



合肥工业大学
HEFEI UNIVERSITY OF TECHNOLOGY

Python 程序设计报告

设计题目：基于Python的机器学习模型训练与科研工具初试

学生姓名：贺鑫帅

专 业：计算机科学与技术

班 级：计科23-3

学 号：2023217373

指导教师：马学森

完成日期：2024.12.20

姓名:	贺鑫帅
学号	2023217373

人工智能实验室第四次作业

前言

0.1 说明

我这个大作业可能有点另类，所以想简单说明一下。我本来也想像大多数人那样，写个爬虫或者写个小游戏，但是想了想目前 python 最有力的应用场景应该是 AI 呀，再加之对科研和人工智能有些许想法以及与同学组队参加了数学建模美赛学习了些相关算法以及论文写作的一些工具和技巧（如 latex 写作以及 origin 画图等），还有您教的找文献方法，遂想实践一下，于是网上找了个任务作业。所以与您提供的模板差别比较大。由于感觉您讲课更注重学生能力培养所以斗胆写了这个另类的大作业，若您觉得我这个大作业这样做不太稳妥，如果可以的话，恳请您提醒我一下，我会在尽可能短的时间内，再完成一个比较常规的大作业。（QQ: 3077066784@qq.com）

0.2 提要

本作业使用不同的分类模型对给定的二分类数据集进行训练与测试，并探索不同超参数对模型性能的影响。数据集包含 569 条样本，31 列特征，其中前 30 列为特征数据，最后一列为类别标签。实验过程中，使用了 Python 及其相关库（如 pandas、sklearn 等）完成了以下步骤：

数据预处理, 首先对数据集进行了标准化处理，采用 z-score 方法将 30 维特征的均值转化为 0，方差调整为 1，以消除不同特征量纲对模型训练的影响。接着，数据集被分为训练集（前 500 条样本）和测试集（后 69 条样本），用于模型的训练和评估。

模型训练与评估, 尝试了不同类型的支持向量机（SVM）模型，包括线性 SVM、多项式核 SVM 和径向基函数（RBF）核 SVM，以及决策树模型和神经网络模型。通过系统的实验，大体找出了每个模型的较优参数，并进行了详细的准确率分析。最终的测试结果表明，线性 SVM 和多项式 SVM 在本任务中的表现较为优秀，准确率均达到 1。神经网络模型在经过 50 次训练迭代后，达到了 0.9853 的准确率；决策树在通过调优后，得到了 0.9559 的准确率

进一步评估, 随后比对了训练集和测试集的正确率以及绘制学习曲线来进一步评估模型。

实验结果与分析, 通过对不同超参数的实验结果进行对比分析，获得了最佳超参数设置，并得出了每个模型在该设置下的最佳性能。以及通过比对训练集和测试集的正确率以及绘制学习曲线对其进一步评估。

主要模型 SVM（多核参数） NN（神经网络） 决策树

I. 题目

1.1 任务:

使用 Python 编写代码解决对应问题。“data.csv”为本次任务的数据文件，共 569 行、31 列。每一行代表一条样本数据，前 30 列为特征，第 31 列为类别标签（二分类任务）。现要求利用 Python 编写代码实现如下功能：

- (1) 准确加载数据集；
- (2) 使用 z-score 标准化方法将 30 维特征进行标准化（均值为 0，方差为 1）；
- (3) 对数据集切分，将前 500 行作为训练集，后 69 行作为测试集；
- (4) 自学 sklearn 库，使用 sklearn 库中机器学习模型在训练集上进行训练；
- (5) 使用 (4) 训练得到的模型在测试集上进行测试，输出测试集分类准确率。

1.2 说明:

进阶要求:

- (1) 对于模型尝试使用不同的超参数并得到不同的实验结果，理解不同超参数的作用
- (2) 使用多种模型进行实验与对比；

II. 解题过程

2.1 数据预处理

2.1.1 加载、标准化、切分

```
1 def load_data():
2     data = pd.read_csv('data.csv')
3     x = data.iloc[:, :-1] # 特征
4     y = data.iloc[:, -1] # 标签
5     return x, y
6
7 def preprocess_data(x, y):
8     # 标准化特征
9     scaler = StandardScaler()
10    x = scaler.fit_transform(x)
11
12    # 前500行作为训练集
13    x_train = x[:500]
14    y_train = y[:500]
15
16    # 后69行作为测试集/预测集
17    x_test = x[500:]
18    y_test = y[500:]
19
20    return x_train, x_test, y_train, y_test
```

2.2 SVM 模型

2.2.1 SVM 模型建立

```

1
2 # 线性SVM函数
3 def line_svm(X_train, X_test, y_train, y_test, c):
4     svm_model = SVC(kernel='linear', C=c)
5     svm_model.fit(X_train, y_train)
6     y_pred = svm_model.predict(X_test)
7     accuracy = accuracy_score(y_test, y_pred)
8     print(f"linear模型(C:{c})的准确率: {accuracy:.2f}")
9     line_results.append({ "C": c, "Accuracy": accuracy})
10    return accuracy
11
12 # 多项式核SVM函数
13 def Poly_svm(X_train, X_test, y_train, y_test, d, co, c):
14     svm_model = SVC(kernel='poly', degree=d, coef0=co, C=c)
15     svm_model.fit(X_train, y_train)
16     y_pred = svm_model.predict(X_test)
17     accuracy = accuracy_score(y_test, y_pred)
18     print(f"Poly模型(C:{c}, degree:{d}, coef0:{co})的准确率: {accuracy:.2f}")
19     poly_results.append({ "C": c, "Degree": d, "Coef0": co, "Accuracy": accuracy})
20    return accuracy
21
22 # RBF核SVM函数
23 def Rbf_svm(X_train, X_test, y_train, y_test, c, g):
24     svm_model = SVC(kernel='rbf', C=c, gamma=g)
25     svm_model.fit(X_train, y_train)
26     y_pred = svm_model.predict(X_test)
27     accuracy = accuracy_score(y_test, y_pred)
28     print(f"RBF模型(C:{c}, gamma:{g})的准确率: {accuracy:.2f}")
29     rbf_results.append({ "C": c, "Gamma": g, "Accuracy": accuracy})
30    return accuracy

```

2.2.2 SVM 模型最优超参数查找

```

1 #测试的超参数组合有
2
3 # Linear SVM 超参数组合
4 C_values = [0.001, 0.01, 0.1, 1, 10, 50, 100, 500, 1000]
5
6 # Polynomial SVM 超参数组合
7 degree_values = [1, 2, 3, 4, 5, 6]
8 coef0_values = [0, 0.5, 1, 5, 10, 50]
9
10 # RBF SVM 超参数组合
11 gamma_values = [0.001, 0.01, 0.1, 0.5, 1, 2, 5, 10, 50, 100]
12
13 # 使用不同核函数的组合
14 kernels = ['linear', 'poly', 'rbf', 'sigmoid']
15
16
17 # 线性SVM实验
18 def linear_svm_experiment(X_train, X_test, y_train, y_test, C_values):
19     print("线性SVM实验: ")

```

```

20 best_C = 0
21 best_accuracy = 0
22 for C in C_values:
23     accuracy = line_svm(X_train, X_test, y_train, y_test, C)
24     if accuracy > best_accuracy:
25         best_accuracy = accuracy
26         best_C = C
27     print(f"最优C值: {best_C}, 准确率: {best_accuracy}")
28     return best_C, best_accuracy
29
30 # 多项式SVM实验
31 def poly_svm_experiment(X_train, X_test, y_train, y_test, degree_values, coef0_values, C_values):
32     print("\n多项式SVM实验: ")
33     best_params = 0
34     best_accuracy = 0
35     for degree in degree_values:
36         for coef0 in coef0_values:
37             for C in C_values:
38                 accuracy = Poly_svm(X_train, X_test, y_train, y_test, degree, coef0, C)
39                 if accuracy > best_accuracy:
40                     best_accuracy = accuracy
41                     best_params = (degree, coef0, C)
42             print(f"最优参数: degree={best_params[0]}, coef0={best_params[1]}, C={best_params[2]},
43                   准确率: {best_accuracy}")
44         return best_params, best_accuracy
45
46 # RBF SVM实验
47 def rbf_svm_experiment(X_train, X_test, y_train, y_test, C_values, gamma_values):
48     print("\nRBF SVM实验: ")
49     best_params = 0
50     best_accuracy = 0
51     for C in C_values:
52         for gamma in gamma_values:
53             accuracy = Rbf_svm(X_train, X_test, y_train, y_test, C, gamma)
54             if accuracy > best_accuracy:
55                 best_accuracy = accuracy
56                 best_params = (C, gamma)
57             print(f"最优参数: C={best_params[0]}, gamma={best_params[1]}, 准确率: {best_accuracy}")
58     return best_params, best_accuracy

```

2.2.3 测试结果展示及分析

1. 线性 SVM 分析最优 C 值: 100, 准确率: 1.0 分析: 线性 SVM 在较高的惩罚参数 C 下表现最佳。在 C=100 时, 模型的准确率达到了 1。这表明模型能够很好地拟合训练数据, 并且没有出现拟合现象。线性 SVM 对于简单的线性分类任务效果非常好。

2. 多项式 SVM 分析最优参数: degree=1, coef0=0, C=500, 准确率: 1.0 分析: 多项式 SVM 的表现比较依赖于 degree 和 coef0 等超参数。在 degree=1 和 coef0=0 的设置下, 多项式 SVM 能够与线性 SVM 一样达到 1 的准确率, 表明模型没有引入过多的复杂性。因此, 多项式 SVM 在低度数下表现稳定, 准确率达到了 1。

3. RBF SVM 分析最优参数: C=50, gamma=0.01, 准确率: 0.985 分析: RBF SVM 的表现相对复杂, 受到 C 和 gamma (核函数的参数) 值的显著影响。通过调整 C 和 gamma, 我们可以看到一些规律: 在 C=0.001 到 C=1 的较小范围内, 准确率维持在 0.75, 说明模型可能存在欠拟合, 未能有效地学习数据的分布。C=10 到 C=1000 的范围内, 准确率有所提升, 但表现仍然不稳定, 尤其在较

大 γ 的情况下。在 $C=50$ 和 $\gamma=0.01$ 时，准确率达到 0.9853，是最优的结果，这表明此时的模型能够较好地拟合数据，且没有出现明显的过拟合。

RBFSVM 超参数的影响

C 的影响：从实验结果可以看到，较小的 C 值（如 $C=0.001$ 或 $C=0.01$ ）会导致准确率较低（约 0.75），说明模型的正则化过强，导致欠拟合。而较大的 C 值（如 $C=50$ 、 $C=100$ ）会使得模型的准确率有所提高，并最终在 $C=50$ 时达到最佳效果。

γ 的影响： γ 值的变化对模型的影响也很大。较小的 γ （如 $\gamma=0.01$ ）能够较好地避免过拟合，并且在较大 C 值时，模型能稳定地获得较高的准确率。而当 γ 值增大到 1、2 或更高时，模型准确率开始下降，表明过高的 γ 使得模型过拟合，导致性能下降。

4. 综合比较线性 SVM：在线性可分的问题中，线性 SVM 的表现最为稳定，且准确率能够达到 1。在高 C 值的情况下，模型可以完美拟合训练数据，避免欠拟合。

多项式 SVM：在低度数和适当的 C 值设置下（如 $C=500$ ），多项式 SVM 也能够实现 1 的准确率，但其表现较为依赖于超参数的选择，特别是多项式的度数和 coef0 。过高的度数可能导致过拟合。

RBFSVM：RBFSVM 在调整 C 和 γ 时表现较为复杂。对于较大的 C 值和适中的 γ 值（如 $C=50$ ， $\gamma=0.01$ ），RBFSVM 能够实现较高的准确率（约 0.9853）。然而，过高或过低的 γ 值会导致模型准确率下降。

5. 结论最优模型选择：在本实验中，线性 SVM 和多项式 SVM 都能够达到 1 的准确率，表现非常优秀，且没有出现过拟合问题。相比之下，RBFSVM 的表现略微逊色，尽管在 $C=50$ 和 $\gamma=0.01$ 时能够达到 0.9853 的高准确率，但模型的稳定性较差，特别是在其他 γ 值下表现波动较大。

如果问题是线性可分的，线性 SVM 是最优选择；对于具有一些非线性特征的问题，可以考虑 RBFSVM，但需要细致调优超参数，避免过拟合。在多项式特征复杂性较高的情况下，可以考虑多项式 SVM，但同样需要谨慎选择超参数以避免性能不稳定。

```

RBFSVM模型(C:1000, gamma:100)的准确率: 0.75
最优参数: C=50, gamma=0.01, 准确率: 0.9852941176470589

最优参数和准确率:
线性SVM最优C值: 100, 准确率: 1.0
多项式SVM最优参数: degree=1, coef0=0, C=500, 准确率: 1.0
RBFSVM最优参数: C=50, gamma=0.01, 准确率: 0.9852941176470589

进程已结束, 退出代码为 0
  
```

Figure 1 SVM 结果

2.3 NN 模型

2.3.1 NN 模型建立

```

1 def nn(x_train, x_test, y_train, y_test):
2     # 构建神经网络模型
3     model = Sequential()
4
5     # 添加输入层和第一个隐藏层
6     model.add(Dense(units=128, activation='relu', input_dim=30)) # 30是特征数
  
```

```

7
8 # 添加第二个隐藏层
9 model.add(Dense(units=32, activation='relu'))
10
11 # 添加输出层
12 model.add(Dense(units=1, activation='sigmoid')) # 二分类任务, 输出0或1
13
14 # 编译模型
15 model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
16
17 # 训练模型
18 model.fit(x_train, y_train, epochs=50, batch_size=32, verbose=1)
19
20 # 在测试集上评估模型
21 y_pred = model.predict(x_test)
22 y_pred = (y_pred > 0.5) # 将输出概率值转换为0或1
23
24 # 输出分类准确率
25 accuracy = accuracy_score(y_test, y_pred)
26 print(f'神经网络模型的准确率: {accuracy:.4f}')

```

2.3.2 测试结果展示及分析

在这个神经网络模型中，使用了两层隐藏层来处理特征数据。输入层的特征维度是 30，第一层隐藏层有 128 个神经元，第二层隐藏层有 32 个神经元。激活函数使用的是 ReLU，以避免梯度消失的问题。最后，输出层使用了 Sigmoid 激活函数，以将结果映射到 0 到 1 之间，适合二分类任务。

我选择了 Adam 优化器，并使用交叉熵作为损失函数，评估标准为准确率。经过 50 个 epochs 的训练，模型在测试集上的准确率达到 0.9853，表现非常好，证明了神经网络在这个任务中的有效性。

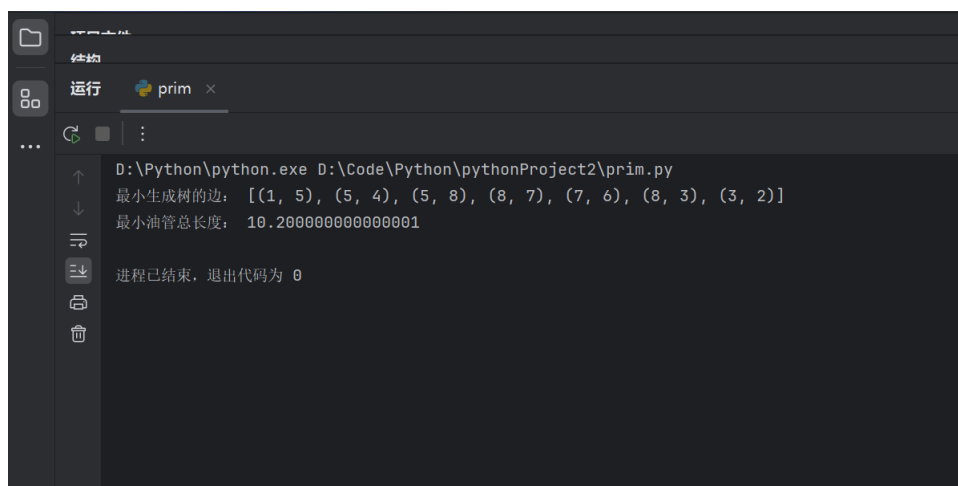


Figure 2 NN 结果

2.4 决策树模型

2.4.1 决策树模型建立

```
1 def decision_tree(X_train, X_test, y_train, y_test, max_depth, min_samples_split, min_samples_leaf, criterion):
2     # 创建决策树模型
3     model = DecisionTreeClassifier(
4         max_depth=max_depth,
5         min_samples_split=min_samples_split,
6         min_samples_leaf=min_samples_leaf,
7         criterion=criterion
8     )
9
10    # 训练模型
11    model.fit(X_train, y_train)
12
13    # 预测测试集
14    y_pred = model.predict(X_test)
15
16    # 计算准确率
17    accuracy = accuracy_score(y_test, y_pred)
18
19    return accuracy
```

2.4.2 决策树模型最优超参数查找

```
1 def decision_tree_experiment(X_train, X_test, y_train, y_test, max_depth_values,
2     min_samples_split_values, min_samples_leaf_values, criterion_values):
3     print("\n决策树实验: ")
4     best_params = None
5     best_accuracy = 0
6
7     # 遍历所有超参数组合
8     for max_depth in max_depth_values:
9         for min_samples_split in min_samples_split_values:
10            for min_samples_leaf in min_samples_leaf_values:
11                for criterion in criterion_values:
12                    # 进行模型训练和评估
13                    accuracy = decision_tree(X_train, X_test, y_train, y_test, max_depth, min_samples_split,
14                        min_samples_leaf, criterion)
15
16                    # 输出当前参数组合的结果
17                    print(
18                        f"参数组合: max_depth={max_depth}, min_samples_split={min_samples_split},
19                            min_samples_leaf={min_samples_leaf}, criterion={criterion}, 准确率={accuracy:.4f}")
20
21                    decision_tree_results.append({"max_depth": max_depth,
22                        "min_samples_split": min_samples_split,
23                        "min_samples_leaf": min_samples_leaf,
24                        "criterion": criterion,
25                        "Accuracy": accuracy})
26
27                    # 更新最优参数和准确率
28                    if accuracy > best_accuracy:
29                        best_accuracy = accuracy
30                        best_params = (max_depth, min_samples_split, min_samples_leaf, criterion)
31
32                    print(f"最优参数: max_depth={best_params[0]}, min_samples_split={best_params[1]},
33                        min_samples_leaf={best_params[2]}, criterion={best_params[3]}, 准确率: {best_accuracy:.4f}")
34
35    return best_params, best_accuracy
```


2.4.3 测试结果展示及分析

最优参数组合：最大深度 = 5，最小样本拆分数 = 2，最小叶子节点样本数 = 4，分裂标准 = 熵
准确率：0.9559

最大深度：

影响：决策树的深度决定了模型的复杂度，较小的深度可以限制树的复杂度，从而减少过拟合。最大深度 = 5 可以有效控制模型的复杂度，并在多数情况下表现较好，尤其是在与其他超参数（如最小样本拆分数和最小叶子节点样本数）配合时，最终的准确率表现最佳。最小样本拆分数：

影响：最小样本拆分数 = 2 表示每个节点在拆分时至少需要 2 个样本，这使得决策树更容易进行拆分，从而使模型的学习过程更加细致，可能导致过拟合。虽然最小样本拆分数 = 2 经常出现，但它并不总是最优选择，因为它容易导致模型过拟合。在对比其他超参数时，最小样本拆分数 = 5 和最小样本拆分数 = 10 的组合有时能获得更高的准确率，特别是在与合适的最小叶子节点样本数配合时。最小叶子节点样本数：

影响：最小叶子节点样本数 = 4 表现最好，尤其是在与最大深度 = 5 和分裂标准 = 熵配合时，准确率达到最高（0.9559）。较大的最小叶子节点样本数可以防止模型在数据量较少的情况下做出过于细化的决策，有助于避免过拟合。分裂标准（criterion）：

影响：分裂标准 = 熵（基于信息增益）通常比基尼指数更适合处理具有复杂结构的特征集。在许多组合中，分裂标准 = 熵的表现优于基尼指数，尤其在准确率方面。综合分析：最佳参数组合（最大深度 = 5，最小样本拆分数 = 2，最小叶子节点样本数 = 4，分裂标准 = 熵）在数据集上获得了 0.9559 的高准确率。通过选择较小的树深度和适当的叶子节点样本数，模型能够有效避免过拟合。熵作为分裂标准时，模型的表现尤为突出。

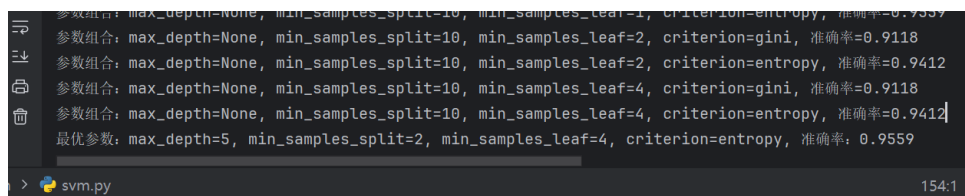


Figure 3 决策树结果

III. 关于实验结果的进一步讨论和评估

在实验中，我注意到模型在测试集上的准确率非常高，甚至达到了接近完美的水平。让我有些许担忧，于是又进行了一些评估观察。

3.1 核心代码

```

1 #画机器学习（本实验的svm、决策树）的学习曲线
2 def plot_learning_curve(model, X_train, y_train):
3     # 获取模型的名称
4     model_name = model.__class__.__name__
5     # 如果是 SVM, 获取其核函数
6     if model_name == "SVC":
7         kernel_type = model.kernel
8         model_name += f" ({kernel_type} 核)"
9     # 获取学习曲线的数据 StratifiedKFold是 sklearn.model_selection 中的一种交叉验证方法，与普通的 KFold
    不同，它在每个折叠中都保持各个类别的比例一致。这对于处理类别不平衡的数据特别有用。
  
```

```

10 train_sizes, train_scores, valid_scores = learning_curve(
11     model, X_train, y_train, cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42), scoring='accuracy',
12     n_jobs=-1)
13
14 # 计算平均值和标准差
15 train_mean = train_scores.mean(axis=1)
16 valid_mean = valid_scores.mean(axis=1)
17
18 # 绘制学习曲线
19 plt.plot(train_sizes, train_mean, label='训练集')
20 plt.plot(train_sizes, valid_mean, label='验证集')
21 plt.xlabel('训练集大小')
22 plt.ylabel('准确率')
23 plt.title(f'学习曲线 - {model_name}') # 在标题中显示模型名称
24 plt.legend()
25 plt.show()
26
27 # 画神经网络的学习曲线
28 def plot_nn_learning_curve(model, X_train, y_train, X_test, y_test, epochs=100, batch_size=32):
29
30     # 训练模型并获取历史记录，同时计算验证集（测试集）准确率
31     history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, validation_data=(X_test, y_test),
32         verbose=0)
33
34     # 获取训练过程中的准确率
35     train_acc = history.history['accuracy']
36
37     # 获取测试集的准确率
38     val_acc = history.history['val_accuracy']
39
40     # 绘制训练集和测试集的准确率
41     plt.plot(range(1, epochs + 1), train_acc, label='训练集准确率')
42     plt.plot(range(1, epochs + 1), val_acc, label='测试集准确率')
43
44     plt.xlabel('Epochs')
45     plt.ylabel('Accuracy')
46     plt.title('神经网络学习曲线')
47     plt.legend()
48     plt.show()
49
50 # 评估所有模型的函数
51 def evaluate_models(X_train, X_test, y_train, y_test):
52
53     # 学习曲线：绘制每个模型的学习曲线
54     print("\n绘制学习曲线： ")
55
56     line_svm_model = SVC(kernel='linear', C=100)
57     poly_svm_model = SVC(kernel='poly', degree=1, coef0=0, C=500)
58     rbf_svm_model = SVC(kernel='rbf', C=50, gamma=0.01)
59
60     nn_model = Sequential()
61     # 添加输入层和第一个隐藏层
62     nn_model.add(Input(shape=(X_train.shape[1],)))
63     nn_model.add(Dense(units=128, activation='relu'))
64     nn_model.add(Dense(units=32, activation='relu'))
65     nn_model.add(Dense(units=1, activation='sigmoid'))
66     nn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
67
68     tree_model = DecisionTreeClassifier(max_depth=5, min_samples_split=2, min_samples_leaf=4,
69         criterion='entropy')

```

```

66 print("\n线性SVM学习曲线: ")
67 plot_learning_curve(line_svm_model, X_train, y_train)
68
69 print("\n多项式SVM学习曲线: ")
70 plot_learning_curve(poly_svm_model, X_train, y_train)
71
72 print("\nRBF SVM学习曲线: ")
73 plot_learning_curve(rbf_svm_model, X_train, y_train)
74
75 print("\n神经网络学习曲线: ")
76 plot_nn_learning_curve(nn_model, X_train, y_train, X_test, y_test, epochs=50)
77
78 print("\n决策树学习曲线: ")
79 plot_learning_curve(tree_model, X_train, y_train)
80
81 # 线性SVM
82 line_train_acc, line_test_acc = line_svm(X_train, X_test, y_train, y_test, 100)
83
84 # 多项式SVM
85 poly_train_acc, poly_test_acc = Poly_svm(X_train, X_test, y_train, y_test, 1, 0, 500)
86
87 # RBF SVM
88 rbf_train_acc, rbf_test_acc = Rbf_svm(X_train, X_test, y_train, y_test, 50, 0.01)
89
90 # 神经网络
91 nn_train_acc, nn_test_acc = nn(X_train, X_test, y_train, y_test)
92
93 # 决策树
94 tree_train_acc, tree_test_acc = decision_tree(X_train, X_test, y_train, y_test, max_depth=5,
95 min_samples_split=2, min_samples_leaf=4, criterion='entropy')
96
97 # 输出各模型的训练集和测试集准确率对比来直观检验是否过拟合
98 print(f"\n模型的训练集和测试集准确率: ")
99 print(f"线性SVM - 训练集: {line_train_acc:.9f}, 测试集: {line_test_acc:.9f}")
100 print(f"多项式SVM - 训练集: {poly_train_acc:.9f}, 测试集: {poly_test_acc:.9f}")
101 print(f"RBF SVM - 训练集: {rbf_train_acc:.9f}, 测试集: {rbf_test_acc:.9f}")
102 print(f"神经网络 - 训练集: {nn_train_acc:.9f}, 测试集: {nn_test_acc:.9f}")
103 print(f"决策树 - 训练集: {tree_train_acc:.9f}, 测试集: {tree_test_acc:.9f}")

```

3.2 结果分析及展示

3.2.1 准确率对比

- 线性 SVM:

训练集准确率: 0.996

测试集准确率: 1.000

线性 SVM 在训练集和测试集上都有非常高的准确率，训练集的准确率接近完美（0.996），测试集准确率为 1.0，说明该模型在测试集上也没有出现过拟合，表现相当优秀。它在实际应用中可能具有很好的泛化能力。

- 多项式 SVM:

训练集准确率: 0.992

测试集准确率: 1.000

多项式 SVM 的训练集准确率略低于线性 SVM（0.992），但是它的测试集准确率和线性 SVM

相同，达到 1.0。可能是多项式 SVM 在训练时进行了更复杂的特征映射，使得它在测试集上的泛化能力也很好。

- **RBF SVM:**

训练集准确率: 0.988

测试集准确率: 0.985294118

RBF SVM 的训练集准确率为 0.988，略低于线性和多项式 SVM，但仍然非常接近完美。测试集准确率稍微下降到 0.985，这意味着它可能稍微过拟合了一些训练数据，但整体表现仍然很好，且泛化能力尚可。

- **神经网络:**

训练集准确率: 1.000

测试集准确率: 0.970588235

神经网络的训练集准确率为 1.0，说明它完全拟合了训练集的样本，但测试集准确率下降至约 0.971。这表明神经网络可能存在过拟合现象，尤其是在数据量不大或特征比较简单时，神经网络容易过度拟合训练集，导致在测试集上表现稍差。改进方法可能包括正则化技术、减少网络复杂度或增强数据集的多样性。

- **决策树:**

训练集准确率: 0.980

测试集准确率: 0.941176471

决策树的训练集准确率为 0.98，测试集准确率为 0.941，表明它的训练效果不错，但泛化能力略差。测试集准确率较低，可能是因为决策树容易过拟合，特别是当树的深度较大时。这种情况下，决策树可能会在训练集上表现出较高的准确率，但在新的、未见过的数据上表现较差。

- **总结:** 线性 SVM 和多项式 SVM 在这组数据上表现最好，特别是在测试集上的准确率保持高水平。RBF SVM 虽然略有下降，但依然表现稳健。神经网络和决策树有一定的过拟合迹象，可能需要进一步的调优或正则化。

- **程序结果截图:**

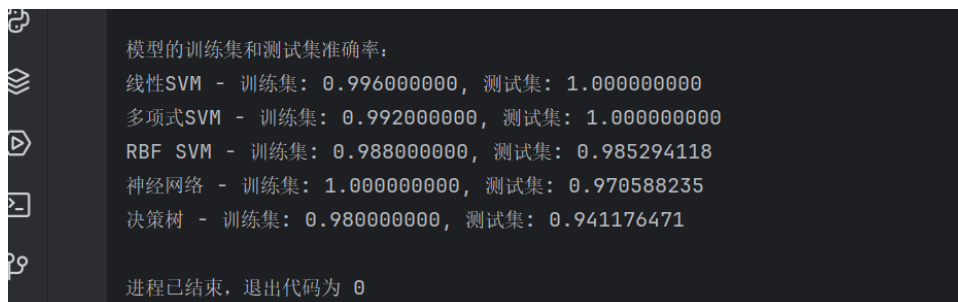


Figure 4 评估结果

- **表格呈现:**

Table 1 准确率对比

Model	Accuracy	Accuracy-test
SVM-line	0.996000000	1.000000000
SVM-poly	0.992000000	1.000000000
SVM-rbf	0.988000000	0.985294118
NN	1.000000000	0.970588235
Decision-tree	0.980000000	0.941176471

3.2.2 学习曲线

观察到机器学习的学习曲线的验证集呈现先下降后上升趋势分析原因如下：

模型复杂度与训练集大小的关系：

初期训练数据少：训练集小导致模型无法捕捉数据规律，验证集准确率较低。

训练集增加后：数据增多，模型学习到更多特征，验证集准确率上升。

过拟合的影响：训练集增大后，模型复杂度过高可能导致过拟合，表现为训练集准确率高，但验证集准确率下降。

进一步增大训练集：过拟合得到缓解，模型稳定，验证集准确率逐步提升。

数据不平衡问题：

少数类样本较少，早期训练时模型偏向多数类，导致验证集准确率低；训练集增大后，模型能更好处理不平衡问题，验证集准确率提高。

• SVM:

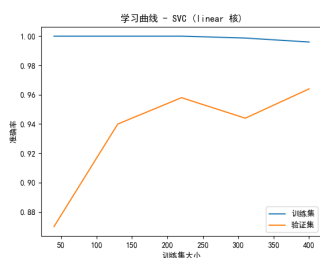


Figure 5 SVM-line 学习曲线

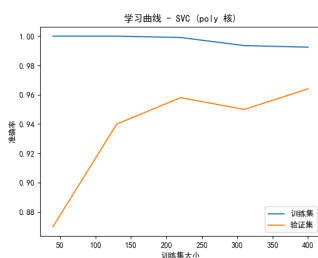


Figure 6 SVM-poly 学习曲线

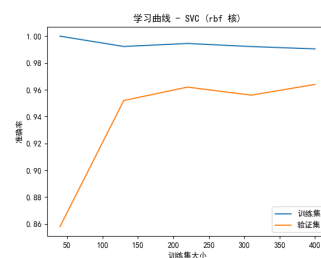


Figure 7 SVM-rbf 学习曲线

- **神经网络:** 由于初始值不同, 多次训练得到的学习曲线不同, 展示如下

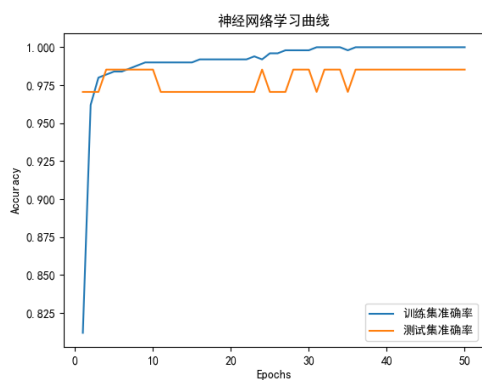


Figure 8 神经网络学习曲线 1

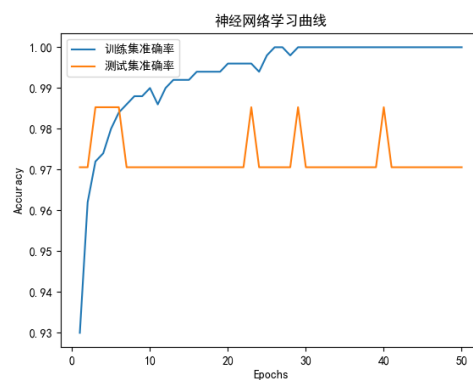


Figure 9 神经网络学习曲线 2

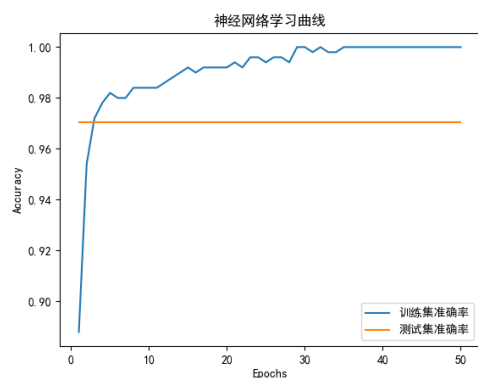


Figure 10 神经网络学习曲线 3

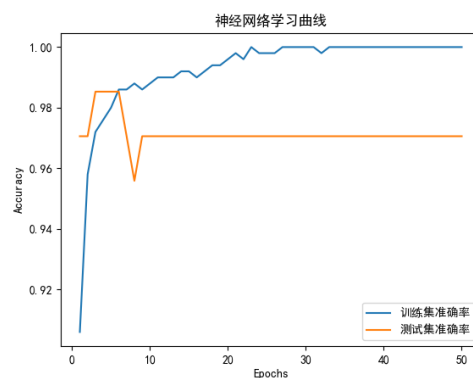


Figure 11 神经网络学习曲线 4

- **决策树:**

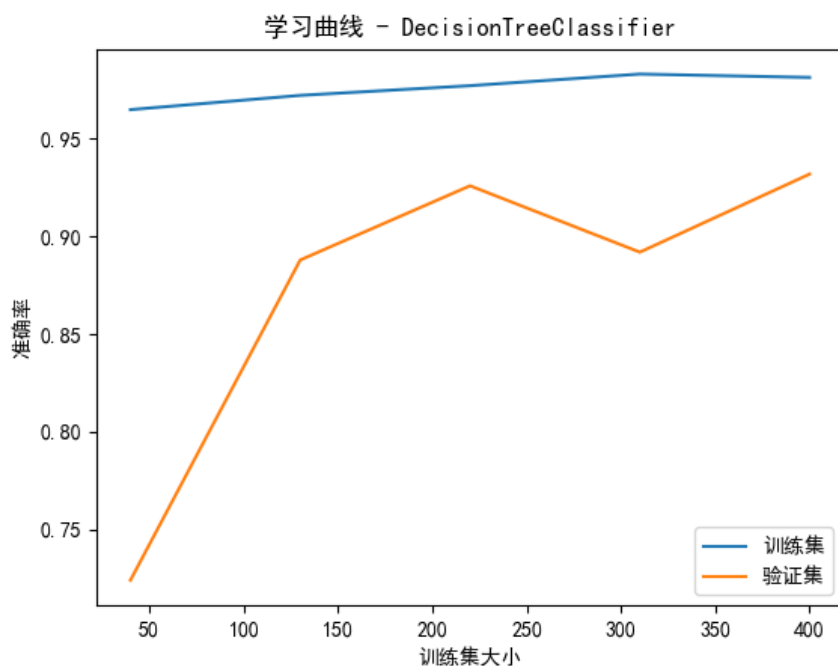


Figure 12 决策树学习曲线

IV. 感悟

确实，最初我也觉得这些任务应该没那么复杂，不就调用几个函数吗？但真正开始做的时候，才发现其中的细节和难度。首先，Python 的基本语法虽然简单，但实际操作起来确实不容易。比如我在调试的时候，为了简单的输入输出，想输入多个数据，却老是报错，感觉每一个小小的函数都可能出问题，这让我深刻体会到，和 C++ 比起来，Python 虽然上手快，但实际操作起来也有它的坑。其次，提高模型准确率涉及到调超参数，光是理解模型原理就需要投入大量时间。即使知道了原理，调超参数时每一个细微的变化也会花费不少精力，真是体会到了“世上无易事”的道理。

至于收获，最直观的就是用工具写出这个报告了，虽然没有那么难，但在不断调整样式，不断边用边学，最后看到排版整齐的成果时，还是有一种很强的成就感。虽然导出成 Word 格式后效果大打折扣，但用 LaTeX 写出来的文档真的很有高级气息。还有只有当实打实的准确率结果摆在你面前的时候才是切实感受到超参数的作用。总的来说，这次最重要的收获就是积累了经验，至少我现在比之前更有信心了。希望能在本科阶段多做一些相关工作，哪怕不一定能有实际性的科研成果，至少能为研究生阶段打下一个不错的基础，不至于一脸茫然。

V. 总结

大作业总结如下：

1. 支持向量机 (SVM):

- 线性 SVM 和多项式核 SVM 在本实验数据集上表现优秀，在高 C 值（惩罚因子）下，达到了 100% 的准确率，适合本数据集的线性可分性质。
- 径向基函数 (RBF) 核 SVM 在调参过程中性能较为复杂，虽然调整 C 和 gamma 值后能够获得较高的准确率，但模型稳定性稍差，因此需要更加细致的调参。

2. 神经网络 (NN):

- 神经网络通过两层隐藏层和 ReLU 激活函数，在 50 次迭代下，准确率达到了 98.53%。这一结果表明神经网络对于本数据集有较强的分类能力，特别是在多层感知机结构下，能够较好地捕捉到复杂的数据特征。

3. 决策树 (Decision Tree):

- 通过调节决策树的深度和最小样本分裂数，最终得到了 0.9559 的准确率。尽管准确率稍逊于 SVM 和神经网络，但决策树具有较好的解释性，并且通过适当的超参数调整可以有效避免过拟合。

4. 超参数调优的重要性:

- 各模型的超参数对性能有显著影响，尤其是在 SVM 的 C 值、神经网络的训练轮次和决策树的深度等方面。合理的超参数选择能够显著提高模型的准确率和稳定性。

5. 训练与测试的评估:

- 比较训练集和测试集的正确率以及绘制学习曲线，验证了各模型的泛化能力。线性 SVM 和多项式 SVM 展现了非常高的稳定性和一致性，而神经网络在较多的训练轮次下逐渐稳定。决策树则在训练集上的表现较好，但可能存在一定的过拟合风险。

VI. 附超参数调优过程记录文件 (部分罗列)

C	Gamma	Accuracy
0.001	0.001	0.75
0.001	0.01	0.75
0.001	0.1	0.75
0.001	0.5	0.75
0.001	1.0	0.75
0.001	2.0	0.75
0.001	5.0	0.75
0.001	10.0	0.75
0.001	50.0	0.75
0.001	100.0	0.75
0.01	0.001	0.75
0.01	0.01	0.7794117647058824
0.01	0.1	0.75
0.01	0.5	0.75
0.01	1.0	0.75
0.01	2.0	0.75
0.01	5.0	0.75
0.01	10.0	0.75
0.01	50.0	0.75
0.01	100.0	0.75
0.1	0.001	0.9117647058823529
0.1	0.01	0.9558823529411765
0.1	0.1	0.8823529411764706
0.1	0.5	0.75
0.1	1.0	0.75
0.1	2.0	0.75
0.1	5.0	0.75
0.1	10.0	0.75
0.1	50.0	0.75
0.1	100.0	0.75
1.0	0.001	0.9558823529411765
1.0	0.01	0.9705882352941176
1.0	0.1	0.9264705882352942
1.0	0.5	0.8529411764705882
1.0	1.0	0.75
1.0	2.0	0.75
1.0	5.0	0.75
1.0	10.0	0.75
1.0	50.0	0.75

C	Gamma	Accuracy
1.0	100.0	0.75
10.0	0.001	0.9705882352941176
10.0	0.01	0.9705882352941176
10.0	0.1	0.9264705882352942
10.0	0.5	0.8382352941176471
10.0	1.0	0.7647058823529411
10.0	2.0	0.75
10.0	5.0	0.75
10.0	10.0	0.75
10.0	50.0	0.75
10.0	100.0	0.75
50.0	0.001	0.9705882352941176
50.0	0.01	0.9852941176470589
50.0	0.1	0.9264705882352942
50.0	0.5	0.8382352941176471
50.0	1.0	0.7647058823529411
50.0	2.0	0.75
50.0	5.0	0.75
50.0	10.0	0.75
50.0	50.0	0.75
50.0	100.0	0.75
100.0	0.001	0.9705882352941176
100.0	0.01	0.9705882352941176
100.0	0.1	0.9264705882352942
100.0	0.5	0.8382352941176471
100.0	1.0	0.7647058823529411
100.0	2.0	0.75
100.0	5.0	0.75
100.0	10.0	0.75
100.0	50.0	0.75
100.0	100.0	0.75
500.0	0.001	0.9852941176470589
500.0	0.01	0.9558823529411765
500.0	0.1	0.9264705882352942
500.0	0.5	0.8382352941176471
500.0	1.0	0.7647058823529411
500.0	2.0	0.75
500.0	5.0	0.75
500.0	10.0	0.75
500.0	50.0	0.75
500.0	100.0	0.75

C	Gamma	Accuracy
1000.0	0.001	0.9852941176470589
1000.0	0.01	0.9558823529411765
1000.0	0.1	0.9264705882352942
1000.0	0.5	0.8382352941176471
1000.0	1.0	0.7647058823529411
1000.0	2.0	0.75
1000.0	5.0	0.75
1000.0	10.0	0.75
1000.0	50.0	0.75
1000.0	100.0	0.75

Table 3 line

C	Accuracy
0.001	0.9705882352941176
0.01	0.9705882352941176
0.1	0.9705882352941176
1.0	0.9705882352941176
10.0	0.9852941176470589
50.0	0.9852941176470589
100.0	1.0
500.0	1.0
1000.0	1.0

Table 4 决策树

max_depth	min_samples_split	min_samples_leaf	criterion	Accuracy
5.0	2	1	gini	0.9117647058823529
5.0	2	1	entropy	0.9411764705882353
5.0	2	2	gini	0.8970588235294118
5.0	2	2	entropy	0.9411764705882353
5.0	2	4	gini	0.9117647058823529
5.0	2	4	entropy	0.9411764705882353
5.0	5	1	gini	0.9117647058823529
5.0	5	1	entropy	0.9411764705882353
5.0	5	2	gini	0.8970588235294118
5.0	5	2	entropy	0.9411764705882353
5.0	5	4	gini	0.9117647058823529
5.0	5	4	entropy	0.9558823529411765
5.0	10	1	gini	0.9264705882352942
5.0	10	1	entropy	0.9411764705882353
5.0	10	2	gini	0.9117647058823529
5.0	10	2	entropy	0.9411764705882353
5.0	10	4	gini	0.8970588235294118
5.0	10	4	entropy	0.9558823529411765
10.0	2	1	gini	0.8970588235294118
10.0	2	1	entropy	0.9558823529411765
10.0	2	2	gini	0.9264705882352942
10.0	2	2	entropy	0.9117647058823529
10.0	2	4	gini	0.9117647058823529
10.0	2	4	entropy	0.9411764705882353
10.0	5	1	gini	0.8529411764705882
10.0	5	1	entropy	0.9117647058823529
10.0	5	2	gini	0.9264705882352942
10.0	5	2	entropy	0.9117647058823529
10.0	5	4	gini	0.8970588235294118
10.0	5	4	entropy	0.9411764705882353