Python编程及人工智能应用

第三章 线性回归及Python编程

内容提要

南京邮电大学 Nanjing University of Posts and Telecommunications

- 线性回归问题简介
- 单变量线性回归问题
- 基于Scikit-learn库求解单变量线性回归
- 自定义求解单变量线性回归
  - 基于最小二乘法
  - 基于梯度下降法
- 多变量线性回归问题

机器学习

- 机器学习“预测”场景
  - 根据细风和晚霞预测明天的气温和天气
  - 根据餐馆的座位情况和香味预测这家店的菜品
  - 根据地铁、学区、居室情况估算房价
- 机器学习
  - 让计算机模仿人类，从过去经验中学习一个“模型”，通过学到的模型再对新情况给出一个预测
  - “经验”通常是以“数据”的形式存在
- 分类（classification）：预测的值是“离散”的
- 回归（regression）：预测的值是“连续”的

线性回归问题与模型

南京邮电大学 Nanjing University of Posts and Telecommunications

- 基本形式
  - 给定有*d*个属性的样例$\boldsymbol{x}=(x_1; x_2; x_3; \ldots; x_d)$，其中$x_i$是$\boldsymbol{x}$在第*i*个属性上的取值，线性回归试图学习得到一个预测函数$f(x)$，该函数的值是各属性值的线性加权和，公式如下

$$f(x) = w_1x_1 + w_2x_2 + \ldots + w_dx_d + b$$

  - 向量形式为：　$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$
  - **w和b是可学习和调整的参数，可根据经验手动设定，或自动从数据中学习获得**
  - 预测某套商品房的总价

$$f(\mathbf{x}) = 3\times\text{面积大小} + 0.5\times\text{楼层指数} + 0.2\times\text{卧室数量指数}$$

- 单变量线性回归与多变量线性回归

单变量线性回归问题

- 当样本仅1个属性时（即只有x_1），只要求解两个参数（w_1和b），是**单变量**线性回归模型

$$f(x) = w_1 x_1 + b$$

- 案例描述：设某小区通过某房产中介处已售出5套房，房屋总价与房屋面积之间有如下的数据关系。现有该小区的一位业主想要通过该房产中介出售房屋，在业主报出房屋面积后，根据训练数据，中介能否能估算出该房屋的合适挂售价格？

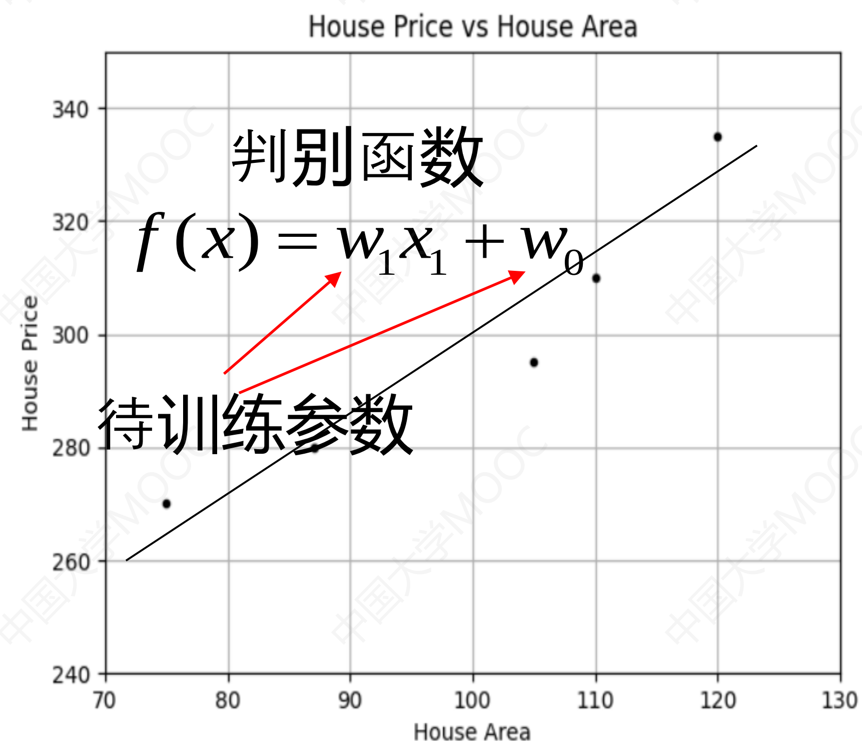
训练样本	房屋面积（平方米）	房屋总价（万元）
1	75	270
2	87	280
3	105	295
4	110	310
5	120	335

案例分析

- 把房屋面积看成自变量*x*，房屋总价看成因变量*y*，先通过绘图看出二者之间的关系。

1. 房屋总价随着房屋面积的变化，大致呈现线性变化趋势；
2. 如果根据现有的训练样本数据能够拟合出一条直线，使之与各训练样本数据点都比较接近，那么根据该直线，就可以计算出任意房屋面积的房屋总价了。

```
xTrain = np.array([75, 87, 105,110,120])
yTrain = np.array([270,280,295,310,335])
plt.plot(xTrain, yTrain, 'k.') #k表示绘制颜色为黑色，点表示绘制散点图
plt.show()
```



LinearRegression类

- **Scikit-learn提供了sklearn.linear_model.LinearRegression线性回归类，可以解决大部分常见的线性回归操作**
- **构造方法**

model = LinearRegression (fit_intercept = True, normalize = False, copy_X = True, n_jobs = 1)

fit_intercept：是否计算模型的截距，默认值是True，为False时则进行数据中心化处理；

normalize：是否归一化，默认值是False；

copy_X：默认True，否则X会被改写；

n_jobs：表示使用CPU的个数，默认1，当-1时，代表使用全部的CPU

LinearRegression类

- LinearRegression类的属性和方法
  - coef_：训练后的输入端模型系数，如果label有两个，即y值有两列，是一个2D的数组；
  - intercept_：截距，即公式 中的w_0值；
  - fit(x, y)：拟合函数，通过训练数据x和训练数据的标签y来拟合模型；
  - predict(x)：预测函数，通过拟合好的模型，对数据x预测y值；
  - score：评价分数值，用于评价模型好坏；

求解步骤

- 第一步：准备训练数据
  - #或xTrain=np.array([75, 87, 105, 110, 120])[:, np.newaxis]
  - xTrain=np.array([[75], [87], [105], [110], [120]])
  - yTrain = np.array ([270, 280, 295, 310, 335)
- 第二步：创建模型对象
  - Model = LinearRegression ( )
- 第三步：执行拟合
  - model.fit (xTrain, yTrain)
  - print ("截距b或w0：", model.intercept_)
  - print ("斜率w1：", model.coef_)
- 第四步：预测新数据
  - mode.predict (np.array ([[70]]))

代码3.2

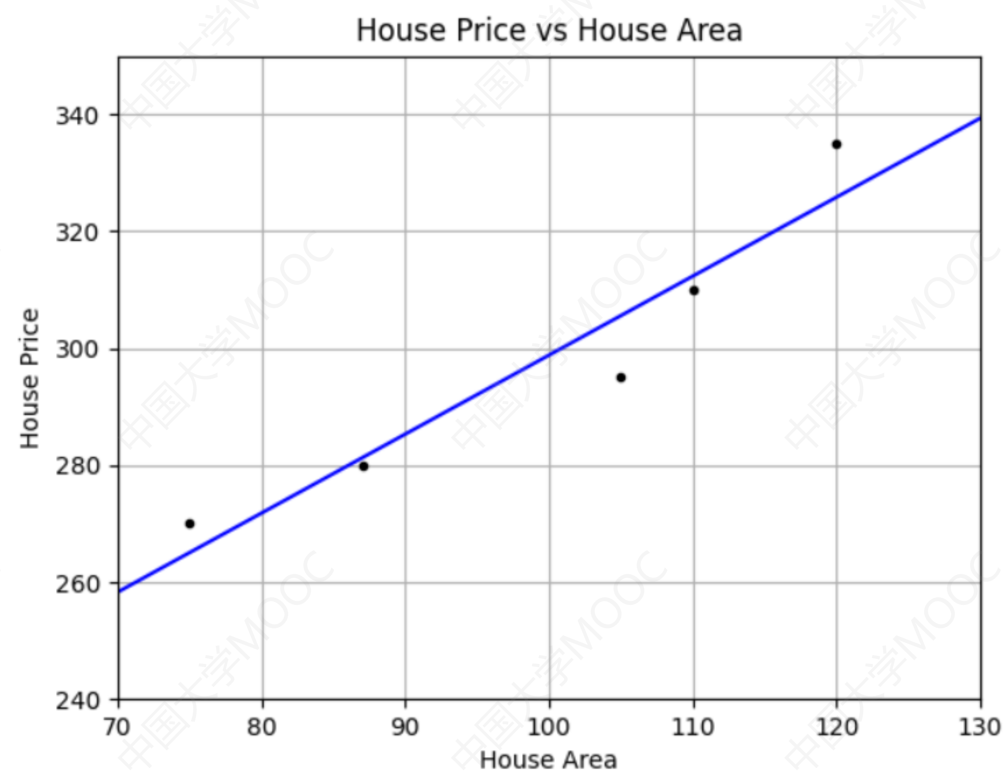

```
#代码3.2 基于Scikit-learn实现房价预测线性规划代码
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
# 以矩阵形式表达(对于单变量，矩阵就是列向量形式
xTrain = np.array ( [ [ 75 ], [ 87 ], [ 105 ], [ 110 ], [ 120 ] ] ) )
#为方便理解，也转换成列向量
yTrain = np.array ( [ 270, 280, 295, 310, 335 ] )
model = LinearRegression ( ) # 创建模型对象
model.fit ( xTrain, yTrain )      # 根据训练数据拟合出直线(以得到假设函数)
print ( "截距b或w0=", model.intercept_ )     # 截距
print ( "斜率w1=", model.coef_ )         # 斜率
# 预测面积为70的房源的总价
print ( "预测面积为70的房源的总价：", model.predict ( [ [ 70 ] ] ) )
# 也可以批量预测多个房源，注意要以列向量形式表达
xTest = np.array ( [ 85, 90, 93, 109 ] ) [ :, np.newaxi s]
yTestPredicted = model.predict ( xTest )
print ( "新房源数据的面积：", xTest )
print ( "预测新房源数据的总价：", yTestPredicted )
```

```
def initPlot ( ) :
    plt.figure ( )
    plt.title ( 'House Price vs House Area' )
    plt.xlabel ( 'House Area' )
    plt.ylabel ( 'House Price’ )
# 设置x轴和y轴的值域分别为70~130和240~350
    plt.axis ( [ 70, 130, 240, 350 ] )
    plt.grid ( True )
    return plt
plt = initPlot ( )
plt.plot ( xTrain, yTrain, 'k.' )  #格式字符串'k.'，表示绘制黑色的散点
#画出蓝色的拟合线
plt.plot ( [ [ 70 ], [ 130 ] ], model.predict ( [ [ 70 ], [ 130 ] ] ), 'b-' )
plt.show ( )
```

代码3.2运行结果

- **输出**

```
截距b或w0= 163.75113877922868
斜率w1= [1.35059217]
预测面积为70的房源的总价： [258.29259034]
新房源数据的面积： [[ 85]
 [ 90]
 [ 93]
 [109]]
预测新房源数据的总价：
[278.55147282 285.30443365 289.35621014 310.96568479]
```



模型评价

- 如何评价该模型的好坏：
  - 用什么数据对拟合模型进行评价？
    - 用训练数据计算的模型误差称之为训练误差，用测试数据计算的模型误差称之为测试误差
    - 在训练过程中，只有训练数据是可见的
    - 训练误差的最小化，不一定能使得测试误差最小化
  - 用什么指标对拟合模型进行评价？
    - 计算线性回归误差的指标主要包括残差平方和与R方（r-squared）

残差

- 残差（Residual）
  - 实际 之间的差异
  - 残差 有样本残差的
  - 残差 残差为0时，

R方

- R方（R-Square），又称确定系数
  - 表达因变量与自变量之间的总体关系
  - 与残差平方和在方差中所占的比率有关

$$SS_{total} = \sum_{i=1}^{m} (\overline{y} - y^{(i)})^2 \qquad R^2 = 1 - \frac{SS_{res}}{SS_{total}}$$

残差与R方的计算

- 手动计算：
  - 训练数据残差平方和：
    - ssResTrain=sum((yTrainPredicted - yTrain) ** 2)
  - 测试数据残差平方和：
    - ssResTest= sum((yTestPredicted - yTest) ** 2)
  - 测试数据方差：
    - ssTotalTest= sum((np.mean(yTest) - yTest) ** 2)
  - 测试数据R方：
    - rsquareTest=1 – ssResTest / ssTotalTest
- LinearRegression提供的自动计算方法：
  - 训练数据残差平方和：model._residues，通过训练数据训练模型后自动获得；
  - 自动计算R方的函数：model.score(xTest, yTest)

代码3.3

```
#代码3.3 加入评价的基于Scikit-lea
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import I
xTrain = np.array ( [ [ 75 ], [ 87 ], [
积)
yTrain = np.array ( [ 270,280,295,3
xTest = np.array ( [ 85, 90, 93, 109 ]
yTest = np.array ( [ 280, 282, 284, 3
model = LinearRegression ( )
model.fit ( xTrain, yTrain )
yTrainPredicted = model.predict ( xTrain ) # 针对训练数据进行预测以计算训练误差
yTestPredicted = model.predict ( xTest )  # 针
ssResTrain = sum ( ( yTrainPredicted – yTrai
数据集残差
print ( ssResTrain )
print ( model._residues ) # 自动计算的训练数
ssResTest = sum ( ( yTestPredicted – yTest ) **
ssTotalTest = sum ( ( np.mean ( yTest ) – yTes
数据集y值偏差平方和
```


```
st          # 手动计算测试数据R
            # 自动计算的测试数
a' )
plt.axis ( [ 70, 130, 250, 350 ] ) # 设置x轴和y轴的值域分别为70~130和240~350
' )         # 训练点数据(红色，小点)
            # 测试点数据(蓝色，大点)
edicted, 'g-' ) # 拟合的函数直线(绿色)
```

227.29653811114474

227.296538111449

0.8090280549127149

0.8090280549127149

最小二乘法求解

南京邮电大学
Nanjing University of Posts and Telecommunications

- 根据残差的定义，单个训练数据点的残差是$f(x^{(i)}) - y^{(i)}$
- 训练目标是最小化训练数据残差绝对值之和 $\sum_{i=1}^{m}|f(\mathbf{x}^{(i)})-y^{(i)}|$
- 绝对值不容易进行包括导数运算，因此采用数据的残差平方和 $\sum_{i=1}^{m}(f(\mathbf{x}^{(i)})-y^{(i)})^2$ 替代残差绝对值之和作为优化目标，使得残差平方和最小化，这种方法被称为最小二乘法
- 对最小二乘法的优化目标进行求解，有两种具体的方法，分别是导数法和矩阵法

导数法求解最小二乘法

- 训练目标： $L(w_0,w_1)=\sum_{i=1}^{m}(f(\mathbf{x}^{(i)})-y^{(i)})^2=\sum_{i=1}^{m}(w_1x_1^{(i)}+w_0-y^{(i)})^2$
- 函数L也可被称为“损失”（Loss）函数
  - 分别对w0和w1求一阶偏导，并使之为0

$$\begin{cases}\dfrac{\partial L}{\partial w_0}=2\sum_{i=1}^{m}(w_1x_1^{(i)}+w_0-y^{(i)})=0\\[2ex]\dfrac{\partial L}{\partial w_1}=2\sum_{i=1}^{m}x_1^{(i)}(w_1x_1^{(i)}+w_0-y^{(i)})=0\end{cases}$$

  - 求得 $w_0=\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}-w_1x_1^{(i)})=\overline{y}-w_1\overline{x_1}$，其中 $\overline{y}$ 和 $\overline{x_1}$ 表示平均值
  - 求得 $w_1=\dfrac{\sum x_1^{(i)}y^{(i)}-m\overline{y}\,\overline{x_1}}{\sum(x_1^{(i)})^2-m\overline{x_1}^2}$ （演示推导过程）
- 最终公式： $\begin{cases}w_0=\overline{y}-w_1\overline{x_1}\\[1ex] w_1=\dfrac{\operatorname{cov}(x_1,y)}{\operatorname{var}(x_1)}\end{cases}$ **（演示推导过程）**

代码3.4


```
#代码3.4 使用导数求解最小二乘法
import numpy as np
xTrain = np.array ( [ 75, 87, 105, 110, 120 ] )  # 训练数据(面积)
yTrain = np.array( [ 270,280,295,310,335 ] )         # 训练数据(总价)
w1 = np.cov ( xTrain, yTrain, ddof = 1) [ 1, 0 ] / np.var ( xTrain, ddof = 1 )
w0 = np.mean ( yTrain ) - w1 * np.mean ( xTrain )
print ( "w1=", w1 )
print ( "w0=", w0 )
```

```
w1= 1.350592165198907
w0= 163.7511387792286 3
```

使用矩阵运算求解

- 下面在使用矩阵法对最小二乘法的优化目标进行求解

$$y = w_0 \bullet 1 + w_1 \bullet x_1 + w_2 \bullet x_2 + \ldots + w_d \bullet x_d$$

- 转为 $y = \mathbf{w}^T \mathbf{x}$ 和 $\mathbf{Y} = \mathbf{w}^T \mathbf{X}$

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ x_1^{(1)} & x_1^{(2)} & x_1^{(3)} & \cdots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} & \cdots & x_2^{(m)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_d^{(1)} & x_d^{(2)} & x_d^{(3)} & \cdots & x_d^{(m)} \end{bmatrix}$$

- 求得结果 $\mathbf{w} = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{Y}^T$ （演示推导过程）

代码3.5和3.6

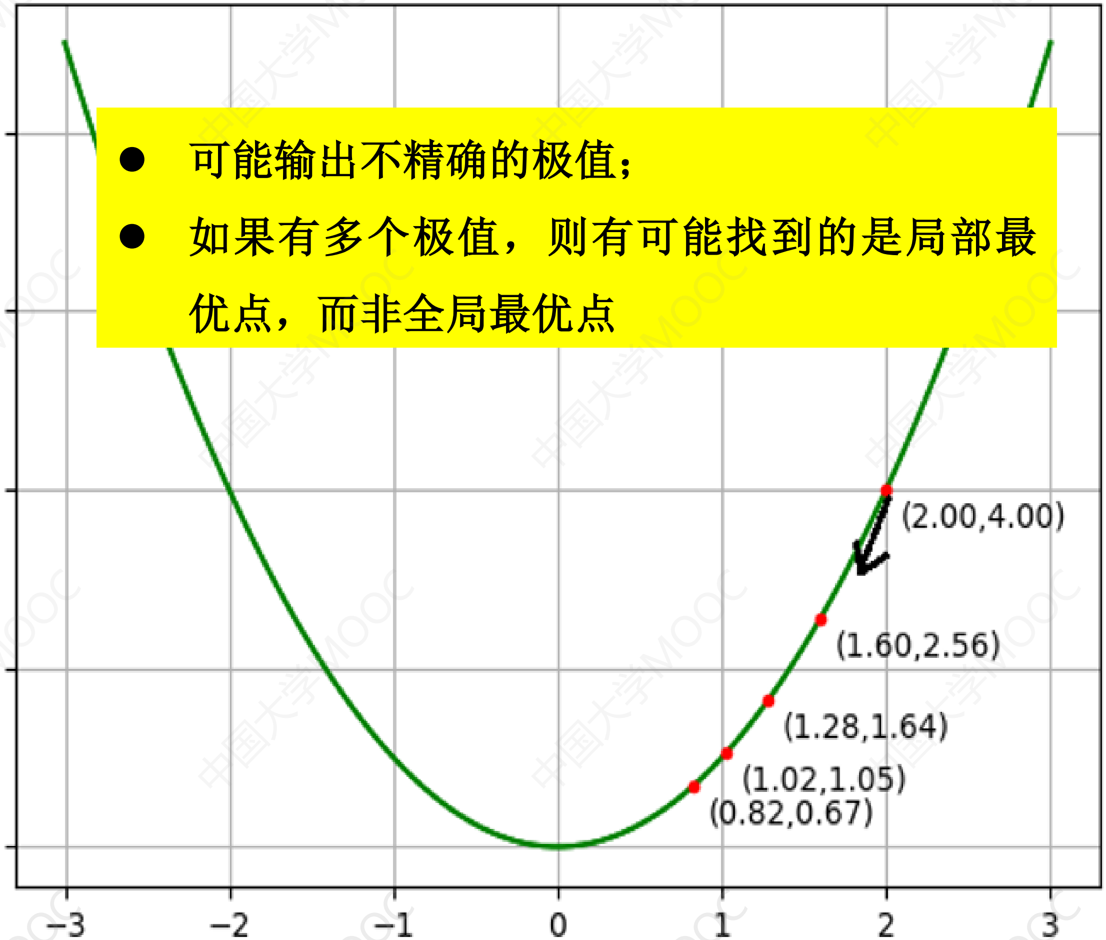

```
#代码3.5 求参数w的矩阵方法代码：linreg_matrix.py
import numpy as np
def linreg_matrix ( x, y ) :
X_X_T = np.matmul ( x, x.T )
X_X_T_1 = np.linalg.inv ( X_X_T )
X_X_T_1_X = np.matmul ( X_X_T_1, x )
X_X_T_1_X_Y_T = np.matmul ( X_X_T_1_X, y.T )
return X_X_T_1_X_Y_T
```

```
x= [[  1   1   1   1   1]
 [ 75  87 105 110 120]]
y= [[270 280 295 310 335]]
w= [[163.75113878]
 [  1.35059217]]
```

```
#代码3.6 使用矩阵方法求解房价预测问题
import numpy as np
from linreg_matrix import linreg_matrix
# 训练数据(面积)，每行表示一个数据点
xTrain = np.array ( [ [ 75 ], [ 87 ], [ 105 ], [ 110 ], [ 120 ] ] )
# 训练数据(总价)，每行表示一个数据点
yTrain = np.array ( [ 270, 280, 295, 310, 335 ] ) [ :, np.newaxis ]
def make_ext ( x ) : #对x进行扩展，加入一个全1的行
    ones = np.ones ( 1 ) [ :, np.newaxis ] #生成全1的行向量
    new_x = np.insert ( x, 0, ones, axis = 0 )
    return new_x
#为适应公式3.11的定义，将xTrain和yTrain进项转换，使得每一列表示一个数据点
x = make_ext ( xTrain.T )
y = yTrain.T
print ( "x=", x )
print ( "y=", y )
w = linreg_matrix ( x, y )
print ( "w=", w )
```

二次函数梯度下降法求解

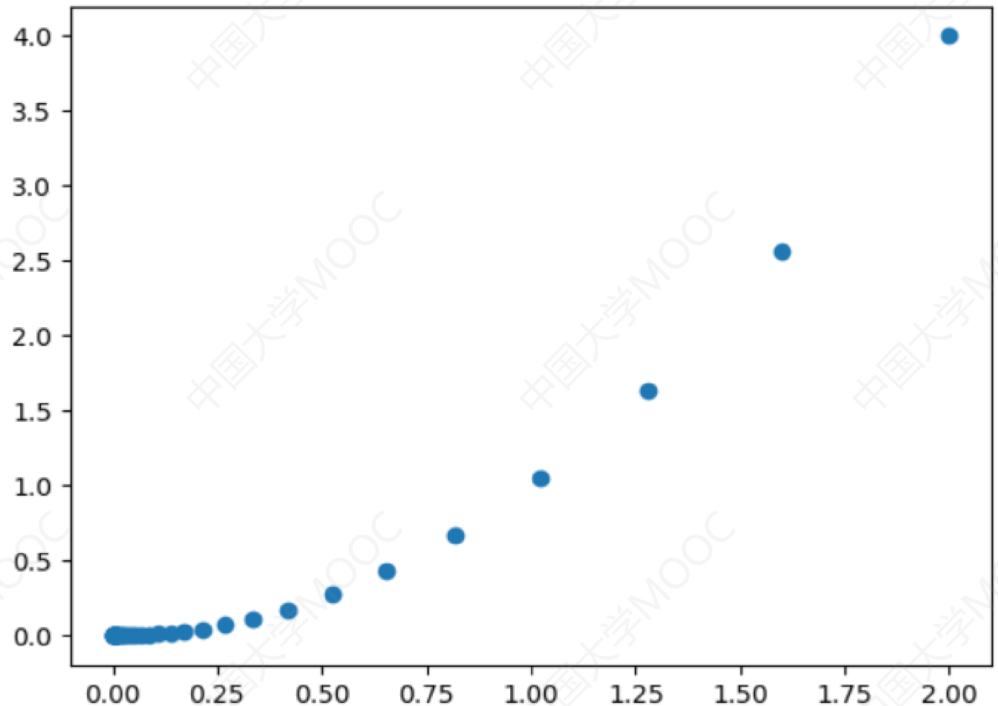
- 梯度下降法 思想是对目
  标函数选取 导数）的方
  向步步运算 部最优解
- 简单二次函
  - 第一步：
  - 第二步：
  - 第三步：
    - *lr*是学
    - $f'(x_0)$ 前进
  - 第四步： 值的绝对值小
    于某个临
  - 第五步： 如果达到则结
    束，如果未达到则设$x_0=x_1$，跳转到第二步继续执行



代码3.7

```
#代码3.7 对y=x*x函数的梯度下降优化过程
import numpy as np
import matplotlib.pyplot as plt
def obj_fun(x): return x * x  #需要求极值的目标函数
def dir_fun(x): return 2 * x  #目标函数的导数
x_list = [ ]
y_list = [ ] # 用于保存经过的点
def minimize ( init_x, lr = 0.1, dif = 1e-9, max_iter = 100(
   # init_x: 初始点，lr: 学习速率，dif: 相邻两步差异临
_iter: 最大迭代次数
   x0 = init_x
   y0 = obj_fun ( x0 )
   x_list.append ( x0 )
   y_list.append ( y0 )
   for i in range ( max_iter ) :
      x1 = x0 - lr * dir_fun ( x0 )
      y1 = obj_fun ( x1 )
      x_list.append ( x1 )
      y_list.append ( y1 )
```

```
      if abs ( y1 - y0 ) <= dif : #达到收敛条件
         print ( "是否收敛：True", "极小点：(%e, %e)"%(x1, y1),
%d"
```



是否收敛：True 极小点：(3.794275e-09, 1.439652e-17) 循环次数：89

批量梯度下降法

- 对于最小二乘法，其目标函数是：

$$L(w_0, w_1) = \frac{1}{2m}\sum_{i=1}^{m}(f(\mathbf{x}^{(i)}) - y^{(i)})^2 = \frac{1}{2m}\sum_{i=1}^{m}(w_1 x_1^{(i)} + w_0 - y^{(i)})^2$$

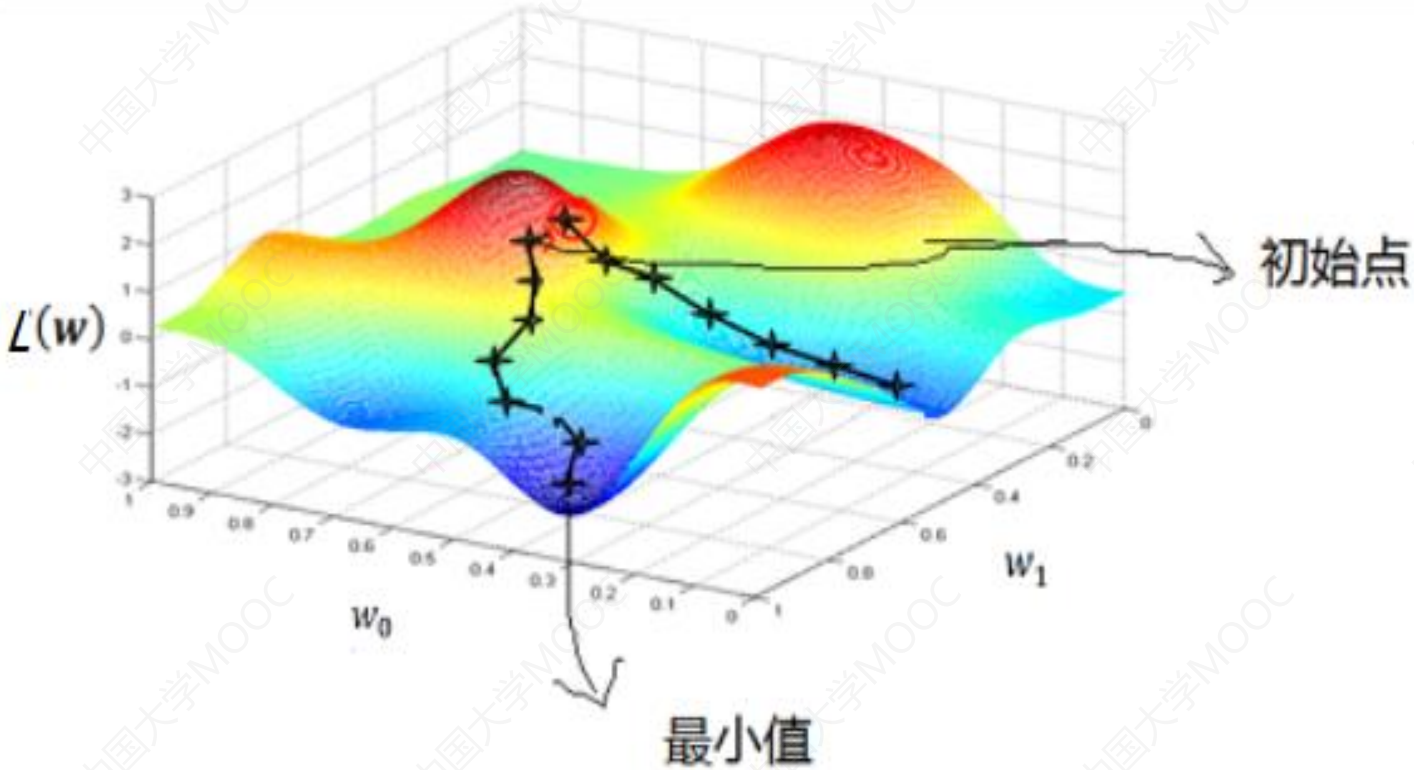
  - 优化的参数有两个，分别是 $\mathbf{w_0}$和 $\mathbf{w_1}$
  - 需要一次性批量地使用全部训练数据，被称为批量梯度下降法（**Batch Gradient Decent**）

算法步骤

南京邮電大學
Nanjing University of Posts and Telecommunications

- 算法步骤

  - 算
  - 算 化最快
  - 算 环迭代
    次



- 演

$$\frac{\partial L(w_0,w_1)}{\partial w_0}=\frac{1}{m}\sum_{i=1}^{m}(w_1 x_1^{(i)}+w_0-y^{(i)})$$

$$\frac{\partial L(w_0,w_1)}{\partial w_1}=\frac{1}{m}\sum_{i=1}^{m}[(w_1 x_1^{(i)}+w_0-y^{(i)})\bullet x_1^{(i)}]$$

代码3.8 批量梯度下降实现


```
#代码3.8 批量梯度下降法：bgd_optimizer.py
def bgd_optimizer ( target_fn, grad_fn, init_W, X, Y, lr=0.0001,tolerance=1e-12, max_iter=100000000 ) :
    W = init_W
    target_value = target_fn ( W, X, Y )        # 计算当前w值下的L(w)值
    for i in range ( max_iter ) :
        grad = grad_fn ( W, X, Y )              # 计算梯度
        next_W = W - grad * lr        # 向量计算，调整了w
        next_target_value = target_fn ( next_W, X, Y )    # 计算新值
        # 如果两次计算之间的误差小于tolerance，则表明已经收敛
        if abs ( next_target_value - target_value ) < tolerance:
            return i, next_W #返回迭代次数和参数w的值
        else: W, target_value = next_W, next_target_value        # 继续进行下一轮计算
    return i, None #返回迭代次数，由于未收敛，w没有优化的值
```

代码3.9

```
#代码3.9 基于梯度下降法求解房价预测问题
import numpy as np
from bgd_optimizer import bgd_optimizer
import matplotlib.pyplot as plt
def target_function ( W, X, Y ):  # 定义目标函数
    w0, w1 = W   # W:[w0, w1]
    # 应使用np.sum，而不要使用sum。
    #np.sum支持向量/矩阵运算
    return np.sum ( ( w0 + X * w1 – Y ) ** 2 ) / ( 2 * len ( X ) )
# 根据目标函数定义梯度，对x0和x1求导数，
# 累计各点导数平均值
def grad_function ( W, X, Y ):
    w0, w1 = W
    # 对应w0的导数
    w0_grad = np.sum ( w0 + X * w1 – Y ) / len ( X )
     # 对应w1的导数。注意采用向量运算
    w1_grad = X.dot ( w0 + X * w1 – Y ) / len ( X )
    return np.array ( [ w0_grad, w1_grad ] )
```

```
# 训练数据(面积)
x = np.array ( [ 75, 87, 105, 110, 120 ], dtype = np.float64 )
# 训练数据(总价)
y = np.array ( [ 270, 280, 295, 310, 335 ], dtype = np.float64 )
np.random.seed ( 0 )
init_W = np.array ( [ np.random.random ( ), np.random.random ( ) ] )
# 随机初始化W值
i, W= bgd_optimizer ( target_function, grad_function, init_W, x, y )
if W is not None :
    w0, w1 = W
    print ( "迭代次数：%d，最优的w0和w1:(%f, %f)" % ( i, w0, w1 ) )
else: print ( "达到最大迭代次数，未收敛" )
```

迭代次数：4051854，

最优的w0和w1:(163.746743, 1.350635)

思考：如果将*lr*设为较大值（如设置为0.01或0.1）会出现什么状况？

随机梯度下降法

- 批量梯度下降法的缺陷：当样本数量非常庞大时，计算复杂、费时且极有可能超出硬件能力（内存容量、CPU计算能力等）限制。
- 随机梯度下降法（Stochastic Gradient Decent，简称SGD），其原理是每次随机选取一部分样本对目标函数进行优化。
- 随机梯度下降法能够达到不弱于批量梯度下降法的优化效果，在实际场景中应用更加广泛。

代码3.10


```
#代码3.10 随机梯度下降法：sgd_optimizer.py
import numpy as np
def sgd_optimizer ( target_fn,grad_fn, init_W, X, Y, lr=0.0001, tolerance=1e-12, max_iter=100000000 ) :
    W, rate = init_W, lr
    min_W, min_target_value = None, float ( "inf" )
    no_improvement = 0
    target_value = target_fn ( W, X, Y )
    for i in range ( max_iter ) :
        index = np.random.randint ( 0, len ( X ) )          # 获得一组随机数据的索引值
        gradient = grad_fn ( W, X [ index ], Y [ index ] )      # 计算该数据点处的导数
        W = W - lr * gradient
        new_target_value = target_fn ( W, X, Y )
        if abs ( new_target_value - target_value ) < tolerance :
            return i, W
    return i, None
```

多变量线性回归问题

南京邮电大学
Nanjing University of Posts and Telecommunications

- 现实生活中，房价不仅与面积有关，也与户型、楼层等相关。增加了房屋户型作为因变量，变化后的数据如表所示
  - 户型类型：1（一居室）、2（二居室）、3（三居室）、4（四居室）

训练样本	房屋面积（平方米）	房屋户型	房屋总价（万元）
1	75	1	270
2	87	3	280
3	105	3	295
4	110	3	310
5	120	4	335

测试样本	房屋面积（平方米）	房屋户型	房屋总价（万元）
1	85	2	280
2	90	3	282
3	93	3	284
4	109	4	305

基于Scikit-learn库求解


```
#代码3.12 使用Scikit-learn库实现多变量房价预测问题求解
import numpy as np
from sklearn.linear_model import LinearRegression
xTrain = np.array ( [ [ 75, 1 ], [ 87, 3 ], [ 105, 3 ], [ 110, 3 ], [ 120, 4 ] ] )  # 训练数据(面积，户型)
yTrain = np.array ( [ 270,280,295,310,335 ] )          # 训练数据(总价)
xTest = np.array ( [ [ 85, 2 ], [ 90, 3 ], [ 93, 3 ], [ 109, 4 ] ] ) # 测试数据(面积，户型)
yTest = np.array ( [ 280, 282, 284, 305 ] )            # 测试数据(总价)
model = LinearRegression ( )
model.fit ( xTrain, yTrain )
print ( model._residues )                     # 训练数据集残差：226.8291
print ( model.score ( xTest, yTest ) )               #测试数据集的R方： 0.8374
```

残差 ：226.82910636037715

R方 ：0.8373813078020356

基于最小二乘法求解

南京郵電大學 Nanjing University of Posts and Telecommunications

```
#代码3.13 使用矩阵方法求解多变量房价预测问题
import numpy as np
from linreg_matrix import linreg_matrix
xTrain = np.array ( [ [ 75, 1 ], [ 87, 3 ], [ 105, 3 ], [ 110, 3 ], [ 120,
4 ] ] ) # 训练数据(面积，户型)
# 训练数据(总价
yTrain = np.a
# 测试数据(面
xTest = np.ar
# 测试数据(总
yTest = np.ar
def make_ext
    ones = np.ones ( 1 ) [ :, np.newaxis ] #生成全1的行向量
    new_x = np.insert ( x, 0, ones, axis = 0 )
    return new_x
#为适应公式3.9的定义，将xTrain和yTrain进项转换，使得每一
列表示一个数据点
x = make_ext ( xTrain.T )
y = yTrain.T
```

```
w = linreg_matrix ( x, y )
print ( "w=", w )
 # 针对训练数据进行预测以计算训练误差
yTrainPredicted = w.dot ( x )
# 针对测试数据进行预测

# 手动计算测试数据集y值偏差平方和
ssTotalTest = sum ( ( yTest - np.mean ( yTest ) ) ** 2 )
# 手动计算测试数据R方
rsquareTest = 1 - ssResTest / ssTotalTest
 # 测试数据集的R方
print ( "测试数据R方： ", rsquareTest )
```

w= [162.19293587   1.38427832   -0.63935732]

训练数据残差平方和： 226.8291063603765

测试数据R方： 0.837381307801918

基于梯度下降法求解

- 梯度下降方法同样可以用于多变量房价预测求解，需要对目标函数和梯度函数做一些扩展。

$$L(\mathbf{w}) = \frac{1}{2m}\sum_{i=1}^{m}\left(\sum_{k=1}^{d} w_k x_k^{(i)} + w_0 - y^{(i)}\right)^2$$

$$\frac{\partial L(\mathbf{w})}{\partial w_0} = \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{k=1}^{d} w_k x_k^{(i)} + w_0 - y^{(i)}\right)$$

- 对于k>0，有 $\frac{\partial L(w)}{\partial w_k} = \frac{1}{m}\sum_{i=1}^{m}\left[\left(\sum_{k=1}^{d} w_k x_k^{(i)} + w_0 - y^{(i)}\right)\bullet x_k^{(i)}\right]$

代码3.14

南京邮电大学 Nanjing University of Posts and Telecommunications

```
#代码3.14 基于批量梯度下降法求解多变量房价预测问题
import numpy as np
from bgd_optimizer import bgd_optimizer
import matplotlib.pyplot as plt
def target_function ( W, X, Y ) :  # 定义目标函数
    return np.sum ( ( W [ 0 ] + X.dot ( W [ 1: ]) – Y ) ** 2 ) / ( 2 * len ( X ) )
def grad_function ( W, X, Y ) : # 根据目标函数定义梯度，对w0, w1, w2求导数平均值
    w0_grad = np.sum ( W [ 0 ] + X.dot ( W [ 1: ] ) – Y ) / len ( X ) #对应w0的导数
    w1_grad = X [ :, 0 ].dot ( np.array ( W [ 0 ] ) + X.dot ( W [ 1: ] ) – Y ) / len ( X ) #对应w1的导数
    w2_grad = X [ :, 1 ].d
    return np.array ( [ w
x = np.array ( [ [ 75, 1 ]
y = np.array ( [ 270, 280, 295, 310, 335 ], dtype = np.float64 )
np.random.seed ( 0 )
init_W = np.array ( [ np.random.random(), np.random.random ( ), np.random.random ( ) ] )#随机初始化W
i, W = bgd_optimizer ( target_function, grad_function, init_W, x, y )
if W is not None :
    w0, w1, w2 = W
    print ( "迭代次数：  %d，最优的w0, w1, w2:(%f, %f, %f)" % ( i, w0, w1, w2 ) )
else: print ( "达到最大迭代次数，未收敛" )
```

迭代次数：6598762，
最优的w0, w1, w2:(162.185449, 1.384389, -0.640646)

数据归一化问题

- 在多变量情况下，各变量的值域有很大差别，比如房屋面积的范围是几十到几百的浮点数，房屋户型的值域是1到5之间的整数
- 值域差异过大，容易造成计算过程中出现溢出或无法收敛，导致各个变量作用权重受到影响
- 可以对数据进行归一化（Normalization）处理来解决这一问题
- 归一化方法：

$$x\_norm_i = \frac{x_i - \overline{x_i}}{std(x_i)}$$

代码3.15

南京邮电大学
Nanjing University of Posts and Telecommunications

```
#代码3.15 先进行归一化处理，再使用Scikit-learn库求解
import numpy as np
from sklearn.linear_model import LinearRegression
def normalize ( X ) :
    X_mean = np.mean ( X, 0 ) #计算均值
    X_std = np.var ( X, 0 ) #计算标准差
    return ( X - X_mean ) / X_std
xTrain = np.array ( [ [ 75, 1 ], [ 87, 3 ], [ 105, 3 ], [ 110, 3 ], [ 120, 4 ] ] )  # 训练数据(面积，户型)
yTrain = np.array ( [ 270, 280, 295, 310, 335 ] )          # 训练数据(总价)
xTrain = ( normalize ( xTrain ) )
model = LinearRegression ( )
model.fit ( xTrain, yTrain )
print( model._residues )                    # 训练数据集残差
```

输出结果为：226.82910636037712

高阶拟合问题

- 单变量房价预测问题中只有一个自变量“房屋面积”，因此拟合出来的是一条直线；多变量房价预测问题中有两个自变量“房屋面积”和“房屋户型”，因此拟合出来的是一个直平面
- 采用“曲线”或“曲面”模型来拟合能够对训练数据产生更逼近真实值的效果，这就是高阶拟合，有可能是非线性的。

高阶拟合问题

- 对于一个自变量的场景，可以采用多阶函数：

$$f_n(x) = \sum_{k=0}^{n} w_k x^k$$

- 二阶单变量函数为：$f_2(x) = w_0 + w_1 x + w_2 x^2$
- 三阶单变量函数为：$f_3(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3$
- 四阶单变量函数为：$f_4(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4$
- 可以通过如下方式进行变换：
  - 加入新变量x_1，设$x_1=x$；
  - 加入新变量x_2，设$x_2=x^2$；
  - 加入新变量x_3，设$x_3=x^3$；
  - 加入新变量x_4，设$x_4=x^4$；

代码3.16

南京邮电大学 Nanjing University of Posts and Telecommunications

```
#代码3.16 使用Scikit-learn库实现高阶拟合单变量房价预测
import numpy as np
from sklearn.linear
xTrain = np.array (
x1 = xTrain
x2 = xTrain ** 2
x3 = xTrain ** 3
x4 = xTrain ** 4
new_xTrain2 = np.
new_xTrain3 = np.
new_xTrain4 = np.
yTrain = np.array (
xTest = np.array ( [
x1 = xTest
x2 = xTest ** 2
x3 = xTest ** 3
x4 = xTest ** 4
new_xTest2 = np.co
new_xTest3 = np.concatenate ( [ x1, x2, x3 ], axis = 1 )
new_xTest4 = np.concatenate ( [ x1, x2, x3, x4 ], axis = 1 )
```

```
new_xTest4 = np.concatenate ( [ x1, x2, x3, x4 ], axis = 1 )
yTest = np.array ( [ 280, 282, 284, 305 ] )        # 测试数据(总价)
)
est, yTest ) )
)
w_xTest2, yTest ) )
)
w_xTest3, yTest ) )
)
print ( "四阶测试数据集R方： ", model4.score ( new_xTest4, yTest ) )
```

一阶训练数据集残差： 227.2965381111449
一阶测试数据集R方： 0.8090280549127149
二阶训练数据集残差： 38.38456969539346
二阶测试数据集R方： 0.8714891148616254
三阶训练数据集残差： 16.488230700095926
三阶测试数据集R方： 0.9885676670452613
四阶训练数据集残差： 5.498381308907731e-18
四阶测试数据集R方： 0.8830805089175938

绘制图形

House Price vs House Area
三阶函数
二阶函数
四阶函数
340
320
300
280
260
House Price
70
80
90
100
110
120
130
House Area

本章小结

南京邮电大学
Nanjing University of Posts and Telecommunications

- 本章介绍了线性回归问题的定义，以及求解线性回归问题的多种方法，包括Scikit-Learn库函数求解法、最小二乘法、梯度下降法等。通过本章的学习，读者能够对线性回归问题有一个基本的了解，并能掌握通过Python编码求解线性回归问题的基本方法，为后续章节的学习打下良好的基础。

输入理想的程序

输出快乐的人生