《机器学习》

实验指导书 1

杭州电子科技大学

华为技术有限公司

目录

一. 线性模型

1.1. 线性回归实验

1.1.1 实验介绍

1.1.1.1 简介

线性回归（Linear Regression）是机器学习最经典的算法之一，具有如下特点：

- 自变量服从正态分布；
- 因变量是连续性数值变量；
- 自变量和因变量呈线性关系。

本实验主要介绍使用 MindSpore 在模拟数据上进行线性回归实验，分析自变量和因变量之间的线性关系，即求得一个线性函数。

1.1.1.2 实验目的

- 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。

1.1.2 实验环境要求

- MindSpore 1.3（MindSpore 版本会定期更新，本指导也会定期刷新，与版本配套）；
- 华为云 ModelArts（控制台左上角选择“华北-北京四”）：ModelArts 是华为云提供的面向开发者的一站式 AI 开发平台，集成了昇腾 AI 处理器资源池，用户可以在该平台下体验 MindSpore。

1.1.3 实验步骤

1.1.3.1 ModelArts 开发平台环境准备

ModelArts 是面向开发者的一站式 AI 开发平台，为机器学习与深度学习提供海量数据预处理及半自动化标注、大规模分布式 Training、自动化模型生成，及端-边-云模型按需部署能力，帮助用户快速创建和部署模型，管理全周期 AI 工作流。

步骤 1 登录华为云，进入 ModelArts 控制台

图 1-1 ModelArts 平台入口

图 1-2 点击进入控制台

图 1-3 ModelArts 主页面

步骤 2 创建 ModelArts Notebook

ModelArts Notebook 提供网页版的 Python 开发环境，可以方便的编写、运行代码，并查看运行结果。

（1）在 ModelArts 服务主界面依次点击“开发环境”、“创建”。首次使用时，可能需要验证 AK/SK，按照之前获取的秘钥进行认证即可。

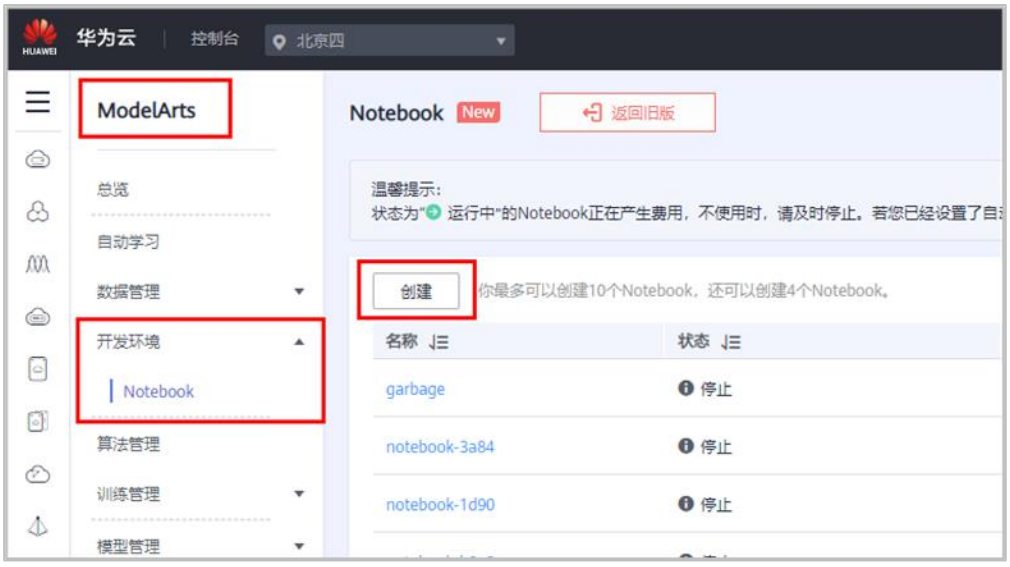


图 1-4 创建 ModelArts Notebook

（2）填写 Notebook 所需的参数，例如：

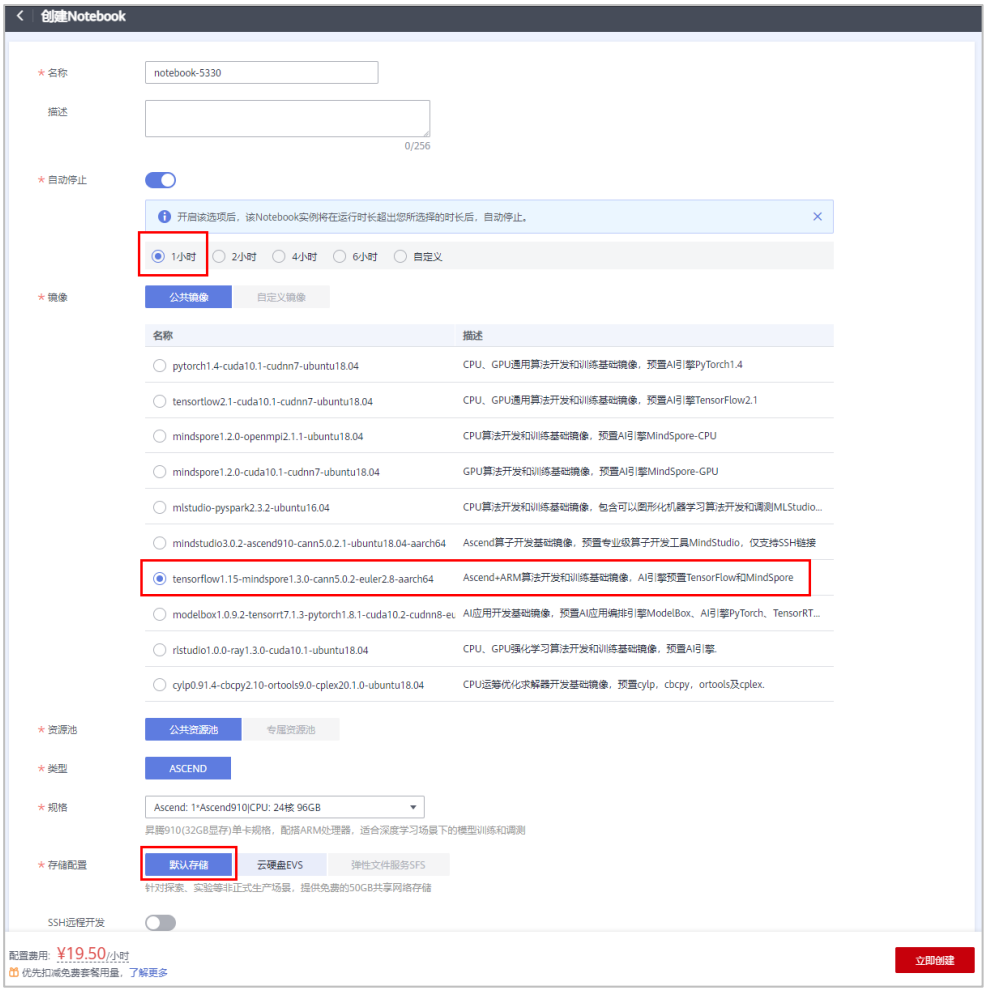


图 1-5 填写 Notebook 参数

表 1-1 Notebook 参数说明

参数名称	说明
“计费方式”	按需计费。当前仅支持按需计费，无需修改。
“名称”	Notebook 的名称。只能包含数字、大小写字母、下划线和中划线，长度不能超过 64 位且不能为空。
“描述”	对 Notebook 的简要描述。
“自动停止”	默认开启，且默认值为“1 小时后”，表示该 Notebook 实例将在运行 1 小时之后自动停止，即 1 小时后停止计费。 开启自动停止功能后，可选择“1 小时后”、“2 小时后”、“4 小时后”、“6 小时后”或“自定义”几种模式。选择“自定义”模式时，可指定 1~24 小时范围内任意整数。
“工作环境”	当前支持 2 种工作环境，分别为“Python2”和“Python3”，不同工作环境其对应可使用的 AI 引擎不同，详细支持列表请参见支持的 AI 引擎。 如果需要使用 TensorFlow 2.X、PyTorch 1.4.0 或者 R 语言版本的 AI 框架，

	则需要选择“TF-2.1.0&Pytorch-1.4.0-python3.6”的工作环境。如果选择此类型的工作环境，暂时无法使用免费规格，建议选择其他规格。 每个工作环境多种 AI 引擎，可以在同一个 Notebook 实例中使用所有支持的 AI 引擎，不同的引擎之间可快速、方便的切换，并且有独立的运行环境。您可以在 Notebook 实例创建完成后，进入 Jupyter 页面创建对应 AI 引擎的开发环境。 **说明：** ModelArts 还支持 Keras 引擎，详细说明请参见 ModelArts 是否支持 Keras 引擎？
“资源池”	可选公共资源池和专属资源池，关于 ModelArts 专属资源池的介绍和购买，请参见资源池。
“类型”	支持 CPU、GPU 和 Ascend 类型。GPU 性能更佳，但是相对 CPU 而言，费用更高。Ascend 类型为公测资源，请提前完成 Ascend 910 公测申请。
“规格”	CPU 规格支持：“2 核 8GiB”、“8 核 32GiB”；GPU 规格支持：“8 核 64GiB 1*p100”；只有选择“公共资源池”时，需要选择规格。根据选择的类型不同，可选规格也不同。Ascend 规格支持：“Ascend: 1*Ascend 910 CPU: 24 核 96GiB” CPU 规格支持：“[限时免费]体验规格 CPU 版”、“2 核 8GiB”、“8 核 32GiB”；GPU 规格支持：“[限时免费]体验规格 GPU 版”、“GPU: 1*v100NV32 CPU: 8 核 64GiB”。如果选择“限时免费”规格，请仔细阅读界面提示，并勾选“我已阅读并同意以上内容”。
“存储配置”	存储配置可选“云硬盘”和“对象存储服务”。 • 选择“云硬盘”作为存储位置 根据实际使用量设置磁盘规格。磁盘规格默认 5GB。ModelArts 提供 5GB 容量供用户免费使用。超出 5GB 时，超出部分每 GB 按“超高 IO”类型的收费标准进行按需收费。磁盘规格的取值范围为 5GB～4096GB。 选择此模式，用户在 Notebook 列表的所有文件读写操作都是针对容器中的内容操作，与 OBS 无关；重启该实例，内容不丢失。 选择“对象存储服务”作为存储位置 在“存储位置”右侧单击“选择”，设置用于存储 Notebook 数据的 OBS 路径。如果想直接使用已有的文件或数据，可将数据提前上传至对应的 OBS 路径下。“存储位置”不能设置为 OBS 桶的根目录，需设置为对应 OBS 桶下的具体目录。 选择此模式，用户在 Notebook 列表的所有文件读写操作是基于所选择的 OBS 路径下的内容操作，与当前实例空间无关。如果您需要将内容同步到实例空间，先选中该内容，单击“Sync OBS”，即可将所选内容同步到当前容器空间，详细操作可参见与 OBS 同步文件。重启该实例时，内容不丢失。
“Git 存储库”	只有当“存储配置”选择“云硬盘”时，支持此参数。

	开启此功能后，可创建一个带有 Git 存储库的 Notebook 实例，系统将自动从 Github 同步代码库。详细配置说明请参见创建带有 Git 存储库的 Notebook 实例。

（3）配置好 Notebook 参数后，点击下一步，进入 Notebook 信息预览。确认无误后，点击“立即创建”。

（4）创建完成后，返回开发环境主界面，等待 Notebook 创建完毕后，打开 Notebook，进行下一步操作。

注意：开发框架使用 MindSpore 的实验，需要选择 Notebook 的配置为：Python3、Ascend、存储位置默认为 OBS，如下图。

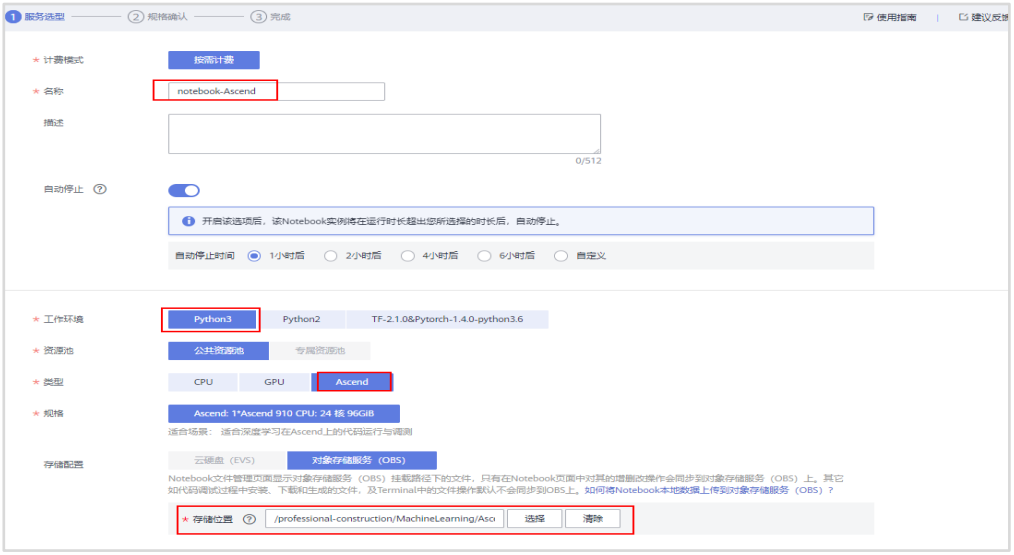


图 1-6 使用 MindSpore 时所需填写的 Notebook 参数

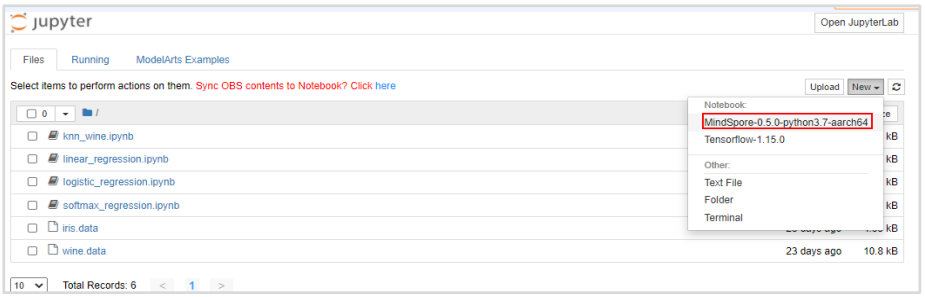


图 1-7 创建 MindSpore 的开发环境

步骤 3 在 Notebook 中创建开发环境

（1）点击下图所示的“打开”按钮，进入刚刚创建的 Notebook。

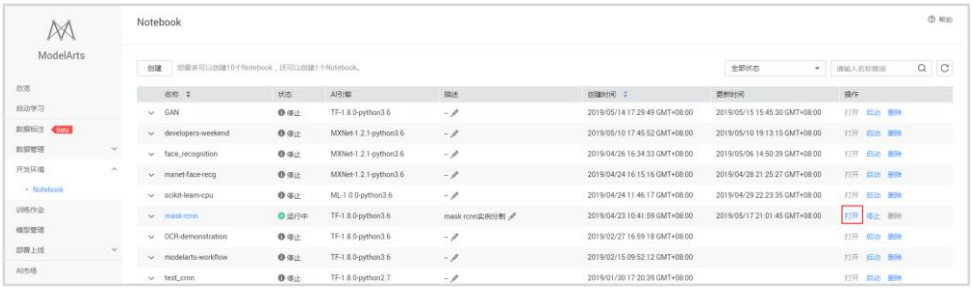
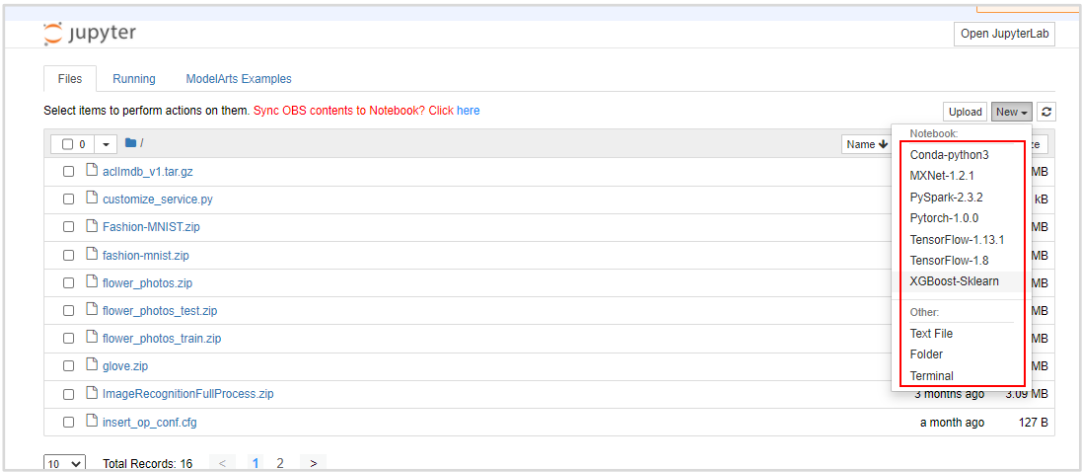


图 1-8 打开 notebook

（2）创建一个 Python3、CPU 环境的 Notebook。点击右上角的"New"，可以选择所创建的开发环境。



（3）点击左上方的文件名"Untitled"，并输入一个与本实验相关的名称，如"movie_recommendation。

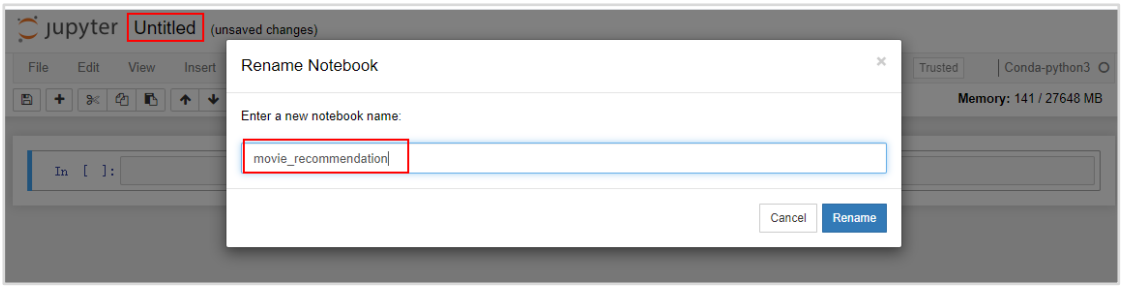


图 1-9 重命名 notebook

步骤 4 在 Notebook 中编写并执行代码

在 Notebook 中，我们输入一个简单的打印语句，然后点击上方的运行按钮，可以查看语句执行的结果：

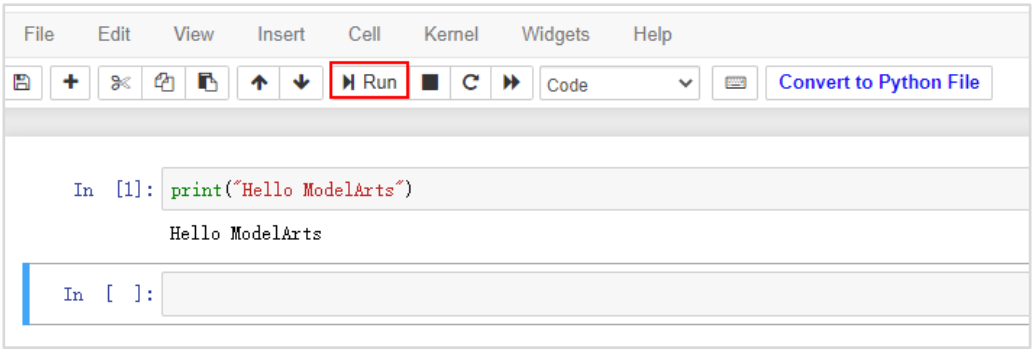


图 1-10 测试 notebook

步骤 5 保存代码

代码编写完之后，我们点击下图所示的"保存"按钮，保存代码和代码执行结果。

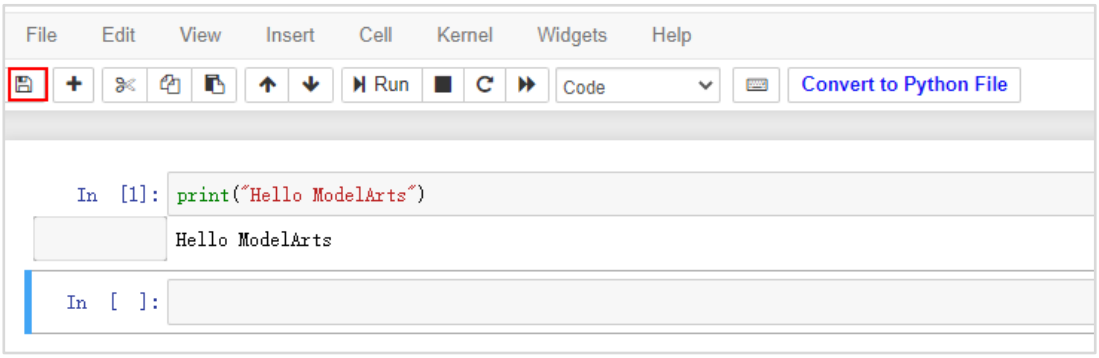
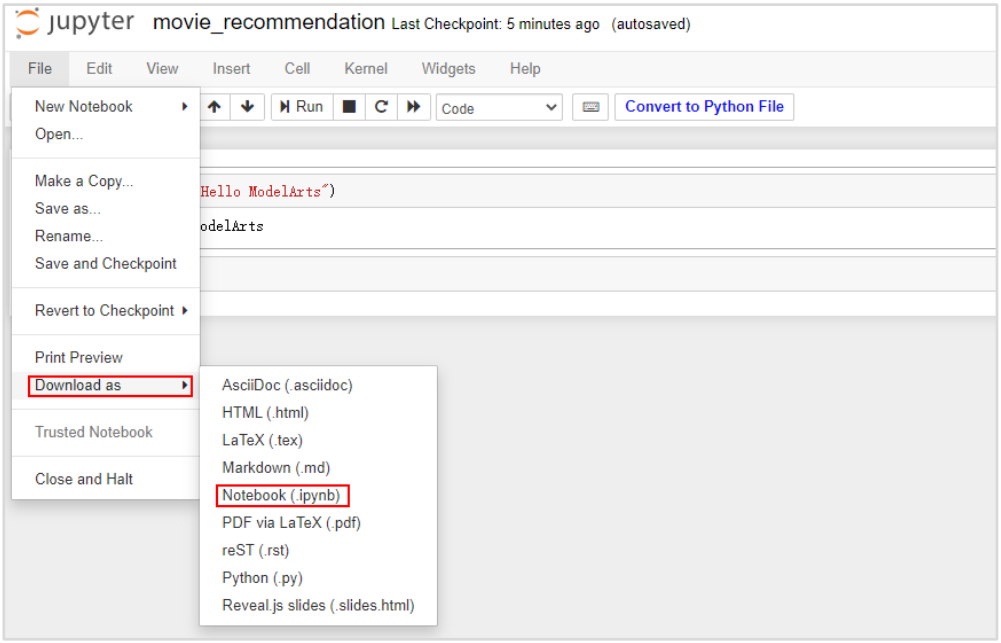
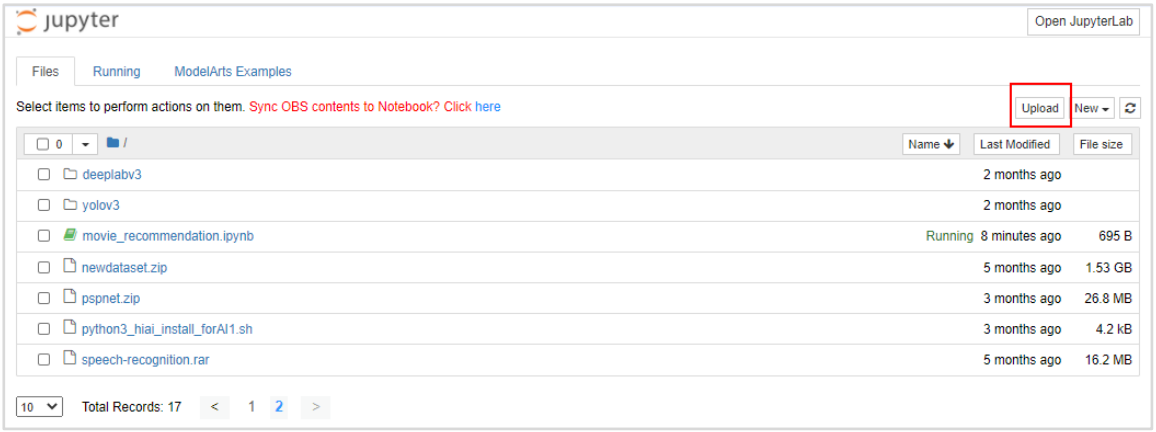


图 1-11 保存代码

步骤 6 从 ModelArts 下载代码到本地



步骤 7 从本地上传代码到 ModelArts



1.1.3.2 导入 MindSpore 模块和辅助模块

代码：

```
import os
# os.environ['DEVICE_ID'] = '0'
import numpy as np

import mindspore as ms
from mindspore import nn
from mindspore import context

context.set_context(mode=context.GRAPH_MODE, device_target="Ascend")
```

1.1.3.3 生成模拟数据

根据线性函数 y = -5 * x + 0.1 生成模拟数据，并在其中加入少许扰动。

代码：

```
x = np.arange(-5, 5, 0.3)[:32].reshape((32, 1))
y = -5 * x +   0.1 * np.random.normal(loc=0.0, scale=20.0, size=x.shape)
```

1.1.3.4 建模

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。

代码：

```
net = nn.Dense(1, 1)
loss_fn = nn.loss.MSELoss()
```

```
opt = nn.optim.SGD(net.trainable_params(), learning_rate=0.01)
with_loss = nn.WithLossCell(net, loss_fn)
train_step = nn.TrainOneStepCell(with_loss, opt).set_train()
```

1.1.3.5 使用模拟数据训练模型

代码：

```
for epoch in range(20):
    loss = train_step(ms.Tensor(x, ms.float32), ms.Tensor(y, ms.float32))
    print('epoch: {0}, loss is {1}'.format(epoch, loss))
```

输出：

```
epoch: 0, loss is 191.03662
epoch: 1, loss is 137.56923
epoch: 2, loss is 99.519455
epoch: 3, loss is 72.41922
epoch: 4, loss is 53.13565
epoch: 5, loss is 39.367752
epoch: 6, loss is 29.570751
epoch: 7, loss is 22.636707
epoch: 8, loss is 17.659052
epoch: 9, loss is 14.094089
epoch: 10, loss is 11.589828
epoch: 11, loss is 9.775642
epoch: 12, loss is 8.497474
epoch: 13, loss is 7.5611076
epoch: 14, loss is 6.9201765
epoch: 15, loss is 6.452571
epoch: 16, loss is 6.100116
epoch: 17, loss is 5.856675
epoch: 18, loss is 5.6804886
epoch: 19, loss is 5.5560365
```

1.1.3.6 使用训练好的模型进行预测

训练一定的代数后，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。

代码：

```
wb = [x.asnumpy() for x in net.trainable_params()]
w, b = np.squeeze(wb[0]), np.squeeze(wb[1])
print('The true linear function is y = -5 * x + 0.1')
print('The trained linear model is y = {0} * x + {1}'.format(w, b))

for i in range(-10, 11, 5):
    print('x = {0}, predicted y = {1}'.format(i, net(ms.Tensor([[i]], ms.float32))))
```

输出：

```
The true linear function is y = -5 * x + 0.1
```

```
The trained linear model is y = -4.842680931091309 * x + 0.03442131727933884
x = -10, predicted y = [[49.714813]]
x = -5, predicted y = [[24.974724]]
x = 0, predicted y = [[0.23463698]]
x = 5, predicted y = [[-24.505451]]
x = 10, predicted y = [[-49.245537]]
```

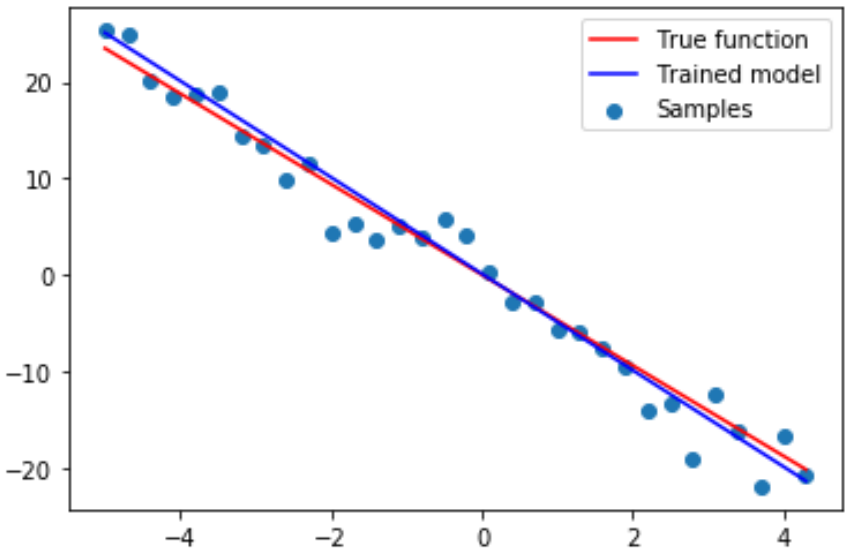
1.1.3.7 可视化

模拟的样本数据、真实的线性函数和训练得到的线性模型，如下图所示：

代码：

```
from matplotlib import pyplot as plt
plt.scatter(x, y, label='Samples')
plt.plot(x, w * x +  b, c='r', label='True function')
plt.plot(x, -5 * x +  0.1, c='b', label='Trained model')
plt.legend()
```

输出：



1.2. 鸢尾花二分类实验

1.2.1 实验介绍

1.2.1.1 简介

逻辑回归(Logistic Regression)是机器学习最经典的算法之一,与线性回归有很多不同,这两种回归都属于广义线性回归（Generalized Linear Regression）的范畴。逻辑回归具有如下特点:

- 逻辑回归对自变量分布没有要求;
- 因变量是离散型变量，即分类变量;
- 逻辑回归分析的是因变量取某个值的概率与自变量的关系。

本实验主要介绍使用 MindSpore 在 2 分类数据集上进行逻辑回归实验，分析自变量和因变量（概率）之间的关系，即求得一个概率函数。

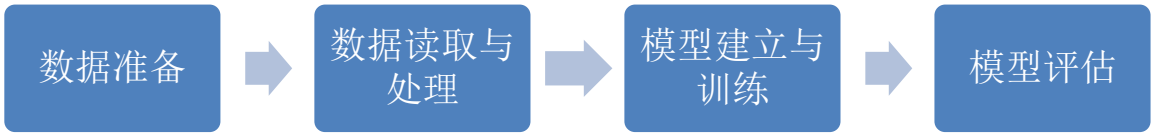
1.2.1.2 实验目的

- 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。

1.2.2 实验环境要求

- MindSpore 1.3（MindSpore 版本会定期更新，本指导也会定期刷新，与版本配套）;
- 华为云 ModelArts（控制台左上角选择“华北-北京四”）: ModelArts 是华为云提供的面向开发者的一站式 AI 开发平台，集成了昇腾 AI 处理器资源池，用户可以在该平台下体验 MindSpore。

1.2.3 实验总体设计



1.2.4 实验过程

1.2.4.1 数据准备

步骤 1 下载数据

Iris 数据集是模式识别最著名的数据集之一。数据集包含 3 类，每类 50 个实例，其中每个类都涉及一种鸢尾植物。 第一类与后两类可线性分离，后两类之间不能线性分离，所以本实验取前两类数据，做一个 2 分类数据集。

Iris 数据集的官网：[Iris Data Set](http://archive.ics.uci.edu/ml/datasets/Iris)。

方式一，从 Iris 数据集官网下载[iris.data 文件]

(http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data)。

方式二，从华为云 OBS 中下载[iris.data 文件]

(https://share-course.obs.cn-north-4.myhuaweicloud.com/dataset/iris.data)

每个样本含有 4 个数值属性和一个类别属性：

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
    - Iris Setosa
    - Iris Versicolour
    - Iris Virginica

概括统计：

	Min	Max	Mean	SD	Class Correlation	
sepal length:	4.3	7.9	5.84	0.83	0.7826	
sepal width:	2.0	4.4	3.05	0.43	-0.4194	
petal length:	1.0	6.9	3.76	1.76	0.9490	(high!)
petal width:	0.1	2.5	1.20	0.76	0.9565	(high!)

步骤 2 上传数据到实验环境

在新建的 notebook 实验环境中，通过如图所示的“上传”按钮，然后选择自己本地已下载好的数据文件“iris.data”，将数据文件上传到实验环境中。

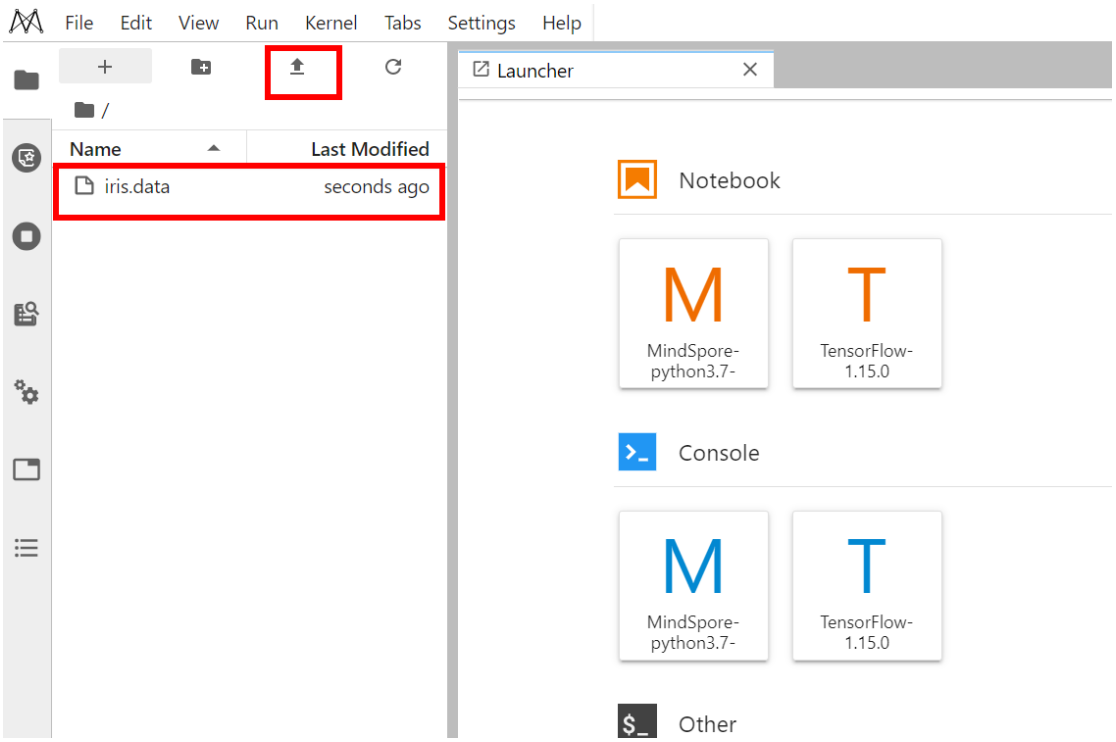


图 2-1 上传数据集到实验环境

1.2.4.2 数据读取与处理

步骤 1 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

context.set_context(mode=context.GRAPH_MODE, device_target="Ascend")
```

步骤 2 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 3 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)
```

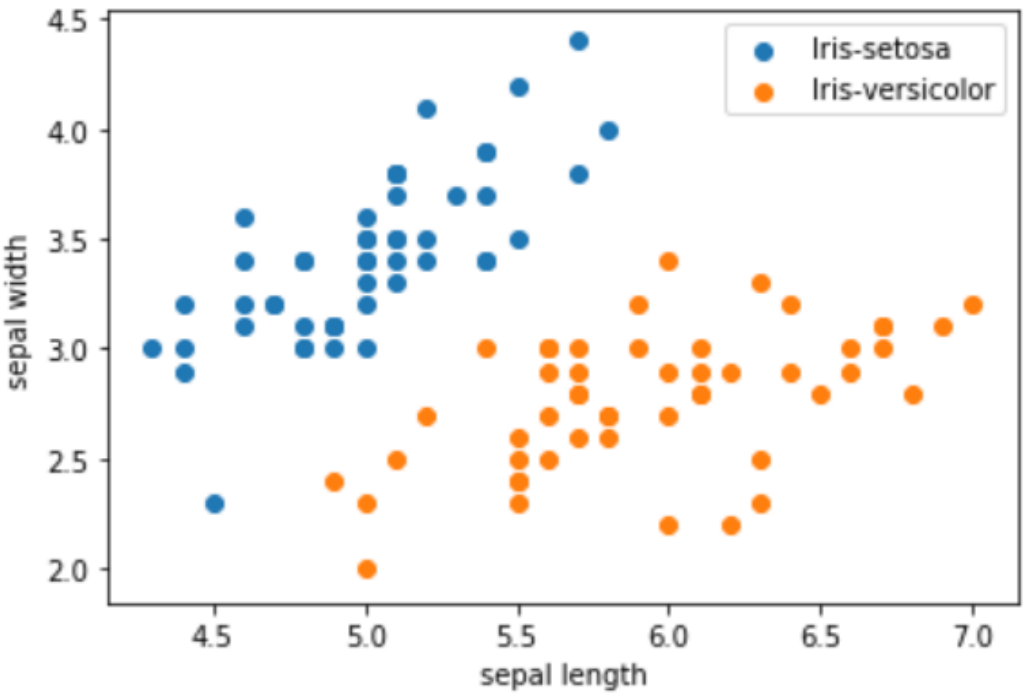
步骤 4 样本可视化

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
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：

```
<matplotlib.legend.Legend at 0xffff5d6599d0>
```



步骤 5 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 6 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))
XY_train_qie = XY_train[:80]
ds_train = dataset.GeneratorDataset(XY_train_qie, ['x', 'y'])
ds_train = ds_train.shuffle(buffer_size=80).batch(32, drop_remainder=True)
```

1.2.4 3 模型建立与训练

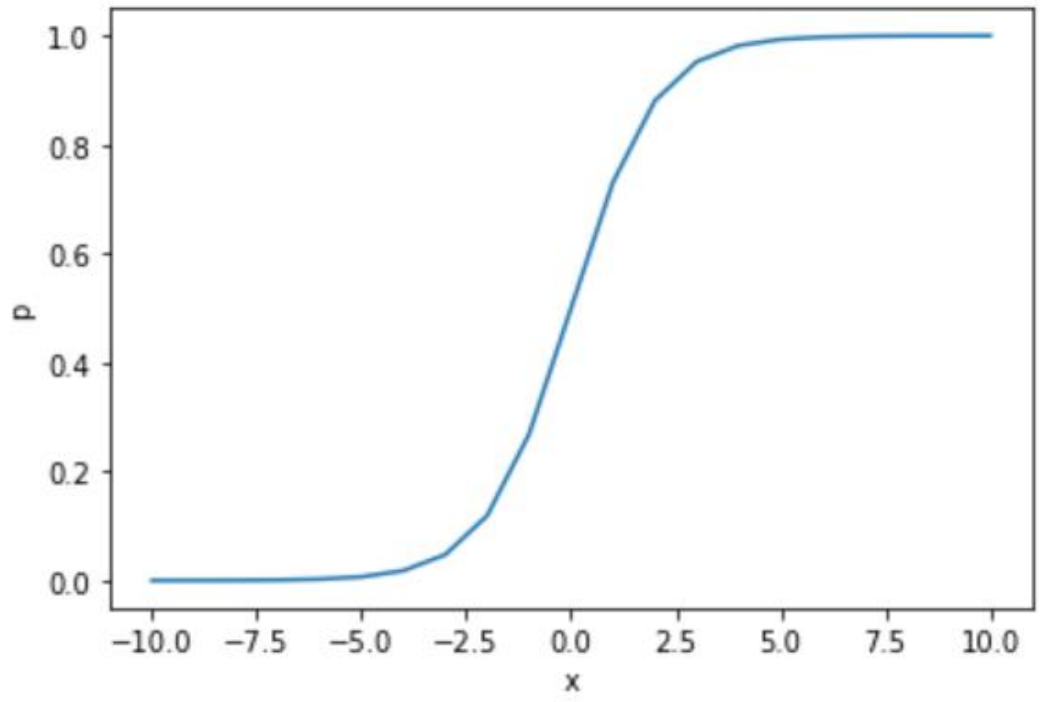
步骤 1 可视化逻辑回归函数

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：

```
Text(0, 0.5, 'p')
```



步骤 2 建模

使用 MindSpore 提供的 nn.Dense(4, 1) 算子（https://www.mindspore.cn/api/zh-CN/0.2.0-alpha/api/python/mindspore/mindspore.nn.html#mindspore.nn.Dense)作为线性部分，其中(4, 1)表示每个样本的输入是含 4 个元素的向量，输出是含 1 个元素的向量，即 W 是 1x4 的矩阵。算子会随机初始化权重 W 和偏置 b。使用 SigmoidCrossEntropyWithLogits 算子作为非线性部分：

对于每个样本 N_i，模型的计算方式如下：

$$z_i = wx_i + b$$

$$p_i = sigmoid(z_i) = \frac{1}{1 + e^{-z_i}}$$

$$loss = \frac{1}{n}\sum_{i=1}^{n}(y_i * \ln(p_i) + (1 - y_i) * \ln(1 - p_i))$$

其中，x_i是 1D Tensor（含 4 个元素），z_i是 1D Tensor（含 1 个元素），y_i是真实类别（2 个类别{0, 1}中的一个），p_i是 1D Tensor（含 1 个元素，表示属于类别 1 的概率，值域为[0, 1]），loss 是标量。

```
# 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 3 模型训练

使用 2 分类的 Iris 数据集对模型进行几代（Epoch）训练：

代码：

```
model = ms.train.Model(net, loss, opt)
model.train(5, ds_train, callbacks=[LossMonitor(per_print_times=ds_train.get_dataset_size())],
dataset_sink_mode=False)
```

输出：

```
    epoch: 1 step 2, loss is 0.6358570456504822
    Epoch time: 9946.221, per step time: 4973.111, avg loss: 0.666
    ************************************************************
    epoch: 2 step 2, loss is 0.5617856979370117
    Epoch time: 132.066, per step time: 66.033, avg loss: 0.595
    ************************************************************
    epoch: 3 step 2, loss is 0.5153790712356567
```

```
Epoch time: 4.302, per step time: 2.151, avg loss: 0.540
************************************************************
epoch: 4 step 2, loss is 0.5422952771186829
Epoch time: 4.457, per step time: 2.229, avg loss: 0.512
************************************************************
epoch: 5 step 2, loss is 0.42156651616096497
Epoch time: 4.481, per step time: 2.241, avg loss: 0.439
************************************************************
```

1.2.4.4 模型评估

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

代码：

```
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
```

1.2.5 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。

1.3. 鸢尾花多分类实验

步骤 1 导入 MindSpore 模块和辅助模块

```
import os
# os.environ['DEVICE_ID'] = '7'
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

context.set_context(mode=context.GRAPH_MODE, device_target="Ascend")
```

步骤 2 读取 Iris 数据集`iris.data`，并作检查

```
with open('iris.data') as csv_file:
    data = list(csv.reader(csv_file, delimiter=','))
print(data[0:5]); print(data[50:55]); print(data[100:105]) #  打印部分数据
```

步骤 3 数据集的 3 类样本共 150 条，将样本的 4 个属性作为自变量X，将样本的 3 个类别映射为{0, 1, 2}，作为因变量Y。

```
label_map = {
    'Iris-setosa': 0,
    'Iris-versicolor': 1,
    'Iris-virginica': 2
}
X = np.array([[float(x) for x in s[:-1]] for s in data[:150]], np.float32)
Y = np.array([label_map[s[-1]] for s in data[:150]], np.int32)
```

步骤 4 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其中一类和余下两类是线性可分的，而余下两类之间线性不可分。

```
from matplotlib import pyplot as plt
plt.scatter(X[:50, 0], X[:50, 1], label='Iris-setosa')
plt.scatter(X[50:100, 0], X[50:100, 1], label='Iris-versicolor')
plt.scatter(X[100:, 0], X[100:, 1], label='Iris-virginica')
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.legend()
```

步骤 5 将数据集按 8:2 划分为训练集和验证集

```
train_idx = np.random.choice(150, 120, replace=False)
test_idx = np.array(list(set(range(150)) - 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 6 使用 MindSpore`GeneratorDataset`接口将 numpy.ndarray 类型的数据转换为 Dataset

```
XY_train = list(zip(X_train, Y_train))
XY_train_qie = XY_train[:120]
```

```
ds_train = dataset.GeneratorDataset(XY_train_qie, ['x', 'y'])
ds_train = ds_train.shuffle(buffer_size=120).batch(32, drop_remainder=True)

XY_test = list(zip(X_test, Y_test))
XY_test_qie = XY_test[:30]
ds_test = dataset.GeneratorDataset(XY_test_qie, ['x', 'y'])
ds_test = ds_test.batch(30)
```

步骤 7 模型训练与评估

```
net = nn.Dense(4, 3)
loss = nn.loss.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
opt = nn.optim.Momentum(net.trainable_params(), learning_rate=0.05, momentum=0.9)

model = ms.train.Model(net, loss, opt, metrics={'acc', 'loss'})
model.train(25, ds_train, callbacks=[LossMonitor(per_print_times=ds_train.get_dataset_size())],
dataset_sink_mode=False)
metrics = model.eval(ds_test)
print(metrics)
```

1.4.实验拓展

使用 Scikit-Learn 实现以上实验任务。

阅读 Scikit-learn 手册中相关算法的使用方法

最小二乘线性回归的用法（https://scikit-learn.org/0.20/modules/linear_model.html#ordinary-least-squares）。

LogisticRegression 的用法（https://scikit-learn.org/0.20/modules/linear_model.html#logistic-regression）。

参考代码：

```
from sklearn import linear_model
reg = linear_model.LinearRegression()
reg.fit(X,y)
#.....
```

二． 信用违约预测

2.1 实验说明

风险管控已经成为了今年金融市场的重要主题之一，银行作为贷方，随时都面临着借贷者违约的风险。传统的专家规则在金融科技时代逐渐过时，机器学习和金融业务的交叉也延伸到信贷领域。违约预测就是其中一重要应用。本实验基于信贷业务场景中一个月内的抽样数据，数据集有 34 个维度，Target 表示客户在接下来一个月是否有违约。模型生成后可使用当前月的数据预测接下来一个月客户是否会违约。

本地离线数据地址和名称：dataset-credit-default.csv

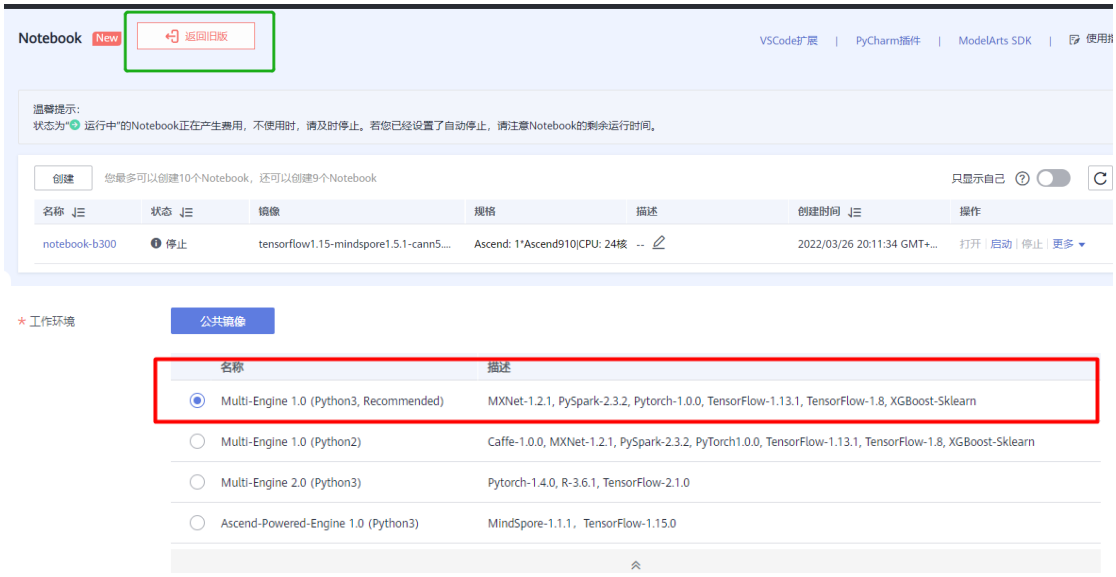
2.2 实验建模流程要求

违约预测只有违约和没有预约两种结果，属于二分类问题。针对二分类问题，可使用的算法有逻辑回归、朴素贝叶斯、支持向量机、树模型等。考虑到实验的完整性和实用性，本实验选用业界常用的决策树和随机森林来做对比。考虑到样本极度不均衡，模型评价选用综合指标 f1_score。

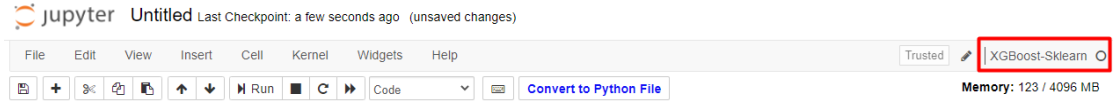
2.2.1 环境要求

ModelArts 云平台：

选择工作环境，若新版找不到 XGBoost-Sklearn 实验环境，可返回旧版华为云寻找创建：



选择 kernel 为 XGboost-sklearn：



2.2.2 实验实现步骤要求

步骤 1：要求导入相关数据读取、处理、分析、可视化，算法模块等

步骤 2：数据读取，数据框类型，命名为 df

步骤 3：查看 Target 的分布 ,是否违约（1 是，0 否）

步骤 4：可视化观察特征相关性

步骤 5：存储相关性过高的特征对,对于相关性过高的的特征，删除其中一个（根据工程经验，以 0.8 为界）

```
# 选择出符合内容的单元格对应的行、列标签
cols_pair_to_drop = []
for index_ in corr_matrix.index:
    for col_ in corr_matrix.columns:
        if corr_matrix.loc[index_,col_] >= 0.8 and index_!=col_ and (col_,index_) not in
cols_pair_to_drop:
            cols_pair_to_drop.append((index_,col_))
#丢弃特征对中的一个
cols_to_drop = np.unique([col[1] for col in cols_pair_to_drop])  #对于一维数组或者列
表，unique 函数去除其中重复的元素，并按元素由大到小返回一个新的无元素重复的元组或
者列表
df.drop(cols_to_drop,axis=1,inplace=True)
df.head()
```

步骤 6：打印出缺失率最高的前 15 个特征以及对应的缺失率

步骤 7：可视化：针对 Couple_Year_Income 和 Couple_L12_Month_Pay_Amount 制作箱型图来判定下如何填充。

步骤 8：选择 IOR 方法筛选 Couple_Year_Income 异常值

```
item = 'Couple_Year_Income'
iqr = df[item].quantile(0.75) - df[item].quantile(0.25)
q_abnormal_L = df[item] < df[item].quantile(0.25) - 1.5 * iqr
q_abnormal_U = df[item] > df[item].quantile(0.75) + 1.5 * iqr
#取异常点的索引
print(item + '中有' + str(q_abnormal_L.sum() + q_abnormal_U.sum()) + '个异常值')
item_outlier_index = df[q_abnormal_L|q_abnormal_U].index
```

步骤 9：根据筛选出的异常值索引，删除 Couple_Year_Income 的异常值并用中位数填补缺失值。

步骤 10：选择 IOR 方法筛选 Couple_L12_Month_Pay_Amount 异常值

步骤 11：根据筛选出的异常值索引，删除 Couple_L12_Month_Pay_Amount 的异常值并用中位数填补缺失值

步骤 12：查看 df 中仍有少量缺失值的特征是哪些，提取出列名，封装到列表结构中，列表命名为 null_col

步骤 13：使用众数填充 null_col 中每列缺失值，直接改变 df。

步骤 14：从 df 中删除无分类意义的特征列 Cust_No。

步骤 15：使用 factorize 函数，对 df 剩余全部名称型特征进行标签编码，部分提示如下：

```
### 查看数据集剩余的名称性特征
con_col=[]
for col in df.columns:
    if df.dtypes[col] == np.object:
        con_col.append(col)
con_col
```

步骤 16：将数据集作 9:1 的切分成训练集和测试集，并查看两个集中不同两类别的数量。

步骤 17：引入 StandardScaler 标准化工具库，对训练集和测试集做标准化

步骤 18：使用决策树对标准化后的数据进行建模、预测与评价

步骤 19：使用网格搜索法对决策树进行调参

步骤 20：使用 RandomForest 对标准化后的数据（***注意此处不是过采样后的数据***）进行建模，设置样本权重参数 class_weight='balanced'，然后对测试集进行预测，并选择多个评估指标 accuracy_score、precision_score 、recall_score 、f1_score 查看预测得分。

步骤 21：参数调优，选择坐标下降方式对 class_weight 进行搜索，部分提示如下:

```
param_test0 = {'class_weight':[{0:1,1:3},{0:1,1:5},{0:1,1:10},{0:1,1:20},'balanced']}
```

步骤 22：参数调优，选择坐标下降方式对 n_estimators 进行搜索，部分提示如下:

```
param_test1 = {'n_estimators':range(10,101,10)}
```

步骤 23：根据两次参数调整后的随机森林模型重新对测试集进行预测，选择多个评估指标 accuracy_score、precision_score 、recall_score 、f1_score，对预测结果进行评估，并打印出在各个指标上得分。

2.3 实验要求

请设计实验分析使用特征数量、是否标准化、决策树结构参数等因素对分类结果的影响。

2.4 参考答案

参考：信用违约预测.ipynb

三． 神经网络

3.1. 前馈神经网络实验

3.1.1 实验介绍

3.1.1.1 简介

Mnist 手写体图像识别实验是深度学习入门经典实验。Mnist 数据集包含 60,000 个用于训练的示例和 10,000 个用于测试的示例。这些数字已经过尺寸标准化并位于图像中心，图像是固定大小(28x28 像素)，其值为 0 到 255 为简单起见，每个图像都被平展并转换为 784(28*28)个特征的一维 numpy 数组。参考 mindspore/course 仓库：

https://gitee.com/mindspore/course/tree/master/lenet5

3.1.1.2 实验目的

- 学会如何搭建全连接神经网络。
- 掌握搭建网络过程中的关键点。
- 掌握分类任务的整体流程。

3.1.2 实验环境要求

推荐在华为云 ModelArts 实验平台完成实验，也可在本地搭建 python3.7.5 和 MindSpore1.3.0 环境完成实验。

3.1.3 实验总体设计

创建实验环境：在 ModelArts 平台创建 Ascend+MindSpore 环境。

导入实验所需模块：该步骤通常都是程序编辑的第一步，将实验代码所需要用到的模块包用 import 命令进行导入。

导入数据集并预处理：神经网络的训练离不开数据，这里对数据进行导入。同时，因为全连接网络只能接收固定维度的输入数据，所以，要对数据集进行预处理，以符合网络的输入维度要求。同时，设定好每一次训练的 Batch 的大小，以 Batch Size 为单位进行输入。

模型搭建：利用 mindspore.nn 的 cell 模块搭建全连接网络，包含输入层，隐藏层，输出层。同时，配置好网络需要的优化器，损失函数和评价指标。传入数据，并开始训练模型。

模型评估：利用测试集进行模型的评估。

3.1.4 实验过程

3.1.4.1 创建实验环境

在开始本实验前，需要完成实验环境搭建工作。

步骤 1：创建 Ascend 开发环境

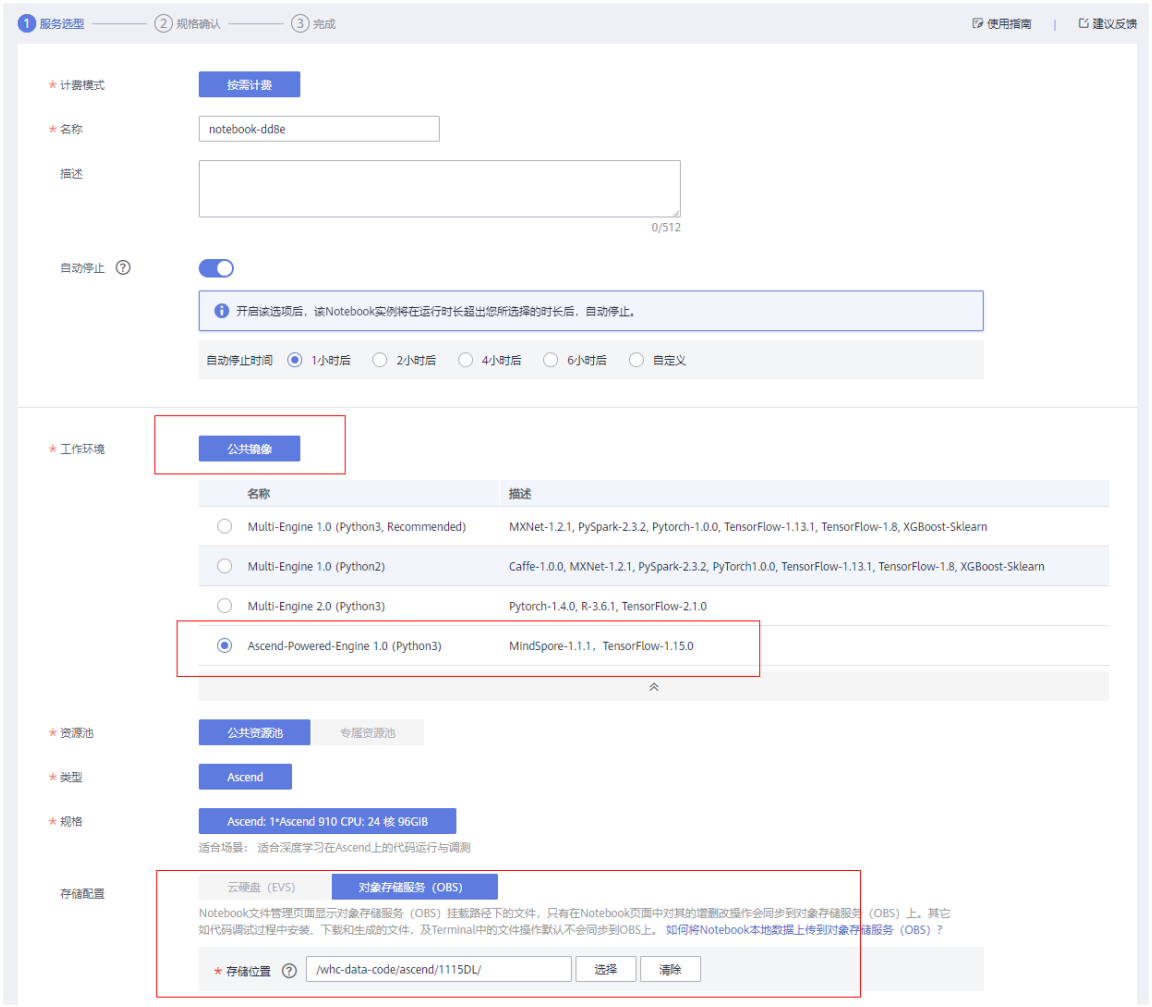


图 1-1 创建 notebook

创建的 notebook 开发环境名称需要自己修改，资源池类型选择 Ascend，现阶段必须使用 OBS 服务存储实验代码和数据集，所以需要制定 OBS 路径。

步骤 2：上传数据集

实验中需要用到 MNIST 手写体字符的数据集。在新建的 notebook 实验环境中，通过如图所示的“上传”按钮，然后选择自己本地已下载好的数据文件“MINST”，将数据文件上传到实验环境中。

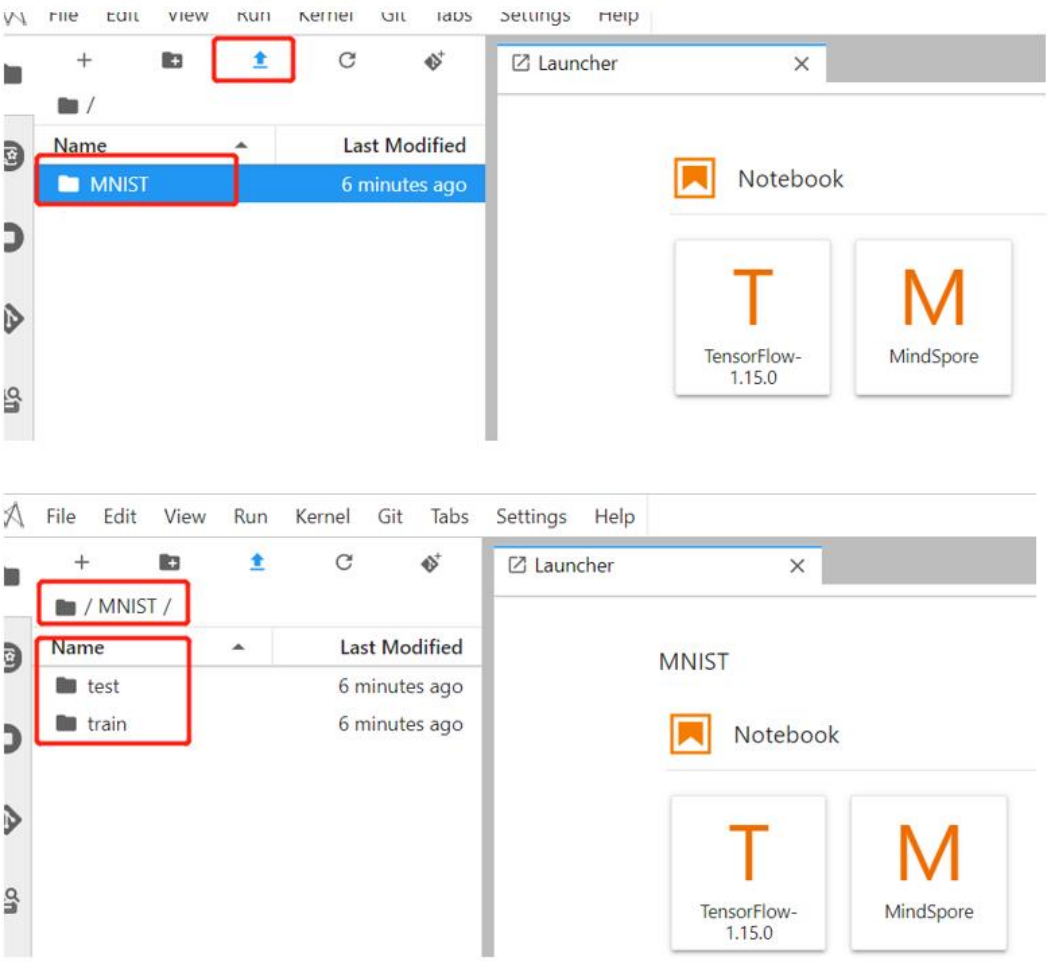


图 1-2 数据上传

在 MNIST 文件夹下建立 train 和 test 两个文件夹，train 中存放 train-labels-idx1-ubyte 和 train-images-idx3-ubyte 文件，test 中存放 t10k-labels-idx1-ubyte 和 t10k-images-idx3-ubyte 文件。

步骤 3：打开 notebook

打开 Notebook 控制台后，新建或打开 ipynb 文件，选择 MindSpore 环境作为 Kernel，即可开始编辑实验代码。

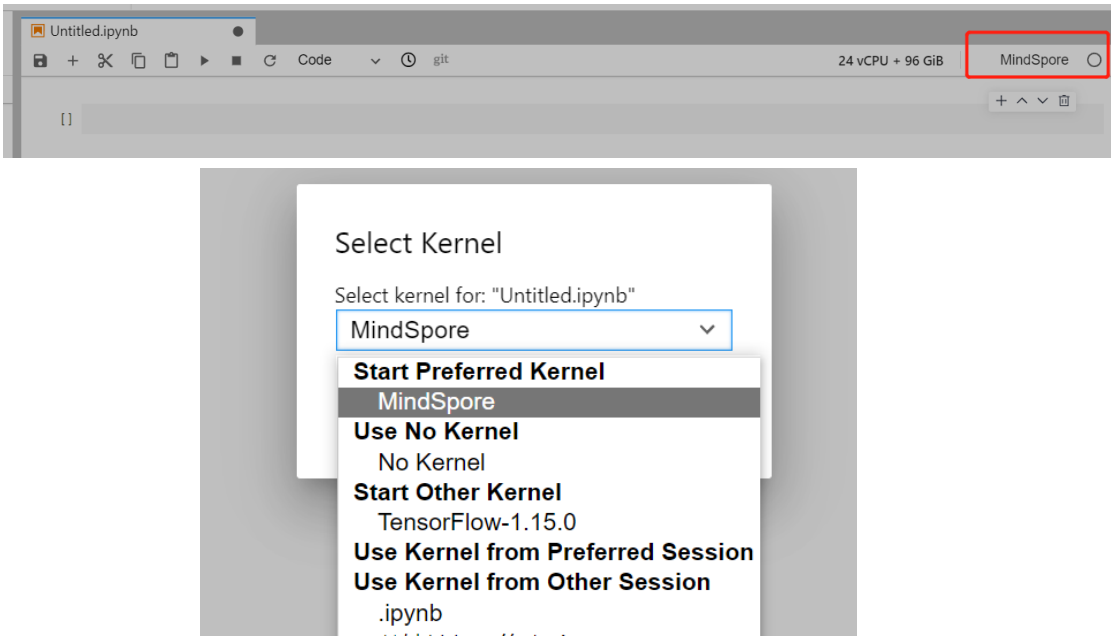


图 1-3 创建 ipynb 文件

3.1.4.2 导入实验所需模块

Os 模块主要用于对系统路径和文件进行处理。Numpy 模块主要用于数据的基本运算操作。Matplotlib 模块主要用于画图。MindSpore 相关模块主要用于搭建网络、调用优化器、读取数据集和将数据集处理成网络的标准输入格式。

```
#导入相关依赖库
import    os
import numpy as np
from matplotlib import pyplot as plt

import mindspore as ms
#context 模块用于设置实验环境和实验设备
import mindspore.context as context
#dataset 模块用于处理数据形成数据集
import mindspore.dataset as ds
#c_transforms 模块用于转换数据类型
import mindspore.dataset.transforms.c_transforms as C
#vision.c_transforms 模块用于转换图像，这是一个基于 opencv 的高级 API
import mindspore.dataset.vision.c_transforms as CV
#导入 Accuracy 作为评价指标
from mindspore.nn.metrics import Accuracy
#nn 中有各种神经网络层如：Dense，ReLu
from mindspore import nn
#Model 用于创建模型对象，完成网络搭建和编译，并用于训练和评估
from mindspore.train import Model
#LossMonitor 可以在训练过程中返回 LOSS 值作为监控指标
```

```
from mindspore.train.callback import    LossMonitor
#设定运行模式为动态图模式，并且运行设备为昇腾芯片,CPU 运行时需要将 Ascend 改为 CPU
context.set_context(mode=context.GRAPH_MODE, device_target='Ascend')
```

3.1.4.3 导入实验数据集

MNIST 是一个手写数字数据集，训练集包含 60000 张手写数字，测试集包含 10000 张手写数字，共 10 类。可在 MNIST 数据集的官网下载数据集，解压到当前代码目录下。MindSpore 的 dataset 模块有专门用于读取和解析 Mnist 数据集的源数据集，可直接读取并生成训练集和测试集。

步骤 1：加载并查看数据集

```
#MindSpore 内置方法读取 MNIST 数据集
ds_train = ds.MnistDataset(os.path.join(r'MNIST', "train"))
ds_test = ds.MnistDataset(os.path.join(r'MNIST', "test"))

print('训练数据集数量：',ds_train.get_dataset_size())
print('测试数据集数量：',ds_test.get_dataset_size())
#该数据集可以通过 create_dict_iterator()转换为迭代器形式，然后通过__next__()一个个输出样本
image=ds_train.create_dict_iterator().__next__()
print(type(image))
print('图像长/宽/通道数：',image['image'].shape)
#一共 10 类，用 0-9 的数字表达类别。
print('一张图像的标签样式：',image['label'])
```

输出结果：

```
训练数据集数量：  60000
测试数据集数量：  10000
图像长/宽/通道数：  (28, 28, 1)
一张图像的标签样式：  7
```

步骤 2：生成测试集和训练集

创建数据集，为训练集设定 Batch Size，这是因为我们通常会采用小批量梯度下降法（MBGD）来训练网络，所以 batch size 作为一个非常重要的超参数需要提前设定好。在本代码中，batch size 为 128，意味着每一次更新参数，我们都用 128 个样本的平均损失值来进行更新。

```
DATA_DIR_TRAIN = "MNIST/train" #  训练集信息
DATA_DIR_TEST = "MNIST/test" #  测试集信息

def create_dataset(training=True, batch_size=128, resize=(28, 28),rescale=1/255, shift=-0.5, buffer_size=64):
    ds = ms.dataset.MnistDataset(DATA_DIR_TRAIN if training else DATA_DIR_TEST)

    #定义改变形状、归一化和更改图片维度的操作。
```

```
    #改为（28,28）的形状
    resize_op = CV.Resize(resize)
    #rescale 方法可以对数据集进行归一化和标准化操作，这里就是将像素值归一到 0 和 1 之间，shift 参数可以让值域偏移至-0.5 和 0.5 之间
    rescale_op = CV.Rescale(rescale, shift)
    #由高度、宽度、深度改为深度、高度、宽度
    hwc2chw_op = CV.HWC2CHW()

    #  利用 map 操作对原数据集进行调整
    ds = ds.map(input_columns="image", operations=[resize_op, rescale_op, hwc2chw_op])
    ds = ds.map(input_columns="label", operations=C.TypeCast(ms.int32))
    #设定洗牌缓冲区的大小，从一定程度上控制打乱操作的混乱程度
    ds = ds.shuffle(buffer_size=buffer_size)
    #设定数据集的 batch_size 大小，并丢弃剩余的样本
    ds = ds.batch(batch_size, drop_remainder=True)

    return ds
```

步骤 3：查看数据

```
#显示前 10 张图片以及对应标签,检查图片是否是正确的数据集
dataset_show = create_dataset(training=False)
data = dataset_show.create_dict_iterator().__next__()
images = data['image'].asnumpy()
labels = data['label'].asnumpy()

for i in range(1,11):
    plt.subplot(2, 5, i)
    #利用 squeeze 方法去掉多余的一个维度
    plt.imshow(np.squeeze(images[i]))
    plt.title('Number: %s' % labels[i])
    plt.xticks([])
plt.show()
```

输出结果：

3.1.4.4 模型搭建与训练

手写数字图像数据集准备完成，接下来我们就需要构建训练模型，本实验采用的是全连接神经网络算法，所以我们首先需要建立初始化的神经网络。

步骤 1：创建网络

手写数字图像数据集准备完成，接下来我们就需要构建训练模型，本实验采用的是全连接神经网络算法，所以我们首先需要建立初始化的神经网络。nn.cell 能够用来组成网络模型;模型包括 5 个卷积层和 RELU 激活函数，一个全连接输出层并使用 softmax 进行多分类，共分成（0-9）10 类

```
#利用定义类的方式生成网络，Mindspore 中定义网络需要继承 nn.cell。在 init 方法中定义该网络需要的神经网络层
#在 construct 方法中梳理神经网络层与层之间的关系。
class ForwardNN(nn.Cell):
    def __init__(self):
        super(ForwardNN, self).__init__()
        self.flatten = nn.Flatten()
        self.relu = nn.ReLU()
        self.fc1 = nn.Dense(784, 512, activation='relu')
        self.fc2 = nn.Dense(512, 256, activation='relu')
        self.fc3 = nn.Dense(256, 128, activation='relu')
        self.fc4 = nn.Dense(128, 64, activation='relu')
        self.fc5 = nn.Dense(64, 32, activation='relu')
        self.fc6 = nn.Dense(32, 10, activation='softmax')

    def construct(self, input_x):
        output = self.flatten(input_x)
        output = self.fc1(output)
        output = self.fc2(output)
        output = self.fc3(output)
        output = self.fc4(output)
        output = self.fc5(output)
        output = self.fc6(output)
        return output
```

步骤 2：设定参数

指定模型所需的损失函数、评估指标、优化器等参数。

```
lr = 0.001
num_epoch = 10
momentum = 0.9

net = ForwardNN()
#定义 loss 函数，改函数不需要求导，可以给离散的标签值，且 loss 值为均值
loss = nn.loss.SoftmaxCrossEntropyWithLogits( sparse=True, reduction='mean')
#定义准确率为评价指标，用于评价模型
metrics={"Accuracy": Accuracy()}
```

```
#定义优化器为 Adam 优化器，并设定学习率
opt = nn.Adam(net.trainable_params(), lr)
```

步骤 3：训练模型并保存

将创建好的网络、损失函数、评估指标、优化器等参数装入模型中对模型进行训练。

```
#生成验证集，验证机不需要训练，所以不需要 repeat
ds_eval = create_dataset(False, batch_size=32)
#模型编译过程，将定义好的网络、loss 函数、评价指标、优化器编译
model = Model(net, loss, opt, metrics)

#生成训练集
ds_train = create_dataset(True, batch_size=32)
print("============== Starting Training ==============")
#训练模型，用 loss 作为监控指标，并利用昇腾芯片的数据下沉特性进行训练
model.train(num_epoch, ds_train,callbacks=[LossMonitor()],dataset_sink_mode=True)
```

输出结果：

```
============== Starting Training ==============
epoch: 1 step: 1875, loss is 1.7190123
epoch: 2 step: 1875, loss is 1.5236794
epoch: 3 step: 1875, loss is 1.5861311
epoch: 4 step: 1875, loss is 1.6486512
epoch: 5 step: 1875, loss is 1.5861502
epoch: 6 step: 1875, loss is 1.5549002
epoch: 7 step: 1875, loss is 1.6174002
epoch: 8 step: 1875, loss is 1.6176393
epoch: 9 step: 1875, loss is 1.9296615
epoch: 10 step: 1875, loss is 1.5236502
```

3.1.4.5 模型评估

利用模型对测试集的数据进行预测，并与标签对比，用准确率 accuarcy 进行评估。

```
#使用测试集评估模型，打印总体准确率
metrics_result=model.eval(ds_eval)
print(metrics_result)
```

输出结果：

```
{'Accuracy': 0.8710666666666667}
```

3.1.5 实验要求

请设计实验分析不同网络结构与训练的超参数对分类结果的影响。

3.2. 卷积神经网络实验

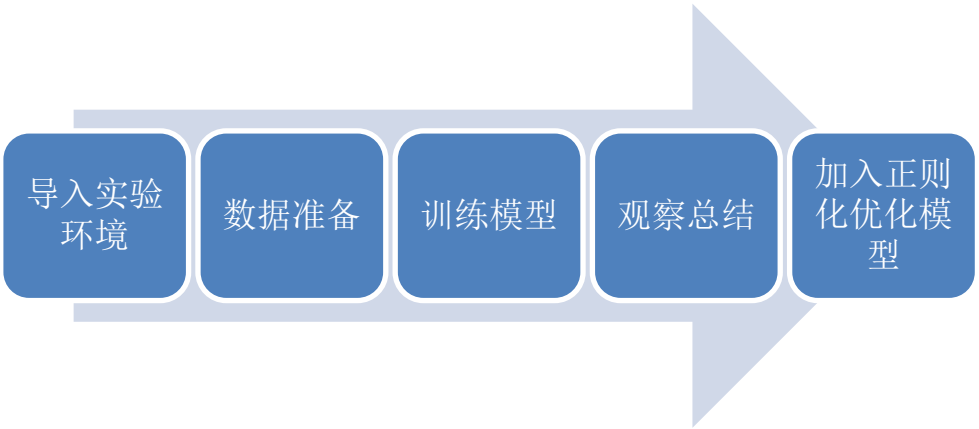
3.2.1 实验介绍

本实验使用 Fashion-MNIST 数据集，它是一个替代 MNIST 手写数字集的图像数据集，由 Zalando（一家德国的时尚科技公司）旗下的研究部门提供。其涵盖了来自 10 种类别的共 7 万个不同商品的正面图片。Fashion-MNIST 的大小、格式和训练集/测试集划分与原始的 MNIST 完全一致。60000/10000 的训练测试数据划分，28x28 的灰度图片。通过上述实验我们对比不同正则化技术效果，此实验我们应用正则化技术做案例分析，并对比有无正则化的训练结果。

3.2.2 实验环境要求

- ModelArts 平台：Mindspore-1.3.0-python3.7-aarch64

3.2.3 实验总体设计



3.2.4 实验过程

本节将详细介绍实验的设计与实现。3.4.1 导入实验环境；3.4.2 数据准备；3.4.3 训练模型；3.4.4 观察总结。

3.2.4.1 导入实验环境

步骤 1：导入库

```
import os
import struct
import sys
from easydict import EasyDict as edict

import matplotlib.pyplot as plt
import numpy as np

import mindspore
import mindspore.dataset as ds
import mindspore.nn as nn
from mindspore import context
from mindspore.nn.metrics import Accuracy
from mindspore.train import Model
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor, TimeMonitor
from mindspore import Tensor

context.set_context(mode=context.GRAPH_MODE, device_target='Ascend')
```

步骤 2：定义常量

步骤 3：导入数据集

https://professional-construction.obs.cn-north-4.myhuaweicloud.com:443/deep-learning/Fashion-MNIST.zip

```
cfg = edict({
    'train_size': 60000,  # 训练集大小

    'test_size': 10000,  # 测试集大小

    'channel': 1,  # 图片通道数

    'image_height': 28,  # 图片高度
```

```
    'image_width': 28,  # 图片宽度
    'batch_size': 64,
    'num_classes': 10,  # 分类类别
    'lr': 0.001,  # 学习率
    'epoch_size': 20,  # 训练次数
    'data_dir_train': os.path.join('fashion-mnist', 'train'),
    'data_dir_test': os.path.join('fashion-mnist', 'test'),
})
```

步骤 4：定义函数用于读取数据

```
def read_image(file_name):
    '''
    :param file_name: 文件路径
    :return:  训练或者测试数据
    如下是训练的图片的二进制格式
    [offset] [type]          [value]          [description]
    0000     32 bit integer  0x00000803(2051) magic number
    0004     32 bit integer  60000            number of images
    0008     32 bit integer  28               number of rows
    0012     32 bit integer  28               number of columns
    0016     unsigned byte   ??               pixel
    0017     unsigned byte   ??               pixel
    ........
    xxxx     unsigned byte   ??               pixel
    '''
    file_handle = open(file_name, "rb")  # 以二进制打开文档
    file_content = file_handle.read()  # 读取到缓冲区中
    head = struct.unpack_from('>IIII', file_content, 0)  # 取前 4 个整数，返回一
个元组
    offset = struct.calcsize('>IIII')
```

```
    imgNum = head[1]    # 图片数

    width = head[2]    # 宽度

    height = head[3]    # 高度

    bits = imgNum * width * height    # data 一共有 60000*28*28 个像素值

    bitsString = '>' + str(bits) + 'B'    # fmt 格式：'>47040000B'

    imgs = struct.unpack_from(bitsString, file_content, offset)    # 取 data 数据，
返回一个元组

    imgs_array = np.array(imgs, np.float32).reshape((imgNum, width * height))
# 最后将读取的数据 reshape 成 【图片数，图片像素】二维数组

    return imgs_array


def read_label(file_name):
    '''
    :param file_name:
    :return:
    标签的格式如下：
    [offset] [type]          [value]          [description]
    0000     32 bit integer  0x00000801(2049) magic number (MSB first)
    0004     32 bit integer  60000            number of items
    0008     unsigned byte   ??               label
    0009     unsigned byte   ??               label
    ........
    xxxx     unsigned byte   ??               label
    The labels values are 0 to 9.
    '''

    file_handle = open(file_name, "rb")    # 以二进制打开文档

    file_content = file_handle.read()    # 读取到缓冲区中

    head = struct.unpack_from('>II', file_content, 0)    # 取前 2 个整数，返回一个
```

```
元组
    offset = struct.calcsize('>II')
    labelNum = head[1]  # label 数
    bitsString = '>' + str(labelNum) + 'B'  # fmt 格式：'>47040000B'
    label = struct.unpack_from(bitsString, file_content, offset)  # 取 data 数据，
返回一个元组
    return np.array(label, np.int32)


def get_data():
    # 文件获取
    train_image = os.path.join(cfg.data_dir_train, 'train-images-idx3-ubyte')
    test_image = os.path.join(cfg.data_dir_test, "t10k-images-idx3-ubyte")
    train_label = os.path.join(cfg.data_dir_train, "train-labels-idx1-ubyte")
    test_label = os.path.join(cfg.data_dir_test, "t10k-labels-idx1-ubyte")
    # 读取数据
    train_x = read_image(train_image)
    test_x = read_image(test_image)
    train_y = read_label(train_label)
    test_y = read_label(test_label)
    return train_x, train_y, test_x, test_y
```

3.2.4.2 数据准备

步骤 1：数据集准备

本实验需要用到的是 Fashion-MNIST 数据集，在新建的 notebook 实验环境中，通过如图所示的“上传”按钮，然后选择自己本地已下载好的数据文件“Fashion-MNIST”，将数据文件上传到实验环境中。

```
!wget
https://professional-construction.obs.cn-north-4.myhuaweicloud.com:443/deep-lea
rning/fashion-mnist.zip

#解压文件

!unzip fashion-mnist.zip
```

Fashion Mnist 有 10 个标签，如下表所示：

标签	描述
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

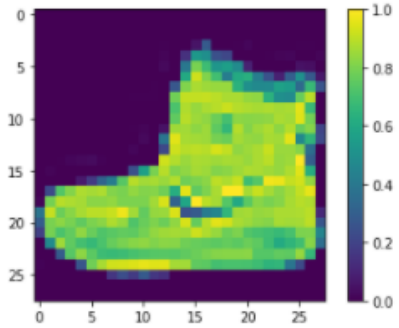
步骤 2：数据预处理

图像最后一个维度，即通道（颜色），使用 reshape()函数将其添加到 train_images 和 test_images 的维度中。在这种情况下，它是单一颜色，因此通道为 1，即"灰度"。为了减少计算量，还需要把图片的像素值进行归一化，八位图像的像素值范围在 0-255 之间，将所有像素值除以 255，使得像素值范围控制在 0-1 之间。并打印数据集形状和一张图片作为例子。

```
train_x, train_y, test_x, test_y = get_data()
train_x = train_x.reshape(-1, 1, cfg.image_height, cfg.image_width)
test_x = test_x.reshape(-1, 1, cfg.image_height, cfg.image_width)
train_x = train_x / 255.0
test_x = test_x / 255.0


print('训练数据集样本数：', train_x.shape[0])

print('测试数据集样本数：', test_y.shape[0])

print('通道数/图像长/宽：', train_x.shape[1:])

print('一张图像的标签样式：', train_y[0])  # 一共 10 类，用 0-9 的数字表达类别。


plt.figure()
plt.imshow(train_x[0,0,...])
plt.colorbar()
plt.grid(False)
plt.show()
```

输出结果：

```
训练数据集样本数： 60000
测试数据集样本数： 10000
通道数/图像长/宽： (1, 28, 28)
一张图像的标签样式： 9
```



步骤 3：数据集预处理

在训练之前，需要先对数据集中的数据进行“洗牌”，打乱数据集的顺序。

```
# 转换数据类型为 Dataset

def create_dataset():
    XY_train = list(zip(train_x, train_y))
    ds_train = ds.GeneratorDataset(XY_train, ['x', 'y'])
    ds_train = ds_train.shuffle(buffer_size=1000).batch(cfg.batch_size,
drop_remainder=True)
    XY_test = list(zip(test_x, test_y))
    ds_test = ds.GeneratorDataset(XY_test, ['x', 'y'])
    ds_test = ds_test.shuffle(buffer_size=1000).batch(cfg.batch_size,
drop_remainder=True)
    return ds_train, ds_test
```

3.2.4.3 训练模型

步骤 1：创建卷积神经网络

该实验中，可以选择使用不含正则化的卷积神经网络或者选择加入正则化的卷积神经网络，用于对比结果。

下面这部分用于创建不加入正则化的卷积神经网络，网络结构为：卷积层 1→卷积层 2→卷积层 3→最大池化层→全连接层 1→全连接层 2。

```
# 定义卷积神经网络，无正则化

class ForwardFashion(nn.Cell):

    def __init__(self, num_class=10):  # 一共分十类，图片通道数是 1

        super(ForwardFashion, self).__init__()
```

```
        self.num_class = num_class
        self.conv1 = nn.Conv2d(1, 32,kernel_size=3, stride=1, padding=0, has_bias=False, pad_mode="valid")
        self.conv2 = nn.Conv2d(32, 64,kernel_size=3, stride=1, padding=0, has_bias=False, pad_mode="valid")
        self.conv3 = nn.Conv2d(64, 128,kernel_size=3, stride=1, padding=0, has_bias=False, pad_mode="valid")
        self.maxpool2d = nn.MaxPool2d(kernel_size=2, stride=2)
        self.relu = nn.ReLU()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Dense(128 * 11 * 11, 128)
        self.fc2 = nn.Dense(128, self.num_class)

    def construct(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = self.conv2(x)
        x = self.relu(x)
        x = self.conv3(x)
        x = self.relu(x)
        x = self.maxpool2d(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x
```

下面这部分用于创建加入正则化的卷积神经网络，网络结构为卷积层 1→dropout 层 1→卷积层 2→dropout 层 1→卷积层 3→dropout 层 1→最大池化层→ dropout 层 2→全连接层 1→dropout 层 2→全连接层 2：

```
# 定义卷积神经网络，有正则化
class ForwardFashionRegularization(nn.Cell):
    def __init__(self, num_class=10):  # 一共分十类，图片通道数是 1
        super(ForwardFashionRegularization, self).__init__()
        self.num_class = num_class
        self.conv1 = nn.Conv2d(1, 32,kernel_size=3, stride=1, padding=0, has_bias=False, pad_mode="valid")
        self.conv2 = nn.Conv2d(32, 64,kernel_size=3, stride=1, padding=0, has_bias=False, pad_mode="valid")
```

```
        self.conv3 = nn.Conv2d(64, 128,kernel_size=3, stride=1, padding=0, has_bias=False, pad_mode="valid")
        self.maxpool2d = nn.MaxPool2d(kernel_size=2, stride=2)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Dense(3200, 128)
        self.bn = nn.BatchNorm1d(128)
        self.fc2 = nn.Dense(128, self.num_class)

    def construct(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = self.conv2(x)
        x = self.relu(x)
        x = self.maxpool2d(x)
        x = self.dropout(x)
        x = self.conv3(x)
        x = self.relu(x)
        x = self.maxpool2d(x)
        x = self.dropout(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.bn(x)
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

步骤 2：启动训练

为这个模型指定优化器（adam）、损失函数（crossentropy）和度量(accuracy)，然后启动训练，最后进行验证。

```
def train(Net):
    ds_train, ds_test = create_dataset()
    # 构建网络
    network = Net(cfg.num_classes)
    # 定义模型的损失函数，优化器
```

```
    net_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")
    net_opt = nn.Adam(network.trainable_params(), cfg.lr)

    # 训练模型

    model = Model(network, loss_fn=net_loss, optimizer=net_opt, metrics={'acc': Accuracy()})
    loss_cb = LossMonitor()
    print("============== Starting Training ==============")
    model.train(30, ds_train, callbacks=[loss_cb], dataset_sink_mode=True)

    # 验证

    metric = model.eval(ds_test)
    print(metric)


    return model
```

训练并验证无正则化的网络。

```
# 训练无正则化的网络
model = train(ForwardFashion)
```

输出结果：

```
============== Starting Training ==============
epoch: 1 step: 937, loss is 0.36209568
epoch: 2 step: 937, loss is 0.11113132
epoch: 3 step: 937, loss is 0.107788906
epoch: 4 step: 937, loss is 0.12908919
epoch: 5 step: 937, loss is 0.078461185
epoch: 6 step: 937, loss is 0.18977618
epoch: 7 step: 937, loss is 0.15168177
epoch: 8 step: 937, loss is 0.06739945
epoch: 9 step: 937, loss is 0.14379226
epoch: 10 step: 937, loss is 0.076876596
epoch: 11 step: 937, loss is 0.055951692
epoch: 12 step: 937, loss is 0.022819532
epoch: 13 step: 937, loss is 0.10054826
epoch: 14 step: 937, loss is 0.012528818
epoch: 15 step: 937, loss is 0.0076259132
epoch: 16 step: 937, loss is 0.07877082
epoch: 17 step: 937, loss is 0.031406786
epoch: 18 step: 937, loss is 0.009203883
epoch: 19 step: 937, loss is 0.005287296
epoch: 20 step: 937, loss is 0.0929834
```

```
epoch: 21 step: 937, loss is 0.0015465739
epoch: 22 step: 937, loss is 0.03491202
epoch: 23 step: 937, loss is 0.0005662525
epoch: 24 step: 937, loss is 0.010102608
epoch: 25 step: 937, loss is 0.003999765
epoch: 26 step: 937, loss is 0.011775437
epoch: 27 step: 937, loss is 0.080310896
epoch: 28 step: 937, loss is 0.0018242185
epoch: 29 step: 937, loss is 0.007360851
epoch: 30 step: 937, loss is 0.003999797
{'acc': 0.9147636217948718}
```

训练并验证有正则化的网络。

```
# 训练有正则化的网络
model = train(ForwardFashionRegularization)
```

输出结果：

```
============== Starting Training ==============
epoch: 1 step: 937, loss is 0.29856867
epoch: 2 step: 937, loss is 0.28910726
epoch: 3 step: 937, loss is 0.18035105
epoch: 4 step: 937, loss is 0.2785972
epoch: 5 step: 937, loss is 0.21400028
epoch: 6 step: 937, loss is 0.27920294
epoch: 7 step: 937, loss is 0.17452516
epoch: 8 step: 937, loss is 0.309029
epoch: 9 step: 937, loss is 0.30411178
epoch: 10 step: 937, loss is 0.2842149
epoch: 11 step: 937, loss is 0.22666603
epoch: 12 step: 937, loss is 0.16507925
epoch: 13 step: 937, loss is 0.17004505
epoch: 14 step: 937, loss is 0.23396353
epoch: 15 step: 937, loss is 0.20207018
epoch: 16 step: 937, loss is 0.43118417
epoch: 17 step: 937, loss is 0.23762615
epoch: 18 step: 937, loss is 0.24660718
epoch: 19 step: 937, loss is 0.12197974
epoch: 20 step: 937, loss is 0.22719634
epoch: 21 step: 937, loss is 0.2809552
epoch: 22 step: 937, loss is 0.21124852
epoch: 23 step: 937, loss is 0.2100177
epoch: 24 step: 937, loss is 0.29766798
epoch: 25 step: 937, loss is 0.115716025
```

```
epoch: 26 step: 937, loss is 0.41360933
epoch: 27 step: 937, loss is 0.11700327
epoch: 28 step: 937, loss is 0.2552187
epoch: 29 step: 937, loss is 0.14747506
epoch: 30 step: 937, loss is 0.19088028
{'acc': 0.9234775641025641}
```

步骤 3：预测模型

使用上述训练好的模型对测试数据集进行预测。打印预测结果

```
# 预测
ds_test, _ = create_dataset()
test_ = ds_test.create_dict_iterator(output_numpy=True).__next__()
predictions = model.predict(Tensor(test_['x']))
predictions = predictions.asnumpy()
for i in range(15):
    p_np = predictions[i, :]
    p_list = p_np.tolist()
    print('第' + str(i) + '个 sample 预测结果：', p_list.index(max(p_list)), '    真实结果：',
test_['y'][i])
```

输出结果：

```
第 0 个 sample 预测结果：  3     真实结果：  3
第 1 个 sample 预测结果：  5     真实结果：  5
第 2 个 sample 预测结果：  6     真实结果：  2
第 3 个 sample 预测结果：  4     真实结果：  4
第 4 个 sample 预测结果：  3     真实结果：  3
第 5 个 sample 预测结果：  3     真实结果：  3
第 6 个 sample 预测结果：  7     真实结果：  7
第 7 个 sample 预测结果：  8     真实结果：  8
第 8 个 sample 预测结果：  3     真实结果：  3
第 9 个 sample 预测结果：  5     真实结果：  5
第 10 个 sample 预测结果：  9     真实结果：  9
第 11 个 sample 预测结果：  2     真实结果：  4
第 12 个 sample 预测结果：  6     真实结果：  6
第 13 个 sample 预测结果：  5     真实结果：  5
第 14 个 sample 预测结果：  0     真实结果：  0
```

步骤 4：可视化结果

定义可视化函数。

```
# ------------------定义可视化函数------------------------------

# 输入预测结果序列，真实标签序列，以及图片序列
```

```
# 目标是根据预测值对错，让其标签显示为红色或者蓝色。对：标签为红色；错：
标签为蓝色
def plot_image(predictions_array, true_label, img):
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    # 显示对应图片
    plt.imshow(img, cmap=plt.cm.binary)
    # 显示预测结果的颜色，如果对上了是蓝色，否则为红色
    predicted_label = np.argmax(predictions_array)
    if predicted_label == true_label:
        color = 'blue'
    else:
        color = 'red'
    # 显示对应标签的格式，样式
    plt.xlabel('{},{:2.0f}% ({})'.format(class_names[predicted_label],
                                         100 * np.max(predictions_array),
                                         class_names[true_label]),
color=color)

# 将预测的结果以柱状图形状显示蓝对红错
def plot_value_array(predictions_array, true_label):
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    this_plot = plt.bar(range(10), predictions_array, color='#777777')
    plt.ylim([0, 1])
    predicted_label = np.argmax(predictions_array)
    this_plot[predicted_label].set_color('red')
    this_plot[true_label].set_color('blue')

import numpy as np
def softmax_np(x):
    x = x - np.max(x)
```

```
    exp_x = np.exp(x)
    softmax_x = exp_x/np.sum(exp_x)
    return softmax_x
```

预测结果可视化，输入预测结果序列，真实标签序列，以及图片序列。目标是根据预测值对错，让其标签显示为红色或者蓝色。对：标签为蓝色；错：标签为红色。最后预测 15 个图像与标签，将预测的结果以柱状图形状显示蓝对红错。

```
# 预测 15 个图像与标签，并展现出来
num_rows = 5
num_cols = 3
num_images = num_rows * num_cols
plt.figure(figsize=(2 * 2 * num_cols, 2 * num_rows))
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
for i in range(num_images):
    plt.subplot(num_rows, 2 * num_cols, 2 * i + 1)
    pred_np_ = predictions[i, :]
    pred_np_ = softmax_np(pred_np_)
    plot_image(pred_np_, test_['y'][i], test_['x'][i, 0, ...])
    plt.subplot(num_rows, 2 * num_cols, 2 * i + 2)
    plot_value_array(pred_np_, test_['y'][i])
plt.show()
```

输出结果：

3.2.4.4 观察总结

如果神经网络具有较少的层数和神经元数量，则它可能在训练数据集上表现不佳，即可能导致欠拟合问题。但是，如果神经网络具有很多的层数和神经元数量，它可能在训练数据集上表现得很好（高精度），但是可能在测试数据集上表现不好，即可能导致过拟合问题。

不幸的是，没有确定公式来确定模型的正确结构（层数和每层中的神经元数量），搭建网络时将需要不断尝试，来找到表现不错的架构。

虽然对于神经网络，我们还有其他正则化技术，如 L1, L2 正则化，Dropout 是所有正则化技术中最有效的，也是最常用的。Dropout 会随机地丢弃层的一些输出特征（神经元），即设置为零。假设在应用 Dropout 之前层的输出为[0.5, 0.8, 2.2, 0.9, 0.1]，在应用 Dropout 之后，输出将为[0, 0.8, 2.2, 0, 0.1]。 “Dropout 率” 是层中被置为零的特征（神经元）的分数。一般来说，Dropout 率保持在 0.2 至 0.5 之间。

请注意，Dropout 只在训练阶段使用。在对测试数据集进行预测时，我们不需要从模型中手动移除 Dropout 层。 在测试阶段，不会应用 Dropout。

因此，为了减少本案例中的过拟合问题，让我们应用正则化技术-Dropout 来提升测试表现。

3.2.5 实验要求

请设计实验分析不同网络结构与训练的超参数对分类结果的影响。

四． 糖尿病预测

4.1 实验说明

该数据集(pima-indians-diabetes.data)是来自美国疾病控制预防中心的数据，背景是记录美国的糖尿病症状信息，现在美国 1/7 的成年人患有糖尿病。但是到 2050 年，这个比例将会快速增长至高达 1/3。可以利用从 UCI 机器学习数据库里一个关于印第安人糖尿病数据集，通过数据挖掘相关算法来预测糖尿病，该问题本质上是一个二元分类问题。

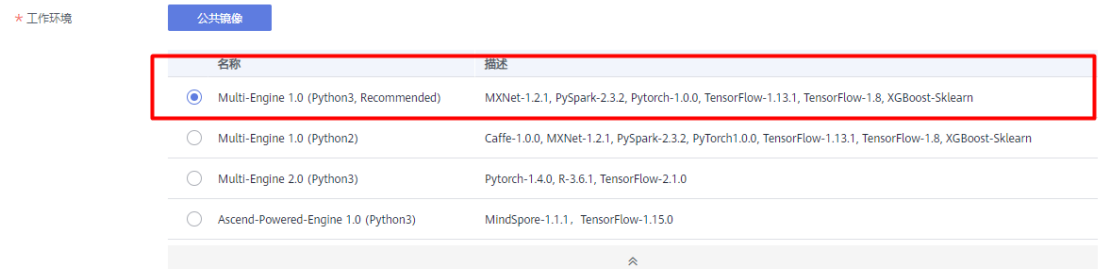
4.2 实验建模流程要求

基于 Diabete 数据，使用支持向量机进行糖尿病预测。

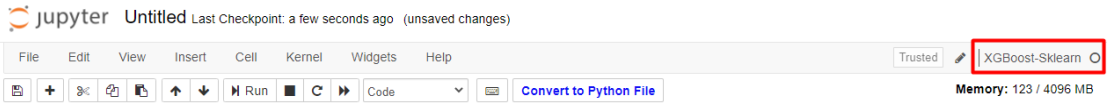
4.2.1 环境要求

ModelArts 平台：

选择工作环境，若新版找不到 XGBoost-Sklearn 实验环境，可返回旧版华为云寻找创建：



选择 kernel 为 XGboost-sklearn:



4.2.2 实验实现步骤要求

(1) 相关模块导入

步骤 1：要求导入相关数据读取、处理、分析、可视化，算法模块等

(2) 数据导入与初步探索

步骤 1：要求载入本地数据集(pima-indians-diabetes.data)，以 dataframe 形式存放后，命名为 df

步骤 2：查看数据尺寸、打印信息，判断特征的类型（名称性、数值型），目标变量分布以及查看是否均衡

步骤 3：对 df 特征进行相关性可视化

步骤 4：对 df 每个特征的分布进行可视化查看

步骤 5：对输入特征进行降维，选择 PCA，并按提示补充如下代码中划线部分内容。

```
### 对输入特征进行降维处理
from sklearn.decomposition import PCA
from sklearn import preprocessing                    #调用标准化模块
                                                     #降维训练前需要对数据标准化
pca = PCA(              )                            # 保留 99%信息的主成分个主成
分
X_pca =pca.fit(X_std).transform(X_std)
```

步骤 6：结合相关性分析和降维后的结论，选择数据进行拆分为训练集和测试集，拆分比例设置为 0.1，指定以 Target 的比例做分层抽样。部分提示如下：

```
from collections import Counter
from sklearn.model_selection import train_test_split
```

步骤 7：选择支持向量机算法对拆分后的数据进行建模训练和预测，其中要求将原始模型做 5 折交叉验证，评估指标选择 f1。

部分提示如下：

```
#引入支持向量机和交叉验证的库
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score

#引入评价指标的库
from sklearn.metrics import f1_score
```

步骤 8：对支持向量机的惩罚因子 C 和核函数进行网格搜索，网格设置参考如下：

```
C=[0.001,0.01,0.1,1.0]
kernel = ['linear','poly','rbf','sigmoid']
tuned_parameters= dict(C=C, kernel=kernel)
```

步骤 9：根据搜索参数，最后确认模型，进行预测。

```
#网格搜素
from sklearn.model_selection import GridSearchCV
grid= GridSearchCV(clf, tuned_parameters, cv=5, scoring='f1')
grid.fit(X_train,y_train)
print('best score for model {}'.format(grid.best_score_))
print('best parameters for model {}'.format(grid.best_params_))
print('best parameters for model {}'.format(grid.best_estimator_))
```

```
#### 根据选择后的参数，最后预测
clf_final = SVC(C=1, kernel='rbf')
clf_final.fit(X_train,y_train)
y_train_pred = clf_final.predict(X_train)
print('final score of model version2: {}'.format(f1_score(y_train,y_train_pred)))
```

4.3 实验要求

请设置合理的惩罚因子区间，设计实验分析惩罚因子、核函数、降维操作等因素对分类结果的影响。

4.4 参考答案

参考：SVM-糖尿病预测.ipynb