SHANDONG UNIVERSITY

机器学习第 3 次实验报告

姓名: 谢子洋
学院: 网络空间安全学院 (研究院)
专业: 网络空间安全
班级: 网安三班
学号:202100460116

2023 年 10 月 30 日

目录

1 实验目的 **2**

2 实验环境 **2**

3 实验方法 **2**

4 结果分析与评估 **3**

4.1 数据预处理 3

4.2 模型选择 4

4.3 梯度下降 5

4.4 模型初始值 6

4.4.1 参数初始值选择. 6

4.4.2 学习率初始值 6

4.5 特征选择 8

4.6 模型最终参数及结果 10

4.7 使用模型进行预测 11

5 结论 **11**

参考文献 **12**

A Code **13**

1 实验目的

使用给出的大量气象观测数据作为训练数据进行线性回归, 根据测试数据给出的前 9 小时特征取值预测第 10 小时空气污染指数 (即 PM2.5) 的数值.

2 实验环境

训练平台:

表 1: 训练平台信息

CPU	INTEL CORE i5-12400
Memory	DDR4 3200Hz 16GB
OS	Windows10
Language	Python3.9

3 实验方法

1) 数据预处理.

输入数据标准化

$$x = \frac{X-\mu}{\sigma_X}$$

2) 线性回归模型.

选择 n 个合适特征作为模型输入, 定义线性模型.

$$y = w_0 + w_1x_i + \cdots w_nx_n$$

3) 梯度下降.

计算 Loss(w) 函数的梯度 g_t, 对参数进行梯度下降.

$$\theta_{t+1} = \theta_t - \eta_t g_t$$

其中学习率 η 可按多种方式计算.

(1) 学习率随时间减小

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{t+1}} g_t$$

(2) Adagrad

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\sum_{i=0}^{t} g_i^2}} g_t$$

4 结果分析与评估

4.1 数据预处理

1) 数据格式化.

   模型共 162 个输入, 在线性模型下对应 163 个参数.

$$w = \begin{pmatrix} w_0 & w_1 & \cdots & w_{dim} \end{pmatrix}$$

   可将原始数据排列为矩阵格式, 每一行对应一组样本输入, 每一列对应输入的一种特征. 最终得到 $m \times dim(5652 \times 162)$ 的输入数据矩阵.

$$X = \begin{pmatrix} x_{1,1} & \cdots & x_{1,dim} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \cdots & x_{m,dim} \end{pmatrix}$$

2) 数据标准化.

   对于一个输入, 其具有多个特征, 不同特征之间具有不同的量纲, 因此我们对输入数据进行标准化. 对每个特征计算其均值和标准差, 再对该特征下的所有值进行标准化.

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad \sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

$$x^{'} = \frac{x - \mu}{\sigma}$$

   标准化后不同特征变成统一单位, 各特征对 Loss 的影响度量程度统一. 消除了样本不同属性具有不同量级时的影响, 使得不同数量级的特征地位一致, 提高了梯度下降迭代收敛速度.

3) 交叉验证.

   评估模型不仅需要在训练数据上计算模型准确度, 还需要在测试集上进行测试. 在训练数据上表现优秀的模型可能存在过拟合等现象, 使用测试集能判断模型是否过拟合. 测试集完全不参与模型训练, 为了防止影响测试模型的" 真实" 准确度.

   但有时可能不存在测试数据集, 这时我们可以将训练集划分为训练集和验证集. 验证集对训练集生成的参数进行测试，相对客观的判断这些参数对训练集之外的数据的符合程度。

   对与本次实验数据, 我们进行如下处理. (1). 将原训练集划分为 80% 的训练数据和 20% 的验证数据. (2). 使用新训练数据集训练模型. (3). 使用验证集的保留样本测试模型准确度. (4). 模型在验证数据上较准确，则说明当前的模型较优.

4.2 模型选择

以每连续的十小时为一组, 前 9 个小时的所有观测值作为特征, 第 10 小时的 PM2.5 值作为预测值. 训练数据: $<x,y>$, 其中输入 $x=(x_1,\cdots,x_{dim})$, 输出 y.

选择线性模型进行训练:

$$y=\sum_{i=1}^{dim} w_i x_i + w_0$$

因为训练数据和验证数据规模相差较大, 两者计算出的 Loss 值会明显不同, 难以判断模型拟合程度. 因此本文选择使用平均 Loss 作为衡量标准, 以此衡量在两数据集下的损失.

$$averLoss=\frac{1}{N}\sum(\hat{y}-y)^2$$

记录模型训练过程中在 trainData 和 validationData 上的平均 Loss.

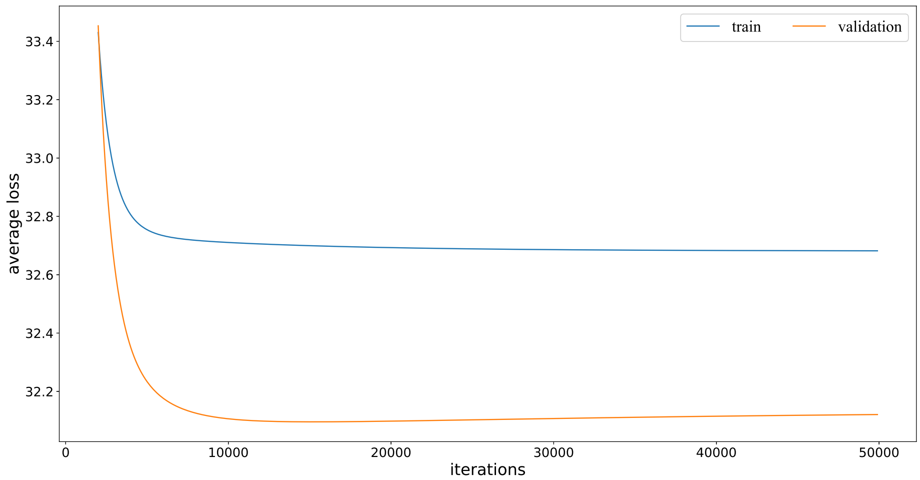


图 1: train and validation

观察上图曲线可发现, 模型在训练数据和验证数据上的平均损失差距不明显, 并且在验证数据上结果更优, 可认为线性模型适用于该预测问题.

4.3 梯度下降

1) Loss 函数.

该线性模型 Loss 函数如下:

$$L(w)=\sum_{r=1}^{m}(w_0+\sum_{i=1}^{dim}w_i x_{r,i}-y_r)^2$$

2) 计算梯度.

进行梯度下降需要先求出 Loss 函数对不同参数的偏导.

$$\begin{aligned}\frac{\partial L(w)}{w_k}&=2\cdot\sum_{r=1}^{m}x_{r,k}(w_0+\sum_{i=1}^{dim}w_i x_{r,i}-y_r)\\ \frac{\partial L(w)}{w_0}&=2\cdot\sum_{r=1}^{m}(w_0+\sum_{i=1}^{dim}w_i x_{r,i}-y_r)\end{aligned}\tag{4.1}$$

为提高模型训练速度, 可以矩阵运算形式计算梯度, 而非嵌套 for 循环. numpy 库进行矩阵运算效率更高, 相比直接循环计算效率能提高许多倍.

首先计算矩阵 C.

$$\begin{pmatrix}c_1\\ \vdots\\ c_m\end{pmatrix}=\begin{pmatrix}x_{1,1}&\cdots&x_{1,dim}\\ \vdots&\ddots&\vdots\\ x_{m,1}&\cdots&x_{m,dim}\end{pmatrix}\times\begin{pmatrix}w_1\\ \vdots\\ w_{dim}\end{pmatrix}+\begin{pmatrix}w_0\\ \vdots\\ w_0\end{pmatrix}-\begin{pmatrix}y_1\\ \vdots\\ y_m\end{pmatrix}$$

根据 C 可以计算 Loss 函数对各参数的偏导.

$$\begin{pmatrix}\frac{\partial L(w)}{w_1}\\ \vdots\\ \frac{\partial L(w)}{w_k}\\ \vdots\\ \frac{\partial L(w)}{w_{dim}}\end{pmatrix}^T=2\times\begin{pmatrix}c_1\\ \vdots\\ c_m\end{pmatrix}^T\times\begin{pmatrix}x_{1,1}&\cdots&x_{1,dim}\\ \vdots&\ddots&\vdots\\ x_{m,1}&\cdots&x_{m,dim}\end{pmatrix}$$

$$\frac{\partial L(w)}{w_0}=\begin{pmatrix}c_1\\ \vdots\\ c_m\end{pmatrix}^T\times\begin{pmatrix}2\\ \vdots\\ 2\end{pmatrix}$$

转化为矩阵运算形式后, 模型训练速度大幅提升. 在使用所有的特征作为输入的情况下, 每分钟可训练大约 18000 次.

3) 调整学习率.

模型的学习率影响着模型训练的速度和稳定性, 合适的学习率能够在确保模型收敛的同时提高训练效率,

常见的的学习率调整方法有:

(1) 随时间减小.

(2) adagrad.

(3) Adam.

(4) RMSprop.

本文选择使用 adagrad 方法.

$$w_{k,t+1} = w_{k,t} - \frac{\eta}{\sqrt{\sum_{s=1}^{t}(\frac{\partial L_s(w)}{\partial w_k})^2}} \cdot \frac{\partial L_t(w)}{\partial w_k}$$

该方法下, 学习率随着训练次数的增加不断减小, 并且能自适应梯度变化及时调整学习率, 从而加快训练速度.

4.4 模型初始值

4.4.1 参数初始值选择.

该线性模型中共 163 个参数, 可大体分为两部分, 即常数部分的 w_0 和线性部分的 $w_i \; for \; i \in [1,163]$, 可分别为这两部分设定不同初值 v1 和 v2.

在一定区间范围内遍历 v1 和 v2 并计算对应的 Loss 函数, 做等高图如下.

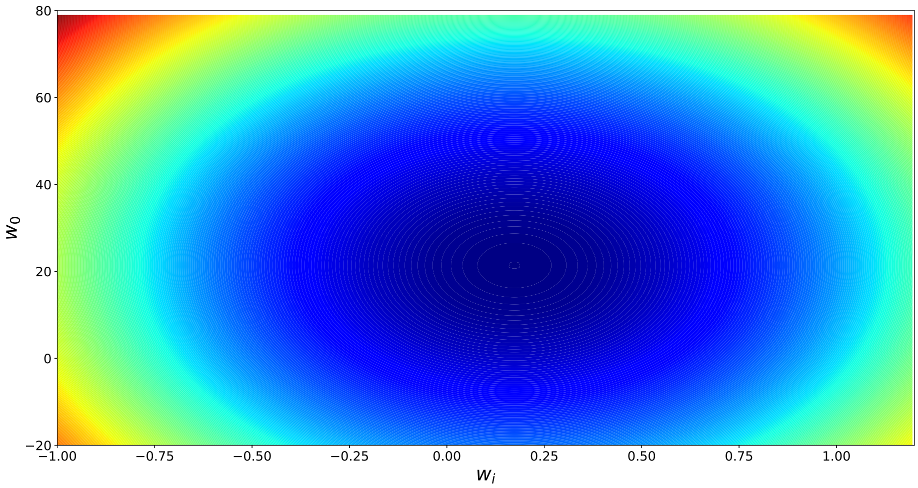


图 2: 等高图

据图可大致确定最优点 $(v1, v2) = (21, 0.17)$, 以此分别作为两类参数的初始值.

4.4.2 学习率初始值

将学习率初始值由低到高不断调整, 当学习率很低时模型学习效率低耗时长, 而当学习率过大时会出现 Loss 函数值突然上升的现象. 实验中我们将学习率由较小值不断向上提高, 最终确定一合适的学习率.

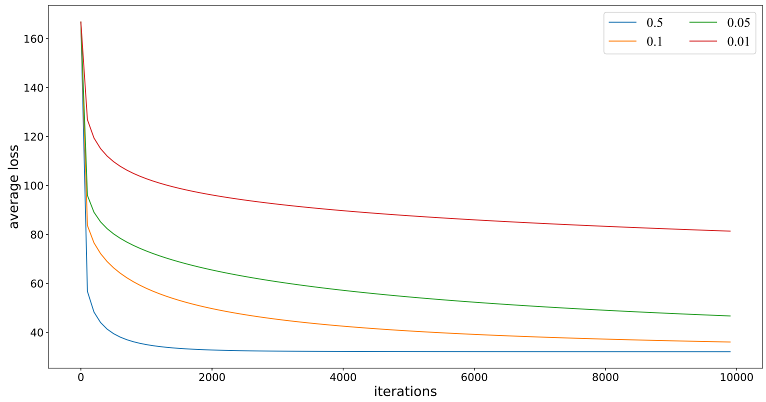
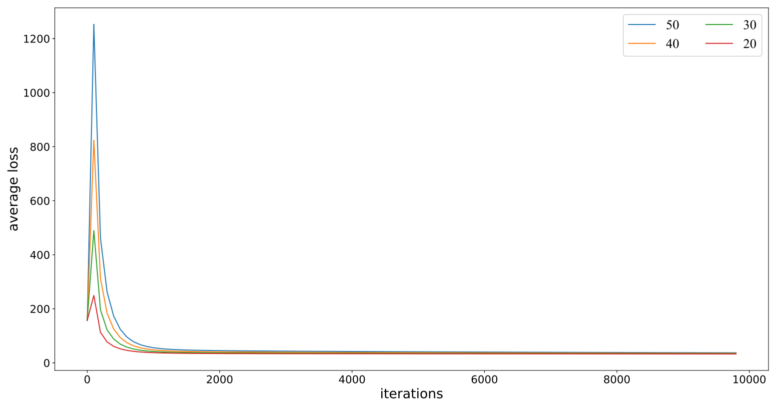
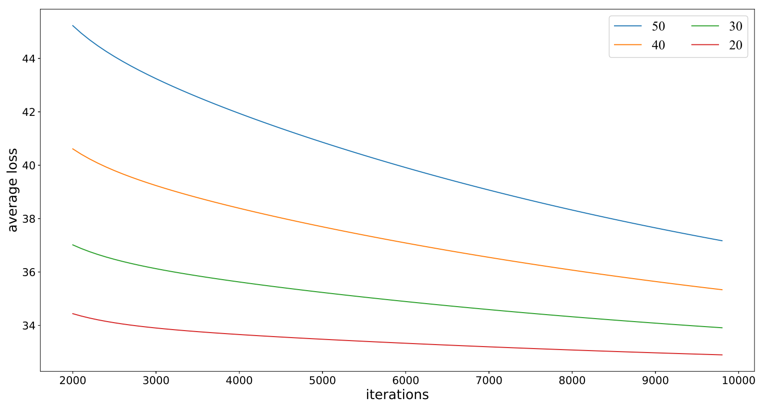


图 3: 学习率较小

观察上图, 当学习率较低时, 模型训练收敛慢, 经过同样训练次数得到的结果差距较大, 难以达到最优解.



(a) 完整训练过程



(b) 后部分过程

图 4: 学习率较大

当学习率设置过大时会导致在初期出现 Loss 函数值暴涨的现象, 经过多次训练后快速下降. 而且学习率对最终的训练准确率也有影响, 在上述选取的学习率初始值中, 取值越大会导致结果越差.

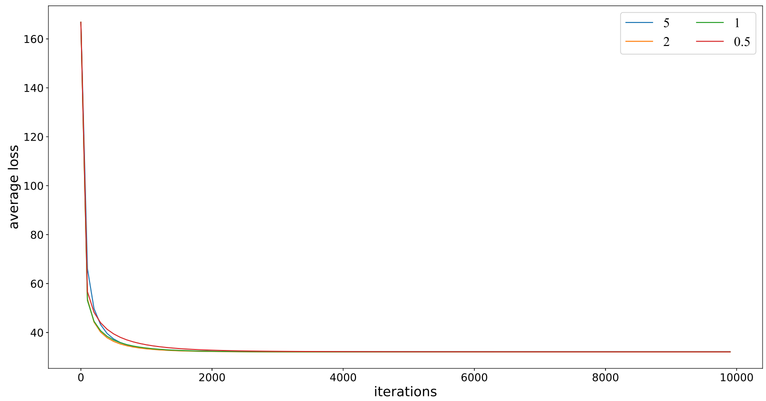


图 5: 学习率适中

而当学习率选定在合适的区间内时, 训练过程稳定, 模型参数收敛速度快, 并且结果一般较优.

综上, 本文在比较多个学习率初值对应的结果后, 最终选定较为合适的 0.4 作为学习率初值.

4.5 特征选择

对全部的 162 个特征进行编号, 编号取值区间为 [1,162]. 本文首先选择全部的特征进行训练, 根据训练结果选出在线性模型中起到主要作用的特征, 即对应系数绝对值更大的特征.

将全部特征作为模型输入, 进行 50000 次训练后得到各参数的线性系数, 对其绝对值逆序排列得到如下结果.(完整数据见附件 1.)

表 2: 主要相关特征

相关性排序	特征编号	特征线性系数	特征
1	90	15.2960	第 9 小时 PM2.5
2	88	-9.1582	第 7 小时 PM2.5
3	87	7.9106	第 6 小时 PM2.5
4	85	-4.1806	第 4 小时 PM2.5
5	84	3.7136	第 3 小时 PM2.5
6	62	3.3499	第 8 小时 NOx
7	53	-3.2081	第 8 小时 NO2
8	58	3.0509	第 4 小时 NOx
9	49	-2.5898	第 4 小时 NO2
10	9	2.5748	第 9 小时 AMB_TEMP
11	4	-1.7976	第 4 小时 AMB_TEMP
12	63	1.7222	第 9 小时 NOx
…	…	…	

选取其中前 n 个特征作为新的模型的输入, 重新训练模型. 下面描述了选取不同数量特征的模型在训练过程中平均损失的变化.

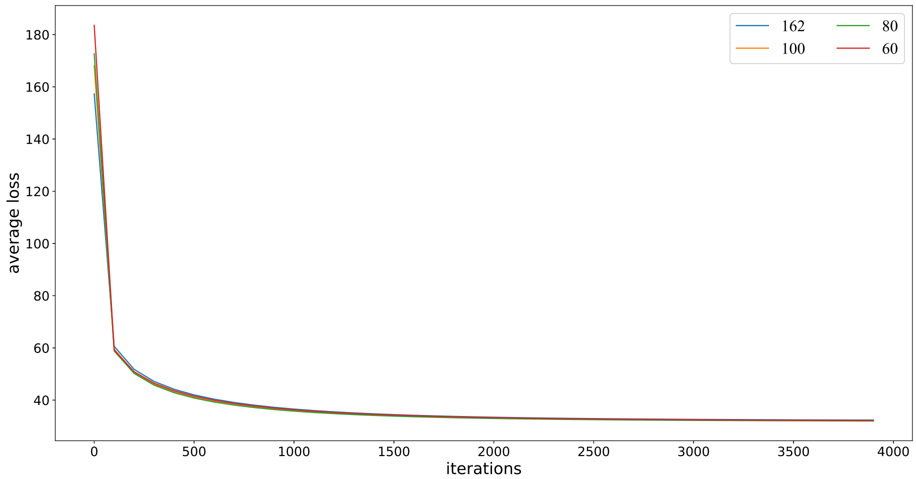


图 6: 不同数量特征模型训练过程

观察上图可发现, 当选取的主要特征足够多时, 不同特征模型的训练速度和最终结果均无较大差异, 这是因为选取的特征中已经包含对目标值 (第 10 天 PM2.5) 起主要作用的特征, 而其他未选用的特征对目标值的影响很小甚至是没有.

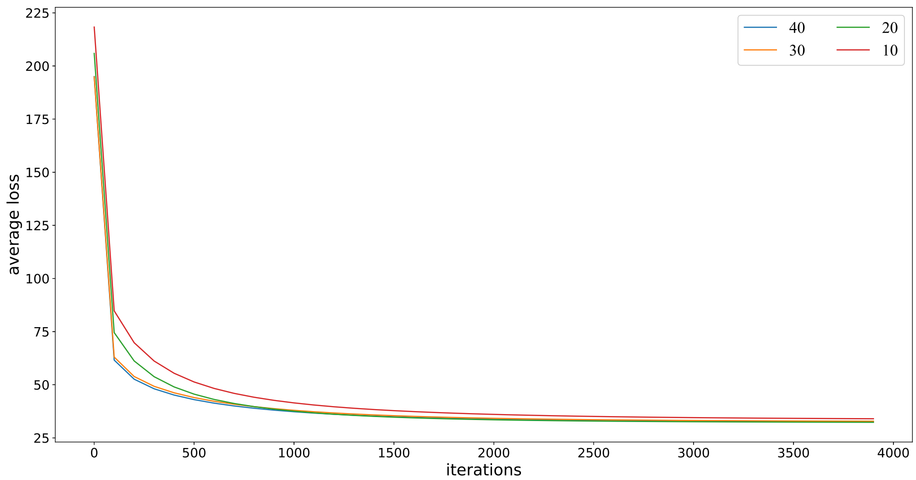


图 7: 不同数量特征模型训练过程

将选取的特征数量再次减少, 可发现不同模型在训练过程中的 Loss 值存在一定差异. 尤其当选取特征数小于 30 时.

为找到最优选取特征数, 本文计算了选取不同数量特征的模型经 50000 次训练后在训练数据和验证数据上的平均 Loss. 绘制曲线图如下.

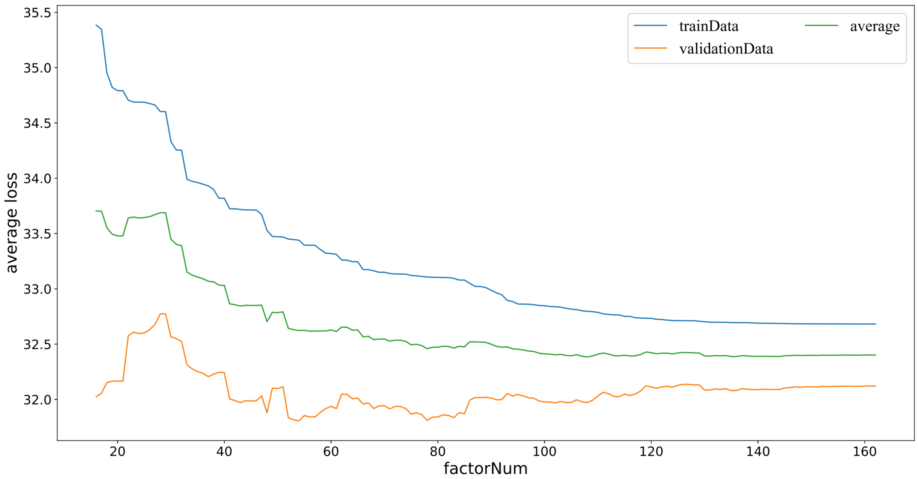


图 8: 选取不同数量的特征

若选取特征数量取值合适, 则模型在训练数据和验证数据中平均 Loss 应均取较小值. 观察上图可知, 在选取特征数 $n = 80$ 时, 训练数据集上的平均 loss 较低且验证数据集上的平均 Loss 几乎为最低. 因此本文选择使用相关性前 80 高的特征作为模型的输入特征.

4.6 模型最终参数及结果

综上设定模型参数, 进行 50000 次训练后得到模型的最终参数及对应 Loss.

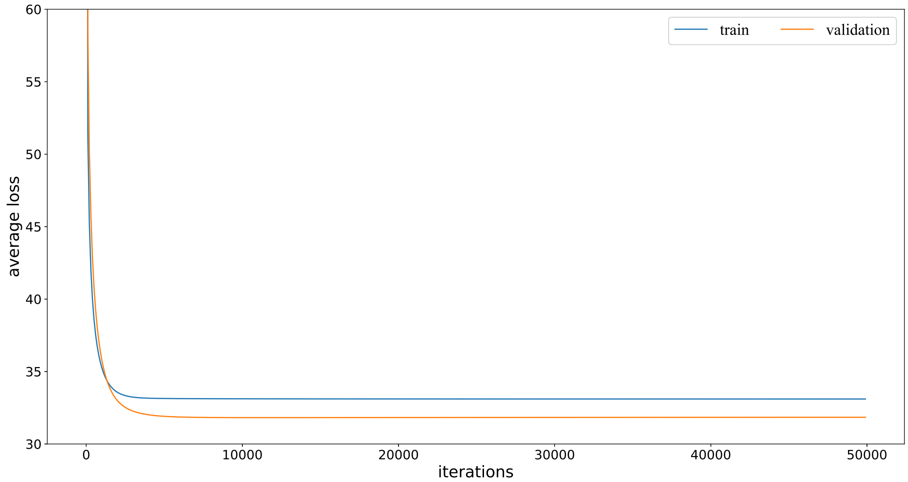


图 9: 模型训练过程

表 3: 模型参数及结果

参数初始值	$\{21, 0.17\cdots, 0.17\}$
学习率初始值	$\{0.4, 0.4, \cdots, 0.4\}$
学习率变化算法	Adagrad
特征选取	80 个
迭代次数	50000
预计耗时	136.08s
trainAverLoss	33.103324
validationAverLoss	31.841386

4.7 使用模型进行预测

将 test 数据处理后作为模型输入, 得到第 10 天 PM2.5 预测值, 详见附件 2.

表 4: 部分预测值

id	value
id_0	24.72525453668088
id_1	64.77028084014955
id_2	20.785554490980296
id_3	29.436868087745665
id_4	9.941318846898326
id_5	32.74499209930922
id_6	40.59656671197534
id_7	18.307240973606966
id_8	54.2695290658516
id_9	33.37695418753495
⋯	⋯

5 结论

数据是模型训练的全部信息来源, 直接决定了模型训练的效果, 因而合理的数据至关重要. 在模型训练前可对数据进行各种预处理, 比如无量纲化能避免不同特征因单位不同而对模型训练造成影响.

模型参数初值能够极大影响模型训练次数, 设置合适的参数初值能节省大量训练时间.

训练的时候需要选取一个合适的学习率，学习率控制着算法迭代过程中的更新步长，如果学习率过大，则不容易收敛，如果过小，收敛速度容易过慢。当学习率较低时, 模型准确率较低, 并且损失值随着迭代轮数增加缓慢减少. 而当学习率较高时, 损失值会在很短迭代次数内迅速减少, 但后期随着迭代轮数增加下降速度也会减少. 因此在模型训练时, 我们可以适当提高学习率, 提高模型训练速度.

对于模型训练, 选择合适的特征作为输入也很重要. 从所有特征中选取主要特征能够减少训练数据量, 提升训练速度, 并且还能提高模型准确性, 减少无关变量对模型参数的影响.

在本次实验中模型能在前期较快的梯度下降, 但在训练次数达到很大值后 Loss 减小速度逐渐放缓, 最后几乎无变化. 这可能是和本文采用的 adagrad 方法有关, 可以改进为其他学

习率调整方法以避免该问题.

参考文献

[1] https://www.runoob.com/numpy/numpy-tutorial.html

A Code

```
2023/10/30
import sys
import pandas as pd
import numpy as np
import math
from tqdm import tqdm,trange
import time
st=time.time()


# 0. 线性模型计算 (1)Loss (2)梯度 (3)预测值.
class linearModel:
    def calLoss(w,x,y):
        y_=w[0]+np.dot(x[:,:],w[1::])
        result=np.sum((y-y_)**2)
        return result
    def calGrad(w,x,y):
        dim=np.size(w,0)
        w_grad=np.zeros([dim,1])
        constVec=(w[0]+np.dot(x[:,:],w[1::])-y[:])*2
        w_grad[0]=np.sum(constVec)
        grad=np.sum(constVec*x[:,:],0)
        w_grad[1:]=grad.reshape(-1,1)
        return w_grad
    def calFuncVal(w,x):
        return np.sum(w[1::]*x.reshape(-1,1))+w[0]


# 1. 训练数据处理
# 1.1 读取并结构化数据
data = pd.read_csv('./train.csv')
data = data.iloc[:, 3:]
data[data == 'NR'] = 0
raw_data = data.to_numpy()
month_data = {}
for month in range(12):
    sample = np.empty([18, 480])
    for day in range(20):
        sample[:, day * 24 : (day + 1) * 24] = raw_data[18 * (20 * month + day) : 18 * (20 * month + day + 1), :]
    month_data[month] = sample
x = np.empty([12 * 471, 18 * 9], dtype = float) # 指标种数18, 特征天数9
y = np.empty([12 * 471, 1], dtype = float)      # 共12*471个输入输出对.
for month in range(12):
    for day in range(20):
        for hour in range(24):
            if day == 19 and hour > 14:
                continue
            x[month * 471 + day * 24 + hour, :] = month_data[month][:,day * 24 + hour : day * 24 + hour + ...
                9].reshape(1, -1)
            y[month * 471 + day * 24 + hour, 0] = month_data[month][9, day * 24 + hour + 9]
# np.savetxt('frame1',x,fmt='%f')
# np.savetxt('frame2',y,fmt='%f')


# 1.2 数据预处理: 标准化
mean_x = np.mean(x, axis = 0) #18 * 9
std_x = np.std(x, axis = 0) #18 * 9
for i in range(len(x)): #12 * 471
    for j in range(len(x[0])): #18 * 9
        if std_x[j] != 0:
            x[i][j] = (x[i][j] - mean_x[j]) / std_x[j]
# np.savetxt('x.txt',x,fmt='%f')
# np.savetxt('y.txt',y,fmt='%f')
# x = np.loadtxt('x.txt').reshape(5652,-1)
# y = np.loadtxt('y.txt').reshape(5652,-1)


def train(factorNum=162):
    # 1.3 选取特征主元素
    isSelect=1
    if(isSelect):
        SelectFactor=\
            [89, 87, 86, 84, 83, 61, 52, 8, 57, 48, 3, 71, 62, 75, 7, 77, 105, 78, 74, 39, 55, 88, 2, 49, 58, 107, 6, ...
                51, 5, 80, 81, 104, 68, 102, 109, 103, 59, 43, 108, 32, 44, 1, 101, 29, 60, 64, 72, 26, 132, 28, ...
                117, 17, 41, 85, 31, 50, 53, 160, 65, 146, 9, 37, 118, 127, 30, 116, 46, 38, 115, 4, 66, 33, 0, ...
                122, 120, 106, 158, 69, 79, 27, 99, 161, 119, 154, 124, 36, 144, 16, 145, 156, 135, 110, 98, 129, ...
```

```
                    24, 40, 73, 155,
    142, 10, 21, 34, 137, 138, 148, 123, 22, 152, 151, 128, 94, 130, 114, 159, 92, 54, 90, 150, 139, 70, 95, 153, ...
         134, 136, 76, 63, 125, 100, 97, 131, 25, 14, 113, 157, 133, 67, 149, 82, 111, 18, 112, 20, 42, 126, 147, ...
         19, 23, 15, 13, 45, 96, 12, 35, 141, 47, 11, 91, 121, 140, 143, 56, 93]
        SelectFactor=SelectFactor[0:factorNum]
        tempx=x[:,SelectFactor]


    # 1.4 将 train 数据按照80:20的比例划分为 train 数据和 validation 数据
    x_train_set = tempx[: math.floor(len(x) * 0.8), :]
    y_train_set = y[: math.floor(len(y) * 0.8), :]
    x_validation = tempx[math.floor(len(x) * 0.8): , :]
    y_validation = y[math.floor(len(y) * 0.8): , :]


    # 2. 使用训练数据进行学习
    # 2.1 设置模型参数
    dim = np.size(x_train_set,1)+1
    w = np.asarray([21]+[0.17 for i in range(dim-1)]).reshape(dim,1)    #w0,w1, ... , w162共163个模型参数,其中 w0=b
    learning_rate = np.array([0.4 for i in range(dim)]).reshape(dim,1) #每个参数单独设置学习率
    #iter_time =3000000  #学习次数
    iter_time =50000  #学习次数
    adagrad = np.zeros([dim, 1])

    # 2.2 模型训练
    for t in range(iter_time):
        if t%100==0:
            loss1 = linearModel.calLoss(w,x_train_set,y_train_set)
            loss2=linearModel.calLoss(w,x_validation,y_validation)
            print(f"{t:6}  {loss1/np.size(x_train_set,0):4.6f}  {loss2/np.size(x_validation,0):4.6f}")
            #print(f"{t:6} trainTotal:{loss1:08.6f} tadinAver:{loss1/np.size(x_train_set,0):4.6f}",end='')
            #print(f" validTotal:{loss2:08.6f} validAver:{loss2/np.size(x_validation,0):4.6f}")
        gradient = linearModel.calGrad(w,x_train_set[:,:],y_train_set[:,:])
        adagrad = adagrad+gradient**2
        curLenRate=learning_rate/(adagrad**0.5)
        w = w-curLenRate*gradient
    #print(f"{factorNum} {loss1/np.size(x_train_set,0):4.6f}  {loss2/np.size(x_validation,0):4.6f}")
    # np.save('weight.npy', w)
    # np.savetxt('weight.txt', w)


    # 3. 使用模型进行预测, 并输出结果
    # 3.1 读取测试数据并转化格式
    testdata = pd.read_csv('./test.csv', header = None, encoding = 'big5')
    test_data = testdata.iloc[:, 2:].copy()
    test_data[test_data == 'NR'] = 0
    test_data = test_data.to_numpy()
    test_x = np.empty([240, 18*9], dtype = float)
    for i in range(240):
        test_x[i, :] = test_data[18 * i: 18* (i + 1), :].reshape(1, -1)

    # 3.2 测试数据标准化
    for i in range(len(test_x)):
        for j in range(len(test_x[0])):
            if std_x[j] != 0:
                test_x[i][j] = (test_x[i][j] - mean_x[j]) / std_x[j]
    #test_x = np.concatenate((np.ones([240, 1]), test_x), axis = 1).astype(float)

    # 3.3 选取主元素
    if (isSelect):
        test_x=test_x[:,SelectFactor]

    # 3.4 计算预测值
    predictVal=np.zeros([np.size(test_x,0),1])
    for i in range(np.size(test_x,0)):
        predictVal[i]=linearModel.calFuncVal(w,test_x[i,:])
        print(predictVal[i])
    print(f"\n*************time cost: {time.time()-st}******************\n")

    # 3.5 结果写入csv文件
    temp=[[f"id_{str(i)}",predictVal[i][0]] for i in range(len(predictVal))]
    city = pd.DataFrame(temp, columns=['id', 'value'])
    city.to_csv('city.csv',index=False)
train(80)
```