安徽大学《机器学习》实验报告 2

学号：WA2214014　　专业：人工智能　　姓名：杨跃浙

实验日期：24.12.02　　教师签字：_____　　成绩：_____

[**实验名称**] 神经网络实验

[**实验目的**]

1．熟悉和掌握感知机神经网络

2．熟悉和掌握随机梯度下降算法

3．了解和掌握第三方机器学习库 Scikit-learn 中的模型调用

[**实验要求**]

1．采用 Python、Matlab 等高级语言进行编程，推荐优先选用 Python 语言

2．核心模型和算法需自主编程实现，不得直接调用 Scikit-learn、PyTorch 等成熟框架的第三方实现（除非实验内容明确指定调用）

3．代码可读性强：变量、函数、类等命名可读性强，包含必要的注释

4．提交实验报告要求：1）报告文件：命名为"学号-姓名-Lab2"； 2）提交时间：下次实验课上课前

[**实验原理**]

1.感知机模型原理

感知机是一种基础的二分类线性分类器，其基本功能是将输入空间(特征空间)中的实例划分为正负两类。感知机模型的决策函数基于特征的线性组

合形式，用数学表达式定义为：

$$f(x) = \text{sign}(w \cdot x + b)$$

其中 w 表示权重向量，x 表示样本特征向量，b 是偏置项，sign 是符号函数：

$$\text{sign}(z) = \begin{cases} +1 & \text{if} z \geqslant 0 \\ -1 & \text{if} z < 0 \end{cases}$$

感知机学习的目的是通过学习找到一组最佳的 w 和 b，使得通过这个模型得到的分类决策能够将训练数据集中的样本正确分类。

2.损失函数与学习策略

感知机的损失函数是基于误分类的度量，即所有误分类点到超平面的总距离。单个样本的误分类损失可以定义为：

$$L(x_i, y_i) = -y_i(w \cdot x_i + b)$$

当样本被正确分类时，$y_i(w \cdot x_i + b) > 0$，此时损失为零。感知机的目标是最小化全体训练样本的总损失，即最小化：

$$L(w, b) = -\sum_{x_i \in M} y_i (w \cdot x_i + b)$$

其中 M 是误分类样本的集合。感知机使用随机梯度下降法(Stochastic Gradient Descent，SGD) 来最小化损失函数。对于训练集中的每个样本，如果它被误分类，参数 w 和 b 将按照以下规则更新：

$$w \leftarrow w + \eta y_i x_i$$
$$b \leftarrow b + \eta y_i$$

其中 η (学习率) 是一个正的常数，决定了学习步长的大小。

3.收敛性分析

对于线性可分的数据集，感知机学习算法能保证找到一个将训练集完全正确划分的超平面，即存在某个时刻 t，算法将在有限步内收敛。这种收敛性由 Novikoff 定理保证。在实际操作中，学习率的选择和初始值的设定对算法的收敛速度有很大的影响。

4.Scikit-learn 感知机模型

Scikit-learn 提供的感知机模型是基于 sklearn.linear_model.Perceptron 类实现的。该模型在内部使用随机梯度下降法（SGD）作为优化算法。Scikit-learn 的感知机模型具有易用的接口和多种可调参数，例如学习率、迭代次数（max_iter）、容忍度（tol）等，从而允许灵活地调整模型以适应不同的数据集。

感知机的决策函数可以用以下公式

定义：

$$f(x) = \text{sign}(w \cdot x + b)$$

其中：w 表示权重向量， x 是输入特征向量，b 是偏置项， sign 是符号函数，输出 +1 或 −1。

Scikit-learn 感知机的优化通过最小化以下代价函数进行：

$$L(w, b) = \sum_{i=1}^{n} \max(0, -y_i(w \cdot x_i + b))$$

这是一个“较链损失”(hinge loss)，适用于二分类任务。每次迭代，算法随机选择一个训练样本，并基于这个样本更新模型参数，更新规则如下：

$$w \leftarrow w + \eta y_i x_i \text{ if } y_i(w \cdot x_i + b) \leq 0$$

$$b \leftarrow b + \eta y_i \quad \text{if } y_i(w \cdot x_i + b) \leq 0$$

Scikit-learn 感知机的主要特点包括：

自动化的学习率调整：Scikit-learn 提供了多种策略，例如常数学习率、递减学习率等，可以通过 eta0 和 learning_rate 参数进行配置。

早停机制：通过 tol 参数，如果模型的改进小于设定的阈值，则提前停止训练，这有助于避免过度拟合并减少计算资源的浪费。

正则化支持：感知机支持 penalty 参数，可用于引入正则化项（如 L1 或 L2），以控制模型的复杂度并防止过拟合。

多类分类支持：虽然基本的感知机模型是一个二分类器，但通过使用一对多（OvR）策略，Scikit-learn 的感知机可以扩展到多类分类问题。

5.随机梯度下降算法(SGD)

随机梯度下降(SGD) 是一种优化算法，常用于大规模机器学习问题。在感知机模型中，SGD 用于高效地找到误分类最少的超平面。它的核心思想是，每次迭代只使用一个样本点来更新模型的权重和偏置，这与批量梯度下降（使用整个数据集更新一次）形成对比。它是标准梯度下降的变种，标准梯度下降的更新规则是基于整个数据集的， 如下所示：

$$w \leftarrow w - \eta \nabla_w J(w)$$

其中 $J(w)$是代价函数，$\nabla_w J(w)$是对权重 w 的代价函数的梯度，η 是学习率。与

此相对，SGD 每次迭代只随机选择一个样本来更新权重：

$$w \leftarrow w - \eta \nabla_w J_i(w)$$

其中$J_i(w)$是第 i 个样本的代价函数。这种方法大大减少了计算负荷，使得训练过程更快速，并能够在线更新模型。然而，SGD 的主要挑战是它的方差较大，可能导致优化路径波动，难以收敛到最小值。为了解决这一问题，通常会使用如动量或学习率衰减等技术来帮助稳定训练过程。

SGD 的优点包括：

高效性：由于每次只处理一个样本，SGD 能够快速进行模型更新，尤其是在数据集很大时。

易于在线学习：SGD 可以逐个样本进行训练，非常适合在线学习场景。

逃离局部最小点：SGD 由于其随机性，有可能逃离局部最小点，找到更全局的最优解。

[实验内容]

(一) 感知机神经网络分类

1. 从 iris 数据集中取出[sepal length，sepal width]两个属性作为样本特征，保持类别标记不变，训练单隐层感知机网络进行二分类实验。注意**取前 100 个样本作为数据集：两类均取前 40 个构建训练集，两类均取后 10 个构建测试集。**

   提示：

   - Iris 数据集介绍详见：https://archive.ics.uci.edu/dataset/53/iris
   - Scikit-learn 库中预装了 Iris 数据集，安装库后采用“from sklearn.datasets import load_iris”可以直接读取，参考 https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html

2. 借助 matplotlib 画出原始训练数据分布的散点图（x=“sepal length”，y=“sepal

width”，点的颜色代表不同类别）

3．按照下述模型和优化目标，构造感知机模型和损失函数：

a） 模型：$f(x)=\text{sign}(w\cdot x+b)$

b） 优化目标：$\min_{w,b} L(w,b)=-\sum_{x_i\in M} y_i\left(w\cdot x_i+b\right)$

4．编写适用于感知机的随机梯度下降算法（Stochastic Gradient Descent，SGD），对单隐层的感知机进行梯度下降。

提示：

- SGD 更新：$\begin{aligned} w &= w+\eta y_i x_i \\ b &= b+\eta y_i \end{aligned}$
- 此数据集线性可分，可以设置迭代的停止条件为“直到训练集内没有误分类样本为止”
- 学习率设置为 0.1，偏置初始化 0，权重均初始化为 1

结果展示：

- 将模型拟合的分类边界与上述原始数据点画到同一图中，观察训练效果
- 用训练的模型对测试数据进行分类，得到测试错误率
- 将模型拟合的分类边界与测试数据点画到同一图中，观察效果

(二) 神经网络调参

1．调整学习率参数取值分别为[0.01，0.05，0.1，0.5]运行模型，比较模型最后的测试正确率以及模型收敛所需要的训练轮数（epoch）数。

(三) Sciki-learn 机器学习库的调用

1．直接调用机器学习库 Scikit-learn 中感知机模型

（https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html），用上述训练和测试集训练进行训练， 采用上述类似的方法可视化训练集和验证集上模型的分类结果。（调用时保持模型超参数 tol 的取值为默认值 0.001）

2. 比较自己实现的模型和调用的 Scikit-learn 中模型在训练集上的可视化结果图，观察有何不同，分析产生不同的原因。

[实验代码和结果]

（一）感知机神经网络分类

实验代码：

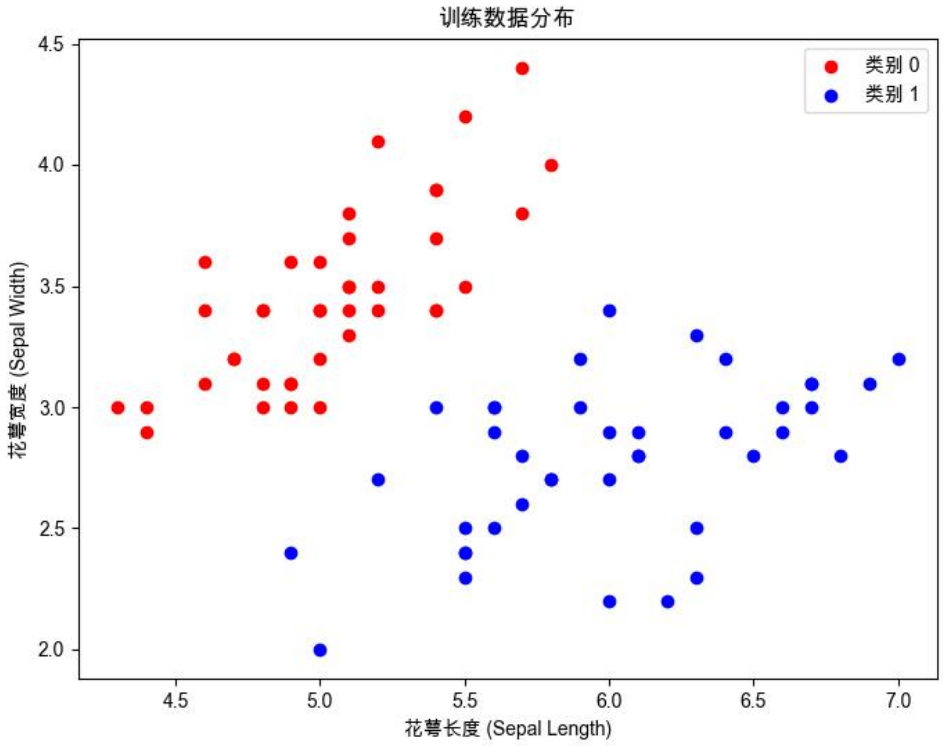
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from matplotlib import rcParams
rcParams['font.sans-serif'] = ['Arial Unicode MS']
rcParams['axes.unicode_minus'] = False
# 加载 Iris 数据集
iris = load_iris()
X = iris.data[:, :2] # 选择前两个属性 [sepal length, sepal width]
y = iris.target

# 过滤出前两个类别的数据用于二分类
binary_filter = y < 2 # 只保留类别 0 和 1
X_binary = X[binary_filter]
y_binary = y[binary_filter]

# 固定划分：每类取前 40 个样本作为训练集，后 10 个作为测试集
X_class_0 = X_binary[y_binary == 0] # 类别 0 的数据
X_class_1 = X_binary[y_binary == 1] # 类别 1 的数据
```

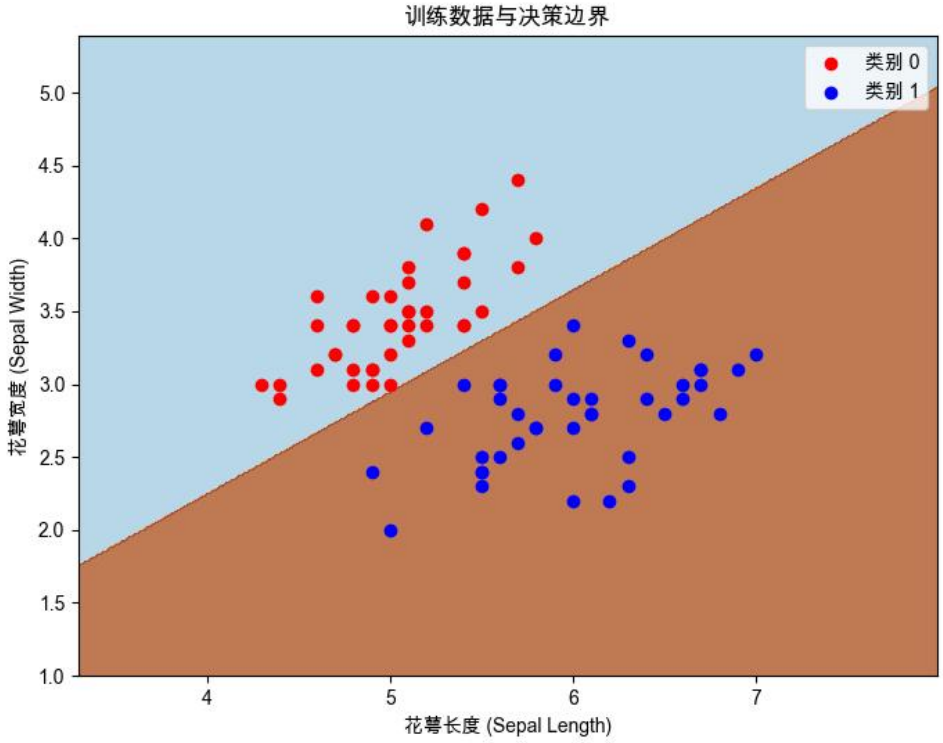
```
y_class_0 = y_binary[y_binary == 0] # 类别 0 的标签
y_class_1 = y_binary[y_binary == 1] # 类别 1 的标签

# 构建训练集
X_train = np.vstack((X_class_0[:40], X_class_1[:40]))
y_train = np.hstack((y_class_0[:40], y_class_1[:40]))

# 构建测试集
X_test = np.vstack((X_class_0[40:50], X_class_1[40:50]))
y_test = np.hstack((y_class_0[40:50], y_class_1[40:50]))
print(f"训练集大小: {len(X_train)}, 测试集大小:
{len(X_test)}")
# 绘制训练数据的分布散点图
plt.figure(figsize=(8, 6))
plt.scatter(X_train[y_train == 0][:, 0], X_train[y_train ==
0][:, 1], color='red', label='类别 0')
plt.scatter(X_train[y_train == 1][:, 0], X_train[y_train ==
1][:, 1], color='blue', label='类别 1')
plt.xlabel("花萼长度 (Sepal Length)")
plt.ylabel("花萼宽度 (Sepal Width)")
plt.title("训练数据分布")
plt.legend()
plt.show()

# 定义感知机模型及其随机梯度下降算法
class Perceptron:
def __init__(self, learning_rate=0.1, max_iter=1000):
self.learning_rate = learning_rate # 学习率
self.max_iter = max_iter # 最大迭代次数
self.weights = None # 权重初始化
self.bias = None # 偏置初始化

def fit(self, X, y):
n_samples, n_features = X.shape
self.weights = np.ones(n_features) # 初始化权重为 1
self.bias = 0 # 初始化偏置为 0

for _ in range(self.max_iter):
errors = 0 # 记录误分类样本数量
```

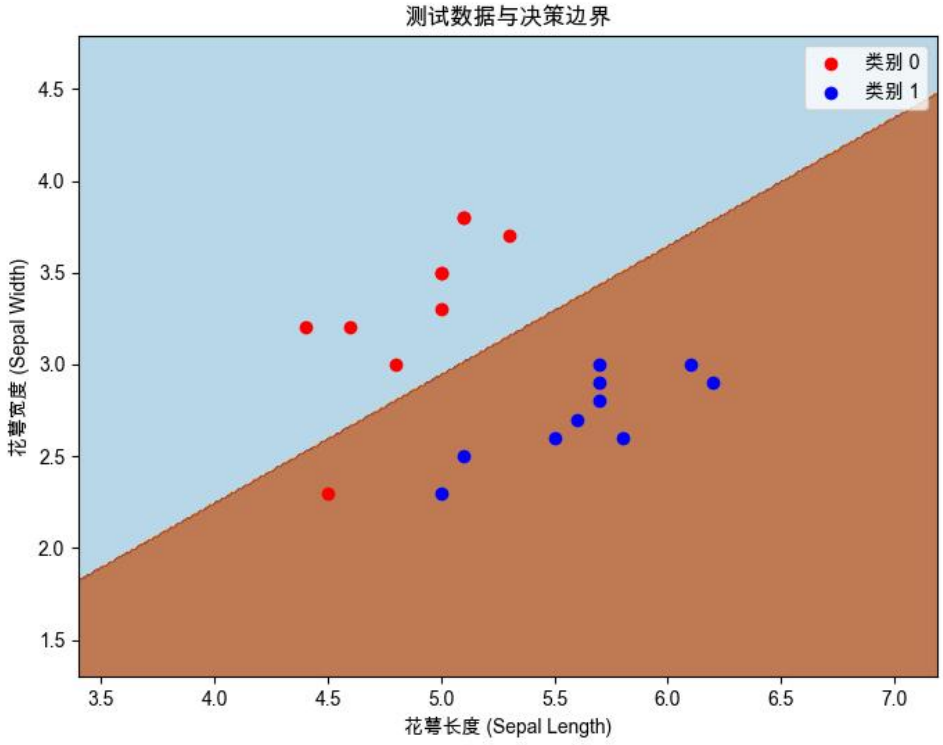
```
for xi, target in zip(X, y):
update = self.learning_rate * (target - self.predict(xi))
self.weights += update * xi # 更新权重
self.bias += update # 更新偏置
errors += int(update != 0.0) # 如果有更新，计入误分类
if errors == 0: # 如果训练集无误分类样本，停止迭代
break

def predict(self, X):
linear_output = np.dot(X, self.weights) + self.bias
return np.where(linear_output >= 0, 1, 0) # 分类阈值：>= 0
为 1，否则为 0

# 绘制决策边界的函数
def plot_decision_boundary(model, X, y, title):
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
np.arange(y_min, y_max, 0.01))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.Paired)
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='red',
label='类别 0')
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='blue',
label='类别 1')
plt.xlabel("花萼长度 (Sepal Length)")
plt.ylabel("花萼宽度 (Sepal Width)")
plt.title(title)
plt.legend()
plt.show()

# 训练感知机模型
perceptron = Perceptron(learning_rate=0.1)
perceptron.fit(X_train, y_train)

# 绘制训练数据的决策边界
```

```
plot_decision_boundary(perceptron, X_train, y_train, "训练数据与决策边界")
# 验证测试集中的类别 0 和类别 1 样本数量
print(f"类别 0 测试样本: {X_test[y_test == 0]}")
print(f"类别 1 测试样本: {X_test[y_test == 1]}")
# 测试模型并计算测试集错误率
y_pred = perceptron.predict(X_test)
test_error_rate = np.mean(y_pred != y_test)
print(f"测试集错误率: {test_error_rate:.2f}")

# 绘制测试数据的决策边界
plot_decision_boundary(perceptron, X_test, y_test, "测试数据与决策边界")
```

实验结果：




训练数据与决策边界
类别 0
类别 1
花萼宽度 (Sepal Width)
花萼长度 (Sepal Length)
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7


测试数据与决策边界
类别 0
类别 1
花萼宽度 (Sepal Width)
花萼长度 (Sepal Length)
4.5
4.0
3.5
3.0
2.5
2.0
1.5
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0

训练集大小: 80, 测试集大小: 20
类别 0 测试样本: [[5. 3.5]
[4.5 2.3]
[4.4 3.2]
[5. 3.5]
[5.1 3.8]
[4.8 3.]
[5.1 3.8]
[4.6 3.2]
[5.3 3.7]
[5. 3.3]]
类别 1 测试样本: [[5.5 2.6]
[6.1 3.]
[5.8 2.6]
[5. 2.3]
[5.6 2.7]
[5.7 3.]
[5.7 2.9]
[6.2 2.9]
[5.1 2.5]
[5.7 2.8]]
测试集错误率: 0.05
图中红色有两个点重合，所以视觉上只有八个点

（二）神经网络调参

实验代码：

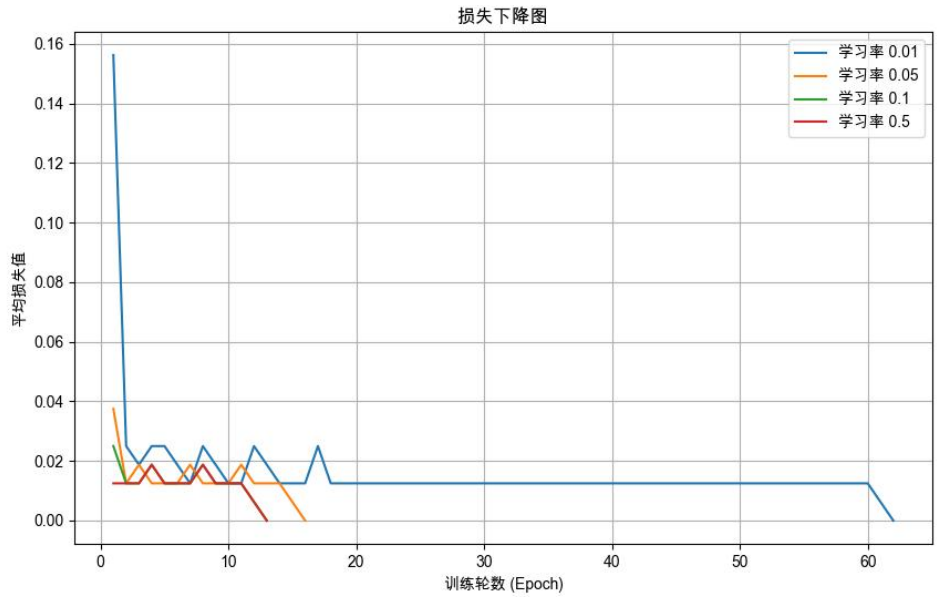
```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.datasets import load_iris
from matplotlib import rcParams
rcParams['font.sans-serif'] = ['Arial Unicode MS']
rcParams['axes.unicode_minus'] = False
# 加载 Iris 数据集
iris = load_iris()
X = iris.data[:, :2] # 选择前两个属性 [sepal length, sepal
width]
y = iris.target

# 过滤出前两个类别的数据用于二分类
binary_filter = y < 2 # 只保留类别 0 和 1
X_binary = X[binary_filter]
y_binary = y[binary_filter]

# 固定划分：每类取前 40 个样本作为训练集，后 10 个作为测试集
X_class_0 = X_binary[y_binary == 0] # 类别 0 的数据
X_class_1 = X_binary[y_binary == 1] # 类别 1 的数据

y_class_0 = y_binary[y_binary == 0] # 类别 0 的标签
y_class_1 = y_binary[y_binary == 1] # 类别 1 的标签

# 构建训练集
X_train = np.vstack((X_class_0[:40], X_class_1[:40]))
y_train = np.hstack((y_class_0[:40], y_class_1[:40]))

# 构建测试集
X_test = np.vstack((X_class_0[40:50], X_class_1[40:50]))
y_test = np.hstack((y_class_0[40:50], y_class_1[40:50]))

# 修改感知机模型以记录每轮的损失和训练轮数
class PerceptronWithTracking:
def __init__(self, learning_rate=0.1, max_iter=1000):
self.learning_rate = learning_rate # 学习率
self.max_iter = max_iter # 最大迭代次数
self.weights = None # 权重初始化
self.bias = None # 偏置初始化
self.epochs = 0 # 实际训练轮数
self.loss_history = [] # 损失值记录
```

```
def fit(self, X, y):
n_samples, n_features = X.shape
self.weights = np.ones(n_features) # 初始化权重为 1
self.bias = 0 # 初始化偏置为 0
self.epochs = 0
self.loss_history = []

for _ in range(self.max_iter):
errors = 0 # 记录误分类样本数量
loss = 0 # 本轮的损失
for xi, target in zip(X, y):
prediction = self.predict(xi)
update = self.learning_rate * (target - prediction)
self.weights += update * xi # 更新权重
self.bias += update # 更新偏置
errors += int(update != 0.0) # 如果有更新，计入误分类
loss += 0.5 * (target - prediction) ** 2 # 计算二分类损失
self.loss_history.append(loss / n_samples) # 平均损失
self.epochs += 1
if errors == 0: # 如果训练集无误分类样本，停止迭代
break

def predict(self, X):
linear_output = np.dot(X, self.weights) + self.bias
return np.where(linear_output >= 0, 1, 0) # 分类阈值：>= 0
为 1，否则为 0

# 定义不同的学习率参数
learning_rates = [0.01, 0.05, 0.1, 0.5]
results = {}

# 运行模型并记录测试正确率和训练轮数
for lr in learning_rates:
model = PerceptronWithTracking(learning_rate=lr)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = np.mean(y_pred == y_test)
```

```
results[lr] = {"accuracy": accuracy, "epochs": model.epochs,
"loss_history": model.loss_history}
print(f"学习率: {lr}, 测试正确率: {accuracy:.2f}, 收敛轮数:
{model.epochs}")

# 绘制损失下降图
plt.figure(figsize=(10, 6))
for lr, result in results.items():
plt.plot(range(1, result["epochs"] + 1),
result["loss_history"], label=f"学习率 {lr}")
plt.xlabel("训练轮数 (Epoch)")
plt.ylabel("平均损失值")
plt.title("损失下降图")
plt.legend()
plt.grid()
plt.show()
```

实验结果：




```
youngbean@YoungBeans Class2 % /usr/local/bin/python3 "/Users/youngbean/Desktop/Experiments in Machine Learning/Class2/Programming/project2.py"
学习率: 0.01, 测试正确率: 0.95, 收敛轮数: 62
学习率: 0.05, 测试正确率: 0.95, 收敛轮数: 16
学习率: 0.1, 测试正确率: 0.95, 收敛轮数: 13
学习率: 0.5, 测试正确率: 0.95, 收敛轮数: 13
2024-12-02 09:19:42.025 Python[41466:1198902] +[IMKClient subclass]: chose IMKClient_Modern
2024-12-02 09:19:42.025 Python[41466:1198902] +[IMKInputSession subclass]: chose IMKInputSession_Modern
```

学习率: 0.01, 测试正确率: 0.95, 收敛轮数: 62
学习率: 0.05, 测试正确率: 0.95, 收敛轮数: 16
学习率: 0.1, 测试正确率: 0.95, 收敛轮数: 13

学习率: 0.5, 测试正确率: 0.95, 收敛轮数: 13

（三）Sciki-learn 机器学习库的调用

实验代码：

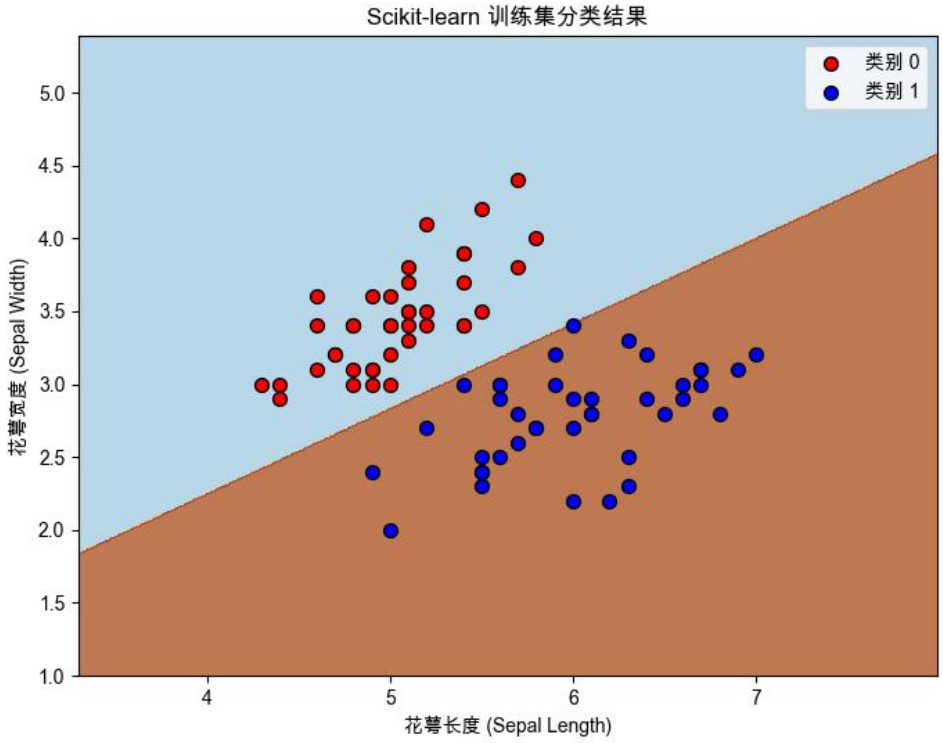
```
from sklearn.linear_model import Perceptron
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import rcParams
rcParams['font.sans-serif'] = ['Arial Unicode MS']
rcParams['axes.unicode_minus'] = False
# 加载 Iris 数据集
iris = load_iris()
X = iris.data[:, :2] # 选择前两个属性 [sepal length, sepal
width]
y = iris.target

# 过滤出前两个类别的数据用于二分类
binary_filter = y < 2 # 只保留类别 0 和 1
X_binary = X[binary_filter]
y_binary = y[binary_filter]

# 固定划分：每类取前 40 个样本作为训练集，后 10 个作为测试集
X_class_0 = X_binary[y_binary == 0] # 类别 0 的数据
X_class_1 = X_binary[y_binary == 1] # 类别 1 的数据

y_class_0 = y_binary[y_binary == 0] # 类别 0 的标签
y_class_1 = y_binary[y_binary == 1] # 类别 1 的标签

# 构建训练集
X_train = np.vstack((X_class_0[:40], X_class_1[:40]))
y_train = np.hstack((y_class_0[:40], y_class_1[:40]))

# 构建测试集
X_test = np.vstack((X_class_0[40:50], X_class_1[40:50]))
y_test = np.hstack((y_class_0[40:50], y_class_1[40:50]))
# 修改感知机模型以记录每轮的损失和训练轮数
class PerceptronWithTracking:
```

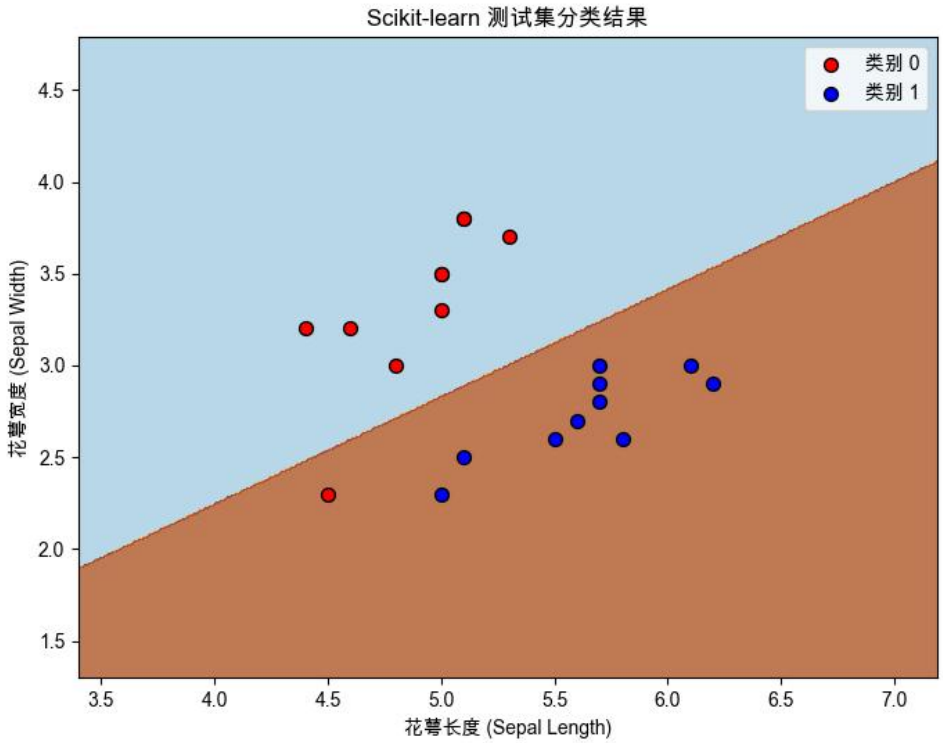
```
def __init__(self, learning_rate=0.1, max_iter=1000):
self.learning_rate = learning_rate # 学习率
self.max_iter = max_iter # 最大迭代次数
self.weights = None # 权重初始化
self.bias = None # 偏置初始化
self.epochs = 0 # 实际训练轮数
self.loss_history = [] # 损失值记录

def fit(self, X, y):
n_samples, n_features = X.shape
self.weights = np.ones(n_features) # 初始化权重为 1
self.bias = 0 # 初始化偏置为 0
self.epochs = 0
self.loss_history = []

for _ in range(self.max_iter):
errors = 0 # 记录误分类样本数量
loss = 0 # 本轮的损失
for xi, target in zip(X, y):
prediction = self.predict(xi)
update = self.learning_rate * (target - prediction)
self.weights += update * xi # 更新权重
self.bias += update # 更新偏置
errors += int(update != 0.0) # 如果有更新，计入误分类
loss += 0.5 * (target - prediction) ** 2 # 计算二分类损失
self.loss_history.append(loss / n_samples) # 平均损失
self.epochs += 1
if errors == 0: # 如果训练集无误分类样本，停止迭代
break

def predict(self, X):
linear_output = np.dot(X, self.weights) + self.bias
return np.where(linear_output >= 0, 1, 0) # 分类阈值：>= 0
为 1，否则为 0
# 绘制决策边界的函数
def plot_decision_boundary(model, X, y, title):
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
```

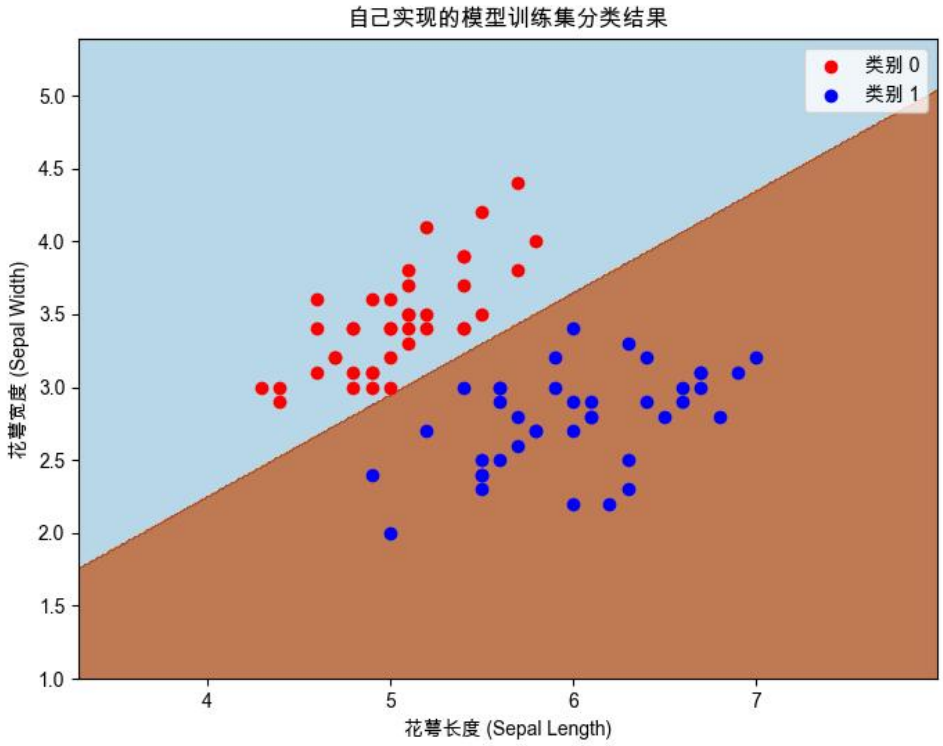
```
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
np.arange(y_min, y_max, 0.01))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.Paired)
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='red',
label='类别 0')
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='blue',
label='类别 1')
plt.xlabel("花萼长度 (Sepal Length)")
plt.ylabel("花萼宽度 (Sepal Width)")
plt.title(title)
plt.legend()
plt.show()

# 使用 Scikit-learn 中的感知机模型
sklearn_model = Perceptron(tol=0.001, random_state=42)
sklearn_model.fit(X_train, y_train)

# 定义绘制分类结果的函数
def plot_decision_boundary_sklearn(model, X, y, title):
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
np.arange(y_min, y_max, 0.01))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.Paired)
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='red',
edgecolor='black', s=50, label='类别 0')
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='blue',
edgecolor='black', s=50, label='类别 1')
plt.xlabel("花萼长度 (Sepal Length)")
plt.ylabel("花萼宽度 (Sepal Width)")
plt.title(title)
plt.legend()
plt.show()
```

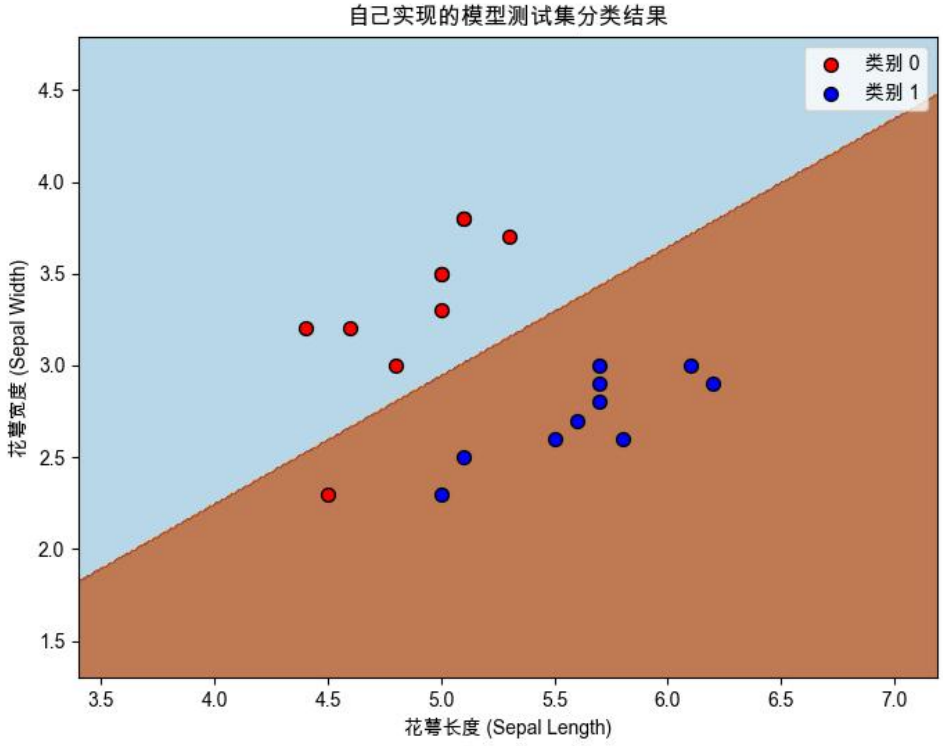
```
# 绘制训练集上的分类结果
plot_decision_boundary_sklearn(sklearn_model, X_train,
y_train, "Scikit-learn 训练集分类结果")

# 绘制测试集上的分类结果
plot_decision_boundary_sklearn(sklearn_model, X_test,
y_test, "Scikit-learn 测试集分类结果")

# 比较两种模型的结果
# 自己实现的模型在训练集上的分类结果
custom_model = PerceptronWithTracking(learning_rate=0.1)
custom_model.fit(X_train, y_train)
plot_decision_boundary(custom_model, X_train, y_train, "自
己实现的模型训练集分类结果")
plot_decision_boundary_sklearn(custom_model, X_test,
y_test, "自己实现的模型测试集分类结果")

# 分析产生不同的原因
# 计算测试集上的分类正确率
y_pred_custom = custom_model.predict(X_test)
y_pred_sklearn = sklearn_model.predict(X_test)
accuracy_custom = np.mean(y_pred_custom == y_test)
accuracy_sklearn = np.mean(y_pred_sklearn == y_test)

# 打印测试正确率
print(f"自己实现的模型测试正确率: {accuracy_custom:.2f}")
print(f"Scikit-learn 模型测试正确率: {accuracy_sklearn:.2f}")

# 打印迭代轮数
print(f"自己实现的模型的迭代轮数: {custom_model.epochs}")
print(f"Scikit-learn 模型的迭代轮数:
{sklearn_model.n_iter_}")
```

实验结果：


Scikit-learn 训练集分类结果
类别 0
类别 1
花萼宽度 (Sepal Width)
花萼长度 (Sepal Length)
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7


Scikit-learn 测试集分类结果
类别 0
类别 1
花萼宽度 (Sepal Width)
花萼长度 (Sepal Length)
4.5
4.0
3.5
3.0
2.5
2.0
1.5
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0






```
youngbean@YoungBeans Class2 % /usr/local/bin/python3 "/Users/youngbean/Desktop/Experiments in Machine Learning/Class2/Programming/project3.py"
2024-12-02 09:22:26.587 Python[41576:1201780] +[IMKClient subclass]: chose IMKClient_Modern
2024-12-02 09:22:26.587 Python[41576:1201780] +[IMKInputSession subclass]: chose IMKInputSession_Modern
自己实现的模型测试正确率: 0.95
Scikit-learn模型测试正确率: 0.95
自己实现的模型的迭代轮数: 13
Scikit-learn模型的迭代轮数: 9
```

```
自己实现的模型测试正确率: 0.95
Scikit-learn 模型测试正确率: 0.95
自己实现的模型的迭代轮数: 13
Scikit-learn 模型的迭代轮数: 9
```

结果分析：

在本次实验中，我们对比了自主实现的感知机模型和 Scikit-learn 库中的感知机模型在 Iris 数据集上的性能。具体而言，两种模型在准确率上均达到了 95%，显示出相似的分类能力，但在迭代次数上，Scikit-learn 模型显示出更高的效率。

尽管两个模型在准确率上都达到了 95%，这种结果主要归因于 Iris 数据集的两个类别（Setosa 和 Versicolor）在所选特征（花萼长度和宽度）上具有明显的线性可分性。这意味着即使是基础的感知机模型也足以学习到有效的分类超平面，从而在这类数据上达到高准确率。

然而，在迭代次数方面，我们观察到显著的差异。自主实现的模型需要 13 次迭代才能收敛，而 Scikit-learn 的模型只需 9 次。这种差异可能是由于 Scikit-learn 内部实现了更为高效的优化算法，例如动态调整学习率或更有效的早停机制，使得模型能够在达到稳定状态时及时停止训练。这些优化措施减少了迭代的次数，从而提高了整体的训练效率。

这一发现突显了利用成熟的机器学习库，如 Scikit-learn，在实现上的优势。这些库不仅提供了高准确率的算法实现，而且通过算法优化和高效的代码实现，在提高训练速度和减少计算资源消耗方面也具有显著优势。对于未来的工作，这强调了在机器学习实践中，选择合适的库和工具以及合理调整模型参数和训练策略的重要性，这是实现高效和有效学习的关键。

[小结或讨论]

在本次实验中，我们围绕感知机神经网络的理论与实践，完成了对其模型原理、自主实现以及与第三方库性能对比的全面探索。实验的核心目标是通过自主编程和调用 Scikit-learn 库，深刻理解感知机模型在分类任务中的表现和优化方法，同时结合调参实验分析模型的敏感性和适用性。

首先，在感知机神经网络分类实验中，我们基于 Iris 数据集的两个属性（花萼长度和花萼宽度），构建了一个单隐层感知机模型，完成了数据的二分类任务。在自主实现的感知机模型中，随机梯度下降算法被用于更新权重和偏置，训练结果展示了模型能够有效拟合训练数据并准确分类测试样本。通过绘制决策边界，我们观察到感知机模型能够找到一个清晰的分类超平面，分隔两个类别的样本点。最终测试错误率为 5%，充分验证了感知机模型在线性可分数据上的高效性和可靠性。

其次，在神经网络调参实验中，我们进一步研究了学习率对感知机模型训练效率和分类准确率的影响。通过设置不同的学习率（0.01，0.05，0.1，0.5），实验结果显示，较低的学习率（如 0.01）尽管能够达到同样的分类准确率，但收敛速度较慢，需要更多的训练轮数。而较高的学习率（如 0.1 和 0.5）显著加快了收敛速度，但同时可能存在过大步长导致优化过程震荡的风险。学习率的合理选择对于模型性能优化至关重要，而这次实验通过记录损失下降趋势和分类准确率，直观展示了学习率对感知机模型训练过程的深远影响。

最后，在 Scikit-learn 机器学习库调用实验中，我们调用了 Scikit-learn 库中的感知机模型，并与自主实现的模型进行了系统性比较。两种模型在分类准确率上均

达到了 95%，显示出对 Iris 数据集线性可分特征的良好适应性。然而，Scikit-learn 模型在迭代次数和训练效率上表现出更为明显的优势，仅用 9 轮迭代即完成训练，而自主实现的模型则需要 13 轮。这种差异主要归因于 Scikit-learn 的优化策略，例如动态学习率调整和早停机制等。在决策边界可视化中，Scikit-learn 模型同样呈现出精准的分类超平面，进一步证明了其在性能和效率上的优势。

综合三个实验的结果，本次实验不仅展示了感知机模型的理论可行性和实践价值，同时通过自主实现和成熟库对比，深入剖析了不同实现方式的优劣。自主实现的过程强化了对感知机模型数学原理和随机梯度下降优化机制的理解，而调用 Scikit-learn 库则体现了现代机器学习工具在实现效率和性能优化上的不可替代性。此外，调参实验中的观察进一步强调了参数选择对模型性能的显著影响，这对后续更复杂模型的设计与优化具有重要参考价值。通过本次实验，我们更加深刻地认识到结合理论理解与实践操作的重要性，为未来更复杂的机器学习任务奠定了坚实的基础。