代码“

#!/usr/bin/env pYthon

# coding: utf-8

import numpy as np

from sklearn import datasets

import pandas as pd  # 导入pandas库

from sklearn.neural\_network import MLPClassifier

from sklearn.datasets import load\_wine

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

import matplotlib.pyplot as plt

from matplotlib.colors import ListedColormap

# ---------------- 导入数据集 -------------------------

iris = datasets.load\_iris()

# print(iris)

X = iris.data[:, [1, 2]]

# print(tYpe(X))

Y = iris.target

# 拆分数据集

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, random\_state=0)

# 定义分类器

mlp = MLPClassifier(solver="lbfgs")

# mlp = MLPClassifier(solver="lbfgs", hidden\_layer\_sizes=[2])  # 设置隐含层节点数减少到2个

mlp = MLPClassifier(

    solver="lbfgs", hidden\_layer\_sizes=[2, 2, 2, 2]

)  # 设置神经网络有两个节点数为10的隐含层

mlp = MLPClassifier(

    solver="lbfgs", hidden\_layer\_sizes=[10, 10], activation="tanh"

)  # 设置激活函数为tanh

mlp = MLPClassifier(

    solver="lbfgs", hidden\_layer\_sizes=[10, 10], activation="tanh", alpha=1

)  # 修改模型的alpha参数

# ------------ 训练 ----------------- #

mlp.fit(X\_train, Y\_train)

# 使用不同色块表示不同分类

cmap\_light = ListedColormap(["#FFAAAA", "#AAFFAA", "#AAAAFF"])

cmap\_bold = ListedColormap(["#FF0000", "#00FF00", "#0000FF"])

x\_min, x\_max = X\_train[:, 0].min() - 1, X\_train[:, 0].max() + 1

Y\_min, Y\_max = X\_train[:, 1].min() - 1, X\_train[:, 1].max() + 1

xx, YY = np.meshgrid(np.arange(x\_min, x\_max, 0.02), np.arange(Y\_min, Y\_max, 0.02))

Z = mlp.predict(np.c\_[xx.ravel(), YY.ravel()])

Z = Z.reshape(xx.shape)

plt.figure()

plt.pcolormesh(xx, YY, Z, cmap=cmap\_light)

# 将数据特征用散点图表示

plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolor="k", s=60)

plt.xlim(xx.min(), xx.max())

plt.ylim(YY.min(), YY.max())

plt.title("train")  # 设定图的标题

plt.show()  # 显示图形

"""mlp.fit(X\_test, Y\_test)

# 使用不同色块表示不同分类

cmap\_light = ListedColormap(["#FFAAAA", "#AAFFAA", "#AAAAFF"])

cmap\_bold = ListedColormap(["#FF0000", "#00FF00", "#0000FF"])

x\_min, x\_max = X\_test[:, 0].min() - 1, X\_test[:, 0].max() + 1

Y\_min, Y\_max = X\_test[:, 1].min() - 1, X\_test[:, 1].max() + 1

xx, YY = np.meshgrid(np.arange(x\_min, x\_max, 0.02), np.arange(Y\_min, Y\_max, 0.02))

Z = mlp.predict(np.c\_[xx.ravel(), YY.ravel()])

Z = Z.reshape(xx.shape)

plt.figure()

plt.pcolormesh(xx, YY, Z, cmap=cmap\_light)

# 将数据特征用散点图表示

plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolor="k", s=60)

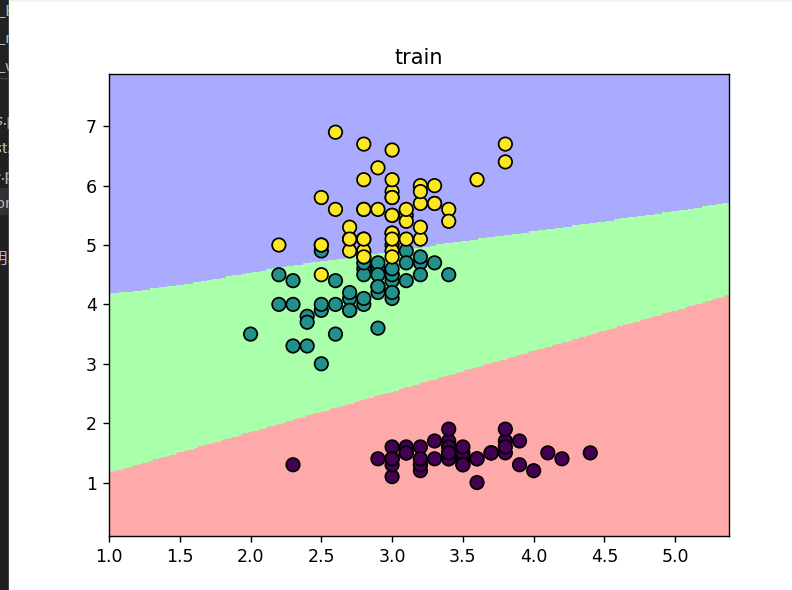
plt.xlim(xx.min(), xx.max())

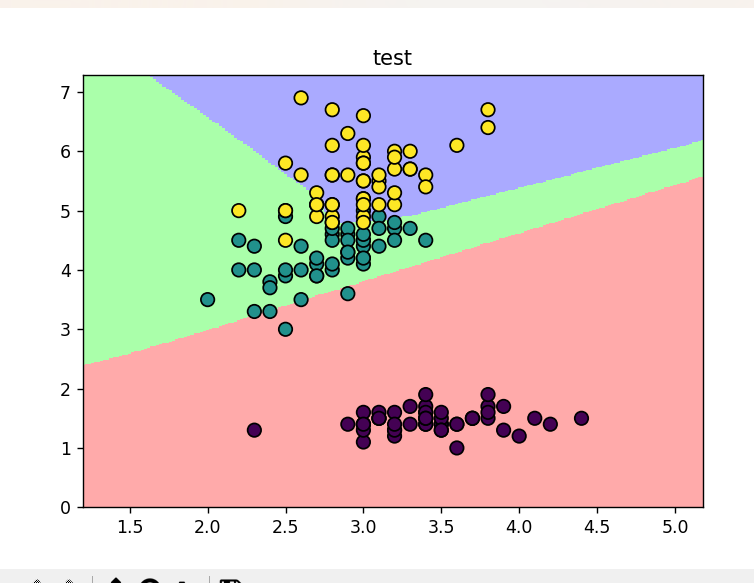
plt.ylim(YY.min(), YY.max())

plt.title("test")  # 设定图的标题

plt.show()  # 显示图形"""

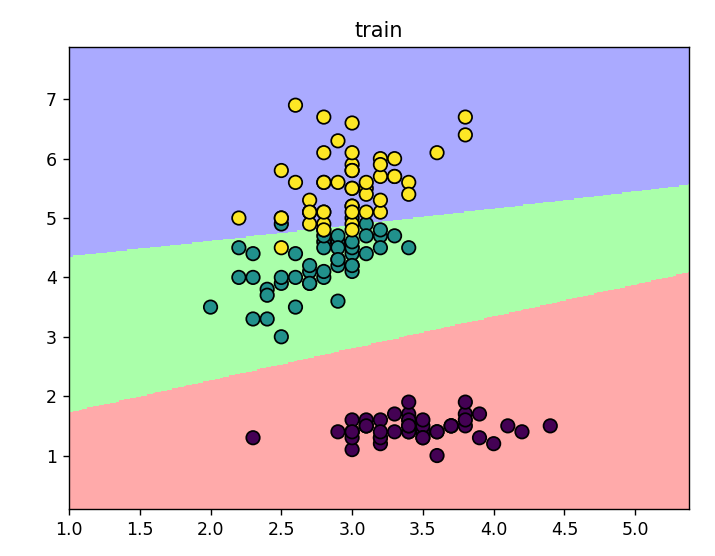
（1）训练集和测试集分类结果如下



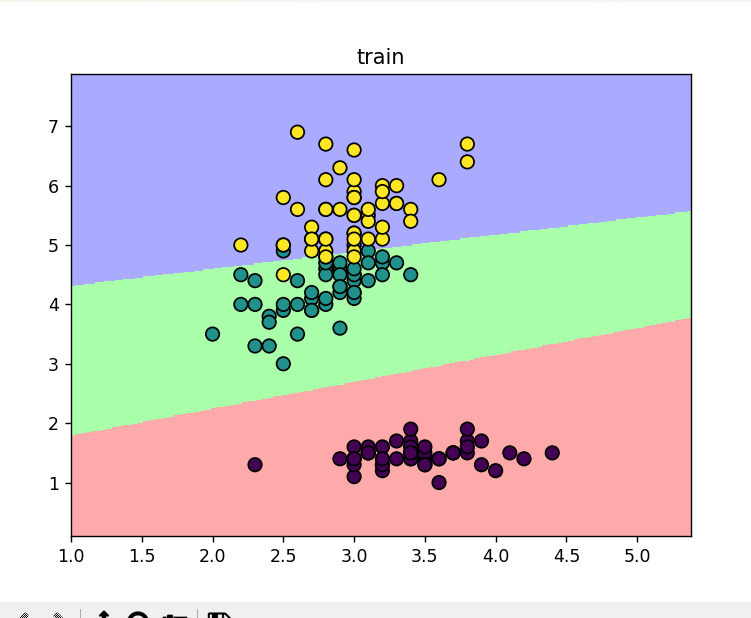


（2）

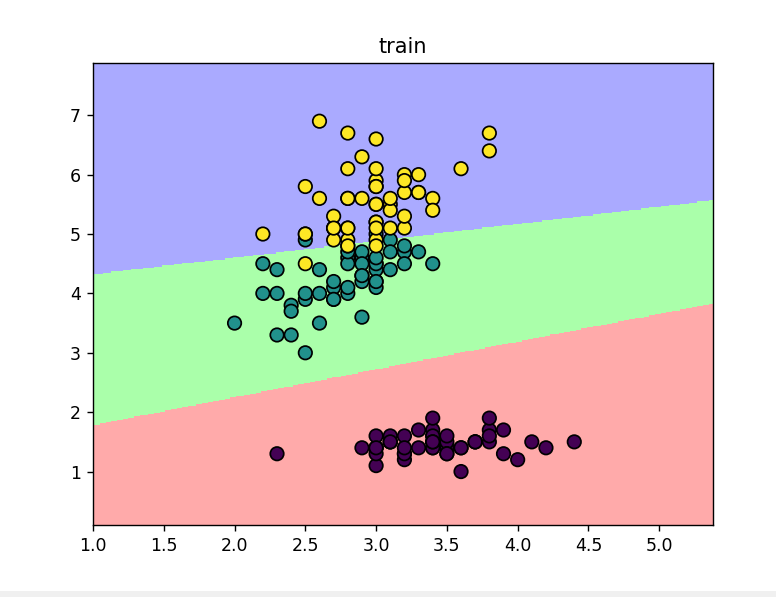
设置神经网络有两个节点数为10的隐含层



设置隐含层节点数减少到10个



设置隐含层节点数减少到2个



由于数据集较小且模型很完备，小幅度调整隐含层相应结构参数、激活函数等相关参数输出结果变化不大

（3）反向传播相关函数

def backward\_propagation(parameters, cache, X, Y):

m = Y.shape[1]

w2 = parameters['w2']

a1 = cache['a1']

a2 = cache['a2']

# 反向传播，计算dw1、db1、dw2、db2

dz2 = a2 - Y

dw2 = (1 / m) \* np.dot(dz2, a1.T)

db2 = (1 / m) \* np.sum(dz2, axis=1, keepdims=True)

dz1 = np.multiply(np.dot(w2.T, dz2), 1 - np.power(a1, 2))

dw1 = (1 / m) \* np.dot(dz1, X.T)

db1 = (1 / m) \* np.sum(dz1, axis=1, keepdims=True)

grads = {'dw1': dw1, 'db1': db1, 'dw2': dw2, 'db2': db2}

return grads