

## 做题笔记

写在前面

### ► 囧 写在前面

---

本项目已经开源在github仓库：

```
https://github.com/wwwwwwwq/lianshidai-ml-3.git
```

文件夹下各个.py文件说明：

`base.py`：基础部分（没有对数转换和R方检测，不过包括了图表的绘画），默认是进行`sklearn`模型训练

`base_manual.py`：手动实现最小二乘法实现线性回归（Build it from scratch应该是这个意思吧qwq）

`base_sklearn.py`：使用`sklearn.linear_model.LinearRegression`来训练模型

`extend.py`：拓展部分，增加了R方检测和对数转换，默认是进行`sklearn`模型训练

`extend_manual.py`：手动实现最小二乘法实现线性回归

`extend_sklearn.py`：使用`sklearn.linear_model.LinearRegression`来训练模型

运行说明：

```
python [base.py/extend.py] [1/2]  
#[1/2]是指，如果输入1，则是sklearn，如果输入2，则是manual
```

如果在文本编辑器里面运行，那么对应功能如上 .py文件说明 所述

需要的包：

```
pandas, numpy, matplotlib, Scikit-Learn
```

注：在此只对基础部分做讲解，拓展部分只给出完整代码

题意理解

### ► ⚡ 题意理解

---

大概就是说，从NASA 系外行星数据库的数据（已放在`exoplanet_data.csv`中）读取轨道半长轴(**pl\_orbsmax**) 和母恒星质量(**st\_mass**)，这两个变量作为输入特征，轨道周期(**pl\_orbper**) 作为目标变量，来构建线性回归模

型。并且要用均方误差(**MSE**)来评估模型的性能。然后的话还有一个手动使用最小二乘法实现线性回归的任务

## 输入输出

### ▶ ⚙ 输入

---

**NASA 系外行星数据库的csv文件(exoplanet\_data.csv)**

### ▶ 📈 输出

---

可视化散点图，预测理想线，解释模型的误差

## 解决题目

### ▶ 📁 读入数据

---

由于接受的是.csv文件的数据，这里为了方便就直接用pandas的read\_csv来读入数据了。不过要注意的是下载下来的exoplanet\_data.csv文件中的内容是以#开头的：

```
# This file was produced by the NASA Exoplanet Archive
http://exoplanetarchive.ipac.caltech.edu
# Wed Feb 19 21:07:49 2025
#
# COLUMN pl_name:          Planet Name
# COLUMN hostname:        Host Name
# COLUMN sy_snum:          Number of Stars
# COLUMN sy_pnum:          Number of Planets
#...
```

所以说我们需要跳过这一部分，就在read\_csv中添加一个 '**comment='#'**' 的参数：

```
#从exoplanet_data.csv文件(当前目录)中读取数据
#由于读取的是DataFrame类型，所以这里用df缩写表示读取的数据
#用pandas库来读取csv文件(这里简写为pd)
df=pd.read_csv('exoplanet_data.csv',comment='#')
```

然后就读取输入变量和目标变量：

```
#自变量(输入变量)x·其值为轨道半长轴a和恒星指令m
x=df[['pl_orbsmax','st_mass']]

#因变量(输出变量)y·其值为轨道周期
y=df['pl_orbper']
```

接下来要处理缺失值（NaN或者null），因为线性回归算法不能处理缺失值。具体的例子和用到的函数可以详见代码的注释：

```
#删除缺失值。
```

```
#因为线性回归算法不能处理缺失值（NaN和null），所以要手动处理缺失值，防止数据中有缺失值出现。
```

```
data=pd.concat([x,y],axis=1)#pandas的连接函数，axis=1表示水平连接，所以这里将x和y水平连接起来
```

```
#举个例子，比如：
```

```
"""
```

```
原始数据：
```

```
x:
```

```
pl_orbsmax st_mass
0 1.0 2.0
1 2.0 NaN#缺失值
2 3.0 3.0
3 4.0 4.0
```

```
y:
```

```
10.0
20.0
NaN#缺失值
40.0
```

```
连接后：
```

```
pl_orbsmax st_mass pl_orbper
0 1.0 2.0 10.0
1 2.0 NaN 20.0 # 有缺失值
2 3.0 3.0 NaN # 有缺失值
3 4.0 4.0 40.0
"""
```

```
data=data.dropna()#删除包含缺失值的行
```

```
#删除缺失值后，数据如下：
```

```
"""
```

```
pl_orbsmax st_mass pl_orbper
0 1.0 2.0 10.0
3 4.0 4.0 40.0
"""
```

```
#接下来就是从删除后的数据中提取自变量x和因变量y
```

```
x=data[['pl_orbsmax','st_mass']]
y=data['pl_orbper']
```

```
#那么处理后x和y就变成了：
```

```
"""
```

```
x:
```

```
pl_orbsmax st_mass
0 1.0 2.0
3 4.0 4.0
```

```

y:
0    10.0
3    40.0
...

```

所以这样了之后，我们就可以写出加载和预处理函数 `load_process_data` 了：

```

#加载和预处理数据函数
def load_process_data():
    #从exoplanet_data.csv文件（当前目录）中读取数据
    #由于读取的是DataFrame类型，所以这里用df缩写表示读取的数据
    #用pandas库来读取csv文件（这里简写为pd）
    df=pd.read_csv('exoplanet_data.csv',comment='#')

    #自变量（输入变量）x，其值为轨道半长轴a和恒星指令m
    x=df[['pl_orbsmax','st_mass']]

    #因变量（输出变量）y，其值为轨道周期
    y=df['pl_orbper']

    #删除缺失值。
    #因为线性回归算法不能处理缺失值（NaN和null），所以要手动处理缺失值，防止数据中有缺失值出现。

    data=pd.concat([x,y],axis=1)#pandas的连接函数，axis=1表示水平连接，所以这里将x和y
    水平连接起来
    #举个例子，比如：
    """
    原始数据：
    x:
        pl_orbsmax  st_mass
    0      1.0      2.0
    1      2.0      NaN#缺失值
    2      3.0      3.0
    3      4.0      4.0

    y:
    0    10.0
    1    20.0
    2    NaN#缺失值
    3    40.0

    连接后：
    pl_orbsmax  st_mass  pl_orbper
    0      1.0      2.0      10.0
    1      2.0      NaN      20.0      # 有缺失值
    2      3.0      3.0      NaN      # 有缺失值
    3      4.0      4.0      40.0
    ...
    """

    data=data.dropna()#删除包含缺失值的行
    #删除缺失值后，数据如下：

```

```

"""
pl_orbsmax  st_mass  pl_orbper
0      1.0        2.0      10.0
3      4.0        4.0      40.0
"""

#接下来就是从删除后的数据中提取自变量x和因变量y
x=data[['pl_orbsmax','st_mass']]
y=data['pl_orbper']

#那么处理后x和y就变成了：
"""
x:
pl_orbsmax  st_mass
0      1.0        2.0
3      4.0        4.0

y:
0      10.0
3      40.0
"""

#返回处理后的自变量x和因变量y
return x,y

```

## ▶ 用 sklearn 训练模型

得到载入和预处理后的变量以后，就可以训练模型了。那么这里先用**sklearn**训练模型。首先利用**train\_test\_split** 函数来分离测试数据和训练数据：

```

#将数据集划分为训练集和测试集

#test_size表示有多少的数据用来测试，random_state=42表示随机种子为42。随机种子的设置
是为了每次运行代码时，划分结果都相同。
#这里选择随机种子为42是因为42是宇宙的终极奥秘（）。

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=test_size,random_stat
e=42)

```

接着，利用LinearRegression类来训练模型：

```

#创建并训练线性回归模型

#这里利用scikit-learn库中的LinearRegression类来创建线性回归模型，其中model就是
LinearRegression类的一个实例。
#LinearRegression类是scikit-learn库中的一个线性回归类，用于构建类似于
y=ax_{1}+bx_{2}+c的方程。其中y代表我们要预测的轨道周期，x_{1}代表轨道半长轴，x_{2}代表
恒星质量。
model=LinearRegression()

```

```
#利用LinearRegression类中的fit方法训练模型  
model.fit(x_train,y_train)
```

然后训练完了之后就拿到预测值和mse：

```
#用LinearRegression类中的predict方法进行预测  
y_pred=model.predict(x_test)  
  
#利用mean_squared_error函数计算均方误差  
mse=mean_squared_error(y_test,y_pred)
```

做完这些之后，我们就可以得到训练和测试模型的函数**train\_test\_model**了：

```
#训练和测试模型函数  
def train_test_model(x,y,test_size=0.7):  
    #将数据集划分为训练集和测试集  
  
    #test_size表示有多少的数据用来测试，random_state=42表示随机种子为42。随机种子的设置  
    #是为了每次运行代码时，划分结果都相同。  
    #这里选择随机种子为42是因为42是宇宙的终极奥秘（）。  
  
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=test_size,random_stat  
    e=42)  
  
    #创建并训练线性回归模型  
  
    #这里利用scikit-learn库中的LinearRegression类来创建线性回归模型，其中model就是  
    #LinearRegression类的一个实例。  
    #LinearRegression类是scikit-learn库中的一个线性回归类，用于构建类似于  
    # $y=ax_1+bx_2+c$ 的方程。其中y代表我们要预测的轨道周期， $x_1$ 代表轨道半长轴， $x_2$ 代表  
    #恒星质量。  
    model=LinearRegression()  
  
    #利用LinearRegression类中的fit方法训练模型  
    model.fit(x_train,y_train)  
  
    #用LinearRegression类中的predict方法进行预测  
    y_pred=model.predict(x_test)  
  
    #利用mean_squared_error函数计算均方误差  
    mse=mean_squared_error(y_test,y_pred)  
  
    #返回model,mse,y_test,y_pred  
    return model,mse,y_test,y_pred
```

## ► 🔍 手动使用最小二乘法来完成线性回归任务

由于题目还说要手动使用最小二乘法来完成线性回归任务，所以我们还需要用程序来实现最小二乘法。

- **最小二乘法**

假设线性回归方程为：

$\$ y=X^{\backslash\beta} + \backslash\varepsilon \$$  我们知道如果要衡量线性回归模型是否有效，可以利用残差平方和来衡量：

$\$ I=\sum_{i=1}^n \backslash\varepsilon_i^2=(y-X^{\backslash\beta})^T(y-X^{\backslash\beta})=(y^T-\backslash\beta^T X^T)(y-X^{\backslash\beta})=y^T y-y^T X^{\backslash\beta}-\backslash\beta^T X y+\backslash\beta^T X^T X^{\backslash\beta}$

$\$ \$ \$$  而由于 $\backslash\beta^T X^T Y=(X^{\backslash\beta})^T Y$ 中 $(X^{\backslash\beta})^T$ 是行向量， $y$ 是列向量，所以 $\backslash\beta^T X^T Y=(\backslash\beta^T X^T Y)^T=y^T X^{\backslash\beta}$

$\$ \$ \$$  所以我们有： $I=y^T y-2\backslash\beta^T X y+\backslash\beta^T X^T X^{\backslash\beta}$   $\$ \$ \$$  而我们要使得残差平方和最小，可以对其进行求导，然后得到使得其最小的解：

$\$ \frac{\partial I}{\partial \beta}=-2X^T y+2X^T X^{\backslash\beta}$   $\$ \$ \$$  令其等于0： $\$ \$ \$$

$-2X^T y+2X^T X^{\backslash\beta}=0$   $\$ \$ \$$  可以得到： $\backslash\beta=(X^T X)^{-1} X^T y$   $\$ \$ \$$  于是这就是最终的解析解。那么接下来我们就用程序实现就好了。以下是手动使用最小二乘法完成线性回归任务的函数：

```
#手动实现最小二乘法实现线性回归 (Build it from scratch应该是这个意思吧( ))
def manual_linear_regression(x,y,test_size=0.7):
    #将数据集划分为训练集和测试集
    x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=test_size, random_state=42)

    #添加偏置项(也就是截距项)，把矩阵x的最左边那一行全部变成1，以便于出现常数项
    #比如：
    """
    原始数据：
    x:
    |2 3|
    |4 5|
    样本：
    x1=2,x2=3
    x1=4,x2=5

    添加偏置项后：
    x:
    |1 2 3|
    |1 4 5|
    就会是：
    result:
    |1xβ1 2xβ1 3xβ2|
    |1xβ1 4xβ1 5xβ2|
    """
    #添加偏置项
    x_train_b=np.c_[np.ones(x_train.shape[0]),x_train]

    #计算最小二乘法的解析解：β=(X^T*X)^-1*X^T*y

    results=np.linalg.inv(x_train_b.T.dot(x_train_b)).dot(x_train_b.T).dot(y_train)
```

```

#计算测试集的预测值
x_test_b=np.c_[np.ones(x_test.shape[0]),x_test]
y_pred=x_test_b.dot(results)

#计算均方误差
mse=mean_squared_error(y_test,y_pred)

#返回结果
#results[0]是截距项，results[1:]是系数项
return results[0],results[1:],y_test,y_pred,mse

```

## ▶ 可视化结果

---

当处理完数据，训练好模型以后，这里我用**matplotlib**库来可视化结果。

那么为了更好的可视化，这里我用了一个**滑动条slider**来控制test\_size，看看不同的测试数据占比会导致结果怎么变化。不过要注意的是更新图表，所以需要**slider**订阅一个**update**函数。由于在代码注释里已经很详细了，所以这里直接给出**visualize\_results**的完整代码：

```

#利用plt可视化预测结果
def visualize_results(x,y,mode=1):
    #创建图形窗口，并将大小设置为10x6英寸
    fig=plt.figure(figsize=(10,6))

    #创建GridSpec对象，用于控制子图的布局
    #2,1表示创建一个2行1列的子图
    #height_ratios=[1,4]表示第一行的高度为1，第二行的高度为4
    #hspace=0.2表示子图之间的垂直间距为0.2
    gs=fig.add_gridspec(2,1,height_ratios=[1,4],hspace=0.2)

    #在网格的第一行创建一个子图，用于显示线性回归方程，评估指标
    #gs[0]表示网格的第一行
    ax_text=fig.add_subplot(gs[0])

    #在网格的第二行创建一个散点图子图
    #gs[1]表示网格的第二行
    ax_scatter=fig.add_subplot(gs[1])

    #调整子图布局，在底部留出0.2的空白区域放置滑动条
    plt.subplots_adjust(bottom=0.2)

    if mode==1:
        #获取train_test_model的返回值
        model,mse,y_test,y_pred=train_test_model(x,y,test_size=0.7)

        #获取线性回归方程的文本
        equ=f'pl_order = {model.coef_[0]:.3f} * pl_orbsmax + {model.coef_[1]:.3f} * st_mass + {model.intercept_:.3f}'
    elif mode==2:
        #获取manual_linear_regression的返回值

    intercept,coef,y_test,y_pred,mse=manual_linear_regression(x,y,test_size=0.7)

```

```
#获取线性回归方程的文本
equ=f'pl_order = {coef[0]:.3f} * pl_orbsmax + {coef[1]:.3f} * st_mass +
{intercept:.3f}'

#获取MSE文本
mse_text=f'均方误差 ( $MSE$ ) : {mse:.3f}'

#plt.scatter()来绘制散点图，其中第一个参数是x轴（表示真实值），第二个参数是y轴（表示预测值），第三个参数是透明度
scatter=ax_scatter.scatter(y_test,y_pred,alpha=0.5,label='pred vs. true')

#获取坐标轴的最小值和最大值
ax_min=min(y_test.min(),y_pred.min())
ax_max=max(y_test.max(),y_pred.max())

#绘制理想的预测线（ideal），也就是对角线
#[y_test.min(),y_test.max()]:x轴的起点和终点
#[y_test.min(),y_test.max()]:y轴的起点和终点
#'r--':红色虚线（r代表red红色，--代表虚线）
#lw=2：线宽为2
line,=ax_scatter.plot([ax_min,ax_max],[ax_min,ax_max], 'r--',
lw=2,label='ideal')

#设置散点图的坐标轴范围，使之变成一个正方形
ax_scatter.set_xlim(ax_min,ax_max)
ax_scatter.set_ylim(ax_min,ax_max)
#设置散点图的坐标轴比例
ax_scatter.set_aspect('equal',adjustable='box')

#创建一个滑动条，用于控制测试集的比例test_size
#0.2：滑动条的x轴
#0.1：滑动条的y轴
#0.6：滑动条的宽度
#0.03：滑动条的高度
ax_slider=plt.axes([0.2,0.1,0.6,0.03])
#创建一个调节范围为0-1的滑动条，并且初始值为0.7
slider=Slider(ax_slider,'Test Size',0,1,valinit=0.7)

#文本信息关闭坐标轴
ax_text.axis('off')

#添加到ax_text中
#0.05代表文本框左上角的x坐标
#0.5代表文本框左上角的y坐标
#transform指定坐标系是transAxes(相对坐标系)，坐标范围为[0,1]
#verticalalignment='center'表示文本垂直居中于(0.05,0.5)
#bbox=dict()设置文本框的背景和边框样式，boxstyle='round'文本框为圆角矩形
facecolor='white'背景颜色为白色, alpha=0.8透明度为0.8
ax_text.text(0.05,0.5,f'线性回归方程 : \n{equ}\n\n评估指标 :
\n{n{mse_text}}',transform=ax_text.transAxes,verticalalignment='center',bbox=dict(box
style='round', facecolor='white', alpha=0.8))

#利用滑动条中的值来更新test_size
```

```
def update(val):
    #获取滑动条中的值并进行赋值
    test_size=slider.val
    if mode==1:
        #重新利用test_size获取测试数据
        model,mse,y_test,y_pred=train_test_model(x,y,test_size=test_size)
        #获取线性回归方程的文本
        equ=f'pl_order = {model.coef_[0]:.3f} * pl_orbsmax +
{model.coef_[1]:.3f} * st_mass + {model.intercept_:.3f}'
    elif mode==2:
        #重新利用test_size获取测试数据

intercept,coef,y_test,y_pred,mse=mannual_linear_regression(x,y,test_size=test_size)
)                                #获取线性回归方程的文本
equ=f'pl_order = {coef[0]:.3f} * pl_orbsmax + {coef[1]:.3f} * st_mass
+ {intercept:.3f}'

#获取MSE文本
mse_text=f'均方误差 ( $MSE$ ) : {mse:.3f}'

#更新散点图
scatter.set_offsets(np.c_[y_test,y_pred])

#更新文本注释
#先清理原本的数据
ax_text.clear()
#关闭坐标轴
ax_text.axis('off')
#添加新的文本注释
ax_text.text(0.05,0.5,f'线性回归方程：\n{equ}\n评估指标：
\n{mse_text}',transform=ax_text.transAxes,verticalalignment='center',bbox=dict(box
style='round',facecolor='white',alpha=0.8))

#更新坐标轴的最大值和最小值
ax_min=min(y_test.min(),y_pred.min())
ax_max=max(y_test.max(),y_pred.max())

#更新理想预测线
line.set_data([ax_min,ax_max],[ax_min,ax_max])

#更新坐标轴的范围
ax_scatter.set_xlim(ax_min,ax_max)
ax_scatter.set_ylim(ax_min,ax_max)
#更新坐标轴的比例
ax_scatter.set_aspect('equal',adjustable='box')

#更新图表，这里利用draw_idle()不用draw()是因为draw_idle()更适合交互式的东西比
如滑动条
fig.canvas.draw_idle()

#利用滑动条的on_changed方法来更新图表
slider.on_changed(update)
```

```
#在左上角添加图例  
ax_scatter.legend(loc='upper left')  
  
#设置x轴标签 ( 示意图中给的是T_true)  
ax_scatter.set_xlabel('T_true')  
  
#设置y轴标签 ( 示意图中给的是T_pred)  
ax_scatter.set_ylabel('T_pred')  
  
#设置图表的标题 ( 示意图中给的是true vs. pred T)  
ax_scatter.set_title('true vs. pred T')  
  
ax_scatter.grid(True)  
  
#显示图表  
plt.show()
```

## 完整代码

### ► 基础部分

```
# -*- coding: utf-8 -*-  
import sys  
import pandas as pd  
import numpy as np  
from sklearn.model_selection import train_test_split  
from sklearn.linear_model import LinearRegression  
from sklearn.metrics import mean_squared_error  
from sklearn.preprocessing import StandardScaler  
from matplotlib.widgets import Slider  
import matplotlib.pyplot as plt  
  
# 设置matplotlib支持中文显示  
plt.rcParams['font.sans-serif']=['SimHei']#用来正常显示中文标签  
plt.rcParams['axes.unicode_minus']=False#用来正常显示负号  
plt.rcParams['mathtext.fontset'] = 'cm'#设置 LaTeX 字体  
  
#加载和预处理数据函数  
def load_process_data():  
    #从exoplanet_data.csv文件 ( 当前目录 ) 中读取数据  
    #由于读取的是DataFrame类型 . 所以这里用df缩写表示读取的数据  
    #用pandas库来读取csv文件 ( 这里简写为pd )  
    df=pd.read_csv('exoplanet_data.csv',comment='#')  
  
    #自变量 ( 输入变量 ) x · 其值为轨道半长轴a和恒星指令m  
    x=df[['pl_orbsmax','st_mass']]  
  
    #因变量 ( 输出变量 ) y · 其值为轨道周期  
    y=df['pl_orbper']  
  
    #删除缺失值 。
```

#因为线性回归算法不能处理缺失值 ( NaN和null ) , 所以要手动处理缺失值 , 防止数据中有缺失值出现。

```
data=pd.concat([x,y],axis=1)#pandas的连接函数·axis=1表示水平连接·所以这里将x和y水平连接起来
```

```
#举个例子·比如:
```

```
"""
```

```
原始数据:
```

```
x:
```

	pl_orbsmax	st_mass
0	1.0	2.0
1	2.0	NaN#缺失值
2	3.0	3.0
3	4.0	4.0

```
y:
```

0	10.0
1	20.0
2	NaN#缺失值
3	40.0

```
连接后:
```

	pl_orbsmax	st_mass	pl_orbper
0	1.0	2.0	10.0
1	2.0	NaN	20.0 # 有缺失值
2	3.0	3.0	NaN # 有缺失值
3	4.0	4.0	40.0

```
"""
```

```
data=data.dropna()#删除包含缺失值的行
```

```
#删除缺失值后·数据如下:
```

```
"""
```

	pl_orbsmax	st_mass	pl_orbper
0	1.0	2.0	10.0
3	4.0	4.0	40.0

```
"""
```

```
#接下来就是从删除后的数据中提取自变量x和因变量y
```

```
x=data[['pl_orbsmax','st_mass']]
```

```
y=data['pl_orbper']
```

```
#那么处理后x和y就变成了:
```

```
"""
```

```
x:
```

	pl_orbsmax	st_mass
0	1.0	2.0
3	4.0	4.0

```
y:
```

0	10.0
3	40.0

```
"""
```

```
#返回处理后的自变量x和因变量y
```

```
return x,y

#训练和测试模型函数
def train_test_model(x,y,test_size=0.7):
    #将数据集划分为训练集和测试集

    #test_size表示有多少的数据用来测试，random_state=42表示随机种子为42。随机种子的设置是为了每次运行代码时，划分结果都相同。
    #这里选择随机种子为42是因为42是宇宙的终极奥秘( )。

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=test_size,random_stat
e=42)

#创建并训练线性回归模型

#这里利用scikit-learn库中的LinearRegression类来创建线性回归模型，其中model就是
LinearRegression类的一个实例。
#LinearRegression类是scikit-learn库中的一个线性回归类，用于构建类似于
y=ax_{1}+bx_{2}+c的方程。其中y代表我们要预测的轨道周期，x_{1}代表轨道半长轴，x_{2}代表
恒星质量。
model=LinearRegression()

#利用LinearRegression类中的fit方法训练模型
model.fit(x_train,y_train)

#用LinearRegression类中的predict方法进行预测
y_pred=model.predict(x_test)

#利用mean_squared_error函数计算均方误差
mse=mean_squared_error(y_test,y_pred)

#返回model,mse,y_test,y_pred
return model,mse,y_test,y_pred

#手动实现最小二乘法实现线性回归（Build it from scratch应该是这个意思吧( )）
def manunal_linear_regression(x,y,test_size=0.7):
    #将数据集划分为训练集和测试集
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=test_size,
random_state=42)

    #添加偏置项（也就是截距项），把矩阵x的最左边那一行全部变成1，以便于出现常数项
    #比如：
    """
    原始数据：
    x:
    |2 3|
    |4 5|
    样本：
    x1=2,x2=3
    x1=4,x2=5

    添加偏置项后：
    x:
    |1 2 3|
```

```
|1 4 5|
```

就会是：

```
result:
```

```
|1xc 2xβ1 3xβ2|
|1xc 4xβ1 5xβ2|
***
```

#添加偏置项

```
x_train_b=np.c_[np.ones(x_train.shape[0]),x_train]
```

#计算最小二乘法的解析解： $\beta = (X^T \cdot X)^{-1} \cdot X^T \cdot y$

```
results=np.linalg.inv(x_train_b.T.dot(x_train_b)).dot(x_train_b.T).dot(y_train)
```

#计算测试集的预测值

```
x_test_b=np.c_[np.ones(x_test.shape[0]),x_test]
y_pred=x_test_b.dot(results)
```

#计算均方误差

```
mse=mean_squared_error(y_test,y_pred)
```

#返回结果

#results[0]是截距项，results[1:]是系数项

```
return results[0],results[1:],y_test,y_pred,mse
```

#利用plt可视化预测结果

```
def visualize_results(x,y,mode=1):
```

#创建图形窗口，并将大小设置为10x6英寸

```
fig=plt.figure(figsize=(10,6))
```

#创建GridSpec对象，用于控制子图的布局

#2,1表示创建一个2行1列的子图

#height\_ratios=[1,4]表示第一行的高度为1，第二行的高度为4

#hspace=0.2表示子图之间的垂直间距为0.2

```
gs=fig.add_gridspec(2,1,height_ratios=[1,4],hspace=0.2)
```

#在网格的第一行创建一个子图，用于显示线性回归方程，评估指标

#gs[0]表示网格的第一行

```
ax_text=fig.add_subplot(gs[0])
```

#在网格的第二行创建一个散点图子图

#gs[1]表示网格的第二行

```
ax_scatter=fig.add_subplot(gs[1])
```

#调整子图布局，在底部留出0.2的空白区域放置滑动条

```
plt.subplots_adjust(bottom=0.2)
```

```
if mode==1:
```

#获取train\_test\_model的返回值

```
model,mse,y_test,y_pred=train_test_model(x,y,test_size=0.7)
```

#获取线性回归方程的文本

```
equ=f'pl_order = {model.coef_[0]:.3f} * pl_orbsmax + {model.coef_[1]:.3f}
```

```
* st_mass + {model.intercept_:.3f}'
```

```
elif mode==2:  
    #获取mannual_linear_regression的返回值  
  
    intercept,coef,y_test,y_pred,mse=mannual_linear_regression(x,y,test_size=0.7)  
  
    #获取线性回归方程的文本  
    equ=f'pl_order = {coef[0]:.3f} * pl_orbsmax + {coef[1]:.3f} * st_mass +  
    {intercept:.3f}'  
  
    #获取MSE文本  
    mse_text=f'均方误差 ($MSE$) : {mse:.3f}'  
  
    #plt.scatter()来绘制散点图·其中第一个参数是x轴(表示真实值)·第二个参数是y轴(表示  
    预测值)·第三个参数是透明度  
    scatter=ax_scatter.scatter(y_test,y_pred,alpha=0.5,label='pred vs. true')  
  
    #获取坐标轴的最小值和最大值  
    ax_min=min(y_test.min(),y_pred.min())  
    ax_max=max(y_test.max(),y_pred.max())  
  
    #绘制理想的预测线(ideal)·也就是对角线  
    #[y_test.min(),y_test.max()]:x轴的起点和终点  
    #[y_test.min(),y_test.max()]:y轴的起点和终点  
    #'r--':红色虚线(r代表red红色·--代表虚线)  
    #lw=2:线宽为2  
    line_=ax_scatter.plot([ax_min,ax_max],[ax_min,ax_max], 'r--  
' ,lw=2,label='ideal')  
  
    #设置散点图的坐标轴范围·使之变成一个正方形  
    ax_scatter.set_xlim(ax_min,ax_max)  
    ax_scatter.set_ylim(ax_min,ax_max)  
    #设置散点图的坐标轴比例  
    ax_scatter.set_aspect('equal',adjustable='box')  
  
    #创建一个滑动条·用于控制测试集的比例test_size  
    #0.2:滑动条的x轴  
    #0.1:滑动条的y轴  
    #0.6:滑动条的宽度  
    #0.03:滑动条的高度  
    ax_slider=plt.axes([0.2,0.1,0.6,0.03])  
    #创建一个调节范围为0-1的滑动条·并且初始值为0.7  
    slider=Slider(ax_slider,'Test Size',0,1, valinit=0.7)  
  
    #文本信息关闭坐标轴  
    ax_text.axis('off')  
  
    #添加到ax_text中  
    #0.05代表文本框左上角的x坐标  
    #0.5代表文本框左上角的y坐标  
    #transform指定坐标系是transAxes(相对坐标系)·坐标范围为[0,1]  
    #verticalalignment='center'表示文本垂直居中于(0.05,0.5)  
    #bbox=dict()设置文本框的背景和边框样式·boxstyle='round'文本框为圆角矩形·  
    facecolor='white'背景颜色为白色, alpha=0.8透明度为0.8  
    ax_text.text(0.05,0.5,f'线性回归方程 : \n{equ}\n\n评估指标 :
```

```
\n{mse_text}', transform=ax_text.transAxes, verticalalignment='center', bbox=dict(box
style='round', facecolor='white', alpha=0.8))

#利用滑动条中的值来更新test_size
def update(val):
    #获取滑动条中的值并进行赋值
    test_size=slider.val
    if mode==1:
        #重新利用test_size获取测试数据
        model,mse,y_test,y_pred=train_test_model(x,y,test_size=test_size)
        #获取线性回归方程的文本
        equ=f'pl_order = {model.coef_[0]:.3f} * pl_orbsmax +
{model.coef_[1]:.3f} * st_mass + {model.intercept_:.3f}'
    elif mode==2:
        #重新利用test_size获取测试数据

intercept,coef,y_test,y_pred,mse=mannual_linear_regression(x,y,test_size=test_size
)
    #获取线性回归方程的文本
    equ=f'pl_order = {coef[0]:.3f} * pl_orbsmax + {coef[1]:.3f} * st_mass
+ {intercept:.3f}'

    #获取MSE文本
    mse_text=f'均方误差 ( $MSE$ ) : {mse:.3f}'

    #更新散点图
    scatter.set_offsets(np.c_[y_test,y_pred])

    #更新文本注释
    #先清理原本的数据
    ax_text.clear()
    #关闭坐标轴
    ax_text.axis('off')
    #添加新的文本注释
    ax_text.text(0.05,0.5,f'线性回归方程： \n{equ}\n\n评估指标：
\n{mse_text}', transform=ax_text.transAxes, verticalalignment='center', bbox=dict(box
style='round', facecolor='white', alpha=0.8))

    #更新坐标轴的最大值和最小值
    ax_min=min(y_test.min(),y_pred.min())
    ax_max=max(y_test.max(),y_pred.max())

    #更新理想预测线
    line.set_data([ax_min,ax_max],[ax_min,ax_max])

    #更新坐标轴的范围
    ax_scatter.set_xlim(ax_min,ax_max)
    ax_scatter.set_ylim(ax_min,ax_max)
    #更新坐标轴的比例
    ax_scatter.set_aspect('equal',adjustable='box')

    #更新图表 · 这里利用draw_idle()不用draw()是因为draw_idle()更适合交互式的东西比
如滑动条
    fig.canvas.draw_idle()
```

```

#利用滑动条的on_changed方法来更新图表
slider.on_changed(update)

#在左上角添加图例
ax_scatter.legend(loc='upper left')

#设置x轴标签 ( 示意图中给的是T_true)
ax_scatter.set_xlabel('T_true')

#设置y轴标签 ( 示意图中给的是T_pred)
ax_scatter.set_ylabel('T_pred')

#设置图表的标题 ( 示意图中给的是true vs. pred T)
ax_scatter.set_title('true vs. pred T')

ax_scatter.grid(True)

#显示图表
plt.show()

#主函数
def main(mode=1):
    #加载和预处理数据
    x,y=load_process_data()

    #可视化预测结果
    visualize_results(x,y,mode)

#如果当前文件是主程序，而不是被用作模块，则执行main函数
if __name__=='__main__':
    if len(sys.argv)>=2:
        main(int(sys.argv[1]))

```

## ► 拓展部分

---

```

# -*- coding: utf-8 -*-
import sys
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from matplotlib.widgets import Slider
import matplotlib.pyplot as plt

#设置matplotlib支持中文显示
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签

```

```
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号
plt.rcParams['mathtext.fontset']='cm'#设置 LaTeX 字体

#加载和预处理数据函数
def load_process_data():
    #从exoplanet_data.csv文件(当前目录)中读取数据
    #由于读取的是DataFrame类型，所以这里用df缩写表示读取的数据
    #用pandas库来读取csv文件(这里简写为pd)
    df=pd.read_csv('exoplanet_data.csv',comment='#')

    #自变量(输入变量)x·其值为轨道半长轴a和恒星质量m
    x=df[['pl_orbsmax','st_mass']]

    #因变量(输出变量)y·其值为轨道周期
    y=df['pl_orbper']

    #删除缺失值。
    #因为线性回归算法不能处理缺失值(NaN和null)，所以要手动处理缺失值，防止数据中有缺失值出现。

    data=pd.concat([x,y],axis=1) #pandas的连接函数·axis=1表示水平连接·所以这里将x和y水平连接起来
    #举个例子·比如：
    """
    原始数据：
    x:
        pl_orbsmax  st_mass
    0      1.0        2.0
    1      2.0        NaN # 缺失值
    2      3.0        3.0
    3      4.0        4.0

    y:
    0    10.0
    1    20.0
    2    NaN # 缺失值
    3    40.0

    连接后：
    pl_orbsmax  st_mass  pl_orbper
    0      1.0        2.0        10.0
    1      2.0        NaN        20.0    # 有缺失值
    2      3.0        3.0        NaN      # 有缺失值
    3      4.0        4.0        40.0
    """

    data=data.dropna()#删除包含缺失值的行
    #删除缺失值后，数据如下：
    """
    pl_orbsmax  st_mass  pl_orbper
    0      1.0        2.0        10.0
    3      4.0        4.0        40.0
    """

```

```
#接下来就是从删除后的数据中提取自变量x和因变量y
x=data[['pl_orbsmax','st_mass']]
y=data['pl_orbper']

#那么处理后x和y就变成了：
"""
x:
pl_orbsmax  st_mass
0      1.0      2.0
3      4.0      4.0

y:
0      10.0
3     40.0
"""

#extend.py拓展部分新加：对数变换，且可以减小数据之间的间隙
x=np.log(abs(x))#加绝对值可以避免负值
y=np.log(abs(y))

#返回处理后的自变量x和因变量y
return x,y
```

```
#训练和测试模型函数
def train_test_model(x,y,test_size=0.7):
    #将数据集划分为训练集和测试集

    #test_size表示有多少的数据用来测试，random_state=42表示随机种子为42。随机种子的设置
    #是为了每次运行代码时，划分结果都相同。
    #这里选择随机种子为42是因为42是宇宙的终极奥秘()。
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=test_size,random_state=42)
```

```
#创建并训练线性回归模型
```

```
#这里利用scikit-learn库中的LinearRegression类来创建线性回归模型，其中model就是
LinearRegression类的一个实例。
```

```
#LinearRegression类是scikit-learn库中的一个线性回归类，用于构建类似于
y=ax_{1}+bx_{2}+c的方程。其中y代表我们要预测的轨道周期，x_{1}代表轨道半长轴，x_{2}代表
恒星质量。
```

```
model=LinearRegression()
```

```
#利用LinearRegression类中的fit方法训练模型
model.fit(x_train,y_train)
```

```
#用LinearRegression类中的predict方法进行预测
y_pred=model.predict(x_test)
```

```
#extend.py拓展部分新加：计算R2分数
r2=model.score(x_test,y_test)
```

```
#利用mean_squared_error函数计算均方误差
mse=mean_squared_error(y_test,y_pred)
```

```
#返回model,mse,r2,y_test,y_pred
return model,mse,r2,y_test,y_pred

#手动实现最小二乘法实现线性回归 ( Build it from scratch应该是这个意思吧( ) )
def manual_linear_regression(x,y,test_size=0.7):
    #将数据集划分为训练集和测试集
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=test_size,
random_state=42)

    #添加偏置项 ( 也就是截距项 ) · 把矩阵x的最左边那一行全部变成1 · 以便于出现常数项
    #比如 :
    """
原始数据 :
x:
|2 3|
|4 5|
样本 :
x1=2,x2=3
x1=4,x2=5

添加偏置项后 :
x:
|1 2 3|
|1 4 5|

就会是 :
result:
|1xc 2xβ1 3xβ2|
|1xc 4xβ1 5xβ2|
"""
#添加偏置项
x_train_b=np.c_[np.ones(x_train.shape[0]),x_train]

#计算最小二乘法的解析解 : β=(X^T*X)^-1*X^T*y

results=np.linalg.inv(x_train_b.T.dot(x_train_b)).dot(x_train_b.T).dot(y_train)

#计算测试集的预测值
x_test_b=np.c_[np.ones(x_test.shape[0]),x_test]
y_pred=x_test_b.dot(results)

#计算均方误差
mse=mean_squared_error(y_test,y_pred)

#extend.py拓展部分新加 : 计算R2分数
#R2=1-(残差平方和/总平方和)
#残差平方和=sum((y_test-y_pred)**2)
l=np.sum((y_test-y_pred)**2)
#总平方和=sum((y_test-y_test.mean())**2)
r=np.sum((y_test-y_test.mean())**2)
r2=1-(l/r)

#返回结果
```

```
#results[0]是截距项，results[1:]是系数项
return results[0],results[1:],y_test,y_pred,mse,r2

#利用plt可视化预测结果
def visualize_results(x,y,mode=1):
    #创建图形窗口，并将大小设置为10x6英寸
    fig=plt.figure(figsize=(10,6))

    #创建GridSpec对象，用于控制子图的布局
    #2,1表示创建一个2行1列的子图
    #height_ratios=[1,4]表示第一行的高度为1，第二行的高度为4
    #hspace=0.2表示子图之间的垂直间距为0.2
    gs=fig.add_gridspec(2,1,height_ratios=[1,4],hspace=0.2)

    #在网格的第一行创建一个子图，用于显示线性回归方程，评估指标
    #gs[0]表示网格的第一行
    ax_text=fig.add_subplot(gs[0])

    #在网格的第二行创建一个散点图子图
    #gs[1]表示网格的第二行
    ax_scatter=fig.add_subplot(gs[1])

    #调整子图布局，在底部留出0.2的空白区域放置滑动条
    plt.subplots_adjust(bottom=0.2)

    if mode==1:
        #获取train_test_model的返回值
        model,mse,r2,y_test,y_pred=train_test_model(x,y,test_size=0.7)

        #获取线性回归方程的文本
        equ=f'pl_order = {model.coef_[0]:.3f} * pl_orbsmax + {model.coef_[1]:.3f} * st_mass + {model.intercept_:.3f}'
    elif mode==2:
        #获取manual_linear_regression的返回值

intercept,coef,y_test,y_pred,mse,r2=manual_linear_regression(x,y,test_size=0.7)

        #获取线性回归方程的文本
        equ=f'pl_order = {coef[0]:.3f} * pl_orbsmax + {coef[1]:.3f} * st_mass + {intercept:.3f}'

        #获取MSE文本
        mse_text=f'均方误差 ($MSE$) : {mse:.3f}'

        #获取R2文本
        r2_text=f'决定系数 ($R^2$) : {r2:.3f}'

        #记录误差文本
        error_text=f'{mse_text}\n{r2_text}'

        #plt.scatter()来绘制散点图，其中第一个参数是x轴（表示真实值），第二个参数是y轴（表示预测值），第三个参数是透明度
        scatter=ax_scatter.scatter(y_test,y_pred,alpha=0.5,label='pred vs. true')
```

```
#获取坐标轴的最小值和最大值
ax_min=min(y_test.min(),y_pred.min())
ax_max=max(y_test.max(),y_pred.max())

#绘制理想的预测线 ( ideal ) · 也就是对角线
#[y_test.min(),y_test.max()]:x轴的起点和终点
#[y_test.min(),y_test.max()]:y轴的起点和终点
#'r--':红色虚线 ( r代表red红色 · --代表虚线 )
#lw=2 : 线宽为2
line,=ax_scatter.plot([ax_min,ax_max],[ax_min,ax_max], 'r--',
',lw=2,label='ideal')

#设置散点图的坐标轴范围 · 使之变成一个正方形
ax_scatter.set_xlim(ax_min,ax_max)
ax_scatter.set_ylim(ax_min,ax_max)
#设置散点图的坐标轴比例
ax_scatter.set_aspect('equal',adjustable='box')

#创建一个滑动条 · 用于控制测试集的比例test_size
#0.2 : 滑动条的x轴
#0.1 : 滑动条的y轴
#0.6 : 滑动条的宽度
#0.03 : 滑动条的高度
ax_slider=plt.axes([0.2,0.1,0.6,0.03])
#创建一个调节范围为0-1的滑动条 · 并且初始值为0.7
slider=Slider(ax_slider,'Test Size',0,1,valinit=0.7)

#文本信息关闭坐标轴
ax_text.axis('off')

#添加到ax_text中
#0.05代表文本框左上角的x坐标
#0.5代表文本框左上角的y坐标
#transform指定坐标系是transAxes(相对坐标系) · 坐标范围为[0,1]
#verticalalignment='center' 表示文本垂直居中于(0.05,0.5)
#bbox=dict()设置文本框的背景和边框样式 · boxstyle='round' 文本框为圆角矩形 ·
facecolor='white' 背景颜色为白色, alpha=0.8透明度为0.8
ax_text.text(0.05,0.5,f'线性回归方程 : \n{equ}\n\n评估指标 :
\n{error_text}',transform=ax_text.transAxes,verticalalignment='center',bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

#利用滑动条中的值来更新test_size
def update(val):
    #获取滑动条中的值并进行赋值
    test_size=slider.val
    if mode==1:
        #重新利用test_size获取测试数据
        model,mse,r2,y_test,y_pred=train_test_model(x,y,test_size=test_size)
        #获取线性回归方程的文本
        equ=f'pl_order = {model.coef_[0]:.3f} * pl_orbsmax +
{model.coef_[1]:.3f} * st_mass + {model.intercept_:.3f}'
    elif mode==2:
        #重新利用test_size获取测试数据
```

```
intercept,coef,y_test,y_pred,mse=manual_linear_regression(x,y,test_size=test_size)
)
#获取线性回归方程的文本
equ=f'pl_order = {coef[0]:.3f} * pl_orbsmax + {coef[1]:.3f} * st_mass
+ {intercept:.3f}'

#获取MSE文本
mse_text=f'均方误差 ( $MSE$) : {mse:.3f}'

#获取R2文本
r2_text=f'决定系数 ( $R^2$) : {r2:.3f}'

#记录误差文本
error_text=f'{mse_text}\n{r2_text}'

#更新散点图
scatter.set_offsets(np.c_[y_test,y_pred])

#更新文本注释
#先清理原本的数据
ax_text.clear()
#关闭坐标轴
ax_text.axis('off')
#添加新的文本注释
ax_text.text(0.05,0.5,f'线性回归方程：\n{equ}\n评估指标：
\n{error_text}',transform=ax_text.transAxes,verticalalignment='center',bbox=dict(boxstyle='round',facecolor='white',alpha=0.8))

#更新坐标轴的最大值和最小值
ax_min=min(y_test.min(),y_pred.min())
ax_max=max(y_test.max(),y_pred.max())

#更新理想预测线
line.set_data([ax_min,ax_max],[ax_min,ax_max])

#更新坐标轴的范围
ax_scatter.set_xlim(ax_min,ax_max)
ax_scatter.set_ylim(ax_min,ax_max)
#更新坐标轴的比例
ax_scatter.set_aspect('equal',adjustable='box')

#更新图表，这里利用draw_idle()不用draw()是因为draw_idle()更适合交互式的东西比如滑动条
fig.canvas.draw_idle()

#利用滑动条的on_changed方法来更新图表
slider.on_changed(update)

#在左上角添加图例
ax_scatter.legend(loc='upper left')

#设置x轴标签（示意图中给的是T_true）
ax_scatter.set_xlabel('T_true')
```

```
#设置y轴标签 (示意图中给的是T_pred)
ax_scatter.set_ylabel('T_pred')

#设置图表的标题 (示意图中给的是true vs. pred T)
ax_scatter.set_title('true vs. pred T')

ax_scatter.grid(True)

#显示图表
plt.show()

#主函数
def main(mode=1):
    #加载和预处理数据
    x,y=load_process_data()

    #可视化预测结果
    visualize_results(x,y,mode)

#如果当前文件是主程序，而不是被用作模块，则执行main函数
if __name__=='__main__':
    if len(sys.argv)>=2:
        main(int(sys.argv[1]))
```

## 运行截图

### ▶ 基础部分

- **base\_sklearn.py**
- **base\_manual.py**

### ▶ 拓展部分

- **extend\_sklearn.py**
- **extend\_manual.py**

## 总结

### ▶ 总结

确实是关于机器学习，线性回归模型的好题，从中也学到了很多新的库，比如scikit-learn, pandas, matplotlib, numpy等。而且以前也只是停留在理论上，这次是通过代码实践了一下，印象更深了