《机器学习》 – 监督学习



华为技术有限公司

目录

[1 实验介绍 3](#_Toc55223209)

[1.1 实验目的 3](#_Toc55223210)

[1.2 实验清单 3](#_Toc55223211)

[2 线性回归模拟实验 5](#_Toc55223212)

[2.1 实验介绍 5](#_Toc55223213)

[2.1.1 简介 5](#_Toc55223214)

[2.1.2 实验目的 5](#_Toc55223215)

[2.2 实验环境要求 5](#_Toc55223216)

[2.3 实验步骤 5](#_Toc55223217)

[2.3.1 导入MindSpore模块和辅助模块 5](#_Toc55223218)

[2.3.2 生成模拟数据 6](#_Toc55223219)

[2.3.3 建模 6](#_Toc55223220)

[2.3.4 使用模拟数据训练模型 6](#_Toc55223221)

[2.3.5 使用训练好的模型进行预测 7](#_Toc55223222)

[2.3.6 可视化 7](#_Toc55223223)

[3 APP评分预测 9](#_Toc55223224)

[3.1 实验介绍 9](#_Toc55223225)

[3.1.1 简介 9](#_Toc55223226)

[3.1.2 实验目的 9](#_Toc55223227)

[3.2 实验环境要求 9](#_Toc55223228)

[3.3 实验总体设计 9](#_Toc55223229)

[3.4 实验步骤 10](#_Toc55223230)

[3.4.1 数据读取 10](#_Toc55223231)

[3.4.2 线性回归 11](#_Toc55223232)

[3.4.3 **SVM回归** 11](#_Toc55223233)

[3.4.4 KNN回归 12](#_Toc55223234)

[3.4.5 思考题 14](#_Toc55223235)

[3.4.6 模型评估与选择 15](#_Toc55223236)

[3.5 实验小结 17](#_Toc55223237)

[3.6 参考答案 17](#_Toc55223238)

[4 鸢尾花二分类实验 19](#_Toc55223239)

[4.1 实验介绍 19](#_Toc55223240)

[4.1.1 简介 19](#_Toc55223241)

[4.1.2 实验目的 19](#_Toc55223242)

[4.2 实验环境要求 19](#_Toc55223243)

[4.3 实验总体设计 20](#_Toc55223244)

[4.4 实验过程 20](#_Toc55223245)

[4.4.1 数据准备 20](#_Toc55223246)

[4.4.2 数据读取与处理 21](#_Toc55223247)

[4.4.3 模型建立与训练 22](#_Toc55223248)

[4.4.4 模型评估 24](#_Toc55223249)

[4.5 实验小结 24](#_Toc55223250)

[4.6 创新设计 24](#_Toc55223251)

[5 糖尿病预测 25](#_Toc55223252)

[5.1 实验说明 25](#_Toc55223253)

[5.2 实验建模流程要求 25](#_Toc55223254)

[5.2.1 环境要求 25](#_Toc55223255)

[5.2.2 实验实现步骤要求 25](#_Toc55223256)

[5.3 参考答案 27](#_Toc55223257)

[6 信用违约预测 27](#_Toc55223258)

[6.1 实验说明 27](#_Toc55223259)

[6.2 实验建模流程要求 27](#_Toc55223260)

[6.2.1 环境要求 27](#_Toc55223261)

[6.2.2 实验实现步骤要求 27](#_Toc55223262)

[6.3 参考答案 30](#_Toc55223263)

# 实验介绍

机器学习分为监督学习、无监督学习、半监督学习、强化学习。监督学习是指利用一组已知类别的样本调整分类器的参数，使其达到所要求性能的过程，也称为监督训练或有教师学习。分类和回归是监督学习中的两种典型任务。

本章实验涉及线性回归、SVM回归、KNN回归等回归算法，以及逻辑回归、决策树、随机森林等分类算法，通过不同算法的效果对比来加深对算法的理解。

本章实验难度包含：初级、中级、高级。

初级实验：线性回归模拟实验、鸢尾花二分类实验

中级实验：APP评分预测实验、糖尿病预测实验

高级实验：信用违约预测实验

## 实验目的

本章实验分为回归与分类两部分，通过本章实验的学习，您将能够：

了解华为云ModelArts自动学习服务

掌握使用MindSpore进行线性回归与分类

掌握回归与分类任务的区别与流程

掌握线性回归、SVM回归、KNN回归等回归算法的原理与使用

掌握逻辑回归、决策树、随机森林等分类算法的原理与使用

掌握模型评估的方法

## 实验清单

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验 | 简述 | 难度 | 软件环境 | 开发环境 |
| 线性回归模拟实验 | 基于随机生成的数据，进行线性回归实验，实现对数据的拟合 | 初级 | MindSpore | ModelArts |
| APP评分预测（回归+分类） | 基于APP下载信息，分别使用线性回归、SVM回归、KNN回归对APP评分进行预测；进而将评分数据离散化，用决策树模型训练分类模型。 | 中级 | Python3 | 本地PC |
| 银行存款预测（分类） | 基于银行客户是否存款数据，使用ModelArts自动学习服务，实现是否存款预测。 | 初级 | Python3 | ModelArts |
| 鸢尾花二分类预测（分类） | 基于两个种类的鸢尾花数据，进行逻辑回归实验，实现鸢尾花的二分类预测 | 初级 | MindSpore | ModelArts |
| 糖尿病预测（分类） | 基于Diabete数据，使用逻辑回归进行糖尿病预测。 | 中级 | Python3 | 本地PC |
| 信用违约预测（分类） | 基于信贷数据，分别使用逻辑回归、随机森林算法实现信用违约预测。 | 高级 | Python3 | 本地PC |

# 线性回归模拟实验

## 实验介绍

### 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。

### 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。

## 实验环境要求

MindSpore 0.5.0（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。

## 实验步骤

环境准备可参考：《机器学习》-数据预处理与特征工程实验手册中的华为云使用指导来准备环境。

### 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")

### 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)

### 建模

使用MindSpore提供的nn.Dense(1, 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()

### 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

### 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.default\_input.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]]

### 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

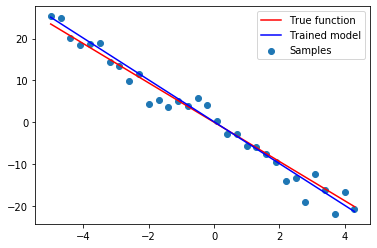
plt.scatter(x, y, label='Samples')

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：



# APP评分预测

## 实验介绍

### 简介

本次实验利用数据预处理与特征工程中处理好的数据集来训练一个回归和分类模型，对评分的预测。

首先线性回归、SVM、KNN回归算法，训练三个回归模型。接下来将评分数据离散化，用决策树模型训练分类模型。这样可以更好学习算法的使用和模型的评估。

### 实验目的

掌握线性回归算法的应用实践

掌握SVM算法的应用实践

掌握决策树算法的应用实践

掌握KNN算法的应用实践

## 实验环境要求

本地PC，Python3

## 实验总体设计

## 实验步骤

### 数据读取

代码：

#导入相关库

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

#读取数据集

data\_after\_pca = pd.read\_csv('after\_pca.csv',index\_col=0)

data = pd.read\_csv('AppDataV2.csv',index\_col=0)

data\_after\_var = pd.read\_csv("data\_after\_var",index\_col=0)

data\_after\_filter = pd.read\_csv("df\_after\_filter.csv",index\_col=0)

#首先确定样本的数据的标签

X = data.drop(["Rating"],axis='columns')

Y = data["Rating"]

X\_var = data\_after\_var.drop(["Rating"],axis='columns')

Y\_var = data\_after\_var["Rating"]

X\_pca = data\_after\_pca.drop(["Rating"],axis='columns')

Y\_pca = data\_after\_pca["Rating"]

X\_filter = data\_after\_filter.drop(["Rating"],axis='columns')

Y\_filter = data\_after\_filter["Rating"]

X.info()

输出：

<class 'pandas.core.frame.DataFrame'>

Int64Index: 10240 entries, 0 to 10239

Data columns (total 40 columns):

Reviews 10240 non-null int64

Size 10240 non-null float64

Installs 10240 non-null float64

Type 10240 non-null int64

Price 10240 non-null float64

Content Rating 10240 non-null int64

Genres 10240 non-null int64

Category\_ART\_AND\_DESIGN 10240 non-null int64

Category\_AUTO\_AND\_VEHICLES 10240 non-null int64

Category\_BEAUTY 10240 non-null int64

Category\_BOOKS\_AND\_REFERENCE 10240 non-null int64

…

代码：

#数据集划分

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size = 0.2, random\_state = 10)

### 线性回归

sklearn.linear\_model.LinearRegression(fit\_intercept=True, normalize=False, copy\_X=True)

参数说明：

fit\_intercept：默认True，是否计算模型的截距，为False时，则数据中心化处理。

normalize：默认False，是否中心化，或者使用sklearn.preprocessing.StandardScaler()。

copy\_X：默认True，否则X会被改写。

代码：

from sklearn.linear\_model import LinearRegression

from sklearn.neighbors import KNeighborsRegressor

from sklearn.metrics import mean\_squared\_error,mean\_absolute\_error,accuracy\_score,accuracy\_score,r2\_score

#初始化线性回归模型

linreg = LinearRegression()

#训练模型

linreg.fit(X\_train,y\_train)

#训练集上的MSE

linreg\_pred\_train = linreg.predict(X\_train)

linreg\_mse\_train = mean\_squared\_error(linreg\_pred\_train,y\_train)

#输出测试集上的测试结果

linreg\_pred\_test=linreg.predict(X\_test)

linreg\_mse\_test = mean\_squared\_error(linreg\_pred\_test,y\_test)

print("训练集MSE：", linreg\_mse\_train)

print("测试集MSE：", linreg\_mse\_test)

输出：

训练集MSE： 0.2308388846541144

测试集MSE： 0.22350603712434897

### **SVM回归**

sklearn.svm.SVR（kernel ='rbf'，degree = 3，gamma ='auto\_deprecated'，coef0 = 0.0，tol = 0.001，C = 1.0，verbose = False，max\_iter = -1 ）

参数说明：

kernel：指定要在算法中使用的内核类型。可以是'linear'，'poly'，'rbf'，'sigmoid'，'precomputed'或者callable之一，默认为rbf。

degree：多项式poly函数的维度，默认是3，选择其他核函数时会被忽略。

gamma：‘rbf’,‘poly’ 和‘sigmoid’的核函数参数。默认是’auto’，则会选择1/n\_features

coef0：核函数的常数项。对于‘poly’和 ‘sigmoid’有用，默认值= 0.0。

tol：停止训练的误差值大小，默认值= 1e-3。

C：惩罚参数，默认= 1.0。C越大，相当于惩罚松弛变量，希望松弛变量接近0，即对误分类的惩罚增大，趋向于对训练集全分对的情况，这样对训练集测试时准确率很高，但泛化能力弱。C值小，对误分类的惩罚减小，允许容错，将他们当成噪声点，泛化能力较强。

verbose：日志。

max\_iter：最大迭代次数。-1为无限制。

**代码：**

from sklearn.svm import SVR

from sklearn.metrics import accuracy\_score

#初始化决策树模型

svr=SVR(kernel='rbf',C=1)

#训练

svr.fit(X\_train,y\_train\_int)

#训练集上的MSE

svr\_pred\_train = svr.predict(X\_train)

svr\_mse\_train = mean\_squared\_error(svr\_pred\_train,y\_train)

#输出测试集上的测试结果

svr\_pred\_test=svr.predict(X\_test)

svr\_mse\_test = mean\_squared\_error(svr\_pred\_test,y\_test)

print("训练集MSE：", svr\_mse\_train)

print("测试集MSE：", svr\_mse\_test)

输出：

训练集MSE： 0.28852538417710893

测试集MSE： 0.44229386180754227

### KNN回归

sklearn.neighbors.KNeighborsRegressor(n\_neighbors=5, weights='uniform', algorithm='auto', leaf\_size=30)：

参数说明：

n\_neighbors：knn算法中指定以最近的几个最近邻样本具有投票权，默认参数为5

algrithm：即内部采用什么算法实现。有以下几种选择参数：

'ball\_tree':球树、

'kd\_tree':kd树、

'brute':暴力搜索、

'auto':自动根据数据的类型和结构选择合适的算法。默认情况下是‘auto’。

暴力搜索就不用说了大家都知道。具体前两种树型数据结构哪种好视情况而定。KD树是对依次对K维坐标轴，以中值切分构造的树,每一个节点是一个超矩形，在维数小于20时效率最高ball tree 是为了克服KD树高维失效而发明的，其构造过程是以质心C和半径r分割样本空间，每一个节点是一个超球体。一般低维数据用kd\_tree速度快，用ball\_tree相对较慢。超过20维之后的高维数据用kd\_tree效果反而不佳，而ball\_tree效果要好，具体构造过程及优劣势的理论大家有兴趣可以去具体学习。

leaf\_size:这个值控制了使用KD树或者球树时， 停止建子树的叶子节点数量的阈值。这个值越小，则生成的KD树或者球树就越大，层数越深，建树时间越长，反之，则生成的KD树或者球树会小，层数较浅，建树时间较短。默认是30.

请根据线性回归的实现和KNN的参数说明，训练一个KNN模型。代码填写：

#初始化knn模型

knn\_model = KNeighborsRegressor(n\_neighbors=50)

#训练

knn\_model.fit(X\_train,y\_train)

#训练集上的MSE

knn\_pred\_train = knn\_model.predict(X\_train)

knn\_mse\_train = mean\_squared\_error(knn\_pred\_train,y\_train)

#输出测试集上的测试结果

knn\_pred\_test=knn\_model.predict(X\_test)

knn\_mse\_test = mean\_squared\_error(knn\_pred\_test,y\_test)

print("训练集MSE：", knn\_mse\_train)

print("测试集MSE：", knn\_mse\_test)

输出：

训练集MSE： 0.2044843076171875

测试集MSE： 0.20693282421875

接下来简单对三个模型的输出精度进行对比。

代码：

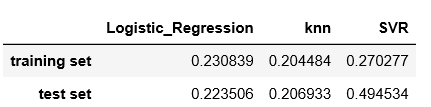
model\_mse = pd.DataFrame(data=[[linreg\_mse\_train,knn\_mse\_train,svr\_mse\_train],

[linreg\_mse\_test,knn\_mse\_test,svr\_mse\_test]],

columns=['Logistic\_Regression','knn','SVR'],index=["training set","test set"])

model\_mse

输出：

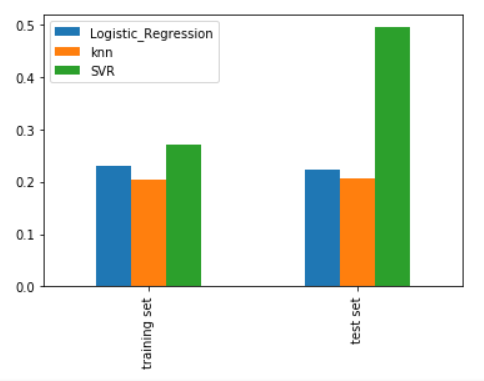


代码：

plt.figure(figsize=(20, 10))

model\_mse.plot(kind = 'bar')

输出：



LR、KNN、SVR算法对比

### 思考题

将数据的标签变为整型，自行编码训练一个决策树分类模型，其中决策树模型使用方法如下：

DecisionTreeClassifier(criterion="mse",splitter="best",max\_depth=None,min\_samples\_split=2,min\_samples\_leaf=1,min\_weight\_fraction\_leaf=0., max\_features=None,random\_state=None,max\_leaf\_nodes=None)

参数说明：

criterion:切分质量的评价准则。默认为'mse'(mean squared error)。

splitter:指定了在每个节点切分的策略。有两种切分策略：

(1).splitter='best':表示选择最优的切分特征和切分点。

(2).splitter='random':表示随机切分。

max\_depth:指定树的最大深度。如果为None，则表示树的深度不限，直到每个叶子都是纯净的。

min\_samples\_split:默认为2。它指定了分裂一个内部节点(非叶子节点)需要的最小样本数。如果为浮点数(0到1之间)，最少样本分割数为ceil(min\_samples\_split \* n\_samples)

min\_samples\_leaf:指定了每个叶子节点包含的最少样本数。如果为浮点数(0到1之间)，每个叶子节点包含的最少样本数为ceil(min\_samples\_leaf \* n\_samples)

min\_weight\_fraction\_leaf:指定了叶子节点中样本的最小权重系数。默认情况下样本有相同的权重。

max\_feature:

(1).如果是整数，则每次节点分裂只考虑max\_feature个特征。

(2).如果是浮点数(0到1之间)，则每次分裂节点的时候只考虑int(max\_features \* n\_features)个特征。

(3).如果是字符串'auto',max\_features=n\_features。

(4).如果是字符串'sqrt',max\_features=sqrt(n\_features)。

(5).如果是字符串'log2',max\_features=log2(n\_features)。

(6).如果是None，max\_feature=n\_feature。

random\_state:随机数生成器

max\_leaf\_nodes:

(1).如果为None，则叶子节点数量不限。

(2).如果不为None，则max\_depth被忽略。

### 模型评估与选择

通过上面的对比，可以看出4个模型都欠拟合的状态。接下来将使用交叉验证、网格搜索和随机搜索的方式，选择模型的超参数。

实验目的：

（1）掌握交叉验证算法的应用实践

（2）掌握网络搜索的实现

（3）掌握随机搜索的实现

#### 交叉验证

将用交叉验证来搜索KNN的模型中n\_neighbors的最佳参数值。

代码：

from sklearn.model\_selection import cross\_val\_score # K折交叉验证模块

#建立测试参数集

k\_range = range(15, 100)

k\_scores = []

#藉由迭代的方式来计算不同参数对模型的影响，并返回交叉验证后的平均准确率

for k in k\_range:

knn = KNeighborsRegressor(n\_neighbors=k)

scores = cross\_val\_score(knn, X\_train, y\_train, cv=10)

k\_scores.append(scores.mean())

#可视化数据

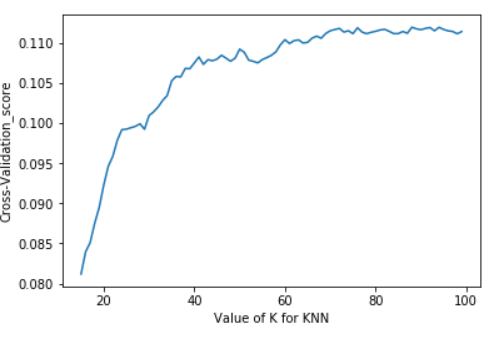
plt.plot(k\_range, k\_scores)

plt.xlabel('Value of K for KNN')

plt.ylabel('Cross-Validation\_score')

plt.show()

输出：



不同K值对模型的影响

根据上图结果，可知n\_neighbors的数值在62左右最佳，大于此值后模型的表现没有明显提升。

#### 参数搜索

本小节分别用将用网格搜索和随机搜索的方式，对决策树分类器和SVM回归模型的超参数进行搜索。

代码：

###决策树分类器

from sklearn.model\_selection import GridSearchCV

params = [{'criterion':['gini'],'max\_depth':[30,50,60,100],'min\_samples\_leaf':[2,3,5,10],'min\_impurity\_decrease':[0.1,0.2,0.5]},

{'criterion':['gini','entropy']},

{'max\_depth': [30,60,100], 'min\_impurity\_decrease':[0.1,0.2,0.5]}]

best\_model = GridSearchCV(dtree, param\_grid=params,cv = 5,scoring ="accuracy")

best\_model.fit(X\_train,y\_train\_int)

print('最优分类器:',best\_model.best\_params\_,'最优分数:', best\_model.best\_score\_) # 得到最优的参数和分值

输出：

最优分类器: {'criterion': 'gini', 'max\_depth': 30, 'min\_impurity\_decrease': 0.1, 'min\_samples\_leaf': 2} 最优分数: 0.781982421875

接下来用随机搜索搜索SVM回归模型的参数。

代码：

from sklearn.model\_selection import RandomizedSearchCV

params\_svr = {'kernel': ['rbf'], 'C': np.logspace(-3, 2, 6), 'gamma':np.arange(0,10,2)}

best\_svr\_model = RandomizedSearchCV(svr, param\_distributions=params\_svr,cv = 3,scoring ="neg\_mean\_squared\_error")

best\_svr\_model.fit(X,Y)

print('最优分类器:',best\_svr\_model.best\_params\_,'最优分数:', best\_svr\_model.best\_score\_) # 得到最优的参数和分值

输出：

最优分类器: {'kernel': 'rbf', 'gamma': 4, 'C': 1.0} 最优分数: -0.24169698734960568

## 实验小结

本章通过代码实践，帮助学习者了解了机器学习算法实践应用的流程，并使用处理过的APP评分数据进行回归和分类建模，最后通过交叉验证、网络搜素和随机搜索等算法对模型进行超参数寻优。

## 参考答案

代码：

y\_train\_int = y\_train.astype(int)

y\_test\_int = y\_test.astype(int)

y\_train\_int.head()

输出：

2077 4

4387 3

8974 4

8189 4

6541 4

Name: Rating, dtype: int32

代码：

from sklearn.tree import DecisionTreeClassifier

初始化决策树模型

dtree=DecisionTreeClassifier(max\_depth=8, min\_samples\_leaf=5, random\_state=42)

#训练

dtree.fit(X\_train,y\_train\_int)

#训练集上的MSE

dtree\_pred\_train = dtree.predict(X\_train)

dtree\_mse\_train = dtree.score(X\_train,y\_train\_int)

#输出测试集上的测试结果

dtree\_pred\_test=dtree.predict(X\_test)

dtree\_mse\_test =dtree.score(X\_test,y\_test\_int)

print("训练集MSE：", dtree\_mse\_train)

print("测试集MSE：", dtree\_mse\_test)

输出：

训练集MSE： 0.793701171875

测试集MSE： 0.7890625

# 鸢尾花二分类实验

## 实验介绍

### 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。

### 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。

## 实验环境要求

MindSpore 0.5.0（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。

## 实验总体设计

## 实验过程

### 数据准备

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!)

上传数据到OBS

点击新建的OBS桶名，通过“上传”、“新建文件夹”等功能，将数据集上传到OBS桶中。



上传数据集到OBS

### 数据读取与处理

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")

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

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)

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()

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]

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)

### 模型建立与训练

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')

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(https://www.mindspore.cn/api/zh-CN/0.3.0-alpha/api/python/mindspore/mindspore.ops.operations.html?#mindspore.ops.operations.SigmoidCrossEntropyWithLogits)作为非线性部分：

对于每个样本N\_i，模型的计算方式如下：

其中，是1D Tensor（含4个元素），是1D Tensor（含1个元素），是真实类别（2个类别{0, 1}中的一个），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)

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### 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

## 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。

## 创新设计

请使用Softmax函数作为联系函数，对完整的Iris数据集实现多分类任务。

# 糖尿病预测

## 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。

## 实验建模流程要求

基于Diabete数据，使用逻辑回归进行糖尿病预测。

### 环境要求

Python 3.7

### 实验实现步骤要求

#### 相关模块导入

要求导入相关数据读取、处理、分析、可视化，算法模块等

#### 数据导入与初步探索

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

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

对df特征进行相关性可视化

对df每个特征的分布进行可视化查看

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)

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

选择逻辑回归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：

#引入逻辑回归和交叉验证的库

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import cross\_val\_score

#引入评价指标的库

from sklearn.metrics import f1\_score

对逻辑回归的几个重要参数进行网格搜索，网格设置参考如下：

c\_range=[0.001,0.01,0.1,1.0]

solvers = ['liblinear','lbfgs','newton-cg','sag']

max\_iters=[80,100,150,200,300]

tuned\_parameters= dict(solver=solvers, C=c\_range,max\_iter=max\_iters)

根据搜索参数，最后确认模型，进行预测。

## 参考答案

参考：逻辑回归-糖尿病预测.ipynb

1. **信用违约预测**

## 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

## 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逻辑回归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。

### 环境要求

Python 3.7

### 实验实现步骤要求

要求导入相关数据读取、处理、分析、可视化，算法模块等

数据读取，数据框类型，命名为 df

查看Target的分布 ,是否违约（1是，0否）

可视化观察特征相关性

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()

打印出缺失率最高的前15个特征以及对应的缺失率

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。

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

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。

选择IOR方法筛选Couple\_L12\_Month\_Pay\_Amount异常值

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

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

使用众数填充null\_col中每列缺失值，直接改变df。

从df中删除无分类意义的特征列Cust\_No。

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

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。

引入StandardScaler标准化工具库，对训练集和测试集做标准化

使用Logistic Regression对标准化后的数据进行建模

对训练集X\_train\_std，y\_train进行过采样，指定过采样比例，此处为2:1。

from imblearn import over\_sampling

print('Original dataset shape {}'.format(Counter(y\_train)))

#ratio指定过采样比例，此处为2:1

smote\_model = over\_sampling.SMOTE(random\_state=7, ratio=0.5)

X\_train\_res,y\_train\_res = smote\_model.fit\_sample(X\_train\_std,y\_train)

print('Resampled dataset shape {}'.format(Counter(y\_train\_res)))

使用Logistic Regression对过采样且标准化后的数据X\_train\_res,y\_train\_res进行建模，且使用交叉验证（5折，默认scoring）进行查看。

对上一步中的Logistic Regression模型，调节常用的C和solver两个参数，使用网格搜索+交叉验证法

查看逻辑回归模型在测试集上的效果，选择多个评估指标accuracy\_score、precision\_score 、recall\_score 、f1\_score

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。

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']}

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)}

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

## 参考答案

参考：信用违约预测.ipynb