中国石油大学（华东）
CHINA UNIVERSITY OF PETROLEUM

2022—2023 学年第 2 学期

《机器学习》课程报告

选题名称		梯度下降法求解线性回归问题—以重力异常为例		
小组成员	学号	姓名	任务分工	备注
	2001030118	王岩	1.1； 1.2；1.3.1； 1.3.4； 3；	
	2001030104	郭文静	2.1 ； 2.2； 2.3；3.3；5	
	2001030116	石博文	1.3.2；1.3.3；1.3.4；4	
	2001030124	赵童	2.4；；3.3；3.6；4	
教师评语				

教师签名:
2023年 4 月 5 日

备注：任务分工是按照章节内容编写划分，每一节长短不一，重合部分为共同编写。

1 线性回归问题

1.1 概念

线性回归（Linear Regression）是利用数理统计中回归分析，来确定两种或两种以上变量间相互依赖的定量关系的一种统计分析方法，运用十分广泛。当只包括一个自变量和一个因变量，且二者的关系可用一条直线近似表示，这种回归分析称为一元线性回归分析。

简言之，当用输入样本的特征的线性组合作为预测值时，就是线性回归。

1.2 表示形式

如果样本为$s = (\boldsymbol{x},\ y)$，其中$\boldsymbol{x}$为二维矩阵，是样本的实例：$\boldsymbol{x} = (x^{(1)}, x^{(2)}, x^{(3)} \dots x^{(n)})$，$x^{(j)}$为样本的第$j$维特征，每一个特征又对应着$m$个特征值，即$x^{(j)} = (x^{(j)}{}_1, x^{(j)}{}_2, x^{(j)}{}_3 \dots x^{(j)}{}_m)$；$y$为样本的标签（可以理解为函数值）。

线性回归模型试图通过n维样本特征学得一个线性组合（函数）来对未来的样本输入进行预测，即：

$$f(x) = \boldsymbol{W} \bullet \boldsymbol{x^T} + b = \sum_{i=0}^{n} \boldsymbol{w}^{(j)} \bullet \boldsymbol{x}^{(j)} + b \quad (1-1)$$

其中$\boldsymbol{W} = (w^{(1)}, w^{(2)}, w^{(3)} \dots w^{(n)})$称为回归系数，即各样本特征的权重，标量b称为偏置，是一个总体的偏差。

线性回归的目的就是求得回归系数矩阵$\boldsymbol{W}$、偏置常数b。

特殊情况下，当样本只有一个特征时，线性回归模型就变为一维函数：

$$f(x) = w^{(1)} \bullet x^{(1)} + b \quad (1-2)$$

1.3 求解方法

1.3.1 问题转化

最小二乘法（least squares）是最常用的用来求解线性回归问题的经典方法。是利用矩阵等数学知识来求取回归函数的解析解。下面介绍最小二乘法来求解线性回归问题的主要步骤。

式子（1-1）不适合矩阵运算，暂时把标量b去掉得：

$$f(x) = \boldsymbol{W} \bullet \boldsymbol{x^T} = \sum_{i=0}^{n} \boldsymbol{w}^{(i)} \bullet \boldsymbol{x}^{(i)} \quad (1-3)$$

其中$\boldsymbol{x} = (x^{(0)}, x^{(1)}, x^{(2)}, x^{(3)} \dots x^{(n)})$为特征矩阵，并指定$x^{(0)} = 1$，$\boldsymbol{W} = (w^{(0)}, w^{(1)}, w^{(2)}, w^{(3)} \dots w^{(n)})$为回归系数矩阵，并指定$w^{(0)} = 1$。那么回归问题就转变为求解回归系数矩阵$\boldsymbol{W}$。

1.3.2 损失函数

损失函数是评价回归模型与原数据拟合情况好坏的重要参量，常用的有残差（或称绝对误差）：

$$s_j = \left| y_j - \left(f(x_j) \right) \right| \quad (1-4)$$

残差方（或称平方损失函数）：

$$s_j = \left| y_j - \left(f(x_j) \right) \right|^2 \quad (1-5)$$

后者方便求导数，所以应用更广泛。s_j越小，则证明回归模型拟合越好，那么只需在规定的残差方最小范围内得到所需的回归系数矩阵$\boldsymbol{W}$即可。

1.3.3 最小二乘法求解

对于前述第j维样本特征来说，$x^{(j)} = (x^{(j)}{}_1, x^{(j)}{}_2, x^{(j)}{}_3 \dots x^{(j)}{}_m)$。其平方损失函数${s_j}^2(W)$就是实际值（标签）$y^{(j)}$与预测值$\left(f(x_j) \right)$之差的平方即：

$${s_j}^2(W) = \left(y^{(j)} - \left(f(x_j) \right) \right)^2 = \left(y^{(j)} - \boldsymbol{W} * x_j \right)^2 \quad (1-6)$$

当训练的样本有n个时，平方损失函数和为：

$$\begin{aligned} L(\boldsymbol{W}) &= s_1^2(\boldsymbol{W}) + s_2^2(\boldsymbol{W}) + \cdots + s_n^2(\boldsymbol{W}) \\ &= (y_1 - \boldsymbol{W} \bullet x_1)^2 + (y_2 - \boldsymbol{W} \bullet x_2)^2 + \cdots + (y_m - \boldsymbol{W} \bullet x_n)^2 \\ &= (y_1 - \boldsymbol{W} \bullet x_1,\ y_2 - \boldsymbol{W} \bullet x_2 \ \dots \ y_n - \boldsymbol{W} \bullet x_n) \begin{pmatrix} y_1 - \boldsymbol{W} \bullet x_1 \\ y_2 - \boldsymbol{W} \bullet x_2 \\ \dots \\ y_n - \boldsymbol{W} \bullet x_n \end{pmatrix} \end{aligned} \quad (1\text{-}7)$$

令$Y=(y_1,y_2\dots y_n)$,$\bar{X}=(x_1,x_2\dots x_n)=\begin{pmatrix} {x^{(0)}}_1\dots {x^{(0)}}_n \\ \cdots \\ \cdots \\ {x^{(m)}}_1\dots {x^{(m)}}_n \end{pmatrix}$

那么式子（1-7）可表示为：

$$L(W)=(Y-\boldsymbol{W}\bar{X})(Y-\boldsymbol{W}\bar{X})^T \quad (1-8)$$

显然上式是仅与$\boldsymbol{W}$有关的变量，$L(W)$也就是需要优化的目标函数，这就把回归问题转变为最优化问题。最优化问题就是求函数的极小值，那么只需对$L(W)$求导数，最终找到特定的$\boldsymbol{W}$使$L(W)$最小化。

$L(W)$关于$\boldsymbol{W^T}$的导数为：

$$\frac{d}{d\boldsymbol{W^T}}(Y-\boldsymbol{W}\bar{X})(Y-\boldsymbol{W}\bar{X})^T=\frac{d}{d\boldsymbol{W^T}}YY^{\boldsymbol{T}}-\frac{d}{d\boldsymbol{W^T}}\boldsymbol{W}\bar{X}Y^{\boldsymbol{T}}-\frac{d}{d\boldsymbol{W^T}}Y\bar{X}^T\boldsymbol{W^T}+\frac{d}{d\boldsymbol{W^T}}\boldsymbol{W}\bar{X}\bar{X}^T\boldsymbol{W^T}$$
$$=\boldsymbol{2}\bar{X}\bar{X}^T\boldsymbol{W^T}-\boldsymbol{2}\bar{X}Y^T=\boldsymbol{2}\bar{X}(\bar{X}\boldsymbol{W^T}-Y^T) \quad (1-9)$$

令导数为零，解得$\boldsymbol{W}$的估计为：

$$\widehat{\boldsymbol{W^T}}=(\bar{X}\bar{X}^T)^{-1}\bar{X}Y^T \quad (1-10)$$

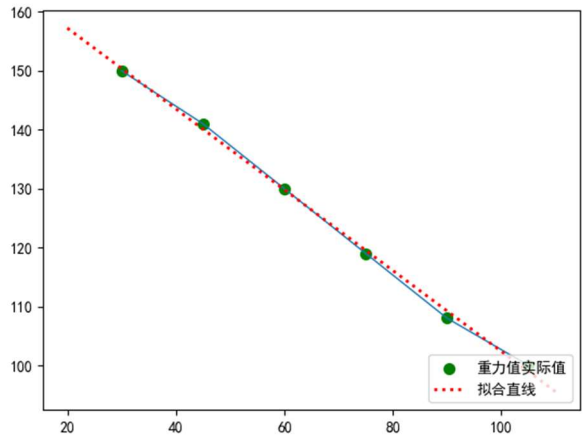
1.3.4 python实现【注意：此处用到的数据表介绍请转至3.2节查看】

${y_i}^{(1)}$重力值（g.u）	150	141	130	119	108	100
${x_i}^{(1)}$测量时间（min）	30	45	60	75	90	105

最小二乘使用数据：唯一（$j=1$）的样本集${x_i}^{(1)}|_{i=1\dots6}$是测量时间，标签${y_i}^{(1)}$是重力值。

```
'''
para X:矩阵，样本特征矩阵
Para Y:矩阵，标签向量
return：矩阵，回归系数
'''
times = [30,45,60,75,90,105]       #测量时间列表（样本）
gravity = [150,141,130,119,108,100]#重力数据列表（标签）
import numpy as np
def least_square(X,Y):          #定义最小二乘拟合函数
     W=(X*X.T).I*X*Y.T         #python提供了方便的转置与逆运算（.T/.I）
     return W
X=np.mat([[1,1,1,1,1,1],times]) #测量时间矩阵（样本）
Y=np.mat(gravity)              #重力数据矩阵（标签）
W=least_square(X,Y)
import matplotlib.pyplot as plt          #导入画图库函数
plt.rcParams['font.sans-serif']=['SimHei']#画图准备
plt.rcParams['axes.unicode_minus']=False
plt.scatter(times,gravity,color="green",label="重力值实际值",linewidth=2)
plt.plot(times,gravity,linewidth=1) #画实际重力测量数据
x1=np.linspace(20,110,5)           #拟合直线x范围
y1=W[1,0]*x1+W[0,0]               #拟合直线纵坐标取值
plt.plot(x1,y1,color="red",label="拟合直线",linewidth=2,linestyle=':')
plt.legend(loc='lower right')       #画拟合直线
plt.show()                         #展示图像
new_times=[120,140,180]
new_times=(np.mat(new_times)).T   #需要预测的时间（新样本）
forecast_gravity=W[1,0]*new_times+W[0,0]   #预测结果（预测值即标签）
print(forecast_gravity)                    #打印
loss_value1=0
for i in range(6):
    loss_value1 +=(W[1,0]*times[i]+W[0,0]-gravity[i])**2
    print(W[1,0]*times[i]+W[0,0])
    print(gravity[i])                  #输出拟合数据与实际数据
print(loss_value1)                      #计算损失值
```

绘图结果：




```
[[88.66666667]
 [74.95238095]
 [47.52380952]]
150.38095238095232
150
140.09523809523807
141
129.8095238095238
130
119.52380952380955
119
109.23809523809527
108
98.952380952381
100
3.9047619047618944
```

右图的数据依次为：前3行，当新的样本集为new_times=[120,140,180]时，最小二乘法预测得到的重力数据为forecast_gravit-y=[88.6666,74.9523,47.5238]；4-15行：实际数据与拟合数据的输出对比；最后一行：损失值为loss_value1 = 3.90476。

2 梯度下降法

2.1 引入

像前面所述最小二乘法在面临大数据量时，存在效率低的问题，并且大部分机器学习问题十分复杂，难以用数学模型来表达，因此机器学习模型需要采用最优化方法来解决。最优化方法有很多，应用最多的是导数方法如：梯度下降法、牛顿法、拟牛顿法等。

对于最优化问题来说，很难像线性回归那样求得解析解，结合计算机运算快速的特点，一般使用迭代法（Iteration）来求数值解。

2.2 迭代法

2.2.1 简介

一言以蔽之，通过迭代关系式，由粗略解逐步逼近精确解。

例如对下面方程求解

$$x^3+\frac{e^x}{2}+5x-6=0 \quad (2-1)$$

按照现在的知识水平，很难将上面方程求出解析解，这是显而易见的。那么就建立迭代关系式：

$$x^*=x=\frac{6-x^3-\frac{e^x}{2}}{5} \quad (2-2)$$

利用计算机语言实现时，只需输入初始x值，通过迭代的方式不断更新x^*。当在给定的循环结束条件结束时，x^*逐渐收敛到一个数值A，那么该值就是方程的解。

2.2.2 python代码

```
import math
x = 0#初值
for i in range(100):
    x = (6-x**3-(math.e**x)/2.0)/5.0
print(str(i)+":"+str(x))#返回string格式
```

2.2.3 运行结果

```
0:1.1
1:0.6333833976053566
2:0.9607831386993697
3:0.7612451427547097
4:0.8976785421774022
5:0.8099353339786866
6:0.8689609915826384
7:0.8303274798888033
8:0.8561001981471914
9:0.8391152621768908
10:0.8504015176669936
11:0.8429422148469066
12:0.8478899896983665
13:0.8446158829987237
14:0.8467858846473397
15:0.8453491541692075
16:0.8463010523357257
17:0.8456706653332824
18:0.8460882605809683
19:0.8458116831312754
20:0.8459948874407963
21:0.8458735439749878
22:0.8459539192397146
23:0.8459006824726572
24:0.8459359448863271
25:0.8459125885326444
26:0.8459280589817704
27:0.8459178119620006
28:0.8459245992200859
29:0.8459201035986089
30:0.8459230813336305
31:0.8459211089938277
32:0.8459224153988485
33:0.8459215500849682
```

显然在第28次循环之后，数值解收敛于$A = 0.8452$，解的精度10^{-4}。

2.3 梯度下降法

2.3.1 简介

梯度下降法(gradient descent)是迭代法种利用导数进行优化的算法，是求解机器学习模型参数时最常用的方法。

其基本思想是通过迭代的方式不断地使x沿着损失函数的负梯度方向前进，最终走到函数的最小值对应的x值。

2.3.2 表示形式

梯度下降法本身也是迭代法的一种，因此建立迭代关系式：

$$x_{i+1} = x_i - \alpha * \frac{df(x)}{dx}|_{x=x_i} \quad (2-3)$$

其中α是步长即学习率。与迭代法类比可得：x_{i+1}就是x^*；x_i就是x。使用梯度下降法仍去解方程（2-1）。

2.3.3 python代码

```
import numpy as np
import math
def f(x):#原函数
    return x**3+(math.e**x)/2.0+5.0*x-6
def loss_fun(x):#损失函数，因为目标是求原函数的根，所以损失函数为（原函数-0）**2
    return (f(x))**2
def calcu_grad(x):#近似求损失函数的导数
    delta = 0.0000001
    return(loss_fun(x+delta)-loss_fun(x-delta))/(2.0*delta)
alpha = 0.01#步长（实际先大后小的思想尝试）
maxtimes =100#迭代次数
x=0.0
for i in range(maxtimes):#迭代思想、梯度下降
    x=x-alpha*calcu_grad(x)
print(str(i)+":"+str(x))
```

2.3.4 运行结果

```
0:0.605000000319933
1:0.8628136609623027
2:0.839024136216164
3:0.8484812908792372
4:0.8449346960598381
5:0.846297217984478
6:0.8457784053613508
7:0.8459766363690868
8:0.8459009940003159
9:0.845929872570837
10:0.8459188494777913
11:0.8459230573531434
12:0.8459214511140406
13:0.8459220642575387
14:0.8459218302055683
15:0.8459219195491066
16:0.8459218854444456
17:0.8459218984630497
```


观察结果可知，在第11次迭代时，数值解已经收敛于0.8452。对比简单迭代法的28次，运行速度是其2.54倍，这也就是梯度下降法的优势。

2.4 梯度下降法解线性回归问题

了解完梯度下降法的基本操作之后，就可以将其应用于线性回归问题，在前面1.3.2中介绍的损失函数仍是本节的优化目标函数。将线性回归问题中m个样本的损失函数表示为：

$$L(\boldsymbol{W}) = \frac{1}{2}[s_1^2(\boldsymbol{W}) + s_2^2(\boldsymbol{W}) + \cdots + s_m^2(\boldsymbol{W})] = \frac{1}{2}\sum_{i=1}^{m} s_i^2(\boldsymbol{W}) \quad (2-4)$$

由（2-3）可知，回归系数矩阵（相当于前面的x^*或x_{i+1}）的更新过程为对每一个样本集j（此处是指每一个样本集都对应m个样本以及一个回归系数矩阵$\boldsymbol{W}^{(j)}$）而言：

$$\boldsymbol{W}_{l+1}^{(j)} = \boldsymbol{W}_{l}^{(j)} - \alpha * \frac{dL(\boldsymbol{W})}{d\boldsymbol{W}^{(j)}} \quad (2-5)$$

其中：

$$\frac{\partial L(\boldsymbol{W})}{\partial \boldsymbol{W}^{(j)}} = \frac{1}{2}\sum_{i=1}^{m}\frac{\partial s_i^2(\boldsymbol{W})}{\partial \boldsymbol{W}^{(j)}} = \frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial \boldsymbol{W}^{(j)}}(y_i - f(x_i))^2$$

$$= -\sum_{i=1}^{m}((y_j - f(x_j))\frac{\partial f(x_i)}{\partial \boldsymbol{W}^{(j)}} = -\sum_{i=1}^{m}((y_i - f(x_i)) * {x_i}^{(j)}$$

$$= -\sum_{i=1}^{m}(y_i - \sum_{k=0}^{n}{x_i}^{(k)} * \boldsymbol{W}^{(k)}) * {x_i}^{(j)} \quad (2-6)$$

式（2-6）是整个梯度下降实现回归问题的核心公式，其中每一个j都对应一个长度为n+1的样本，为将复杂问题简单化，下面一章以一个样本集为例来具体实现整个过程。

3 应用

3.1 背景

当前学期除了《机器学习》这门必修课之外，专业课程《重力与固体潮》的课程讲师也提到了在重力数据处理中线性回归的重要地位。结合当前对人才培养的全面型、综合型、应用型要求，可以把机器学习中的梯度下降法用来解决重力测量数据线性回归问题。

3.2 样本数据及标签

$y_i^{(1)}$重力值（g.u）	150	141	130	119	108	100
$x_i^{(1)}$测量时间（min）	30	45	60	75	90	105

唯一（$j=1$）的样本集$x_i^{(1)}|_{i=1\ldots6}$是测量时间，标签$y_i^{(1)}$是重力值，根据以上分析则可以编写程序。目标是预测测量时间为120、140、180（min）时重力值的大小。

3.3 python代码

```
'''
x是样本集
y是特征（标签）也可以理解为函数值
w是线性回归模型的参数，目标矩阵
gradient是梯度计算函数
loss_function是损失函数
上述三者均为矩阵
'''
#计算梯度
import numpy as np
def gradient(x,y,w):                    #定义梯度函数
    m,n = np.shape(x)                   #获取x的行、列数分别赋给m,n
    g = np.mat(np.zeros((n,1)))    #设置一个n行1列的0值矩阵给变量g
    for i in range(m):
        m_v = y[i,0] - x[i,] * w  #一阶梯度中间值的计算
        for j in range(n):
            g[j,] -= m_v * x[i,j]  #对每一特征j计算一阶梯度
    return g
#计算损失函数
def loss_function(x,y,w):
    k = y - x*w                         #利用误差值的平方来作为此次优化问题的损失函数
    return k.T*k/2                      #矩阵与自身的逆矩阵相乘
#主函数
times = [30,45,60,75,90,105]      #自习时长列表
gravity = [150,141,130,119,108,100]#学生数列表
x = (np.mat([[1,1,1,1,1,1],times])).T   #时间矩阵
y = (np.mat(gravity)).T             #学生数矩阵
w = (np.mat([0.0,0.0])).T          #回归模型系数矩阵
print(w)
loop_number = 0
step = 0.000060                     #步长
loss_change=0.000001                #迭代精度
loss = loss_function(x,y,w)
for i in range(200000):             #这个迭代次数可以设置大一些
    loop_number = loop_number + 1 #每次迭代循环次数加1
    w = w - step * gradient(x,y,w)#回归系数的梯度下降，这是梯度下降的核心*
    loss_value2 = loss_function(x,y,w)#梯度下降后的新损失值
    print(str(i)+":"+str(w[0])+":"+str(w[1]))
    print(loss_value2)
    if abs(loss - loss_value2) < loss_change:
        break                       #如果损失值变化很小了那么结束循环即可
    loss = newloss                  #如果损失值变化还较大，那么就更新损失值并继续
new_time = [120,140,180]           #本模型期望预测的时间列表
new_time = (np.mat([[1,1,1],new_time])).T
forecast_gravity = new_time * w
print(forecast_gravity)
```

3.4 运行结果

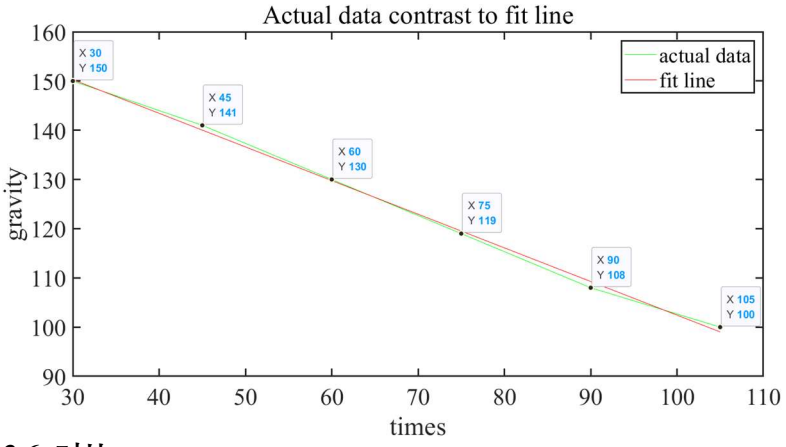
```
152434:[[170.78147695]]:[[-0.68350109]]
[[1.96341282]]
[[88.76134652]
 [75.09132478]
 [47.75128131]]
```

上图数据依次为：迭代次数 N=152434；偏置常量b=170.78148；回归系数矩阵为$\boldsymbol{W}$=[-0.683501]；损失值为loss_value2=1.963412；新样本new_times=[120,140,180]的预测重力值为：forecast_gravity=[88.76134652, 75.09132478, 47.75128131]。上述结果基于步长=0.00006、拟合精度为0.000001时。另外因为只有一个样本集，所以$\boldsymbol{W}$为一常数而不是矩阵。

3.5 matlab画拟合曲线

```
x=[30,45,60,75,90,105]
y=[150,141,130,119,108,100]
plot(x,y,'g')
hold on
y = -0.6835*x+170.7814
plot(x,y,'r')
xlabel('times');
ylabel('gravity');
legend('actual data','fit line')
set(gcf,'Units','centimeters','Position',[20 10 30 15]);%中括号里分别代表成图的位置和大小
title('Actual data contrast to fit line')
set(gca, 'LineWidth',1.5,'Fontname', 'times new roman','fontsize',22)
```

绘制结果：



3.6 对比

前面所用最小二乘法与梯度下降法拟合效果肉眼观察均不错。其中损失值差异明显，最小二乘法实现拟合数据的损失值为loss_value1=3.90476，梯度下降法损失值：为loss_value2=1.963412。不难得出梯度下降法是精确度更高得的最优化方法。尽管在精度为0.000001要求下，梯度下降法的迭代次数达到了15万次，这仍不能撼动其在迭代法中的地位，目前的小数据量可能体现不出它的便捷性，等日后接触专业领域大量的地震、重力数据时，这项工作可以为使用梯度下降法优化数据处理过程提供基础思路。

4 总结

（1）回顾了线性回归问题、最小二乘法求解线性回归问题的基本思路与公式推导，编写了实现最小二乘法的python语言代码并成功运行。

（2）回顾了损失函数、迭代法、梯度下降法的基本概念及相关公式表示形式，编写了实现迭代法解方程根、梯度下降法解方程根的python语言代码并成功运行。

（3）推导了梯度下降法在线性回归问题中的应用公式，编写了迭代形式的梯度下降法求解实际重力数据线性回归问题的python语言代码，利用matlab语言对拟合前后数据进行绘图对比；把最小二乘法与梯度下降法的拟合结果进行对照，通过损失值的对比证明了了梯度下降法的优越性。

（4）将专业知识与《机器学习》课程相联系，并达到了小组成员之间相互配合、共同探讨的目的。

5 不足

在给定的时间内，没有完成本来设想的多个样本集基础上的线性回归计算，在今后的学习过程中小组成员之间还会继续探讨此问题，争取把梯度下降法在线性回归中的应用方法灵活掌握并进一步应用到专业领域。