**武汉大学计算机学院**

**2022 --- 2023学年度第2学期**

**《模式识别》课程实验及实验报告要求**

## 实验环境

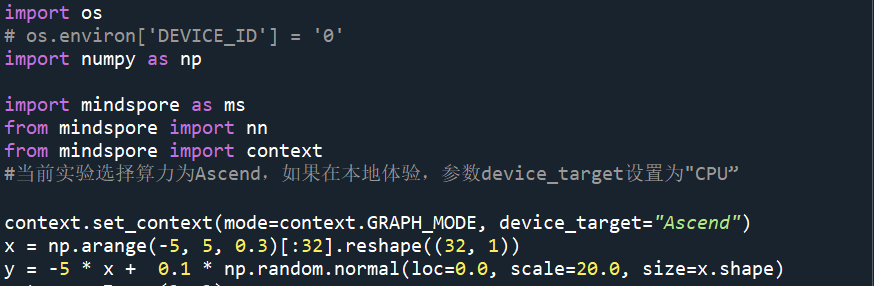
MindSpore 1.7

华为云ModelArts

## 实验内容

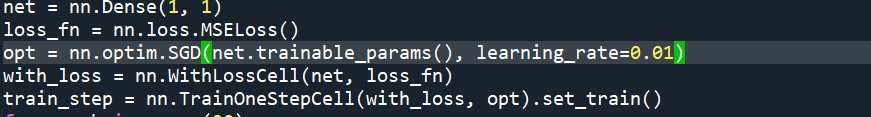
### 2.1 线性回归模拟实验

#### 2.1.1 模拟数据



导入Mindspore模块和辅助模块，并且依据线性函数y = -5 \* x + 0.1生成模拟数据，并在其中加入少许扰动。

#### 2.1.2 建立模型

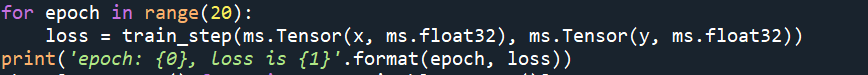


使用MindSpore提供的nn.Dense(1, 1)算子作为线性模型，其中(1, 1)表示线性模型的输入和输出皆是1维，即w是1x1的矩阵。算子会随机初始化权重w和偏置b。

y = w \* x + b

采用均方差（Mean Squared Error, MSE）作为损失函数。采用随机梯度下降（Stochastic Gradient Descent, SGD）对模型进行优化。

#### 2.1.3 使用模拟数据训练模型



输出的结果为：

epoch: 0, loss is 187.36127

epoch: 1, loss is 134.36545

epoch: 2, loss is 96.63767

epoch: 3, loss is 69.77918

epoch: 4, loss is 50.658524

epoch: 5, loss is 37.04642

epoch: 6, loss is 27.35585

epoch: 7, loss is 20.457014

epoch: 8, loss is 15.545614

epoch: 9, loss is 12.049067

epoch: 10, loss is 9.559763

epoch: 11, loss is 7.787512

epoch: 12, loss is 6.525736

epoch: 13, loss is 5.6273723

epoch: 14, loss is 4.9877214

epoch: 15, loss is 4.532255

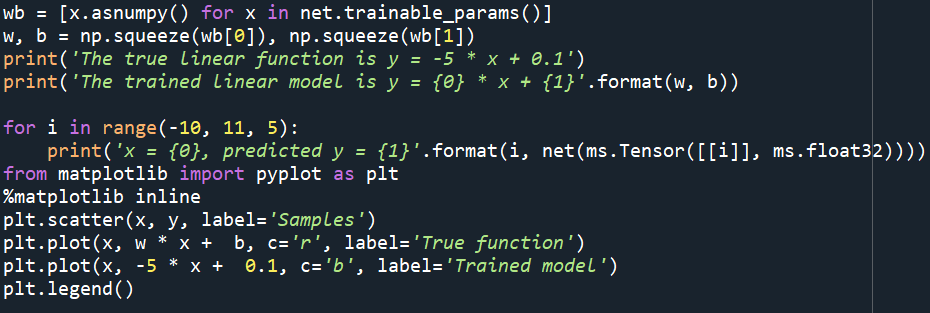
epoch: 16, loss is 4.2079134

epoch: 17, loss is 3.9769204

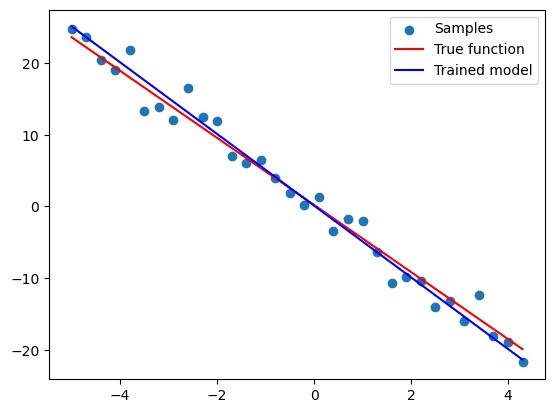
epoch: 18, loss is 3.8123863

epoch: 19, loss is 3.6951683

#### 2.1.4 预测并可视化



训练一定的轮次后，得到的模型已经十分接近真实的线性函数了，使用训练好的模型进行预测。并将模拟的样本数据、真实的线性函数和训练得到的线性模型在同一个图中可视化



模拟数据展示

#### 2.1.5 实验小结

本实验使用MindSpore对线性回归的基本概念和问题模拟；并进行实验。

### 2.2. 鸢尾花二分类实验

#### 2.2.1 数据读取与处理

步骤一：导入MindSpore模块和辅助模块

import os

os.environ['DEVICE\_ID'] = '6'

import csv

import numpy as np

import mindspore as ms

from mindspore import nn

from mindspore import context

from mindspore import dataset

from mindspore.train.callback import LossMonitor

from mindspore.common.api import ms\_function

from mindspore.ops import operations as P

#当前实验选择算力为Ascend，如果在本地体验，参数device\_target设置为"CPU”

# context.set\_context(mode=context.GRAPH\_MODE, device\_target="Ascend")

步骤二：读取Iris数据集，并查看部分数据：

with open('iris.data') as csv\_file:

data = list(csv.reader(csv\_file, delimiter=','))

print(data[40:60]) # 打印部分数据

输出：

[['5.0', '3.5', '1.3', '0.3', 'Iris-setosa'], ['4.5', '2.3', '1.3', '0.3', 'Iris-setosa'], ['4.4', '3.2', '1.3', '0.2', 'Iris-setosa'], ['5.0', '3.5', '1.6', '0.6', 'Iris-setosa'], ['5.1', '3.8', '1.9', '0.4', 'Iris-setosa'], ['4.8', '3.0', '1.4', '0.3', 'Iris-setosa'], ['5.1', '3.8', '1.6', '0.2', 'Iris-setosa'], ['4.6', '3.2', '1.4', '0.2', 'Iris-setosa'], ['5.3', '3.7', '1.5', '0.2', 'Iris-setosa'], ['5.0', '3.3', '1.4', '0.2', 'Iris-setosa'], ['7.0', '3.2', '4.7', '1.4', 'Iris-versicolor'], ['6.4', '3.2', '4.5', '1.5', 'Iris-versicolor'], ['6.9', '3.1', '4.9', '1.5', 'Iris-versicolor'], ['5.5', '2.3', '4.0', '1.3', 'Iris-versicolor'], ['6.5', '2.8', '4.6', '1.5', 'Iris-versicolor'], ['5.7', '2.8', '4.5', '1.3', 'Iris-versicolor'], ['6.3', '3.3', '4.7', '1.6', 'Iris-versicolor'], ['4.9', '2.4', '3.3', '1.0', 'Iris-versicolor'], ['6.6', '2.9', '4.6', '1.3', 'Iris-versicolor'], ['5.2', '2.7', '3.9', '1.4', 'Iris-versicolor']]

步骤三：抽取样本

取前两类样本（共100条），将数据集的4个属性作为自变量X。将数据集的2个类别映射为{0, 1}，作为因变量Y。

label\_map = {

'Iris-setosa': 0,

'Iris-versicolor': 1,

}

X = np.array([[float(x) for x in s[:-1]] for s in data[:100]], np.float32)

Y = np.array([[label\_map[s[-1]]] for s in data[:100]], np.float32)

步骤四：样本可视化

取样本的前两个属性进行2维可视化，可以看到在前两个属性上两类样本是线性可分的。

from matplotlib import pyplot as plt

%matplotlib inline

plt.scatter(X[:50, 0], X[:50, 1], label='Iris-setosa')

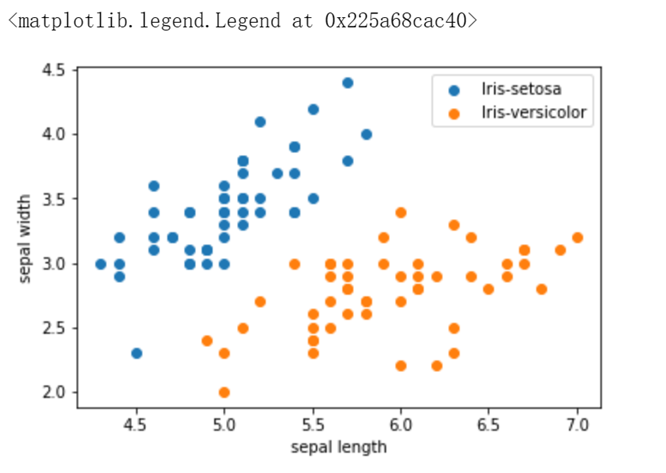
plt.scatter(X[50:, 0], X[50:, 1], label='Iris-versicolor')

plt.xlabel('sepal length')

plt.ylabel('sepal width')

plt.legend()

输出：



步骤五：分割数据集

将数据集按8:2划分为训练集和验证集：

train\_idx = np.random.choice(100, 80, replace=False)

test\_idx = np.array(list(set(range(100)) - set(train\_idx)))

X\_train, Y\_train = X[train\_idx], Y[train\_idx]

X\_test, Y\_test = X[test\_idx], Y[test\_idx]

步骤六：数据类型转换

使用MindSpore的GeneratorDataset接口将numpy.ndarray类型的数据转换为Dataset：

XY\_train = list(zip(X\_train, Y\_train))

ds\_train = dataset.GeneratorDataset(XY\_train, ['x', 'y'])

# ds\_train.set\_dataset\_size(80)

ds\_train = ds\_train.shuffle(buffer\_size=80).batch(32, drop\_remainder=True)

#### 2.2.2 模型建立与训练

步骤一：可视化逻辑回归函数

逻辑回归常用的联系函数是Sigmoid（S形函数），Sigmoid函数如下图所示，可以将连续值映射到{0, 1}，同时也是单调可微的。

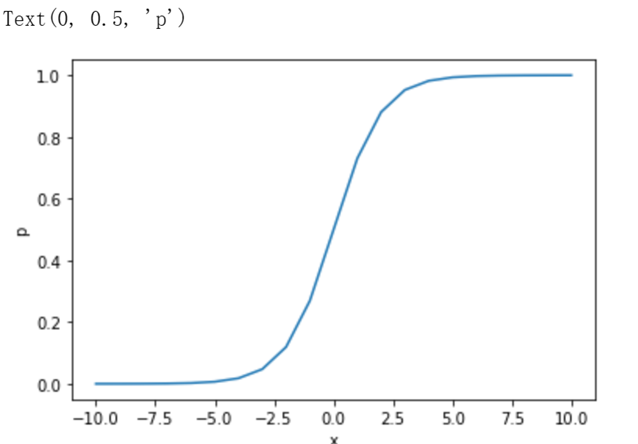
coor\_x = np.arange(-10, 11, dtype=np.float32)

coor\_y = nn.Sigmoid()(ms.Tensor(coor\_x)).asnumpy()

plt.plot(coor\_x, coor\_y)

plt.xlabel('x')

plt.ylabel('p')

输出：

步骤二：建模

# 自定义Loss

class Loss(nn.Cell):

def \_\_init\_\_(self):

super(Loss, self).\_\_init\_\_()

self.sigmoid\_cross\_entropy\_with\_logits = P.SigmoidCrossEntropyWithLogits()

self.reduce\_mean = P.ReduceMean(keep\_dims=False)

def construct(self, x, y):

loss = self.sigmoid\_cross\_entropy\_with\_logits(x, y)

return self.reduce\_mean(loss, -1)

net = nn.Dense(4,1)

loss = Loss()

opt = nn.optim.SGD(net.trainable\_params(), learning\_rate=0.003)

步骤三：模型训练：

使用2分类的Iris数据集对模型进行几代（Epoch）训练：

model = ms.train.Model(net, loss, opt)

model.train(10, ds\_train, callbacks=[LossMonitor(per\_print\_times=ds\_train.get\_dataset\_size())], dataset\_sink\_mode=False)

输出：

epoch: 1 step: 2, loss is 0.6503548622131348

epoch: 2 step: 2, loss is 0.5764381885528564

epoch: 3 step: 2, loss is 0.500321626663208

epoch: 4 step: 2, loss is 0.4692782461643219

epoch: 5 step: 2, loss is 0.4085477590560913

epoch: 6 step: 2, loss is 0.3651609420776367

epoch: 7 step: 2, loss is 0.3684331476688385

epoch: 8 step: 2, loss is 0.3218190670013428

epoch: 9 step: 2, loss is 0.2877786159515381

epoch: 10 step: 2, loss is 0.2668483555316925

#### 2.2.3 模型评估

然后计算模型在测试集上精度，测试集上的准确率达到了1.0左右，即逻辑回归模型学会了区分2类鸢尾花。

x = model.predict(ms.Tensor(X\_test)).asnumpy()

pred = np.round(1 / (1 + np.exp(-x)))

correct = np.equal(pred, Y\_test)

acc = np.mean(correct)

print('Test accuracy is', acc)

输出：

Test accuracy is 1.0

#### 2.2.4 实验小结

本实验使用MindSpore实现了逻辑回归，用来解决2分类问题。在Iris数据集上进行训练后，所得的模型可以很好的表示每个样本类别y和属性x的关系。

#### 2.2.5 创新设计

使用Softmax函数作为概率映射函数，对完整的Iris数据集实现多分类任务。

from sklearn.datasets import load\_iris

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

#读取数据集

data= load\_iris()

iris\_target = data.target #得到数据对应的标签

iris\_features = pd.DataFrame(data=data.data, columns=data.feature\_names) #转化为DataFrame格式

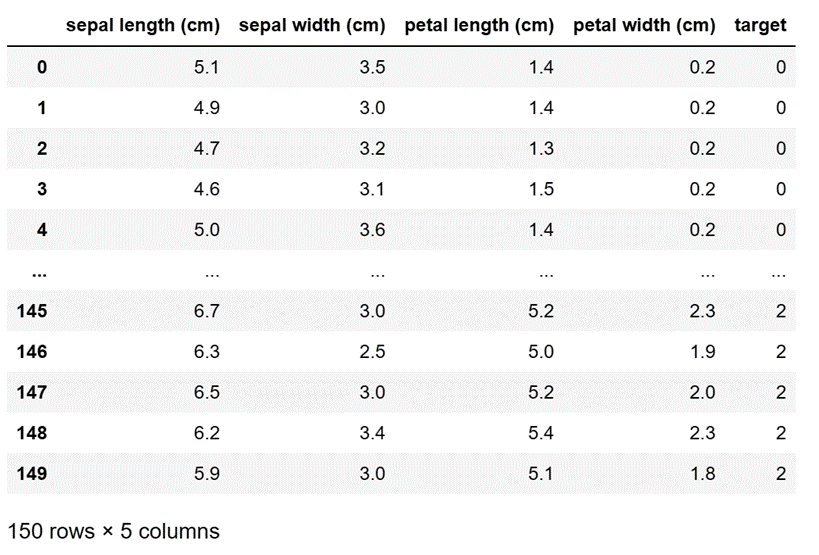
iris\_test=iris\_features.copy()

#样本可视化

iris\_test = iris\_features.copy() ##进行浅拷贝，防止对于原始数据的修改

iris\_test['target'] = iris\_target

iris\_test



# 选取其前二个特征绘制二维散点图

iris\_all=iris\_test

from mpl\_toolkits.mplot3d import Axes3D

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

ax = fig.add\_subplot(111)

iris\_all\_class0 = iris\_all[iris\_all['target']==0].values

iris\_all\_class1 = iris\_all[iris\_all['target']==1].values

iris\_all\_class2 = iris\_all[iris\_all['target']==2].values

# 'setosa'(0), 'versicolor'(1), 'virginica'(2)

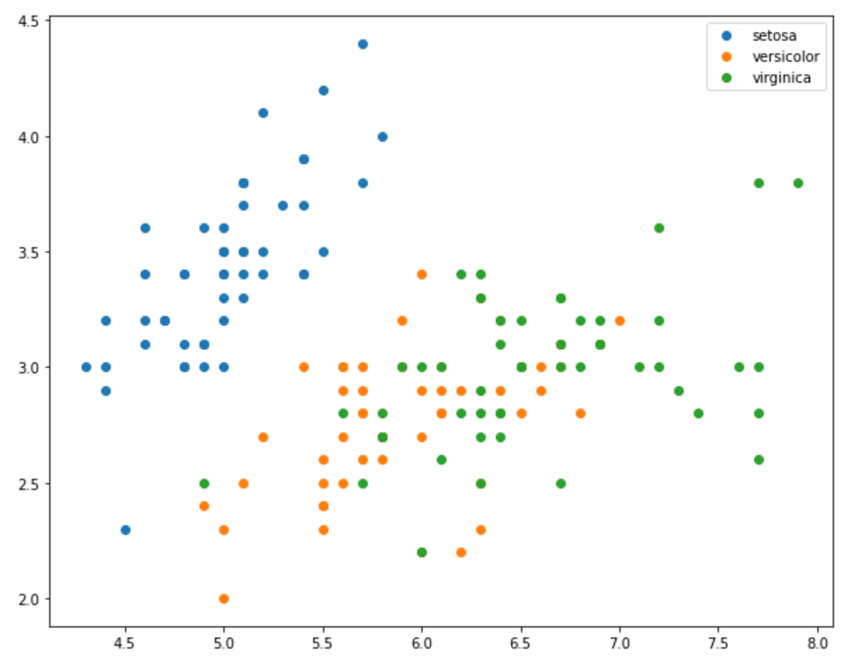
ax.scatter(iris\_all\_class0[:,0], iris\_all\_class0[:,1],label='setosa')

ax.scatter(iris\_all\_class1[:,0], iris\_all\_class1[:,1],label='versicolor')

ax.scatter(iris\_all\_class2[:,0], iris\_all\_class2[:,1],label='virginica')

plt.legend()

plt.show()



#分割数据集(80%/20%区分)

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(iris\_features, iris\_target, test\_size = 0.2, random\_state = 2020)

from sklearn.linear\_model import LogisticRegression

clf = LogisticRegression(random\_state=0, solver='lbfgs')

clf.fit(x\_train, y\_train)

## 在训练集和测试集上分布利用训练好的模型进行预测

train\_predict = clf.predict(x\_train)

test\_predict = clf.predict(x\_test)

## 查看其对应的w

print('the weight of Logistic Regression:',clf.coef\_)

## 查看其对应的b

print('the intercept(w0) of Logistic Regression:',clf.intercept\_)

from sklearn import metrics

## 利用accuracy（准确度）【预测正确的样本数目占总预测样本数目的比例】评估模型效果

print('The accuracy of the Logistic Regression is:',metrics.accuracy\_score(y\_train,train\_predict))

print('The accuracy of the Logistic Regression is:',metrics.accuracy\_score(y\_test,test\_predict))

## 查看混淆矩阵

confusion\_matrix\_result = metrics.confusion\_matrix(test\_predict,y\_test)

print('The confusion matrix result:\n',confusion\_matrix\_result)

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

sns.heatmap(confusion\_matrix\_result, annot=True, cmap='Blues')

plt.xlabel('Predicted labels')

plt.ylabel('True labels')

plt.show()

输出：

the weight of Logistic Regression: [[-0.45928925 0.83069887 -2.26606531 -0.99743981]

[ 0.33117319 -0.72863424 -0.06841147 -0.9871103 ]

[ 0.12811606 -0.10206464 2.33447678 1.98455011]]

the intercept(w0) of Logistic Regression: [ 9.4388067 3.93047364 -13.36928034]

The accuracy of the Logistic Regression is: 0.9833333333333333

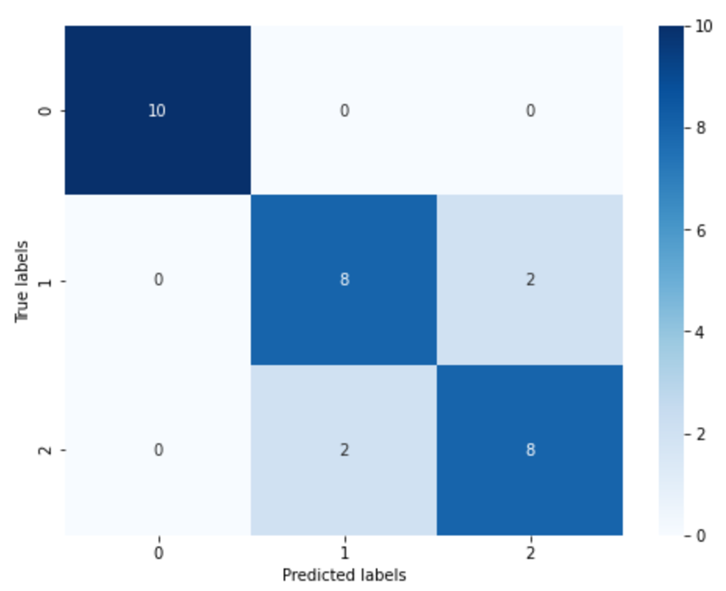
The accuracy of the Logistic Regression is: 0.8666666666666667

The confusion matrix result:

[[10 0 0]

[ 0 8 2]

[ 0 2 8]]



### 2.3 红酒分类实验

#### 2.3.1 数据读取与处理

步骤一：导入MIndSpore模块和辅助模块

import os

# os.environ['DEVICE\_ID'] = '4'

import csv

import numpy as np

import mindspore as ms

from mindspore import context

from mindspore import nn

from mindspore.ops import operations as P

from mindspore.ops import functional as F

context.set\_context(device\_target="Ascend")

步骤二：读取Wine数据集wine.data，并查看部分数据

with open('wine.data') as csv\_file:

data = list(csv.reader(csv\_file, delimiter=','))

print(data[56:62]+data[130:133]) # 打印部分数据

输出：

[['1', '14.22', '1.7', '2.3', '16.3', '118', '3.2', '3', '.26', '2.03', '6.38', '.94', '3.31', '970'], ['1', '13.29', '1.97', '2.68', '16.8', '102', '3', '3.23', '.31', '1.66', '6', '1.07', '2.84', '1270'], ['1', '13.72', '1.43', '2.5', '16.7', '108', '3.4', '3.67', '.19', '2.04', '6.8', '.89', '2.87', '1285'], ['2', '12.37', '.94', '1.36', '10.6', '88', '1.98', '.57', '.28', '.42', '1.95', '1.05', '1.82', '520'], ['2', '12.33', '1.1', '2.28', '16', '101', '2.05', '1.09', '.63', '.41', '3.27', '1.25', '1.67', '680'], ['2', '12.64', '1.36', '2.02', '16.8', '100', '2.02', '1.41', '.53', '.62', '5.75', '.98', '1.59', '450'], ['3', '12.86', '1.35', '2.32', '18', '122', '1.51', '1.25', '.21', '.94', '4.1', '.76', '1.29', '630'], ['3', '12.88', '2.99', '2.4', '20', '104', '1.3', '1.22', '.24', '.83', '5.4', '.74', '1.42', '530'], ['3', '12.81', '2.31', '2.4', '24', '98', '1.15', '1.09', '.27', '.83', '5.7', '.66', '1.36', '560']]

步骤三：抽取样本，取三类样本（共178条），将数据集的13个属性作为自变量X。将数据集的3个类别作为因变量Y。

X = np.array([[float(x) for x in s[1:]] for s in data[:178]], np.float32)

Y = np.array([s[0] for s in data[:178]], np.int32)

步骤四：样本可视化，取样本的某两个属性进行2维可视化，可以看到在某两个属性上样本的分布情况以及可分性。

from matplotlib import pyplot as plt

%matplotlib inline

attrs = ['Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash', 'Magnesium', 'Total phenols',

'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins', 'Color intensity', 'Hue',

'OD280/OD315 of diluted wines', 'Proline']

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

for i in range(0, 4):

plt.subplot(2, 2, i+1)

a1, a2 = 2 \* i, 2 \* i + 1

plt.scatter(X[:59, a1], X[:59, a2], label='1')

plt.scatter(X[59:130, a1], X[59:130, a2], label='2')

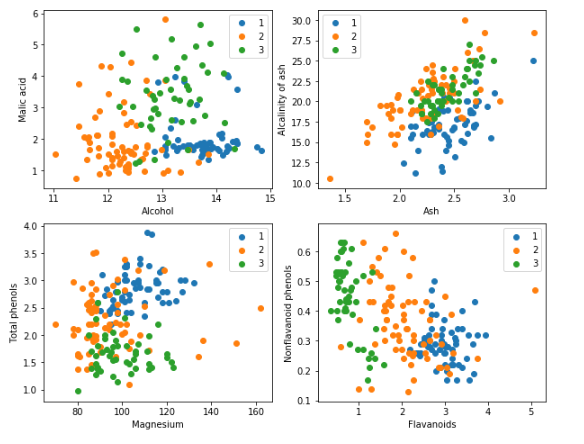
plt.scatter(X[130:, a1], X[130:, a2], label='3')

plt.xlabel(attrs[a1])

plt.ylabel(attrs[a2])

plt.legend()

plt.show()

输出：

步骤五：分割数据集，将数据集按128:50划分为训练集（已知类别样本）和验证集（待验证样本）：

train\_idx = np.random.choice(178, 128, replace=False)

test\_idx = np.array(list(set(range(178)) - set(train\_idx)))

X\_train, Y\_train = X[train\_idx], Y[train\_idx]

X\_test, Y\_test = X[test\_idx], Y[test\_idx]

#### 2.3.2 计算距离

利用MindSpore提供的tile, suqare, ReduceSum, sqrt, TopK等算子，通过矩阵运算的方式同时计算输入样本x和已明确分类的其他样本X\_train的距离，并计算出top k近邻。

class KnnNet(nn.Cell):

def \_\_init\_\_(self, k):

super(KnnNet, self).\_\_init\_\_()

self.tile = P.Tile()

self.sum = P.ReduceSum()

self.topk = P.TopK()

self.k = k

def construct(self, x, X\_train):

# Tile input x to match the number of samples in X\_train

x\_tile = self.tile(x, (128, 1))

square\_diff = F.square(x\_tile - X\_train)

square\_dist = self.sum(square\_diff, 1)

dist = F.sqrt(square\_dist)

# -dist mean the bigger the value is, the nearer the samples are

values, indices = self.topk(-dist, self.k)

return indices

def knn(knn\_net, x, X\_train, Y\_train):

x, X\_train = ms.Tensor(x), ms.Tensor(X\_train)

indices = knn\_net(x, X\_train)

topk\_cls = [0]\*len(indices.asnumpy())

for idx in indices.asnumpy():

topk\_cls[Y\_train[idx]] += 1

cls = np.argmax(topk\_cls)

return cls

#### 2.3.3 预测

acc = 0

knn\_net = KnnNet(5)

for x, y in zip(X\_test, Y\_test):

pred = knn(knn\_net, x, X\_train, Y\_train)

acc += (pred == y)

print('label: %d, prediction: %s' % (y, pred))

print('Validation accuracy is %f' % (acc/len(Y\_test)))

输出：

label: 1, prediction: 1

label: 3, prediction: 3

label: 3, prediction: 2

label: 1, prediction: 1

label: 1, prediction: 1

label: 3, prediction: 2

label: 1, prediction: 3

label: 3, prediction: 2

label: 1, prediction: 3

label: 1, prediction: 3

label: 3, prediction: 2

label: 1, prediction: 1

label: 3, prediction: 2

label: 3, prediction: 2

label: 3, prediction: 2

label: 3, prediction: 3

label: 1, prediction: 1

label: 1, prediction: 1

label: 1, prediction: 3

label: 3, prediction: 2

label: 3, prediction: 3

label: 3, prediction: 3

label: 3, prediction: 3

label: 1, prediction: 1

label: 1, prediction: 1

label: 1, prediction: 1

label: 1, prediction: 1

label: 1, prediction: 1

label: 1, prediction: 1

label: 1, prediction: 1

label: 2, prediction: 2

label: 2, prediction: 2

label: 2, prediction: 2

label: 2, prediction: 2

label: 2, prediction: 1

label: 2, prediction: 1

label: 2, prediction: 2

label: 2, prediction: 2

label: 2, prediction: 2

label: 2, prediction: 2

label: 2, prediction: 3

label: 2, prediction: 2

label: 2, prediction: 2

label: 2, prediction: 2

label: 2, prediction: 2

label: 2, prediction: 2

label: 2, prediction: 2

label: 2, prediction: 2

label: 2, prediction: 3

label: 2, prediction: 2

Validation accuracy is 0.680000

#### 1.3.4 实验小结

本实验使用MindSpore实现了KNN算法，用来解决3分类问题。取wine数据集上的3类样本，分为已知类别样本和待验证样本，从验证结果可以看出KNN算法在该任务上有效，能根据酒的13种属性判断出酒的品种。

## 实验内容对课程目标的支撑

本章实验涉及线性回归、逻辑回归、KNN回归等算法，通过不同算法的效果对比来加深对算法的理解。

## 实验要求

* 掌握使用MindSpore进行线性回归
* 掌握使用MindSpore进行逻辑回归
* 掌握使用MindSpore进行KNN分类
* 掌握回归与分类任务的区别与流程
* 掌握线性回归、逻辑回归、KNN分类等算法的原理与使用
* 掌握模型评估的方法

线性回归模拟实验

了解线性回归的基本概念和问题模拟；

了解如何使用MindSpore进行线性回归实验。

鸢尾花二分类实验

* 了解逻辑回归的基本概念；
* 了解如何使用MindSpore进行逻辑回归实验。

红酒分类实验

* 了解KNN的基本概念；
* 了解如何使用MindSpore进行KNN实验。

## 实验完成时间和提交方式

完成时间：2023/3/23

提交方式：word文档

## 实验报告要求