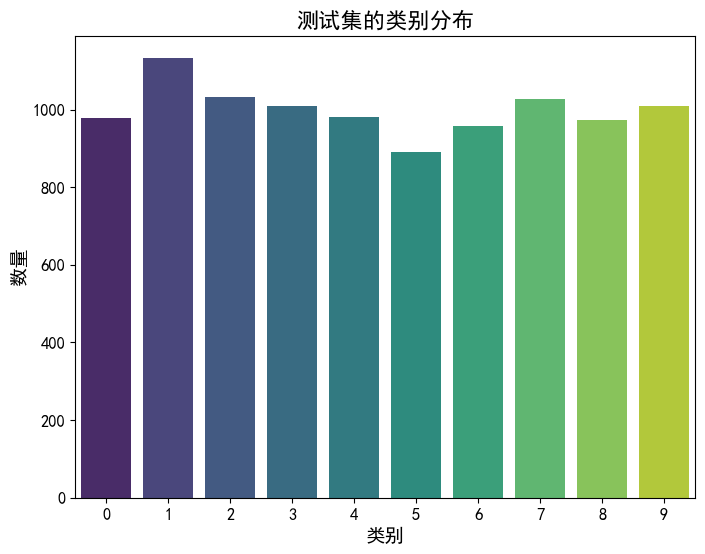
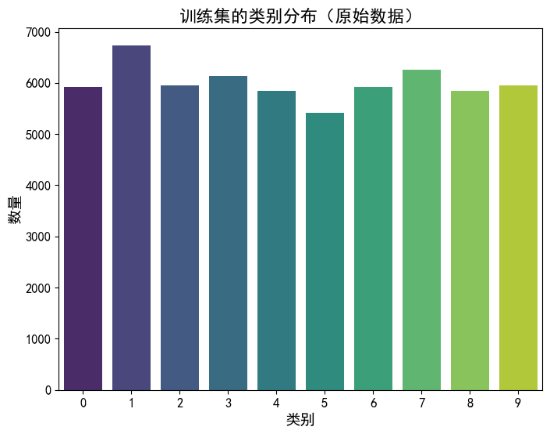


图 9 原始训练集类别分布

欠采样方法通过减少多数类样本的数量以达到与少数样本接近，使样本数据集平衡的效果。欠采样的主要有随机欠采样、Tomek Links、Near Miss和ENN等方法。本文采用欠采样方法中的随机欠采样，随机选择多数样本的一部分进行删除，至与少数样本数量达成一致。

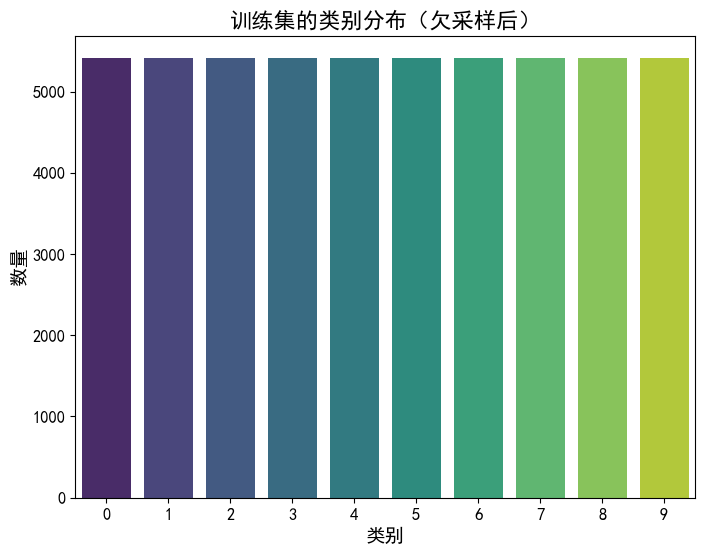


图 10 欠采样后测试集类别分布

过采样方法通过复制少数类样本，使之达到数量接近多数类样本，从而平衡数据集。过采样的主要有随机过采样、SMOTE和ADASYN等方法。本文采用的随机过采样方法实现对样本数据集的平衡。

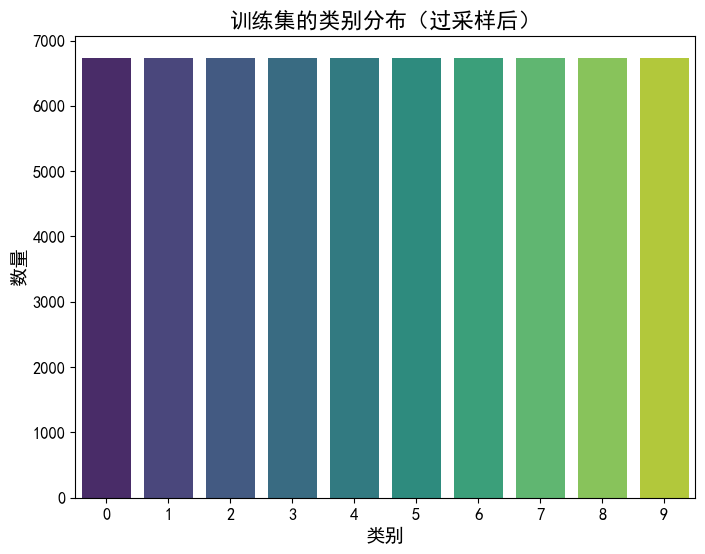


图 11 过采样后测试集类别分布

3.2.3 改进结果评估

通过分别进行欠采样和过采样操作后，与原训练好的随机森林模型进行准确率效果比对。

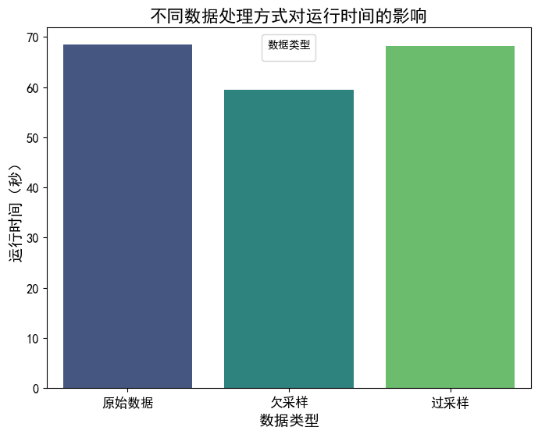
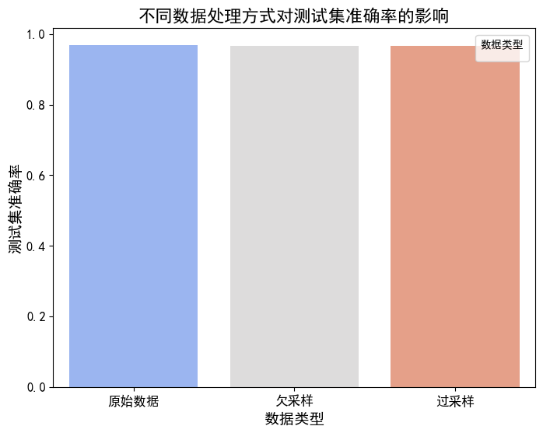
|  |  |  |
| --- | --- | --- |
| 操作类型 | 测试集准确率 | 运行时间（s） |
| 原始数据集 | 0.9680 | 68.485252 |
| 欠采样 | 0.9668 | 59.549360 |
| 过采样 | 0.9658 | 68.230916 |

表格 5 原始测试集

|  |  |  |
| --- | --- | --- |
| 操作类型 | 测试集（欠采样）准确率 | 运行时间（s） |
| 原始数据集 | 0.9674 | 67.155391 |
| 欠采样 | 0.9665 | 58.971178 |
| 过采样 | 0.9659 | 66.295006 |

表格 6 欠采样测试集

将结果进行数据可视化。可以直观看出，欠采样和过采样操作在随机森林模型中对于模型准确率的影响较小，三个情况下的准确率及其相近。但是，进行欠采样操作，可以有效地减少训练模型所需的时间，因此在随机森林模型的训练中，可以引入欠采样操作，节省训练时间，提高效率。



3.3 神经网络

3.3.1 神经网络训练结果

使用Mnist数据集自带的测试集导入到已训练好的卷积神经网络模型进行分类测试，最终**正确率为99.14%**。

可视化CNN模型的训练准确率，可以看到训练模型准确率随着迭代次数的变化情况。

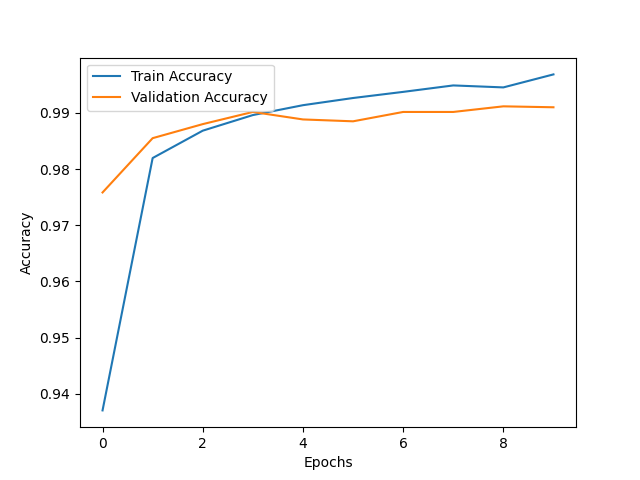


图 12 CNN模型的训练准确率与测试集准确率

卷积神经网络高效地在两个迭代周期内就实现模型准确率的快速提升，保持在97%至99%之间，在后续的训练中逐渐保持准确率的稳定。整个模型在测试集上的识别准确率是很高的，说明卷积神经网络极其适用在Mnist数据集的分类识别中。

绘制卷积神经网络模型的混淆矩阵图。

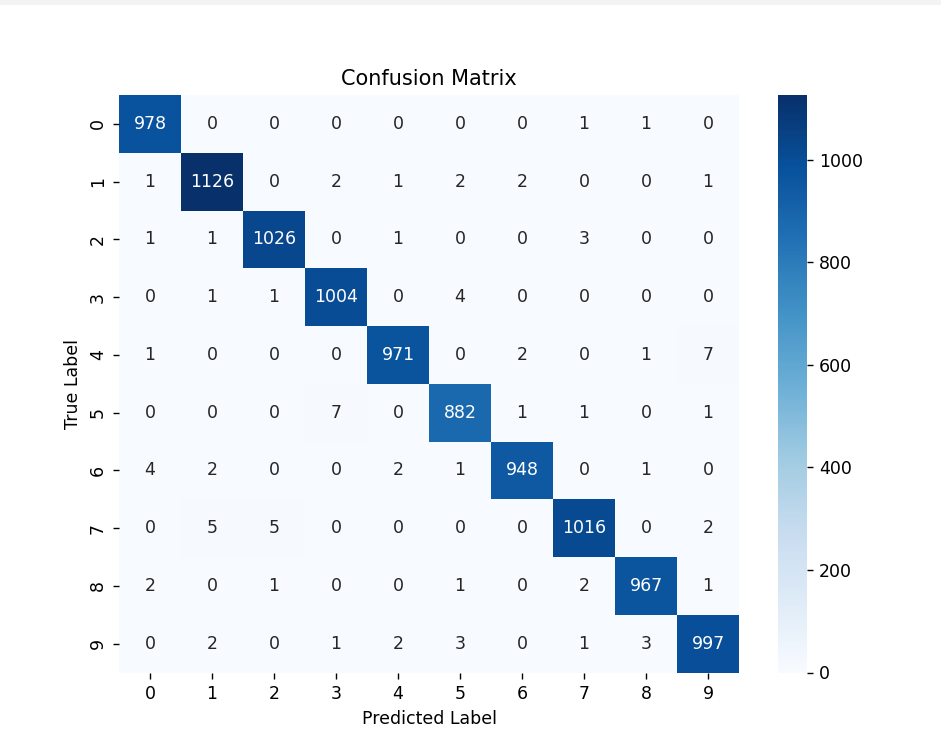


图 13 CNN模型混淆矩阵

图13中可以看出，各个类别错误分类的个数最高不超过7，证明了卷积神经网络模型在手写数字识别上的极高准确度。

对卷积神经网络模型的ROC曲线图像进行可视化。可以看出形状极其接近直角三角形，说明分类效果极佳。

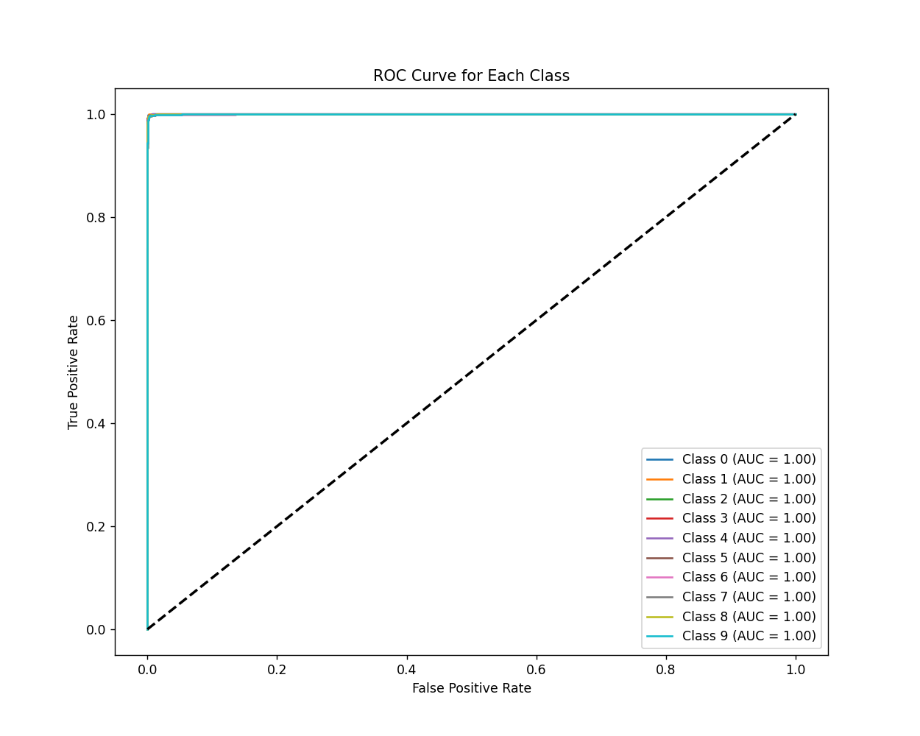


图 14 CNN模型的ROC曲线图

3.4 结果汇总

综合上述三种算法，我们最终得到各算法在最优超参数下的测试集准确率。最终我们选择卷积神经网络作为MNIST数据集的最优训练模型算法。

|  |  |  |
| --- | --- | --- |
| **算法** | **最优超参数** | **测试集准确率** |
| **支持向量机（SVM）** | 正则化参数c=0.1、核函数kernel为linear | 98.82% |
| **随机森林（RF）** | 不进行欠/过采样处理、决策树数量100，随机种子42 | 97.04% |
| **卷积神经网络（CNN）** | 标准卷积Groups=1、无跳步stride=1、无填充padding=0、偏置项bias=True | 99.14% |

3.5 附加内容

3.5.1 提高模型训练方法

MNIST数据集提供的数据元素特征为28x28，特征维数大，在计算过程中直接使用会导致模型训练过程耗时大。因此我们考虑使用一些方法对输入数据进行特征处理，实现对模型训练时间的降低同时尽量减少模型效果的降低。

分箱法是一种将连续变量划分为离散区间的数据预处理技术，广泛应用于统计分析和机器学习中。分箱法的效果主要体现在数据简化和增强模型鲁棒性上，通过将连续变量转化为离散区间，降低数据复杂度，减少异常值和噪声的影响，使模型更加稳定。此外，分箱法能够捕捉变量与目标之间的非线性关系，并调整数据分布，使样本比例更加均衡，从而提高模型训练效果。同时，分箱后的变量更直观易懂，增强了数据的可解释性，便于决策解读和业务规则提取，还能生成统计特征以提升模型的预测能力。这使得分箱法在信用评分、异常值处理、特征工程和营销策略制定等领域广泛应用。

使用不同的分箱大小对模型进行训练，观察模型训练耗时和准确率变化，可以得出最优分箱大小。在本例中使用随机森林模型为基础进行分箱处理。根据结果，最终最优分箱大小为20，在这个分箱下，模型能够保持97.10%的分类准确率，且模型训练时间降低到31s，节省大约15s的时间消耗。

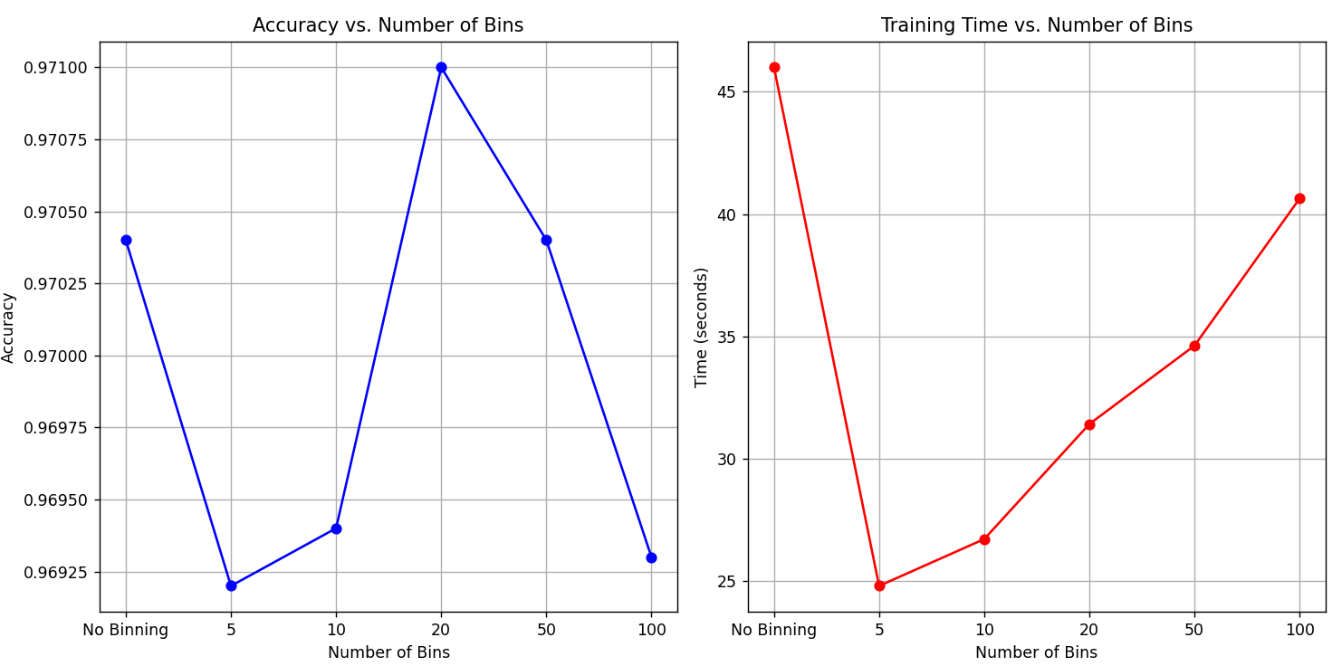


图 15 分箱大小与模型准确率、训练耗时比对图

|  |  |  |
| --- | --- | --- |
| **分箱大小** | **模型准确率** | **训练耗时** |
| 0 | 0.9704 | 46.003335 |
| 5 | 0.9692 | 24.808733 |
| 10 | 0.9694 | 26.722098 |
| 20 | 0.9710 | 31.423074 |
| 50 | 0.9704 | 34.627154 |
| 100 | 0.9693 | 40.640461 |

表格 7 分箱结果表

主成分分析（Principal Component Analysis, PCA）是一种经典的降维算法，用于高维数据的特征提取和数据简化。它通过线性变换，将原始数据映射到一个新的低维空间，同时尽可能保留原始数据的方差信息。PCA广泛应用于模式识别、数据可视化和机器学习特征处理等领域。

在随机森林模型基础上，使用PCA对输入训练集进行降维处理。观察结果可以发现，PCA降维计算反而导致计算耗时提升，推测原因在于降维计算的耗时比直接导入特征进行分类时间更长。因此在效果和训练耗时的结果上，都不应该选择使用PCA对MNIST数据集进行降维。

|  |  |  |
| --- | --- | --- |
| **数据** | **模型准确率** | **训练耗时** |
| 原始数据 | 0.9704 | 45.059127 |
| PCA降维数据 | 0.9552 | 95.759258 |

表格 8 PCA处理结果

3.5.2 手写功能

为了增进本次实践在现实的实际应用意义，我们小组利用python语言设计了一个手写输入应用程序，利用训练好的分类器作为程序的识别模型，使用鼠标进行数字书写并即时分类识别，能够切实地感受手写数字识别的分类效果。

通过python的Tkinter库进行GUI创建，包括一个手写绘图的幕布，和识别、消除两个按钮。绘制界面提供28x28像素的网格，用户使用时直接进行绘制，程序将其转换为Mnist数据集一样的特征，导入到已训练好的模型中，进行识别，最终在GUI界面输出识别结果。

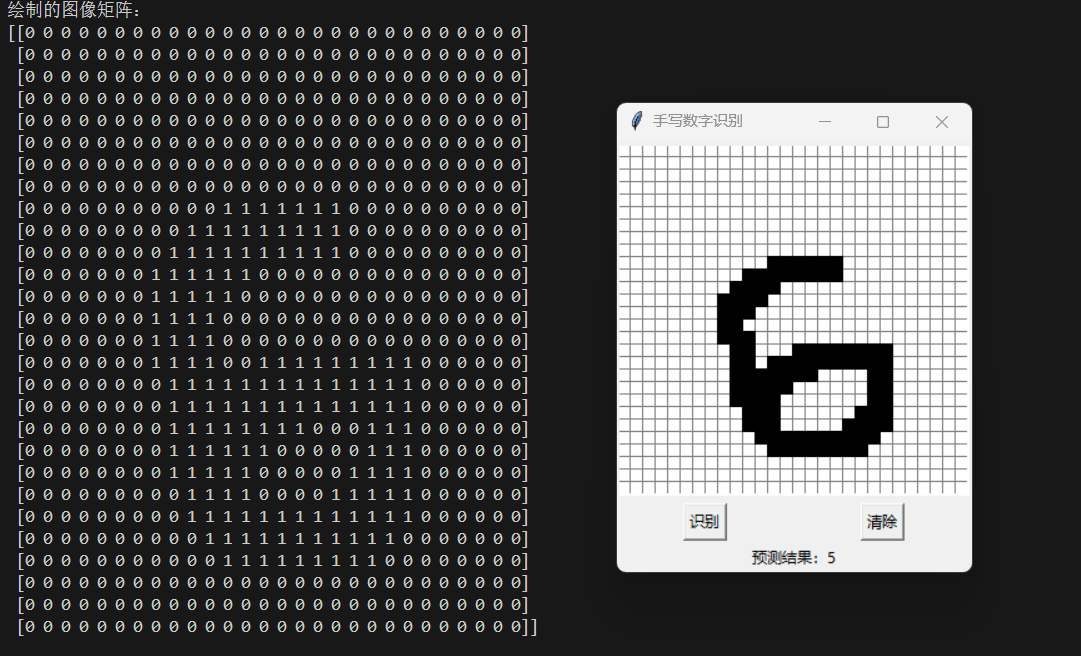


图 16 错误识别样例

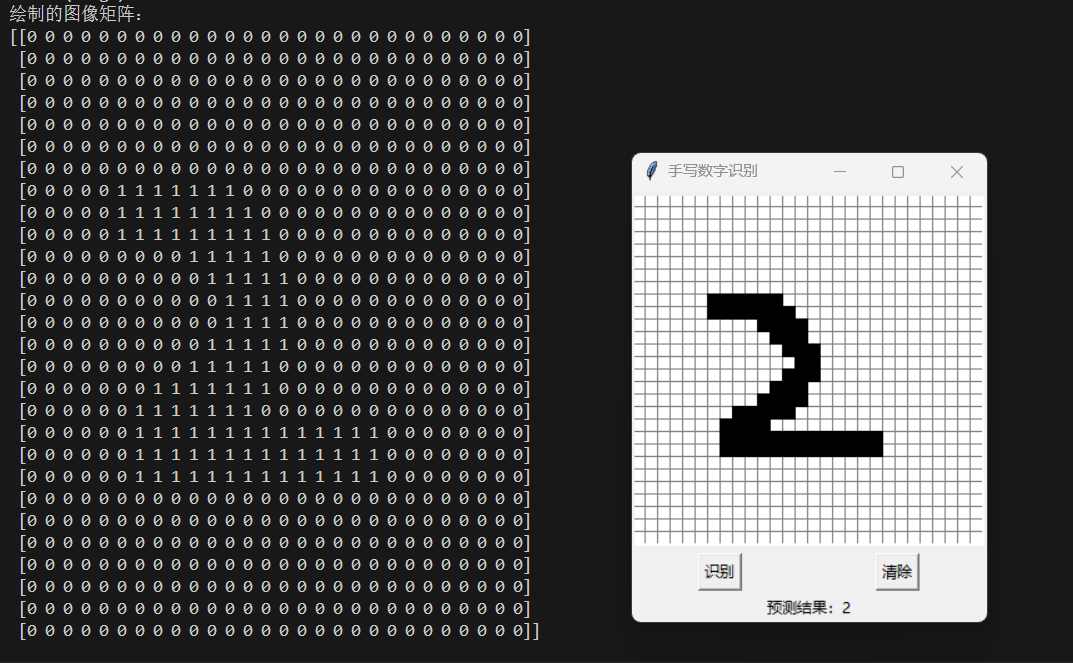


图 17 正确识别样例

# 第4章 结论与展望

4.1 结论

本文基于Mnist手写数据集，实现利用支持向量机算法、随机森林算法、神经网络算法对手写数字的识别模型训练。目的是探求这哪个算法能实现对于Mnist数据集的最优分类识别训练，同时进行评估和改进，使得能够获取一个最优算法的最优参数。最终得到利用卷积神经网络，对于Mnist手写数据集可以实现识别**准确率99.14%**的结果。

对于支持向量机算法方面，我们首先利用Mnist数据集进行模型训练，接着利用ROC曲线可视化评估了支持向量机算法在本数据集上的分类效果。接着选择了三种最常见的核函数进行网格搜索，使其分别在最优超参数条件下达到自身最优效果，最终选择线性核函数作为支持向量机最优核函数使用。

对于随机森林算法方面，我们首先进行随机森林模型训练，结果数据可视化了模型的ROC曲线、学习率变化图，评估了随机森林算法的分类效果。接着，在随机森林模型改进方面进行对训练集、测试集的欠采样和过采样操作，得到在欠采样下，能够有效提升模型的训练速度并且不会对模型识别效果有很大损失的结果。

对于神经网络算法方面，我们实现对于Mnist数据集的基于卷积神经网络的模型训练。卷积神经网络十分适合在图像识别方面，因此得到一个极其高效的分类模型。接着进行对神经网络模型的相关评估。

在附加内容方面，我们首先探求如何提升模型训练效率。发现分箱法适合在MNIST数据集使用，而PCA降维处理反而导致结果更差。接着我们设计了一个手写识别程序，可以通过导入已有训练好的模型，实现用户实时书写，实时输出识别的效果。对于不同算法情况下，手写识别程序的识别结果有所不同。因为像素网络数量设置，和程序实现上的原因，在实际应用中仍需要进一步调整改进，但是基本识别效果良好。

4.2 展望

随着手写识别应用愈发融入我们的生活，越来越多更复杂、高效的算法会不断涌现。本文基于经典的机器学习算法进行实践，目的在于增进对于优秀分类识别机器学习算法的理解和在实践中提升技能。

文章对于各个算法的调整，主要集中在参数方面，未来进一步方向可以是进入对算法流程、增添实际条件情况的进一步改进中。通过进一步增添额外条件，使得算法更加符合实际情况，达到更好分类效果的方向进行。

本文的手写识别应用中，仅是基于训练好的模型，简单实现一个识别程序，未来可以进一步完善手写应用程序，如增添橡皮擦功能、增添灰度色调调整等，不仅局限于纯色黑白的手写，提供更符合现实书写图像的模拟和更流畅的绘图书写体验的方向，对手写识别程序进行改进。

未来的识别技术不会只是在手写数字识别，可以向着图像识别方向进一步扩展，甚至是美术艺术方面也有学习和拓展的实际应用意义。

# 参 考 文 献

[1]张燕宁,陈海燕,常莹,等.基于KNN算法的手写数字识别技术研究[J].电脑编程技巧与维护,2021,(11):123-124+132.DOI:10.16184/j.cnki.comprg.2021.11.046.

[2]杨永琪,徐欣蕾,陈曦.基于SVM技术实现手写数字分类识别的研究[J].电脑知识与技术,2020,16(06):195-196.DOI:10.14004/j.cnki.ckt.2020.0701.

[3]余国庆,杨燕婷,宗兆星,等.基于卷积神经网络的手写数字识别技术研究[J].安徽电子信息职业技术学院学报,2024,23(03):1-5.

[4]赵力衡.基于决策树的手写数字识别的应用研究[J].软件,2018,39(03):90-94.

[5]贯宸,杨云峰.基于四种算法下的手写数字识别准确率对比[J].新型工业化,2020,10(07):1-3.DOI:10.19335/j.cnki.2095-6649.2020.07.001.

[6]周勃恺.基于随机森林的手写数字识别技术[J].电脑编程技巧与维护,2021,(11):112-114.DOI:10.16184/j.cnki.comprg.2021.11.042.

[7]黄旻浩.基于线性分类器的手写数字识别[J].科学技术创新,2019,(33):58-59.

[8]蒙庚祥,方景龙.基于支持向量机的手写体数字识别系统设计[J].计算机工程与设计,2005,(06):1592-1594+1598.DOI:10.16208/j.issn1000-7024.2005.06.062.

# 附 录（小二号黑体居中）

SVM算法实现代码

# 库导入

import random

import matplotlib.pyplot as plt

import numpy as np

from sklearn import datasets

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score,confusion\_matrix

from sklearn.preprocessing import StandardScaler

#Mnist数据集导入

data=np.load("Projects/Mnist.npz")

#数据项提取

x\_train=data['x\_train']

y\_train=data['y\_train']

x\_test=data['x\_test']

y\_test=data['y\_test']

#软间隔优化的支持向量机的实现

class SVM:

    #SVM类初始化函数

    def \_\_init\_\_(self,dataset,target,C,toler,Iter\_max):

        '''

        dataset:训练数据集

        target:数据集标签

        N:样本数量

        M:特征数量

        C:惩罚参数

        toler:容忍度

        b:偏置项

        Alpha:拉格朗日乘子

        Iter\_max:最大迭代次数

        '''

        self.dataset=dataset

        self.target=target

        self.N=len(self.dataset)

        self.M=len(dataset[0])

        self.C=C

        self.b=0

        self.Alpha=np.zeros(self.N)#长度为self.N的以为数组，初始化为0

        self.iter\_max=Iter\_max #最大迭代次数——在所有样本符合KTT条件时停止训练

        self.w=np.zeros(self.M)

    #分类决策值计算函数——计算属于某一类别的得分

    def Fx(self,i):

        f=0

        #利用所有拉格朗日乘子和标签计算当前决策值

        for k in range(self.N):

            f+=self.Alpha[k]\*self.target[k]\*np.matmul(self.dataset[i],self.dataset[k].T) #矩阵乘法

        f+=self.b

        return f

    #高斯核函数——计算两个样本的内积

    def Kernel(self,i,j):

        sigma=1

        distance\_squared=np.linalg.norm(i-j)\*\*2 #欧氏距离平方

        result=np.exp(-distance\_squared/(2\*sigma\*\*2))#sigema设置为1的情况下的高斯核函数

        return result

    #选择与i不同的样本

    def random\_j(self,i):

        while True:

            j=random.choice(range(self.N))

            if j!=i:

                return j

    #Alpha[i]、Alpha[j]的边界L、H计算

    def get\_L\_H(self,i,j):

        L,H=0,0

        #根据标签是否相同确定L、H的范围

        if self.target[i]!=self.target[j]:

            L=max([0,self.Alpha[j]-self.Alpha[i]])

            H=min([self.C,self.C+self.Alpha[i]+self.Alpha[j]])

        else:

            L = max([0, self.Alpha[j] + self.Alpha[i] - self.C])

            H = min([self.C, self.Alpha[i] + self.Alpha[j]])

        return L, H

    # 将Alpha[j]值约束在L和H之间

    def filter(self, L, H, alpha\_j):

        if alpha\_j < L:

            alpha\_j = L

        if alpha\_j > H:

            alpha\_j = H

        return alpha\_j

    # SMO算法，优化SVM的拉格朗日乘子Alpha值

    def SMO(self):

        iter = 0

        while iter < self.iter\_max:

            change\_num = 0  # 记录Alpha值是否有更新

            for i in range(self.N):

                Fx\_i = self.Fx(i)

                Ex\_i = Fx\_i - self.target[i]  # 误差计算

                # 检查是否满足KKT条件，不满足则更新Alpha

                if self.target[i] \* Ex\_i < -self.toler and self.Alpha[i] < self.C or self.target[i] \* Ex\_i > self.toler and self.Alpha[i] > 0:

                    j = self.random\_j(i)  # 随机选择另一个样本j

                    print('i:{},j:{}'.format(i, j))

                    Fx\_j = self.Fx(j)

                    Ex\_j = Fx\_j - self.target[j]

                    alpha\_i = self.Alpha[i]

                    alpha\_j = self.Alpha[j]

                    L, H = self.get\_L\_H(i, j)  # 获取L和H

                    if L == H:

                        print('L == H')

                        continue

                    eta = self.Kernel(i, i) + self.Kernel(j, j) - 2 \* self.Kernel(i, j)

                    if eta <= 0:

                        print('eta <= 0')

                        continue

                    # 更新Alpha[j]值

                    self.Alpha[j] += self.target[j] \* (Ex\_i - Ex\_j) / eta

                    self.Alpha[j] = self.filter(L, H, self.Alpha[j])  # 约束在L和H之间

                    if abs(self.Alpha[j] - alpha\_j) < 0.00001:  # 检查Alpha[j]的更新是否足够大

                        print('alpha够精确了')

                        continue

                    # 更新Alpha[i]值

                    self.Alpha[i] += self.target[i] \* self.target[j] \* (alpha\_j - self.Alpha[j])

                    # 更新偏置项b

                    b1 = self.b - Ex\_i - self.target[i] \* self.Kernel(i, i) \* (self.Alpha[i] - alpha\_i) - self.target[j] \* self.Kernel(i, j) \* (self.Alpha[j] - alpha\_j)

                    b2 = self.b - Ex\_j - self.target[i] \* self.Kernel(i, j) \* (self.Alpha[i] - alpha\_i) - self.target[j] \* self.Kernel(j, j) \* (self.Alpha[j] - alpha\_j)

                    if 0 < self.Alpha[i] < self.C:

                        self.b = b1

                    elif 0 < self.Alpha[j] < self.C:

                        self.b = b2

                    else:

                        self.b = (b1 + b2) / 2.0

                    print(self.Alpha[i], self.Alpha[j])

                    change\_num += 1

            if change\_num == 0:

                iter += 1  # 若没有更新，迭代次数加1

            else:

                iter = 0  # 若有更新，重置迭代次数

        # 计算权重向量w

        for i in range(self.N):

            self.w += self.target[i] \* self.Alpha[i] \* self.dataset[i]

# 可视化分类结果

    def display(self):

        svm\_point = []

        for i in range(100):

            if self.Alpha[i] > 0:  # 识别支持向量

                print('第{}个是支持向量'.format(i), self.dataset[i], self.target[i])

                svm\_point.append(i)

        x\_point = np.array([i[0] for i in dataset])

        y\_point = np.array([i[1] for i in dataset])

        x = np.linspace(4, 6, 5)  # 绘制超平面

        y = -(self.w[0] \* x + self.b) / self.w[1]

        plt.scatter(x\_point[:50], y\_point[:50], color='red')  # 类别1

        plt.scatter(x\_point[-50:], y\_point[-50:], color='blue')  # 类别-1

        support\_vector = np.array([dataset[i] for i in svm\_point])  # 支持向量

        plt.scatter(support\_vector[:, 0], support\_vector[:, 1], color='black')

        plt.plot(x, y)  # 画决策边界

        plt.show()

# 主程序入口

if \_\_name\_\_ == '\_\_main\_\_':

    # 加载Iris数据集的前100个样本（两类），并将标签转换为SVM格式

    dataset, target = load\_iris()['data'][:100, :2], load\_iris()['target'][:100]

    target = np.array([1 if i == 1 else -1 for i in target])

    # 初始化SVM模型，设置参数C=100，容忍度0.01，最大迭代40次

    model = SVM(dataset, target, 100, 0.01, 40)

    model.SMO()  # 训练模型

    print(model.w)  # 输出权重向量w

    print(model.b)  # 输出偏置项b

    model.display()  # 可视化结果

随机森林欠拟合过拟合可视化展现代码

import os

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from time import time

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix

from sklearn.model\_selection import GridSearchCV

# 设置 Matplotlib 支持中文

plt.rcParams['font.sans-serif'] = ['SimHei'] # 设置字体为黑体

plt.rcParams['axes.unicode\_minus'] = False # 解决坐标轴负号显示问题

# 加载 MNIST 数据集

path = 'C:/Users/Lenovo/Downloads/mnist.npz'

data = np.load(path)

x\_train = data['x\_train']

y\_train = data['y\_train']

x\_test = data['x\_test']

y\_test = data['y\_test']

# 数据预处理

x\_train = x\_train.reshape(-1, 28 \* 28) # 展平

x\_test = x\_test.reshape(-1, 28 \* 28) # 展平

x\_train = x\_train / 255.0 # 归一化

x\_test = x\_test / 255.0 # 归一化

# 设置临时目录（避免并行搜索内存不足）

os.environ["JOBLIB\_TEMP\_FOLDER"] = "D:/temp"

# 数据分布可视化

def plot\_class\_distribution(y, title):

plt.figure(figsize=(8, 6))

sns.countplot(x=y, palette="viridis")

plt.title(title, fontsize=16)

plt.xlabel("类别", fontsize=14)

plt.ylabel("数量", fontsize=14)

plt.xticks(fontsize=12)

plt.yticks(fontsize=12)

plt.show()

plot\_class\_distribution(y\_train, "训练集的类别分布（原始数据）")

# 手动实现欠采样

def under\_sample(x, y):

unique\_classes, class\_counts = np.unique(y, return\_counts=True)

min\_class\_count = min(class\_counts) # 找到最少样本类别的数量

sampled\_indices = np.hstack([

np.random.choice(np.where(y == cls)[0], min\_class\_count, replace=False)

for cls in unique\_classes

])

return x[sampled\_indices], y[sampled\_indices]

x\_train\_under, y\_train\_under = under\_sample(x\_train, y\_train)

plot\_class\_distribution(y\_train\_under, "训练集的类别分布（欠采样后）")

# 手动实现过采样

def over\_sample(x, y):

unique\_classes, class\_counts = np.unique(y, return\_counts=True)

max\_class\_count = max(class\_counts) # 找到最多样本类别的数量

sampled\_indices = np.hstack([

np.random.choice(np.where(y == cls)[0], max\_class\_count, replace=True)

for cls in unique\_classes

])

return x[sampled\_indices], y[sampled\_indices]

x\_train\_over, y\_train\_over = over\_sample(x\_train, y\_train)

plot\_class\_distribution(y\_train\_over, "训练集的类别分布（过采样后）")

# 定义随机森林模型和超参数

rf\_model = RandomForestClassifier(random\_state=42)

param\_grid = {

'n\_estimators': [100],

'max\_depth': [20],

'min\_samples\_split': [5],

'min\_samples\_leaf': [2],

'max\_features': ['sqrt']

}

# 定义函数：训练模型并输出结果

def train\_and\_evaluate(x\_train, y\_train, x\_test, y\_test, description):

start\_time = time()

grid\_search = GridSearchCV(estimator=rf\_model, param\_grid=param\_grid,

scoring='accuracy', cv=3, verbose=2, n\_jobs=-1)

grid\_search.fit(x\_train, y\_train)

elapsed\_time = time() - start\_time

# 输出最佳参数和测试集准确率

print(f"{description}最佳参数: {grid\_search.best\_params\_}")

print(f"{description}最佳验证准确率: {grid\_search.best\_score\_:.2%}")

print(f"{description}运行时间: {elapsed\_time:.2f} 秒")

best\_model = grid\_search.best\_estimator\_

y\_pred = best\_model.predict(x\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"{description}测试集准确率: {accuracy:.2%}")

# 绘制混淆矩阵

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(10, 8))

sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)

plt.title(f"{description}混淆矩阵", fontsize=16)

plt.xlabel("预测标签", fontsize=14)

plt.ylabel("真实标签", fontsize=14)

plt.xticks(fontsize=12)

plt.yticks(fontsize=12)

plt.show()

return accuracy, elapsed\_time

# 测试模型

print("开始训练和测试...")

acc\_original, time\_original = train\_and\_evaluate(x\_train, y\_train, x\_test, y\_test, "原始数据")

acc\_under, time\_under = train\_and\_evaluate(x\_train\_under, y\_train\_under, x\_test, y\_test, "欠采样数据")

acc\_over, time\_over = train\_and\_evaluate(x\_train\_over, y\_train\_over, x\_test, y\_test, "过采样数据")

# 汇总结果

results = pd.DataFrame({

"数据类型": ["原始数据", "欠采样", "过采样"],

"测试集准确率": [acc\_original, acc\_under, acc\_over],

"运行时间（秒）": [time\_original, time\_under, time\_over]

})

print(results)

# 可视化测试集准确率

plt.figure(figsize=(8, 6))

sns.barplot(x="数据类型", y="测试集准确率", hue="数据类型", dodge=False, data=results, palette="coolwarm")

plt.title("不同数据处理方式对测试集准确率的影响", fontsize=16)

plt.xlabel("数据类型", fontsize=14)

plt.ylabel("测试集准确率", fontsize=14)

plt.xticks(fontsize=12)

plt.yticks(fontsize=12)

plt.legend(title="数据类型", fontsize=12)

plt.show()

# 可视化运行时间

plt.figure(figsize=(8, 6))

sns.barplot(x="数据类型", y="运行时间（秒）", hue="数据类型", dodge=False, data=results, palette="viridis")

plt.title("不同数据处理方式对运行时间的影响", fontsize=16)

plt.xlabel("数据类型", fontsize=14)

plt.ylabel("运行时间（秒）", fontsize=14)

plt.xticks(fontsize=12)

plt.yticks(fontsize=12)

plt.legend(title="数据类型", fontsize=12)

plt.show()

CNN卷积神经网络代码：

import tensorflow as tf

from tensorflow.keras import layers, models

import matplotlib.pyplot as plt

# 加载 MNIST 数据集

(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.mnist.load\_data()

# 数据预处理

x\_train = x\_train.reshape(-1, 28, 28, 1).astype("float32") / 255.0  # 归一化并调整维度

x\_test = x\_test.reshape(-1, 28, 28, 1).astype("float32") / 255.0

y\_train = tf.keras.utils.to\_categorical(y\_train, 10)  # One-hot 编码

y\_test = tf.keras.utils.to\_categorical(y\_test, 10)

# 构建 CNN 模型

model = models.Sequential([

    layers.Conv2D(32, (3, 3), activation="relu", input\_shape=(28, 28, 1)),  # 卷积层1

    layers.MaxPooling2D((2, 2)),  # 池化层1

    layers.Conv2D(64, (3, 3), activation="relu"),  # 卷积层2

    layers.MaxPooling2D((2, 2)),  # 池化层2

    layers.Conv2D(64, (3, 3), activation="relu"),  # 卷积层3

    layers.Flatten(),  # 展平层

    layers.Dense(64, activation="relu"),  # 全连接层1

    layers.Dense(10, activation="softmax")  # 输出层

])

# 编译模型

model.compile(optimizer="adam",

            loss="categorical\_crossentropy",

            metrics=["accuracy"])

# 训练模型

history = model.fit(x\_train, y\_train, epochs=10, batch\_size=64, validation\_split=0.1)

# 测试模型

test\_loss, test\_acc = model.evaluate(x\_test, y\_test)

print(f"Test accuracy: {test\_acc:.4f}")

# 绘制训练过程

plt.plot(history.history["accuracy"], label="Train Accuracy")

plt.plot(history.history["val\_accuracy"], label="Validation Accuracy")

plt.xlabel("Epochs")

plt.ylabel("Accuracy")

plt.legend()

plt.show()

# 保存模型

model.save("mnist\_cnn\_model.keras")

print("Model saved as mnist\_cnn\_model.keras")

LSTM长短期记忆神经网络代码

import tensorflow as tf

from tensorflow.keras import layers, models

import matplotlib.pyplot as plt

# 加载 MNIST 数据集

(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.mnist.load\_data()

# 数据预处理

x\_train = x\_train.reshape(-1, 28, 28, 1).astype("float32") / 255.0 # 归一化并调整维度

x\_test = x\_test.reshape(-1, 28, 28, 1).astype("float32") / 255.0

y\_train = tf.keras.utils.to\_categorical(y\_train, 10) # One-hot 编码

y\_test = tf.keras.utils.to\_categorical(y\_test, 10)

# 构建 CNN 模型

model = models.Sequential([

layers.Conv2D(32, (3, 3), activation="relu", input\_shape=(28, 28, 1)), # 卷积层1

layers.MaxPooling2D((2, 2)), # 池化层1

layers.Conv2D(64, (3, 3), activation="relu"), # 卷积层2

layers.MaxPooling2D((2, 2)), # 池化层2

layers.Conv2D(64, (3, 3), activation="relu"), # 卷积层3

layers.Flatten(), # 展平层

layers.Dense(64, activation="relu"), # 全连接层1

layers.Dense(10, activation="softmax") # 输出层

])

# 编译模型

model.compile(optimizer="adam",

loss="categorical\_crossentropy",

metrics=["accuracy"])

# 训练模型

history = model.fit(x\_train, y\_train, epochs=10, batch\_size=64, validation\_split=0.1)

# 测试模型

test\_loss, test\_acc = model.evaluate(x\_test, y\_test)

print(f"Test accuracy: {test\_acc:.4f}")

# 绘制训练过程

plt.plot(history.history["accuracy"], label="Train Accuracy")

plt.plot(history.history["val\_accuracy"], label="Validation Accuracy")

plt.xlabel("Epochs")

plt.ylabel("Accuracy")

plt.legend()

plt.show()

# 保存模型

model.save("mnist\_cnn\_model.keras")

print("Model saved as mnist\_cnn\_model.keras")

随机森林最大特征数和最小样本数调参

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_openml

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report

import os

# 指定一个有权限的文件夹路径来存储数据集

data\_home = os.path.join(os.getcwd(), 'sklearn\_data') # 例如，当前工作目录下的 'sklearn\_data' 文件夹

# 加载MNIST数据集并指定存储路径

mnist = fetch\_openml("mnist\_784", data\_home=data\_home)

# 获取数据和标签

X, y = mnist["data"], mnist["target"].astype(int)

# 只选取部分训练数据（减少数据量）

X\_train, \_, y\_train, \_ = train\_test\_split(X, y, train\_size=0.1, random\_state=42) # 使用10%的数据

X\_train /= 255.0 # 归一化

X\_test = X[::100] # 使用每100个数据点中的一个作为测试集（减少测试集大小）

y\_test = y[::100] # 测试集标签

X\_test /= 255.0 # 归一化

# 设置调参范围：max\_depth的不同值

param\_grid = {

'n\_estimators': [100], # 设置一个固定的树的数量

'max\_depth': [None, 10, 20, 30, 40, 50] # 不同的树深度值

}

# 创建RandomForestClassifier模型

rf = RandomForestClassifier(random\_state=42)

# 使用较少的交叉验证折数（cv=2）以减少计算时间

grid\_search = GridSearchCV(estimator=rf, param\_grid=param\_grid, cv=2, verbose=1, n\_jobs=1, scoring='accuracy')

# 训练模型

grid\_search.fit(X\_train, y\_train)

# 获取调参结果

results = grid\_search.cv\_results\_

mean\_test\_scores = results['mean\_test\_score']

max\_depth\_values = results['param\_max\_depth'].data

# 绘制性能图：max\_depth与准确率的关系

plt.figure(figsize=(8, 6))

plt.plot(max\_depth\_values, mean\_test\_scores, marker='o', color='b', linestyle='-', markersize=6)

plt.title('Effect of max\_depth on Accuracy in Random Forest (MNIST)', fontsize=14)

plt.xlabel('max\_depth', fontsize=12)

plt.ylabel('Mean Test Accuracy', fontsize=12)

plt.grid(True)

# 保存图像为图片

plt.savefig('random\_forest\_max\_depth.png')

# 显示图像

plt.show()

# 输出最佳参数

print("Best parameters found: ", grid\_search.best\_params\_)

# 使用最佳模型进行预测

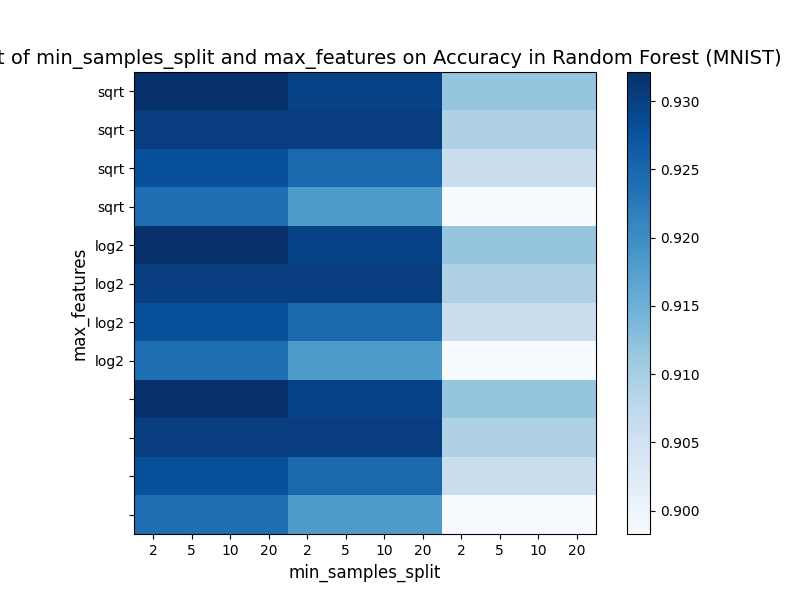
best\_rf = grid\_search.best\_estimator\_

y\_pred = best\_rf.predict(X\_test)

# 打印分类报告

print("Accuracy on Test Set: ", accuracy\_score(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))



随机森林树的数量调参

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_openml

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report

import os

# 指定一个有权限的文件夹路径来存储数据集

data\_home = os.path.join(os.getcwd(), 'sklearn\_data') # 例如，当前工作目录下的 'sklearn\_data' 文件夹

# 加载MNIST数据集并指定存储路径

mnist = fetch\_openml("mnist\_784", data\_home=data\_home)

# 获取数据和标签

X, y = mnist["data"], mnist["target"].astype(int)

# 只选取部分训练数据（减少数据量）

X\_train, \_, y\_train, \_ = train\_test\_split(X, y, train\_size=0.1, random\_state=42) # 使用10%的数据

X\_train /= 255.0 # 归一化

X\_test = X[:1000] # 使用每100个数据点中的一个作为测试集（减少测试集大小）

y\_test = y[:1000] # 测试集标签

X\_test /= 255.0 # 归一化

# 设置较小的参数范围

param\_grid = {

'n\_estimators': [50, 100, 150], # 减少树的数量选择

'max\_depth': [None], # 树的最大深度为 None

'max\_features': ['sqrt'], # 每个树使用 sqrt 特征

'min\_samples\_leaf': [1], # 每个叶子节点至少包含 1 个样本

'min\_samples\_split': [2], # 每个节点至少包含 2 个样本进行分裂

}

# 创建RandomForestClassifier模型

rf = RandomForestClassifier(random\_state=42)

# 使用较少的交叉验证折数

grid\_search = GridSearchCV(estimator=rf, param\_grid=param\_grid, cv=2, verbose=1, n\_jobs=1, scoring='accuracy') # cv=2 可以加速

# 训练模型

grid\_search.fit(X\_train, y\_train)

# 获取调参结果

results = grid\_search.cv\_results\_

mean\_test\_scores = results['mean\_test\_score']

n\_estimators = results['param\_n\_estimators'].data

# 绘制性能图

plt.figure(figsize=(8, 6))

plt.plot(n\_estimators, mean\_test\_scores, marker='o', color='b', linestyle='-', markersize=6)

plt.title('Effect of n\_estimators on Accuracy in Random Forest (MNIST)', fontsize=14)

plt.xlabel('Number of Estimators', fontsize=12)

plt.ylabel('Mean Test Accuracy', fontsize=12)

plt.grid(True)

# 保存图像为图片

plt.savefig('random\_forest\_n\_estimators.png')

# 显示图像

plt.show()

# 输出最佳参数

print("Best parameters found: ", grid\_search.best\_params\_)

# 使用最佳模型进行预测

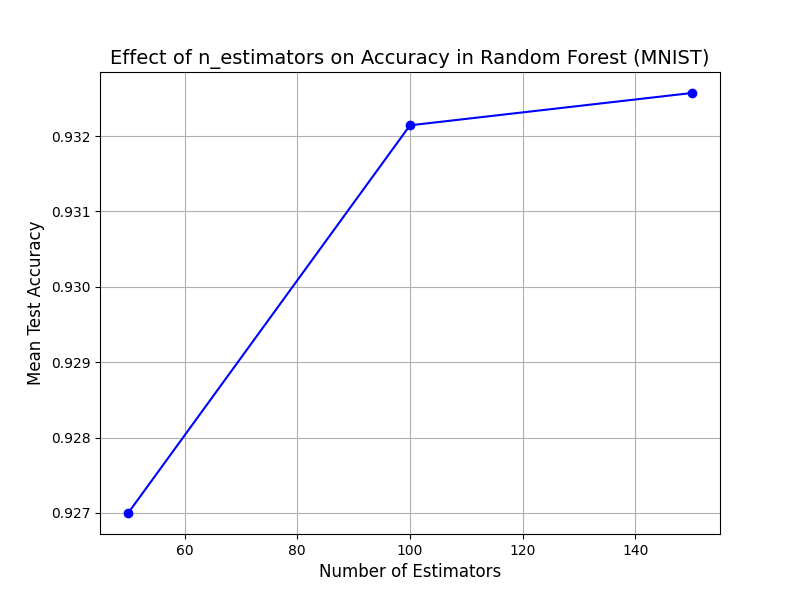
best\_rf = grid\_search.best\_estimator\_

y\_pred = best\_rf.predict(X\_test)

# 打印分类报告

print("Accuracy on Test Set: ", accuracy\_score(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))



随机森林树深度调参

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_openml

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report

import os

# 指定一个有权限的文件夹路径来存储数据集

data\_home = os.path.join(os.getcwd(), 'sklearn\_data') # 例如，当前工作目录下的 'sklearn\_data' 文件夹

# 加载MNIST数据集并指定存储路径

mnist = fetch\_openml("mnist\_784", data\_home=data\_home)

# 获取数据和标签

X, y = mnist["data"], mnist["target"].astype(int)

# 只选取部分训练数据（减少数据量）

X\_train, \_, y\_train, \_ = train\_test\_split(X, y, train\_size=0.1, random\_state=42) # 使用10%的数据

X\_train /= 255.0 # 归一化

X\_test = X[::100] # 使用每100个数据点中的一个作为测试集（减少测试集大小）

y\_test = y[::100] # 测试集标签

X\_test /= 255.0 # 归一化

# 设置调参范围：max\_depth的不同值

param\_grid = {

'n\_estimators': [100], # 设置一个固定的树的数量

'max\_depth': [None, 10, 20, 30, 40, 50] # 不同的树深度值

}

# 创建RandomForestClassifier模型

rf = RandomForestClassifier(random\_state=42)

# 使用较少的交叉验证折数（cv=2）以减少计算时间

grid\_search = GridSearchCV(estimator=rf, param\_grid=param\_grid, cv=2, verbose=1, n\_jobs=1, scoring='accuracy')

# 训练模型

grid\_search.fit(X\_train, y\_train)

# 获取调参结果

results = grid\_search.cv\_results\_

mean\_test\_scores = results['mean\_test\_score']

max\_depth\_values = results['param\_max\_depth'].data

# 绘制性能图：max\_depth与准确率的关系

plt.figure(figsize=(8, 6))

plt.plot(max\_depth\_values, mean\_test\_scores, marker='o', color='b', linestyle='-', markersize=6)

plt.title('Effect of max\_depth on Accuracy in Random Forest (MNIST)', fontsize=14)

plt.xlabel('max\_depth', fontsize=12)

plt.ylabel('Mean Test Accuracy', fontsize=12)

plt.grid(True)

# 保存图像为图片

plt.savefig('random\_forest\_max\_depth.png')

# 显示图像

plt.show()

# 输出最佳参数

print("Best parameters found: ", grid\_search.best\_params\_)

# 使用最佳模型进行预测

best\_rf = grid\_search.best\_estimator\_

y\_pred = best\_rf.predict(X\_test)

# 打印分类报告

print("Accuracy on Test Set: ", accuracy\_score(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

# 

# 致 谢

感谢小组成员的辛勤工作，感谢老师的提点指导。