课程设计报告

课 程 名 称：机器学习课程设计

年 级 专 业：2020 级智能科学与技术

学 生 姓 名：孙成

学　　号：20203101694

课 程 名 称：机器学习

任 课 老 师：胡祝华教授

平时成绩	
报告成绩	
总评成绩	
任课教师签名	

海南大学·信息与通信工程学院

School of Information and Communication Engineering, Hainan University

2022 年 12 月

目录

设计题目：基于线性回归的 PM 指标预测模型设计与实现

1 选题目的

我出生于合肥，一座新兴的科教工业城市，小时候的空气是自由干净的，可如今随着城市的不断发展，空气质量却是肉眼可见的恶化，我无比怀念小时候的干净空气，长大后通过高考我选择了空气质量位居全国前列的城市——海南省海口市。值此机器学习的机会，让我能实现基于线性回归分析空气质量的课程设计。本人负责此次项目的训练和评价部分，数据集源自于 kaggle 的开源数据集，数据收集地点是合肥丰原。

2 机器学习方法原理和方案

2.1 回归模型

2.1.1 什么是线性回归与非线性回归

1) 在统计学中，线性回归是利用称为线性回归方程的最小二乘函数对一个或多个自变量和因变量之间关系进行建模的一种回归分析。这种函数是一个或多个称为回归系数的模型参数的线性组合。当因变量和自变量之间高度相关时，我们就可以使用线性回归来对数据进行预测。
2) 线性回归模型，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。其表达形式为 y = w'x+e，e 为误差服从均值为 0 的正态分布。线性回归模型是利用称为线性回归方程的最小平方函数对一个或多个自变量和因变量之间关系进行建模的一种回归分析。这种函数是一个或多个称为回归系数的模型参数的线性组合。只有一个自变量的情况称为简单回归,大于一个自变量情况的叫做多元回归。
3) 非线性回归，是在掌握大量观察数据的基础上，利用数理统计方法建立因变量与自变量之间的回归关系函数表达式（称回归方程式）。回归分析中，当研究的因果关系只涉及

因变量和一个自变量时，叫做一元回归分析；当研究的因果关系涉及因变量和两个或两个以上自变量时，叫做多元回归分析。

4) 区别：线性回归与非线性回归通过自变量的指数来区分

2.1.2 简单线性回归的原理及数学推导

简单线性回归为一元线性回归模型，是指模型中只含有一个自变量和因变量，该模型的数学公式可以表示为 y=ax+b+ε，a 为模型的斜率，b 为模型的截距，ε 为误差。

最小二乘法(OLS)是最小化残差的平方和。对于简单线性回归，函数可以用可以用一个公式来表示，假设 x 和 y 之间存在这样的关系：

$$\hat{y_i} = ax_i + b$$

其中等号左边带的 y 是我们 y 的预测值，实际情况时我们预测的数值与实际数值之间的差异，就是残差，即：

$$\text{error } = y - \hat{y}$$

残差平方和，就是：

$$RSS(SSE) = \sum_{i=1}^{n} (y_i - \hat{y_i})^2 = \sum_{i=1}^{n} [y_i - (ax_i + b)]^2$$

我们的目的是使得预测值尽可能地接近实际值，即残差越小越好。也就是说，当我们找到一组（a, b），使得残差平方和最小时，就说明在某种程度上，我们找到了预测效果最好的简单线性回归模型。

残差的绝对值是一个很好的指标。但是对机器和数学来说，计算、表示残差的平方和时，远比计算和表示残差的绝对值要来得方便；同时，由于平方和的数据形态是一个 U 型的曲线，方便我们通过求导、随机梯度下降等方式得到最小值。

重点：求导（最小二乘法）、梯度下降两种方式，

接下来求解系数(a, b)，如前边所言，残差平方和的数据形态是一个 U 型曲线，因此当且仅当导数为 0 时，我们得到最低点，重点是导数为 0 时，此时为残差平方和曲线的最低点，即此时的残差最小，结果最接近实际值。

求解过程：求导--->导数结果为 0 时为正确结果--->求出 a、b 的计算公式

1) 求解 b

J(a, b)代表损失函数，在这个例子中，就是我们上边提到的残差平方和。先对 b 求导，并使导数等于 0：

$$J(a,b)=\sum_{i=1}^{n}[y_i-(ax_i+b)]^2$$

$$\Rightarrow\frac{\partial J(a,b)}{\partial b}=\sum_{i=1}^{n}2[y_i-(ax_i+b)](-1)=0$$

$$\Rightarrow\sum_{i=1}^{n}(y_i-ax_i-b)=0$$

$$\Rightarrow\sum_{i=1}^{n}y_i-a\sum_{i=1}^{n}x_i-\sum_{i=1}^{n}b=0$$

$$\Rightarrow b=\frac{1}{n}\sum_{i=1}^{n}y_i-\frac{1}{n}a\sum_{i=1}^{n}x_i$$

$$\Rightarrow b=\bar{y}-a\bar{x}$$

2) 求解 a

$$\frac{\partial J(a,b)}{\partial a}=\sum_{i=1}^{n}2[y_i-(ax_i+b)](-x_i)=0$$

$$\Rightarrow\sum_{i=1}^{n}(y_ix_i-ax_i^2-bx_i)=0$$

$$b=\bar{y}-a\bar{x}\Rightarrow\sum_{i=1}^{n}(y_ix_i-ax_i^2-\bar{y}x_i+a\bar{x}x_i)=0$$

$$\Rightarrow\sum_{i=1}^{n}(y_ix_i-\bar{y}x_i)-a\sum_{i=1}^{n}(x_i^2-\bar{x}x_i)=0$$

$$\Rightarrow a = \frac{\sum_{i=1}^{n}(y_i x_i - \bar{y}x_i)}{\sum_{i=1}^{n}(x_i^2 - \bar{x}x_i)}$$

又因为：

$$\sum_{i=1}^{n} x_i \bar{y} = n\bar{x}\bar{y}, \sum_{i=1}^{n} x_i \bar{x} = n\bar{x}^2$$

$$\Rightarrow a = \frac{\sum_{i=1}^{n}\left(y_i x_i - \bar{y}x_i - \bar{x}y_i + \bar{x}\bar{y}\right)}{\sum_{i=1}^{n}(x_i^2 - \bar{x}x_i - \bar{x}x_i + \bar{x}^2)}$$

$$= \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

最后的结果 现在我们成功地推导出了(a, b)的求解公式，即：

$$a = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$b = \bar{y} - a\bar{x}$$

2.1.3 使用机器学习模块构建线性回归模型

这里使用机器学习的 sklearn 模块，sklearn 是一个 Python 第三方提供的非常强力的机器学习库，它包含了从数据预处理到训练模型的各个方面。在实战使用 scikit-learn 中可以极大的节省我们编写代码的时间以及减少我们的代码量，使我们有更多的精力去分析数据分布，调整模型和修改超参。

sklearn 拥有可以用于监督和无监督学习的方法，一般来说监督学习使用的更多。sklearn 中的大部分函数可以归为估计器(Estimator)和转化器(Transformer)两类。

- 估计器(Estimator)其实就是模型，它用于对数据的预测或回归。基本上估计器都会有以下几个方法：

fit(x,y):传入数据以及标签即可训练模型，训练的时间和参数设置，数据集大小以及数据本身的特点有关

score(x,y)用于对模型的正确率进行评分(范围 0-1)。但由于对在不同的问题下，评判模型优劣的的标准不限于简单的正确率，可能还包括召回率或者是查准率等其他的指标，特别是对于类别失衡的样本，准确率并不能很好的评估模型的优劣，因此在对模型进行评估时，不要轻易的被 score 的得分蒙蔽。

predict(x)用于对数据的预测，它接受输入，并输出预测标签，输出的格式为 numpy 数组。我们通常使用这个方法返回测试的结果，再将这个结果用于评估模型。

- 转化器(Transformer)用于对数据的处理，例如标准化、降维以及特征选择等等。同与估计器的使用方法类似:

fit(x,y):该方法接受输入和标签，计算出数据变换的方式。

transform(x):根据已经计算出的变换方式，返回对输入数据 x 变换后的结果（不改变 x）

fit_transform(x,y) :该方法在计算出数据变换方式之后对输入 x 就地转换。

相关代码如下：

```
# 导入线性回归模块
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']
rng = np.random.RandomState(5) #选定随机数种子，这样生成的随机数不会改变
xtrain = 10 * rng.rand(30)
ytrain = 8+4*xtrain + rng.rand(30)  #样本关系近似 y=8+4x
fig = plt.figure(figsize=(16,5))
ax1 = fig.add_subplot(121)
plt.scatter(xtrain,ytrain,marker='.',color='k')
plt.title('样本数据散点图')
model = LinearRegression() #创建线性回归模型
model.fit(xtrain.reshape(-1, 1),ytrain) #放入参数,需要将 xtrain 转换为列维度，因为是矩阵运算
#查看结果
model.coef_  #查看斜率，即 y=ax+b 中的 a
model.intercept_ #查看截距，即 y=ax+b 中的 b
print('计算出来的值，a=%f,,,,b=%f'%(model.coef_,model.intercept_))
print('拟合函数为，y = %f * x  + %f'%(model.coef_,model.intercept_))
```

```
print('与原散点数值样本关系 y = 4x + 8 近似')
#进行测试
xtest = np.linspace(0,10,100) #返回一个等差数列
ytest = model.predict(xtest.reshape(-1,1))  #使用 predict 进行预测
#新建一个预测值的图表
ax2 = fig.add_subplot(122)
#fig = plt.figure(figsize=(8,5))
plt.scatter(xtrain,ytrain,marker='.',color='k',label='原值')
plt.plot(xtest,ytest,color='b',label='拟合线')
plt.title('回归函数与原值')
plt.legend()
plt.show()
# sklearn 的误差值
plt.scatter(xtrain,ytrain,marker='.',color='k',label='原值')
ytest2 = model.predict(xtrain.reshape(-1,1)) #样本数据在拟合线上的 y 值
plt.plot(xtest,ytest,color='b',label='拟合线')
#plt.scatter(xtrain,ytest2,marker='x',color='k',label='原值')
plt.plot([xtrain,xtrain],[ytrain,ytest2],color='g')  #误差线
plt.title('误差线图')
plt.show()
```

结果如下图：

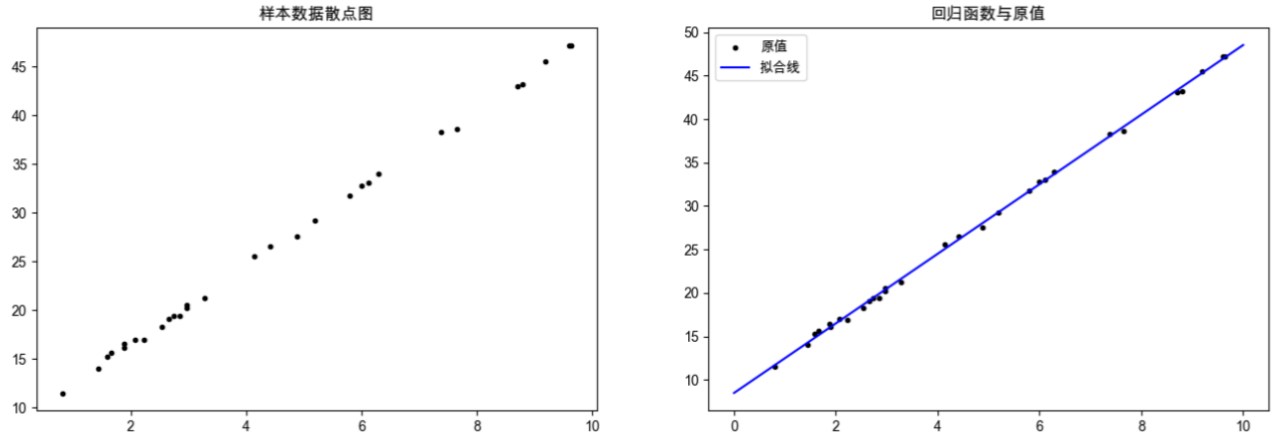

图 1 python 实现线性回归

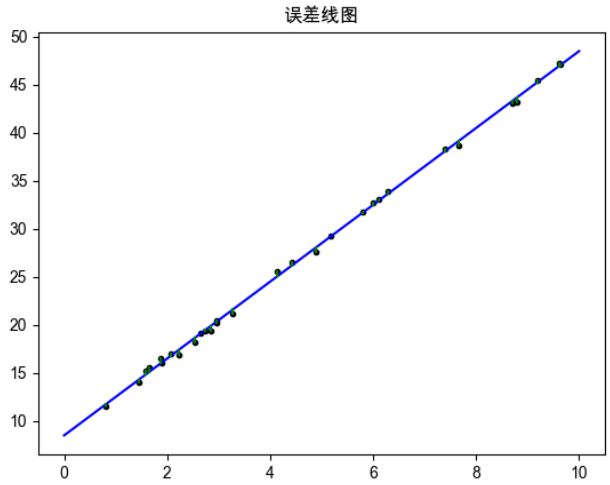

图 2 误差线图

```
/Users/suncheng/opt/anaconda3/envs/SS/bin/python /Users/suncheng/PycharmPr
计算出来的值，a=4.002743,,,,b=8.495176
拟合函数为，y = 4.002743 * x  + 8.495176
与原散点数值样本关系y = 4x + 8 近似

进程已结束,退出代码0
```

图 3 输出结果

2.1.4 线性回归模型评估

这里需要对该模型的进行检验评价，有以下几种方式

1) SSE(误差平方和):即求解实际值与在拟合线上的预测值的误差的平方和

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

其中 SSE 越接近于θ说明拟合的效果越好

2) MSE(均方差、方差)：

$$MSE = SSE/n = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

3) RMSE(均方根、标准差)：

该统计参数，也叫回归系统的拟合标准差，是 MSE 的平方根，公式如下:

$$RMSE = \sqrt{MSE} = \sqrt{\mathrm{SSE}/n} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

4) R-squre

上面的几种衡量标准针对不同的模型会有不同的值， 数字之间没有可比性， 没有什么可读性，这里就有一种新的评价方法那就是 RSquare

计算逻辑为：

SSR:Sum of squares of the regression，即预测数据与原始数据均值之差的平方和

$$SST = \sum_{i=1}^{n}(\hat{y}_i - \bar{y}_i)^2$$

SST:Total sum of squares，即原始数据和均值之差的平方和(即为这组数据的方差)

$$SST = \sum_{i=1}^{n}(y_i - \bar{y}_i)^2$$

前提公式：SST = SSE + SSR

最终的公式为：

$$R-\ \mathrm{square}\ = \frac{SSR}{SST} = \frac{SST - SSE}{SST} = 1 - \frac{SSE}{SST}$$

R-square“确定系数”是定义为 SSR 和 SST 的比值，值越接近 1，表明方程的变量对 y 的解释能力越强，这个模型对数据拟合的也较好

2.2 矩阵分解模型

2.2.1 显式数据和隐式数据

矩阵分解(Matrix Factorization, MF)用到的用户行为数据分为显式数据和隐式数据两种。显式数据是指用户对 item 的显式打分，比如用户对电影、商品的评分，通常有 5 分制和 10 分制。隐式数据是指用户对 item 的浏览、点击、购买、收藏、点赞、评论、分享等数据，其特点是用户没有显式地给 item 打分，用户对 item 的感兴趣程度都体现在他对 item 的浏览、点击、购买、收藏、点赞、评论、分享等行为的强度上[1,4]。

显式数据的优点是行为的置信度高，因为是用户明确给出的打分，所以真实反映了用户对 item 的喜欢程度。缺点是这种数据的量太小，因为绝大部分用户都不会去给 item 评分，这就

导致数据非常稀疏，同时这部分评分也仅代表了小部分用户的兴趣，可能会导致数据有偏。隐式数据的优点是容易获取，数据量很大。因为几乎所有用户都会有浏览、点击等行为，所以数据量大，几乎覆盖所有用户，不会导致数据有偏。其缺点是置信度不如显式数据的高，比如浏览不一定代表感兴趣，还要看强度，经常浏览同一类东西才能以较高置信度认为用户感兴趣。

根据所使用的数据是显式数据还是隐式数据，矩阵分解算法又分为两种[4,6]。使用显式数据的矩阵分解算法称为显式矩阵分解算法，使用隐式数据的矩阵分解算法称为隐式矩阵分解算法。从实际应用的效果来看，隐式矩阵分解的效果一般会更好。

2.2.2 矩阵分解

矩阵分解算法的输入是 user 对 item 的评分矩阵(图 4 等号左边的矩阵)，输出是 User 矩阵和 Item 矩阵(图 4 等号右边的矩阵)，其中 User 矩阵的每一行代表一个用户向量，Item 矩阵的每一列代表一个 item 的向量。User 对 item 的预测评分用它们的向量内积来表示，通过最小化预测评分和实际评分的差异来学习 User 矩阵和 Item 矩阵[7,10]。

隐式矩阵分解与显式矩阵分解的一个比较大的区别，就是它会去拟合评分矩阵中的零，即没有评分的地方也要拟合。

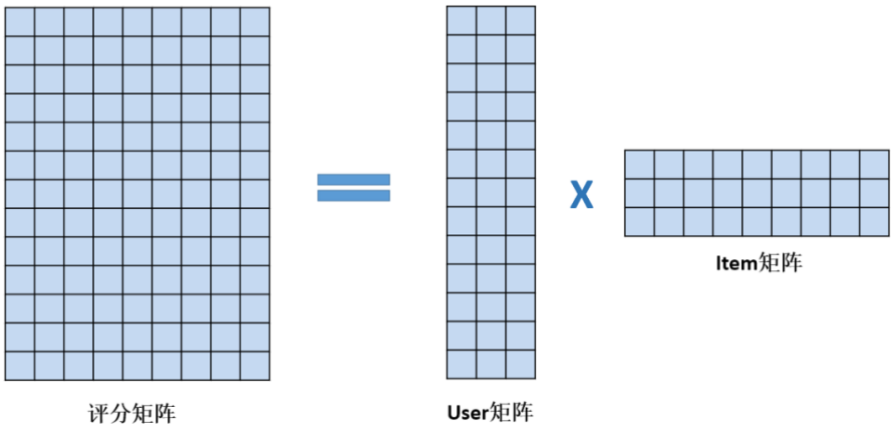

图 4 显式矩阵分解

2.2.3 显式矩阵分解求解方法

为了用数学的语言定量表示上述思想，我们先引入一些符号。设 rui 表示用户 u 对 item i 的显式评分，当 rui >0 时，表示用户 u 对 item i 有评分，当 rui =0 时，表示用户 u 对 item i 没有评分，xu 表示用户 u 的向量，yi 表示 item i 的向量，则显式矩阵分解的目标函数为：

$$\min_{X,Y} \sum_{r_{ii}\neq 0} (r_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u \|x_u\|_2^2 + \sum_i \|y_i\|_2^2 \right)$$

其中 *xu* 和 *yi* 都是 *k* 维的列向量，*k* 为隐变量的个数，

$$X = [x_1, x_2, \cdots, x_N]$$

是所有 *xu* 构成的矩阵，

$$Y = [y_1, y_2, \cdots, y_M]$$

为所有 *yi* 构成的矩阵，*N* 为用户数，*M* 为 item 数，λ 为正则化参数。

在上述公式中

$$x_u^T y_i$$

为用户向量与物品向量的内积，表示用户 *u* 对物品 *i* 的预测评分，目标函数通过最小化预测评分和实际评分 r_{ui} 之间的残差平方和，来学习所有用户向量和物品向量。这里的残差项只包含了有评分的数据，不包括没有评分的数据。目标函数中第二项是 L2 正则项，用于保证数值计算稳定性和防止过拟合[10,11]。

求解 X 和 Y 采用的是交替最小二乘法(alternative least square, ALS)，也就是先固定 X 优化 Y ，然后固定 Y 优化 X ，这个过程不断重复，直到 X 和 Y 收敛为止。每次固定其中一个优化另一个都需要解一个最小二乘问题，所以这个算法叫做交替最小二乘方法[12]。

1) *Y* 固定为上一步迭代值或初始化值，优化 *X*：

此时，Y 被当做常数处理，目标函数被分解为多个独立的子目标函数，每个子目标函数对应一个用户。对于用户 u ，目标函数为：

$$\min_{x_u} \sum_{r_{iii}\neq 0} (r_{ui} - x_u^T y_i)^2 + \lambda \|x_u\|_2^2$$

这里面残差项求和的个数等于用于 u 评过分的物品的个数，记为 m 个。把这个目标函数化为矩阵形式，得

$$J(x_u) = (R_u - Y_u^T x_u)^T (R_u - Y_u^T x_u) + \lambda x_u^T x_u$$

其中

$$R_u = \left[r_{ui_1}, \cdots, r_{ui_m} \right]^T$$

表示用户 *u* 对这 *m* 个物品的评分构成的向量

$$Y_u = \left[y_{i_1}, y_{i_2}, \cdots, y_{i_m} \right]$$

表示这 *m* 个物品的向量构成的矩阵，顺序跟 *Ru* 中物品的顺序一致。

对目标函数 J 关于 *xu* 求梯度，并令梯度为零，得：

$$\frac{\partial J(x_u)}{\partial x_u} = -2Y_u(R_u - Y_u^T x_u) + 2\lambda x_u = 0$$

解这个线性方程组，可得到 *xu* 的解析解为：

$$x_u = (Y_u Y_u^T + \lambda I)^{-1} Y_u R_u$$

2) *X* 固定为上一步迭代值或初始化值，优化 *Y*：

此时，*X* 被当做常数处理，目标函数也被分解为多个独立的子目标函数，每个子目标函数对应一个物品。类似上面的推导，我们可以得到 y_i 的解析解为：

$$y_i = (X_i X_i^T + \lambda I)^{-1} X_i R_i$$

其中

$$R_i = \left[r_{u_1 i}, \cdots, r_{u_n i}\right]^T$$

表示 n 个用户对物品 i 的评分构成的向量

$$X_i = \left[x_{u_1 i}, x_{u_2 i}, \cdots, x_{u_n i}\right]$$

表示这 n 个用户的向量构成的矩阵，顺序跟 Ri 中用户的顺序一致。

2.2.4 隐式矩阵分解求解方法

我们仍然用 r_{ui} 表示用户 *u* 对物品 *i* 的评分，但这里的评分表示的是行为的强度，比如浏览次数、阅读时长、播放完整度等。当 $r_{ui}>0$ 时，表示用户 *u* 对物品 i 有过行为，当 $r_{ui}=0$ 时，表示用户 *u* 对物品 i 没有过行为。首先，我们定义一个二值变量 p_{ui} 如下[13]：

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases}$$

这个 pui 是一个依赖于 rui 的量，用于表示用户 u 对物品 i 是否感兴趣，也称为用户偏好。当用户 u 对物品 i 有过行为时，我们认为用户 u 对物品 i 感兴趣，此时 pui =1；当用户 u 对物品 i 没有过行为时，我们认为用户 u 对物品 i 不感兴趣，此时 pui =0。

模型除了要刻画用户对物品是否感兴趣外，而且还要刻画感兴趣或不感兴趣的程度，所以这里的隐式矩阵分解还引入了置信度的概念。从直观上来说，当 rui >0 时，rui 越大，我们越确信用户 u 喜欢物品 i ，而当 rui =0 时，我们不能确定用户 u 是否喜欢物品 i ，没有行为可能只是因为用户 u 并不知道物品 i 的存在。

因此，置信度是 rui 的函数，并且当 rui >0 时，置信度是 rui 的增函数；当 rui =0 时，置信度取值要小。论文中给出的置信度 cui 的表达式为：

$$c_{ui} = 1 + \alpha r_{ui}$$

当 rui >0 时，cui 关于 rui 线性递增，表示对于有评分的物品，行为强度越大，我们越相信用户 u 对物品 i 感兴趣；当 rui =0 时，置信度恒等于 1，表示对所有没有评分的物品，用户不感兴趣的置信度都一样，并且比有评分物品的置信度低。用 xu 表示用户 u 的向量，yi 表示 item i 的向量，引入置信度以后，隐式矩阵分解[6]的目标函数为：

$$\min_{X,Y} \sum_{u=1}^{N} \sum_{i=1}^{M} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_{u=1}^{N} \|x_u\|_2^2 + \sum_{i=1}^{M} \|y_i\|_2^2 \right)$$

其中 *xu* 和 *yi* 都是 *k* 维的列向量，*k* 为隐变量的个数

$$X = [x_1, x_2, \cdots, x_N]$$

是所有 *xu* 构成的矩阵

$$Y = [y_1, y_2, \cdots, y_M]$$

为所有 *yi* 构成的矩阵，*N* 为用户数，*M* 为 item 数，λ 为正则化参数。目标函数里的内积用于表示用户对物品的预测偏好，拟合实际偏好 *pui*，拟合强度由 *cui* 控制。并且对于 *pui* =0 的项也要拟合。目标函数中的第二项是正则项，用于保证数值计算稳定性以及防止过拟合。

目标函数的求解仍然可以采用交替最小二乘法。具体如下[14,15]：

1) Y 固定为上一步迭代值或初始化值，优化 X ：

此时，Y 被当做常数处理，目标函数被分解为多个独立的子目标函数，每个子目标函数都是某个 xu 的函数。对于用户 u ，目标函数为：

$$\min_{x_u} \sum_{i=1}^{M} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda \|x_u\|_2^2$$

把这个目标函数化为矩阵形式，得

$$J(x_u) = (P_u - Y^T x_u)^T \Lambda_u (P_u - Y^T x_u) + \lambda x_u^T x_u$$

其中：

$$P_u = [p_{u1}, \cdots, p_{uM}]^T$$

为用户 *u* 对每个物品的偏好构成的列向量

$$Y = [y_1, y_2, \cdots, y_M]$$

表示所有物品向量构成的矩阵，*Λu* 为用户 *u* 对所有物品的置信度 *cui* 构成的对角阵，即：

$$\Lambda_u = \begin{pmatrix} c_{u1} & & \\ & \ddots & \\ & & c_{uM} \end{pmatrix}$$

对目标函数 *J* 关于 *xu* 求梯度，并令梯度为零，得：

$$\frac{\partial J(x_u)}{\partial x_u} = -2Y\Lambda_u(P_u - Y^T x_u) + 2\lambda x_u = 0$$

解这个线性方程组，可得到 xu 的解析解为：

$$x_u = (Y\Lambda_u Y^T + \lambda I)^{-1} Y\Lambda_u P_u$$

2) X 固定为上一步迭代值或初始化值，优化 Y：

此时，X 被当做常数处理，目标函数也被分解为多个独立的子目标函数，每个子目标函数都是关于某个 yi 的函数。通过同样的推导方法，可以得到 yi 的解析解为：

$$y_i = (X\Lambda_i X^T + \lambda I)^{-1} X\Lambda_i P_i$$

其中，

$$P_i = [p_{1i}, \cdots, p_{Ni}]^T$$

为所有用户对物品 i 的偏好构成的向量，

$$X = [x_1, x_2, \cdots, x_N]$$

表示所有用户的向量构成的矩阵，Λ_i 为所有用户对物品 i 的偏好的置信度构成的对角矩阵，即，

$$\Lambda_i = \text{diag}\{c_{1i}, \cdots, c_{Ni}\}$$

3 实验数据与模型的训练过程

3.1 数据集来源

本次实验使用的数据集是 Kaggle 上公开的数据集，采样地为合肥丰原，链接如下：

Kaggle：https://www.kaggle.com/datasets/cangnan/hefeipm25?select=train.csv

3.2 训练过程

1) 数据准备：

本次简单预测 PM2.5 的走向，每个月只取了 20 天的测量数据。共有 18 中污染源，即将 18 种污染源的数据和对应的 PM2.5 的数据进行训练再根据训练结果来预测新的数据下 PM2.5 的值。

先将 18 种污染物存放在 18 个纬度中，每个维度都是一个列表。然后打开数据文件,文件 big5 编码为繁体字，读取名称为 text 的 Excel 文件，返回文件行的累加信息,类型为_csv.reader，设一个列表 r，其中 r 保存了当前行的所有信息，循环遍历累加信息。

2) 选择损失函数

我们选取均方误差为损失函数，L(f) = (y - f)^2 求和再求平局值。

3) 初始化参数

```
w = np.zeros(len(x[0])) # shap: (163,) 0: 163 __len__: 1 size: 163
l_rate = 1 # 学习率，用于更新 w
repeat = 10000 # 迭代次数
```

4) 计算预测值

做矩阵乘法 `np.dot(x,w)`，(5652, 163)*(163, 1) ，shape = (5652, 1)，乘后可得预测值。

5) 计算损失

```
loss = hypo - y # 预测数值 - 真实数值
cost = np.sum(loss**2) / len(x) # L(f) = (y - f)^2 求和再求平局值
```

6) 计算梯度

```
gra = np.dot(x_t,loss) # (163, 5652) * (5652, 1)   shape: (163, 1)
```

7) 更新参数

```
s_gra += gra**2
ada = np.sqrt(s_gra) # 均方根
w = w - l_rate * (gra/ada) # 更新后的 w
```

8) 重复步骤 4 至 7

4 实验结果及分析

4.1 运行结果

初始设置学习率为 1，迭代十万次发现其陷入局部最优解中，因此提高学习率，进行重新计算。

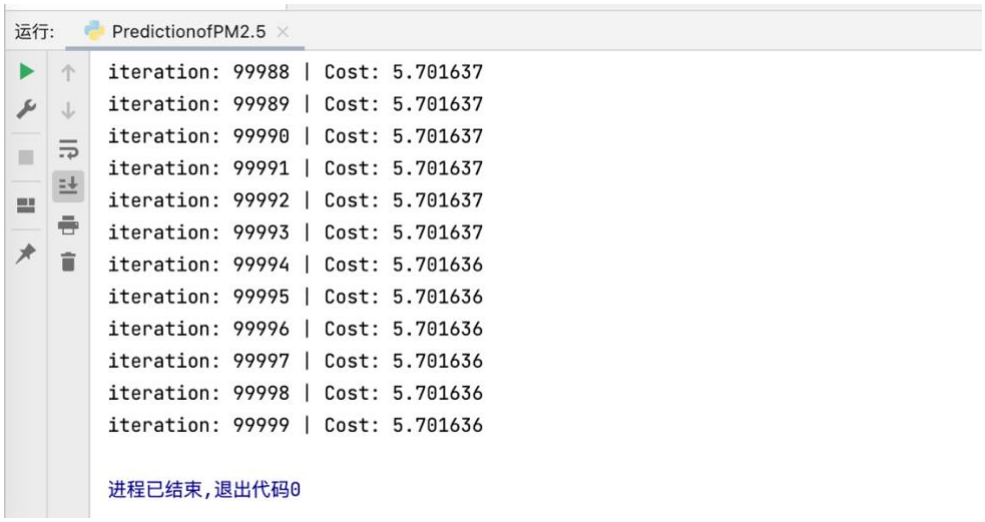

图 5 学习率为 1，迭代十万次

设置学习率为 2，迭代十万次发现其陷入局部最优解中，因此提高学习率，进行重新计算。

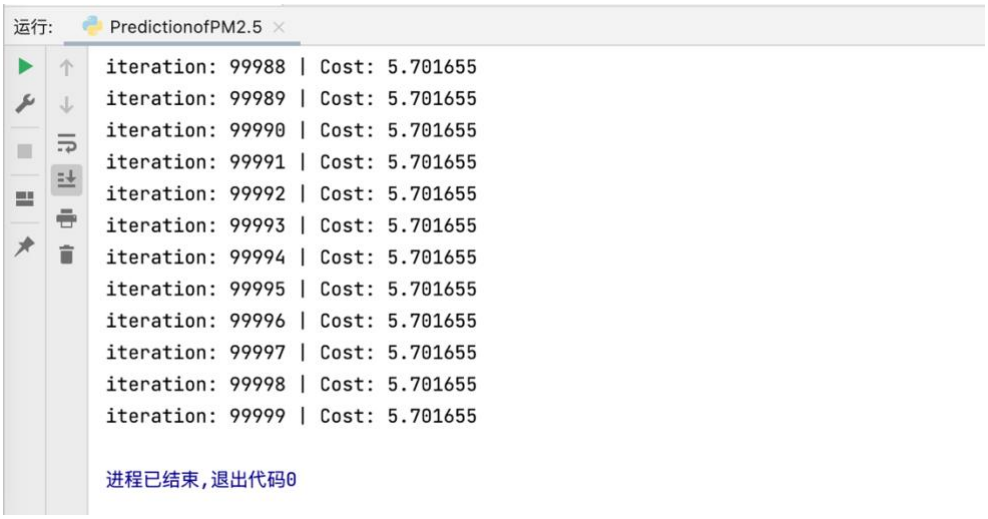

图 6 学习率为 2，迭代十万次

设置学习率为 3，迭代十万次发现其陷入局部最优解中，因此提高学习率，进行重新计算。

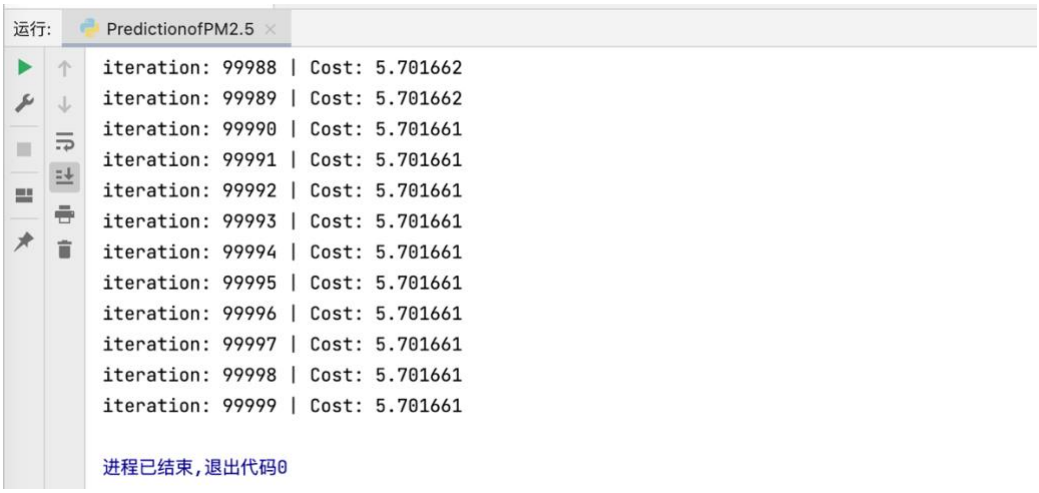

图 7 学习率为 3，迭代十万次

经过机器学习后，我们得到一个训练完成的模型 **model.npy**，将其载入并且使用 **test.csv** 进行测试，得到如下结果。可以看出模型的预测能力基本达到预定期望。

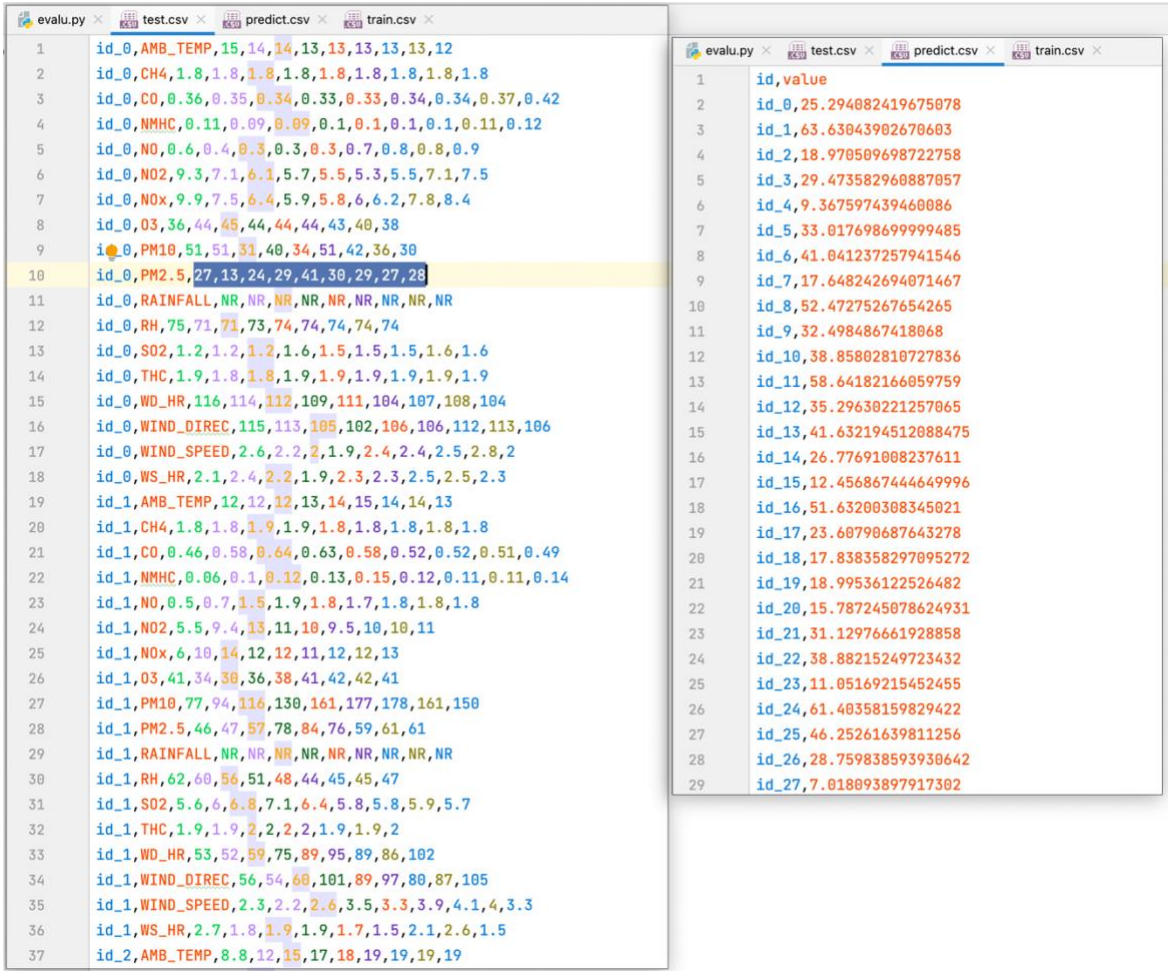

图 8 预测结果与测试结果对比

将 id_0 的 PM2.5 取平均得到 27.56 和预测结果 25.29 相近，同理推算出 id_1 及后面的数据，发现其预测精准度能够达到预期目标。

4.2 源代码

```
'''
本程序简单预测 PM2.5 的走向，每个月只取了 20 天的测量数据。
共有 18 中污染源，即将 18 种污染源的数据和对应的 PM2.5 的数据 进行训练
再根据训练结果来预测新的数据下 PM2.5 的值
'''
#   import library  #
import csv
import numpy as np
from numpy.linalg import inv
import random
import math
import sys
```

```
#   read data   #
data = []
# 每一个维度存储一种污染物的数据,一共有 18 种污染物
for i in range(18):
   data.append([]) # []表示这十八个输入中，每一个输入都是一个列表


n_row = 0 # 初始从第 0 行开始
# 打开数据文件,文件 big5 编码为繁体字
text = open('train.csv', 'r', encoding='big5')
# 读取名称为 text 的 Excel 文件，返回文件行的累加信息,类型为_csv.reader
row = csv.reader(text , delimiter=",")
# r 中保存了当前行的所有信息，r 是一个列表，每次循环到这就更换到下一行
for r in row:
    # 第 0 行没有数据信息
    if n_row != 0:
        # 每一列只有第 3-27 格有值(一天内 24 小时的数值)
        for i in range(3,27):
            if r[i] != "NR":
                # data[]中一共可以存放 18 个列表，每一个列表存放某一种污染物的所有数值
                data[(n_row-1)%18].append(float(r[i]))
            else:
                data[(n_row-1)%18].append(float(0))
    n_row = n_row+1
text.close()
'''
file = open('PM2.5Prediction/testdata.csv','w', encoding = 'utf-8')
for i in range(len(data)):
    s = str(data[i]).replace('[','').replace(']','')#去除[],这两行按数据不同，可以选
择
    s = s + '\n'   #去除单引号，逗号，每行末尾追加换行符
    file.write(s)
file.close()
print("保存文件成功")
'''
#    Parse Data to (x,y)     #
x = []
y = []
# 一共有 12 个月，每个月 20 天
for i in range(12):
    # 每个月取 20 天，一共有 480 个小时，连续取 10 个小时一组的数据，可取 471 组（最后 9 个数据舍
去）
    for j in range(471):
        x.append([]) # 在 x 中加入列表存储数据
```

```
        # 污染物的种类一共有 18 种
        for t in range(18):
            # 取每种污染物前 9 小时的数据,在第 10 小时存放 PM2.5 的值
            for s in range(9):
                x[471*i+j].append(data[t][480*i+j+s] )
        y.append(data[9][480*i+j+9]) # PM2.5 的数据存放在第 10 行
# 每个按行排列
x = np.array(x)
y = np.array(y)

# add square term
# x = np.concatenate((x,x**2), axis=1) 将数据量翻倍，新的数据为 x^2

# add bias
# 5652 行 1 列的 ones，每一行 1 后面直接连接 x 的每行列表
# 此处多的一列存储的是 bias，值为 1
x = np.concatenate((np.ones((x.shape[0],1)),x), axis=1) #x.shape[0]代表行数，
x.shape[1]代表列数。
#print(x.shape) # (5652, 163)

#init weight & other hyperparams#
w = np.zeros(len(x[0])) # shap: (163,) 0: 163 __len__: 1 size: 163
l_rate = 3 # 学习率，用于更新 w
repeat = 100000 # 迭代次数


#check your ans with close form solution#
# use close form to check whether ur gradient descent is good
# however, this cannot be used in hw1.sh
# w = np.matmul(np.matmul(inv(np.matmul(x.transpose(),x)),x.transpose()),y)

'''模型是 Y = b + w * x'''

#start training#
# 将 x 矩阵进行转置，此时第一行全是 1
# shape: (5652,) 0: 5652 len: 1 size: 5652
x_t = x.transpose()
s_gra = np.zeros(len(x[0])) # 163 行的 0

for i in range(repeat):
    hypo = np.dot(x,w) # 做矩阵乘法，(5652, 163)*(163, 1) shape = (5652, 1),预测值
    loss = hypo - y # 预测数值 - 真实数值
    cost = np.sum(loss**2) / len(x) # L(f) = (y - f)^2 求和再求平局值
    cost_a  = math.sqrt(cost) # 开方
```

```
    # 梯度下降
    gra = np.dot(x_t,loss) # (163, 5652) * (5652, 1)   shape: (163, 1)   dL
    s_gra += gra**2
    ada = np.sqrt(s_gra) # 均方根
    w = w - l_rate * (gra/ada) # 更新后的 w
    print ('iteration: %d | Cost: %f  ' % ( i,cost_a))

# save model
np.save('model.npy',w)
# read model
w = np.load('model.npy')

test_x = []
n_row = 0
text = open('test.csv' ,"r")
row = csv.reader(text , delimiter= ",")

for r in row:
    if n_row %18 == 0:
        test_x.append([]) # 没有 18 个数据就生成新的列表
        for i in range(2,11):
            test_x[n_row//18].append(float(r[i]) ) # // 表示向下取整
    else :
        for i in range(2,11):
            if r[i] !="NR":
                test_x[n_row//18].append(float(r[i]))
            else:
                test_x[n_row//18].append(0)
    n_row = n_row+1
text.close()
test_x = np.array(test_x)

# add square term
# test_x = np.concatenate((test_x,test_x**2), axis=1)

# add bias
test_x = np.concatenate((np.ones((test_x.shape[0],1)),test_x), axis=1) # 把 test
里面的数据连接到 ones 后面


ans = []
for i in range(len(test_x)):
    ans.append(["id_"+str(i)])
    a = np.dot(w,test_x[i]) # 用已经训练的 w 来生成预测值，并存入 ans
    ans[i].append(a)
```

```
filename = "predict.csv" # 最后把结果写入 predict 的文件中
text = open(filename, "w+")
s = csv.writer(text,delimiter=',',lineterminator='\n')
s.writerow(["id","value"])
for i in range(len(ans)):
    s.writerow(ans[i])
text.close()
```

4.3 优化模型

1) 方式一:将不同天数据连在一起
2) 方式二:增加模型的参数个数，从 1*9 增加到 18*9
3) 方式三:增加训练轮数
4) 方式四:Adagrad

5. 自我评价和心得体会

本次实验我学到了很多知识也遇到了很多问题，对于新的知识我在整个实验过程中有了很深的体会，对于遇到的问题和困难也通过查阅资料和论文进行逐一击破。

首先遇到的问题就是数据集的获取，最开始打算自己去官网使用爬虫爬取下来整理，但是考虑到后期的整理工作量我改为从网上找公开的数据集，这段时间我学会了使用 kaggle 来找寻公开的数据集，也参与了 kaggle 的相关比赛活动中。

第二个遇到的问题是，关于数据的预处理，本次数据是多个变量影响 PM2.5，之前并未尝试过相关的线性模型计算。在查阅前辈的代码以及专业书后，类比推出本次数据预处理的操作，在后续的实验中起到了很大的作用。

第三个遇到的问题是，在近十万次迭代后，其 Cost 都趋于稳定，在更换过学习率后依旧如此，后期将尝试更大的学习率，避免掉入局部最优解陷阱。

通过本次实验，我们可以得到一个可以用来预测 PM2.5 浓度的线性模型。我们可以评估模型的准确性，并对其进行改进。

本次实验让我更深入地了解了线性模型的原理和应用，并让我有机会将所学知识应用到实际问题中。本次实验对我的数据分析和机器学习技能有很大的提升。

总的来说，本次实验是一次有意义的经历，对我的专业能力和个人成长有很大的帮助。

6 参考文献

[1] 项亮. 推荐系统实践. 北京: 人民邮电出版社. 2012.

[2] Sarwar B M, Karypis G, Konstan J A, et al. Item-based collaborative filtering recommendation algorithms. WWW. 2001, 1: 285-295.

[3] Linden G, Smith B, York J. Amazon. com recommendations: Item-to-item collaborative filtering. IEEE Internet computing. 2003 (1): 76-80.

[4] Koren Y, Bell R, Volinsky C. Matrix factorization techniques for recommender systems. Computer. 2009 (8): 30-37.

[5] Koren Y. Factorization meets the neighborhood: a multifaceted collaborative filtering model. SIGKDD. 2008: 426-434.

[6] Hu Y, Koren Y, Volinsky C. Collaborative filtering for implicit feedback datasets. ICDM. 2008: 263-272.

[7] Mnih A, Salakhutdinov R R. Probabilistic matrix factorization. NIPS. 2008: 1257-1264.

[8] Koren Y. Collaborative filtering with temporal dynamics. SIGKDD. 2009: 447-456.

[9] Pilászy I, Zibriczky D, Tikk D. Fast als-based matrix factorization for explicit and implicit feedback datasets. RecSys. 2010: 71-78.

[10] Johnson C C. Logistic matrix factorization for implicit feedback data. NIPS, 2014, 27.

[11] He X, Zhang H, Kan M Y, et al. Fast matrix factorization for online recommendation with implicit feedback. SIGIR. 2016: 549-558.

[12] Hofmann T. Probabilistic latent semantic analysis. UAI. 1999: 289-296.

[13] Blei D M, Ng A Y, Jordan M I. Latent dirichlet allocation. JMLR. 2003, 3(Jan): 993-1022.

[14] Zhang S, Yao L, Sun A, et al. Deep learning based recommender system: A survey and new perspectives. ACM Computing Surveys (CSUR). 2019, 52(1): 5.

[15] Mikolov, Tomas & Chen, Kai & Corrado, G.s & Dean, Jeffrey. Efficient Estimation of Word Representations in Vector Space. Proceedings of Workshop at ICLR. 2013.

7. 成绩评定

总评分：

评分人：

日　期：