# 线性回归

## 导入实验所需包

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt

## 定义模型

### 参数初始化

w与b的初始化

### 定义预测函数与损失函数

预测函数为f(x)=w^T \* x\_i + b，损失函数为均方误差

### 模型训练

最小二乘法优化

### 模型保存与加载

使用numpy读取与保存模型参数

### 模型预测与评估

评估使用均方误差

# 使用最小二乘法实现线性回归  
class LinearRegression:  
 def \_\_init\_\_(self) -> None:  
 self.w = None  
 self.b = None  
  
 # 最小二乘法训练  
 def fit(self, X, y):  
 X = np.hstack([X, np.ones((X.shape[0], 1))])  
 self.w = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)  
 self.b = self.w[-1]  
 self.w = self.w[:-1]  
   
 # 预测  
 def predict(self, X):  
 X = np.hstack([X, np.ones((X.shape[0], 1))])  
 return X.dot(np.hstack([self.w, self.b]))  
   
 # 加载模型  
 def load(self, path):  
 data = np.load(path)  
 self.w = data['w']  
 self.b = data['b']  
  
 # 保存模型  
 def save(self, path):  
 np.savez(path, w=self.w, b=self.b)  
   
 # MSE评估  
 def mse(self, y\_true, y\_pred):  
 return np.mean((y\_true - y\_pred) \*\* 2)

## 数据处理

### 获取数据集

path = dataset/housing-data.csv

### 缺省值处理

使用均值填充

### 特征标准化

使用Z-score标准化

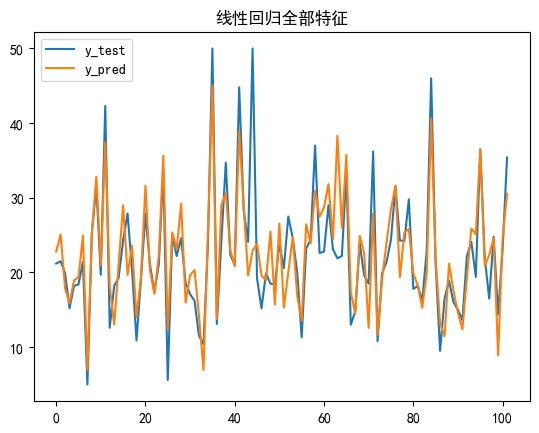
### 分割数据集

80%的数据作为训练集，20%的数据作为测试集

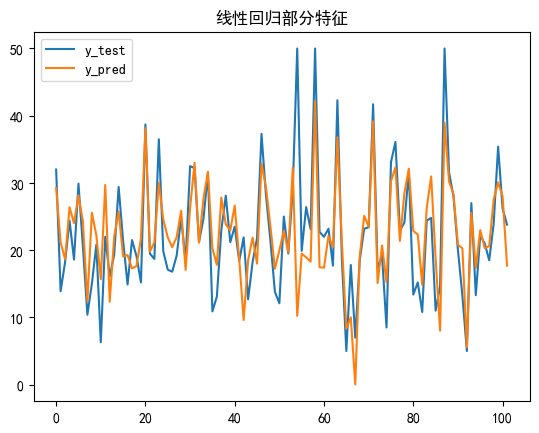
# 数据处理  
path = 'dataset/housing-data.csv'  
# 全部特征  
def data\_process\_full():  
 # 读取数据  
 data = pd.read\_csv(path)  
 # 均值填充缺省值  
 data = data.fillna(data.mean())  
 # Z-Score标准化  
 for col in data.columns[:-1]:  
 data[col] = (data[col] - data[col].mean()) / data[col].std()  
 # 分割数据集  
 data = np.array(data)  
 np.random.shuffle(data)  
 X = data[:, :-1]  
 y = data[:, -1]  
 X\_train = X[:int(0.8 \* len(X))]  
 y\_train = y[:int(0.8 \* len(y))]  
 X\_test = X[int(0.8 \* len(X)):]  
 y\_test = y[int(0.8 \* len(y)):]  
 return X\_train, y\_train, X\_test, y\_test  
  
# 部分特征  
def data\_process\_part():  
 # 读取数据  
 data = pd.read\_csv(path)  
 # 均值填充缺省值  
 data = data.fillna(data.mean())  
 # Z-Score标准化  
 for col in data.columns[:-1]:  
 data[col] = (data[col] - data[col].mean()) / data[col].std()  
 # 分割数据集  
 data = np.array(data)  
 np.random.shuffle(data)  
 X = data[:, [0, 1, 2, 3, 4, 5]]  
 y = data[:, -1]  
 X\_train = X[:int(0.8 \* len(X))]  
 y\_train = y[:int(0.8 \* len(y))]  
 X\_test = X[int(0.8 \* len(X)):]  
 y\_test = y[int(0.8 \* len(y)):]  
 return X\_train, y\_train, X\_test, y\_test

## 主函数

# 全部特征  
X\_train, y\_train, X\_test, y\_test = data\_process\_full()  
# 实例化模型  
model = LinearRegression()  
# 训练模型  
model.fit(X\_train, y\_train)  
# 保存模型  
model.save('model/linear\_model\_implement\_full.npz')  
# MSE评估  
y\_pred = model.predict(X\_test)  
  
# 将预测与实际值可视化  
plt.title('线性回归全部特征')  
plt.plot(y\_test, label='y\_test')  
plt.plot(y\_pred, label='y\_pred')  
plt.legend()  
plt.show()  
print('全部特征MSE:', model.mse(y\_test, y\_pred))  
  
# 部分特征  
X\_train, y\_train, X\_test, y\_test = data\_process\_part()  
# 实例化模型  
model = LinearRegression()  
# 训练模型  
model.fit(X\_train, y\_train)  
# 保存模型  
model.save('model/linear\_model\_implement\_part.npz')  
# MSE评估  
y\_pred = model.predict(X\_test)  
  
# 设置显示中文字体  
from pylab import mpl  
mpl.rcParams["font.sans-serif"] = ["SimHei"]  
  
# 将预测与实际值可视化  
plt.title('线性回归部分特征')  
plt.plot(y\_test, label='y\_test')  
plt.plot(y\_pred, label='y\_pred')  
plt.legend()  
plt.show()  
print('部分特征MSE:', model.mse(y\_test, y\_pred))



全部特征MSE: 22.215766722868906



部分特征MSE: 37.063954447819135

# 逻辑回归

## 导入实验所需包

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt

## 定义模型

### 参数初始化

w与b的初始化

### 定义预测函数与损失函数

预测函数为f(x)= 1 / (1 + e ^ -((w ^ T)x +))，损失函数为极大似然估计

### 模型训练

极大似然法优化

### 模型保存与加载

使用numpy读取与保存模型参数

### 模型预测与评估

评估使用正确率和召回率

# 使用极大似然法实现线性回归  
class LogisticRegression:  
 def \_\_init\_\_(self) -> None:  
 self.w = None  
 self.b = None  
  
 # sigmoid函数  
 def sigmoid(self, x):  
 return 1 / (1 + np.exp(-x))  
   
 # 损失函数  
 def cost(self, hx, y):  
 return -y \* np.log(hx) - (1 - y) \* np.log(1 - hx)  
  
 # 梯度下降  
 def gradient\_descent(self, X, y, lr=0.01):  
 y\_pred = self.sigmoid(X.dot(self.w))  
 grad = X.T.dot(y\_pred - y)  
 return lr \* grad  
  
 # 训练模型  
 def fit(self, X, y, lr=0.01, max\_iter=1000):  
 X = np.hstack([X, np.ones((X.shape[0], 1))])  
 self.w = np.random.randn(X.shape[1])  
 for i in range(max\_iter):  
 self.w -= self.gradient\_descent(X, y, lr)  
   
 # 预测  
 def predict(self, X):  
 X = np.hstack([X, np.ones((X.shape[0], 1))])  
 return self.sigmoid(X.dot(self.w))  
   
 # 加载模型  
 def load(self, path):  
 data = np.load(path)  
 self.w = data['w']  
 self.b = data['b']  
  
 # 保存模型  
 def save(self, path):  
 np.savez(path, w=self.w, b=self.b)  
   
 # 准确率  
 def accuracy(self, y\_true, y\_pred):  
 y\_pred = np.where(y\_pred > 0.5, 1, 0)  
 return np.mean(y\_true == y\_pred)  
  
 # recall   
 def recall(self, y\_true, y\_pred):  
 y\_pred = np.where(y\_pred > 0.5, 1, 0)  
 return np.sum(y\_true \* y\_pred) / np.sum(y\_true)

## 数据处理

### 获取数据集

path = dataset/breast-cancer-wisconsin.data

### 缺省值处理

使用均值填充

### 特征标准化

使用Z-score标准化

### 分割数据集

80%的数据作为训练集，20%的数据作为测试集

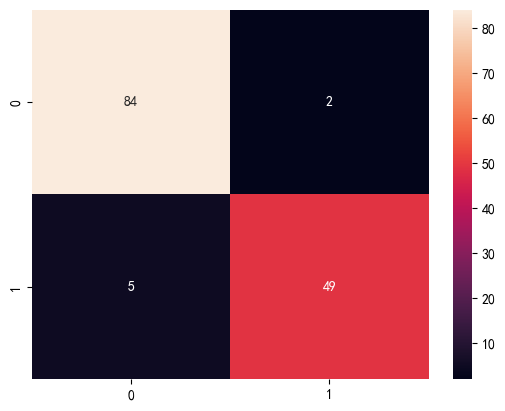
# 数据处理  
path = 'dataset/breast-cancer-wisconsin.data'\  
# 全部特征  
def data\_process\_full():  
 # 将data加入第一行列表头'Sample code number', 'Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of Cell Shape', 'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromatin', 'Normal Nucleoli', 'Mitoses', 'Class'  
 data = pd.read\_csv(path, header=None, names= ['Sample code number', 'Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of Cell Shape', 'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromatin', 'Normal Nucleoli', 'Mitoses', 'Class'])  
 # 去掉'Sample code number'列  
 data = data.drop('Sample code number', axis=1)  
 # 0填充'?'  
 data = data.replace('?', 0)  
 # 将所有数据转换为int类型  
 data = data.astype(int)  
 # Z-Score标准化  
 for col in data.columns:  
 data[col] = (data[col] - data[col].mean()) / data[col].std()  
 # 分割数据集  
 data = np.array(data)  
 np.random.shuffle(data)  
 X = data[:, 1:-1]  
 y = data[:, -1]  
 X\_train = X[:int(0.8 \* len(X))]  
 y\_train = y[:int(0.8 \* len(y))]  
 X\_test = X[int(0.8 \* len(X)):]  
 y\_test = y[int(0.8 \* len(y)):]  
 return X\_train, y\_train, X\_test, y\_test  
  
# 部分特征  
def data\_process\_part():  
 # 将data加入第一行列表头'Sample code number', 'Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of Cell Shape', 'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromatin', 'Normal Nucleoli', 'Mitoses', 'Class'  
 data = pd.read\_csv(path, header=None, names= ['Sample code number', 'Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of Cell Shape', 'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromatin', 'Normal Nucleoli', 'Mitoses', 'Class'])  
 # 去掉'Sample code number'列  
 data = data.drop('Sample code number', axis=1)  
 # 0填充'?'  
 data = data.replace('?', 0)  
 # 将所有数据转换为int类型  
 data = data.astype(int)  
 # Z-Score标准化  
 for col in data.columns:  
 data[col] = (data[col] - data[col].mean()) / data[col].std()  
 # 分割数据集  
 data = np.array(data)  
 np.random.shuffle(data)  
 X = data[:, [1, 2, 3, 4, 5]]  
 y = data[:, -1]  
 X\_train = X[:int(0.8 \* len(X))]  
 y\_train = y[:int(0.8 \* len(y))]  
 X\_test = X[int(0.8 \* len(X)):]  
 y\_test = y[int(0.8 \* len(y)):]  
 return X\_train, y\_train, X\_test, y\_test

## 主函数

import seaborn as sns  
# 全部特征  
X\_train, y\_train, X\_test, y\_test = data\_process\_full()  
# y=2 为benign, y=4 为malignant  
# 实例化模型  
model = LogisticRegression()  
# 训练模型  
model.fit(X\_train, y\_train, lr=0.01, max\_iter=1000)  
# 保存模型  
model.save('model/logistic\_model\_implement\_full.npz')  
# 评估模型  
y\_pred = model.predict(X\_test)  
# y\_pred转化为0,1  
y\_pred = np.where(y\_pred > 0.5, 1, 0)  
# 将y\_test转化为0,1  
y\_test = np.where(y\_test < 0, 0, 1)  
# print(y\_pred, y\_test)  
print('全部特征accuracy:', model.accuracy(y\_test, y\_pred))  
print('全部特征recall:', model.recall(y\_test, y\_pred))  
  
# 设置显示中文字体  
from pylab import mpl  
mpl.rcParams["font.sans-serif"] = ["SimHei"]  
# 混淆矩阵  
from sklearn.metrics import confusion\_matrix  
cm = confusion\_matrix(y\_test, y\_pred)  
sns.heatmap(cm, annot=True, fmt='d')  
plt.show()  
  
# 部分特征  
X\_train, y\_train, X\_test, y\_test = data\_process\_part()  
# y=2 为benign, y=4 为malignant  
# 实例化模型  
model = LogisticRegression()  
# 训练模型  
model.fit(X\_train, y\_train, lr=0.01, max\_iter=1000)  
# 保存模型  
model.save('model/logistic\_model\_implement\_part.npz')  
# 评估模型  
y\_pred = model.predict(X\_test)  
# y\_pred转化为0,1  
y\_pred = np.where(y\_pred > 0.5, 1, 0)  
# 将y\_test转化为0,1  
y\_test = np.where(y\_test < 0, 0, 1)  
# print(y\_pred, y\_test)  
print('部分特征accuracy:', model.accuracy(y\_test, y\_pred))  
print('部分特征recall:', model.recall(y\_test, y\_pred))  
  
# 混淆矩阵  
from sklearn.metrics import confusion\_matrix  
cm = confusion\_matrix(y\_test, y\_pred)  
sns.heatmap(cm, annot=True, fmt='d')  
plt.show()

C:\Users\13107\AppData\Local\Temp\ipykernel\_19108\2051046317.py:9: RuntimeWarning: overflow encountered in exp  
 return 1 / (1 + np.exp(-x))

全部特征accuracy: 0.95  
全部特征recall: 0.9074074074074074



部分特征accuracy: 0.9285714285714286  
部分特征recall: 0.8958333333333334

C:\Users\13107\AppData\Local\Temp\ipykernel\_19108\2051046317.py:9: RuntimeWarning: overflow encountered in exp  
 return 1 / (1 + np.exp(-x))

