1、读取iris.txt数据，划分训练集和测试集，

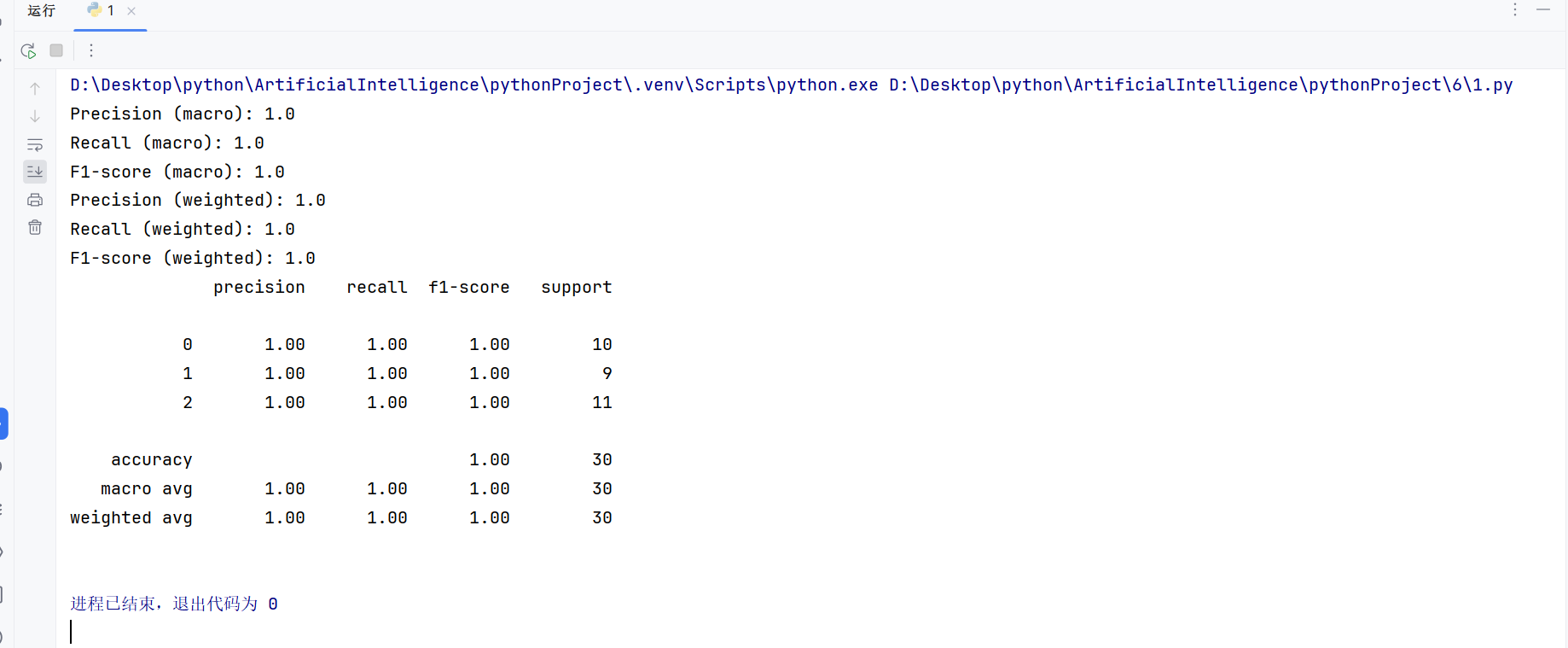
（1）使用逻辑回归模型训练数据，画出训练集的混淆矩阵图。

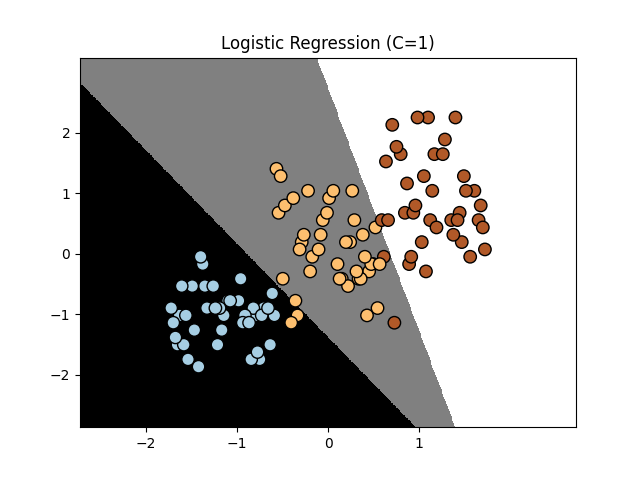
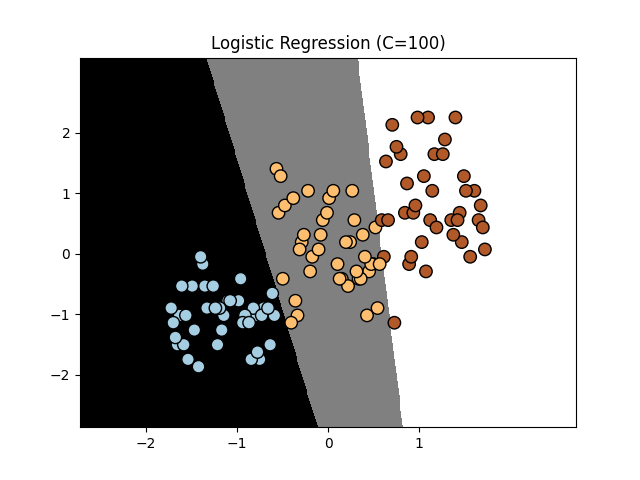
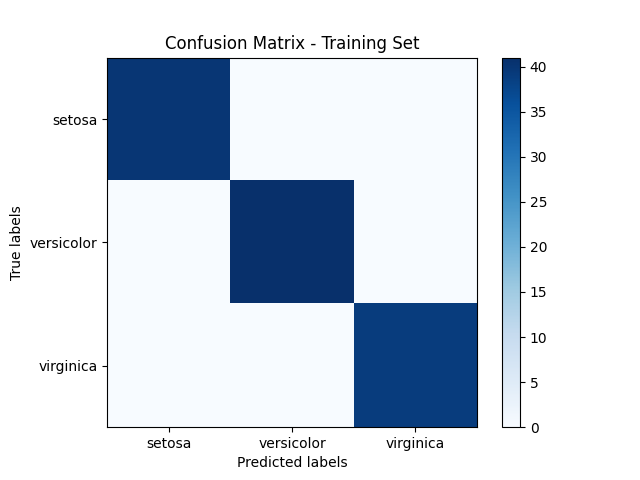
（2）分别输出测试集在average='macro'和average='weighted'时的精确率、召回率和F1分数，

（3）输出测试集预测的性能报告

（4）分别画正则参数C=100 和C=1时的分类器模型图（数据选用前两维或后两维都可以）

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import confusion\_matrix, classification\_report  
from sklearn.preprocessing import StandardScaler, LabelEncoder  
  
# 1. 读取iris数据  
iris\_df = pd.read\_csv('../iris.csv')  
  
# 划分特征和标签  
X = iris\_df.iloc[:, :-1].values  
y = iris\_df.iloc[:, -1].values  
scaler=StandardScaler()  
X=scaler.fit\_transform(X)  
label\_encoder = LabelEncoder()  
y\_encoded = label\_encoder.fit\_transform(y)  
# 划分训练集和测试集  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_encoded, test\_size=0.2, random\_state=42)  
  
# (1) 使用逻辑回归模型训练数据，画出训练集的混淆矩阵图  
log\_reg = LogisticRegression()  
log\_reg.fit(X\_train, y\_train)  
y\_pred\_train = log\_reg.predict(X\_train)  
conf\_matrix\_train = confusion\_matrix(y\_train, y\_pred\_train)  
  
plt.imshow(conf\_matrix\_train, cmap='Blues', interpolation='nearest')  
plt.colorbar()  
plt.xlabel('Predicted labels')  
plt.ylabel('True labels')  
plt.xticks(np.arange(len(iris\_df['Species'].unique())), iris\_df['Species'].unique())  
plt.yticks(np.arange(len(iris\_df['Species'].unique())), iris\_df['Species'].unique())  
plt.title('Confusion Matrix - Training Set')  
plt.show()  
  
# (2) 分别输出测试集在average='macro'和average='weighted'时的精确率、召回率和F1分数  
from sklearn.metrics import precision\_recall\_fscore\_support  
  
def print\_metrics(y\_true, y\_pred, average):  
 precision, recall, f1, \_ = precision\_recall\_fscore\_support(y\_true, y\_pred, average=average)  
 print(f'Precision ({average}): {precision}')  
 print(f'Recall ({average}): {recall}')  
 print(f'F1-score ({average}): {f1}')  
  
print\_metrics(y\_test, log\_reg.predict(X\_test), average='macro')  
print\_metrics(y\_test, log\_reg.predict(X\_test), average='weighted')  
  
# (3) 输出测试集预测的性能报告  
print(classification\_report(y\_test, log\_reg.predict(X\_test)))  
  
# # (4) 分别画正则参数C=100 和C=1时的分类器模型图  
def plot\_decision\_boundary(classifier, X, y, title):  
 x\_min, x\_max = X[:, 0].min() - 1.0, X[:, 0].max() + 1.0  
 y\_min, y\_max = X[:, 1].min() - 1.0, X[:, 1].max() + 1.0  
  
 step\_size = 0.01  
 x\_values, y\_values = np.meshgrid(np.arange(x\_min, x\_max, step\_size), np.arange(y\_min, y\_max, step\_size))  
 mesh\_output = classifier.predict(np.c\_[x\_values.ravel(), y\_values.ravel()])  
 mesh\_output = mesh\_output.reshape(x\_values.shape)  
 # print(x\_values)  
 plt.figure()  
 plt.pcolormesh(x\_values, y\_values, mesh\_output, cmap=plt.cm.gray)  
 plt.scatter(X[:, 0], X[:, 1], c=y, s=80, edgecolors='black', linewidth=1, cmap=plt.cm.Paired)  
 plt.xlim(x\_values.min(), x\_values.max())  
 plt.ylim(y\_values.min(), y\_values.max())  
 # specify the ticks on the X and Y axes  
 plt.xticks((np.arange(int(min(X[:, 0]) - 1), int(max(X[:, 0]) + 1), 1.0)))  
 plt.yticks((np.arange(int(min(X[:, 1]) - 1), int(max(X[:, 1]) + 1), 1.0)))  
 plt.title(title)  
 plt.show()  
  
  
C\_values = [100, 1]  
for C in C\_values:  
 log\_reg = LogisticRegression(C=C)  
 log\_reg.fit(X\_train[:, :2], y\_train) # 使用前两维特征  
 plot\_decision\_boundary(log\_reg,X\_train[:, :2],y\_train , f'Logistic Regression (C={C})')





2、读取car.txt数据，对样本特征和标签数据进行编码（OneHot/LabelEncoder)，

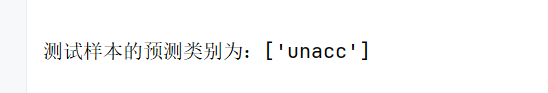
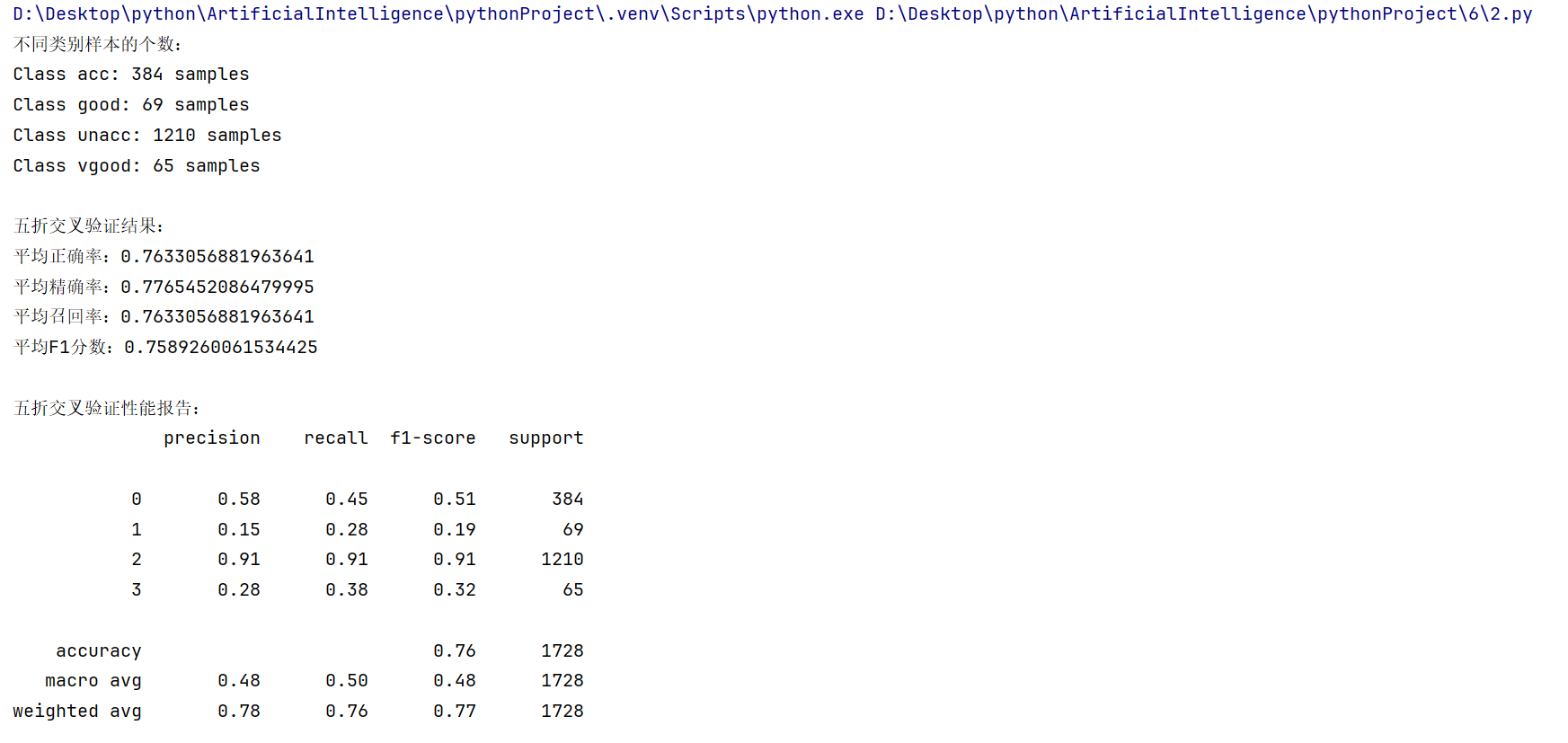
（1）统计不同类别样本的个数并输出，

（2）选择一种average评价指标，使用逻辑回归模型训练数据，并输出五折交叉验证的正确率，精确率、召回率和F1分数

（3）输出五折交叉验证的性能报告

（4）生成一个测试样本，使用模型进行预测，并输出预测值。

import numpy as np  
import pandas as pd  
import sklearn.metrics as ms  
from sklearn.preprocessing import OneHotEncoder, LabelEncoder  
from sklearn.model\_selection import cross\_val\_score, cross\_validate, train\_test\_split, cross\_val\_predict  
from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, classification\_report,make\_scorer  
  
# 读取数据  
car\_df = pd.read\_csv('../3/car.txt', header=None)  
  
# 对特征数据进行OneHot编码  
X = pd.get\_dummies(car\_df.iloc[:, :-1])  
X1=X  
X=np.array(X)  
y\_original = car\_df.iloc[:, -1]  
  
# 统计编码前每个类别的个数  
unique\_labels, counts = np.unique(y\_original, return\_counts=True)  
  
print("不同类别样本的个数：")  
for label, count in zip(unique\_labels, counts):  
 print(f"Class {label}: {count} samples")  
# 对标签数据进行LabelEncoder编码  
label\_encoder = LabelEncoder()  
y = label\_encoder.fit\_transform(car\_df.iloc[:, -1])  
  
# (2) 选择一种average评价指标，使用逻辑回归模型训练数据，并输出五折交叉验证的正确率、精确率、召回率和F1分数  
# scoring = ['accuracy', 'precision\_macro', 'recall\_macro', 'f1\_macro']  
scoring = {  
 'accuracy': make\_scorer(accuracy\_score),  
 'precision': make\_scorer(precision\_score, zero\_division=0,average='weighted'),  
 'recall': make\_scorer(recall\_score, zero\_division=0,average='weighted'),  
 'f1': make\_scorer(f1\_score, zero\_division=0,average='weighted'),  
}  
log\_reg = LogisticRegression()  
  
# cv\_results = cross\_validate(log\_reg, X, y, cv=5, scoring=scoring,error\_score=0)  
for key,value in scoring.items():  
 scores = cross\_val\_score(log\_reg, X, y, cv=5, scoring=value)  
 scoring[key]=scores  
 # print(scores)  
accuracy\_avg = np.mean(scoring['accuracy'])  
precision\_avg = np.mean(scoring['precision'])  
recall\_avg = np.mean(scoring['recall'])  
f1\_avg = np.mean(scoring['f1'])  
  
print("\n五折交叉验证结果：")  
print(f"平均正确率：{accuracy\_avg}")  
print(f"平均精确率：{precision\_avg}")  
print(f"平均召回率：{recall\_avg}")  
print(f"平均F1分数：{f1\_avg}")  
  
# (3) 输出五折交叉验证的性能报告  
print("\n五折交叉验证性能报告：")  
log\_reg.fit(X, y) # 使用全部数据进行训练  
# print(X)  
y\_pred = cross\_val\_predict(log\_reg, X, y, cv=5) # 预测  
print(classification\_report(y, y\_pred))  
  
# (4) 生成一个测试样本，使用模型进行预测，并输出预测值  
# 假设新的测试样本为第一行数据  
test\_sample = X1.iloc[0, :].values.reshape(1, -1)  
predicted\_class = log\_reg.predict(test\_sample)  
predicted\_class\_label = label\_encoder.inverse\_transform(predicted\_class)  
print(f"\n测试样本的预测类别为：{predicted\_class\_label}")



3、读取data\_nn\_classifier.txt数据

（1）使用KNN分类模型训练数据，分别输出数据集在average='macro'和average='weighted'时的精确率、召回率和F1分数

（2）画出测试集的混淆矩阵图，及测试集的性能报告

（3）画KNN分类模型图（包含分类边界）

（4）输入新测试样本，输出新样本的预测值，并对画出KNN分类结果图（包含新样本，新样本的标签及K个邻近点）

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.metrics import confusion\_matrix, classification\_report, precision\_score, recall\_score, f1\_score  
from mlxtend.plotting import plot\_decision\_regions  
import seaborn as sns  
  
# 读取数据  
data = pd.read\_csv("data\_nn\_classifier.txt")  
  
# 提取特征和标签  
X = data.iloc[:, :-1].values  
y = data.iloc[:, -1].values  
  
# 划分训练集和测试集  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# 训练KNN分类器  
knn = KNeighborsClassifier()  
knn.fit(X, y)  
  
# 任务(1): 输出精确率、召回率和F1分数  
for average in ['macro', 'weighted']:  
 y\_pred = knn.predict(X)  
 precision = precision\_score(y, y\_pred, average=average)  
 recall = recall\_score(y, y\_pred, average=average)  
 f1 = f1\_score(y, y\_pred, average=average)  
 print(f"Average: {average}")  
 print(f"Precision: {precision}")  
 print(f"Recall: {recall}")  
 print(f"F1 Score: {f1}")  
  
# 任务(2): 绘制混淆矩阵图和性能报告  
y\_pred = knn.predict(X\_test)  
conf\_matrix = confusion\_matrix(y\_test, y\_pred)  
plt.figure(figsize=(8, 6))  
sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues')  
plt.xlabel('Predicted labels')  
plt.ylabel('True labels')  
plt.title('Confusion Matrix')  
plt.show()  
  
# print("Confusion Matrix:")  
# print(conf\_matrix)  
print("Classification Report:")  
print(classification\_report(y\_test, y\_pred))  
  
# 任务(3): 绘制KNN分类模型图  
plt.figure(figsize=(10, 6))  
plot\_decision\_regions(X\_train, y\_train, clf=knn, legend=2)  
plt.xlabel('Feature 1')  
plt.ylabel('Feature 2')  
plt.title('KNN Classifier Decision Regions')  
plt.show()  
  
# 任务(4): 输入新测试样本，输出预测值，并绘制KNN分类结果图  
# 假设新测试样本为X\_new  
X\_new = np.array([[1.5, 2.5]]) # 请根据实际情况修改新测试样本的值  
y\_new\_pred = knn.predict(X\_new)  
print("Predicted Class for New Sample:", y\_new\_pred)  
  
from sklearn.neighbors import NearestNeighbors  
  
# 找到新样本的K个最近邻点  
k = 5 # 假设K值为5  
neigh = NearestNeighbors(n\_neighbors=k)  
neigh.fit(X)  
distances, indices = neigh.kneighbors(X\_new)  
label = 'New Sample (Predicted to be' + str(y\_new\_pred) + ')'  
# print(indices)  
# 绘制KNN分类结果图  
plt.figure(figsize=(10, 6))  
# plot\_decision\_regions(X, y, clf=knn, legend=0, scatter\_kwargs={'alpha': 0, 'label': None})  
plot\_decision\_regions(X, y, clf=knn, legend=2)  
l1 = plt.scatter(X\_new[:, 0], X\_new[:, 1], c='r', marker='x', label=label)  
l2 = plt.scatter(X[indices[0], 0], X[indices[0], 1], c='g', marker='o', label='Nearest Neighbors')  
plt.xlabel('Feature 1')  
plt.ylabel('Feature 2')  
plt.title('KNN Classifier Decision Regions with New Sample and Nearest Neighbors')  
# plt.get\_legend().remove()  
# plt.legend([l1, l2], [label, 'Nearest Neighbors'])  
plt.legend()  
plt.show()

