# 实验1《线性回归预测PM2.5值》

实验学时： 2 实验地点： 二综204 实验日期： 2019/10/11

1. **问题描述**

学会使用学习到的线性回归的知识，手动使用adagrad梯度下降方法，通过给定的相关数据来完成PM2.5值的回归预测。

1. **设计简要描述**

**1．安装Anaconda**

安装Anaconda会自动安装pyhton，或者也可以去python官网下载安装python3.6以上版本。

**2.数据描述**

本次作业是使用某监控站的观测记录，分成train set 以及 test set，train set 是该站每个月前20天的所有资料，test set 是从该站剩下的资料中取样出来。即：

* Train.csv每个月前20天的完整资料。
* Test.csv从剩下资料中取样出连续的10小时为1笔数据，前9小时所有的观测数据当做feature，第10小时的PM2.5 当做answer，一共可以取出240笔不重复的test data，请根据feature预测这些240笔的PM2.5.

**3.使用梯度下降的简单线性回归(使用adagrad)**

**3.1首先对数据进行处理**

伪代码如下：

{

Declare a 18-dim vector (Data);#声明一个18维向量

for i\_th row in training data:   
 Data[i\_th row%18].append(every element in i\_th row)

}

**3.2 训练数据规整化**

由于要test数据中是通过前9小时数据来预测第10小时数据。因此我们训练数据训练时也是按照该采样方法训练模型。可知每月共有480小时数据，其中每10小时为为1笔训练资料，那么一个月就是一共有471笔的训练资料。

其中伪代码如下：

{

Declare train\_x for previous 9-hr data, and train\_y for 10th-hr pm2.5

#前9列数据作为train\_x,tran\_y作为结果为第10列PM2.5数据

for i in all the given data:  
 sample every10 hrs：  
 train\_x.append(previous 9-hr data)  
 train\_y.append(the value of 10th-hr pm2.5)  
 add a bias term to every data in train\_x

}

**3.3 实现线性回归**

自己需要设置权重w，学习率lr以及迭代次数iteration这些超参数。

线性回归的伪代码如下：

{

Declare weight vector, initial lr ,and # of iteration

#声明迭代的权向量、初始lr,迭代次数  
 for i\_th iteration :  
 y’ = the inner product of train\_x  and weight vector  
 Loss = y’ - train\_y  
 gradient = 2\*np.dot((train\_x)’, L )//点乘  
 weight vector -= learning rate \* gradient

}

这里训练的过程中可以设置一个标准差或者平方差观察训练效果如何;

cost = np.sum(loss\*\*2) / len(x) #平方差

cost\_a = math.sqrt(cost)#标准差

**3.4 存储模型**

训练完成后，需要存储下训练好的模型。

# save model

np.save('model.npy',w)

# read model

w = np.load('model.npy')

**3.5 读取测试数据**

观察测试数据的储存形式，规整化测试数据集，以便能通过训练好的模型直接完成数据集的测试，类似于训练数据的读取。

text = open('data/test.csv' ,"r")

row = csv.reader(text , delimiter= ",")

这里没有繁体字，使用默认编码即可。

**3.6预测PM2.5**

伪代码如下：

{

read test\_x.csv file  
for every 18 rows :  
 test\_x.append([1])  
 test\_x.append(9-hr data)  
 test\_y = np.dot(weight vector, test\_x)

#训练好的weight vector

}

**3.7 测试结果**

测试结果应该分为两列：[“id”,”value”],将其写入的文件中，类似如下;(写文件过程)

filename = "data/predict.csv"

text = open(filename, "w+")

s = csv.writer(text,delimiter=',',lineterminator='\n')

s.writerow(["id","value"])

1. **程序清单**
2. **数据处理**

train\_data = pd.read\_csv("dataset/train.csv", encoding='big5')

test\_data = pd.read\_csv("dataset/test.csv", encoding='big5')

sample\_submission = pd.read\_csv("dataset/sampleSubmission.csv", encoding='big5')

train\_data = train\_data.iloc[:, 3:]

print(train\_data.shape)

print(train\_data.head())

train\_data.replace("NR", 0 , inplace=True)

# month : 12, pollution\_source : 18, days : 20, hours each day : 24

train\_each\_month = np.zeros((12, 18, 24 \* 20), dtype=np.float64)

for i in range(train\_each\_month.shape[0]):

for j in range(20):

train\_each\_month[i, :, j \* 24 : j \* 24 + 24] = train\_data.iloc[i + j \* 18:i + 18 + j \* 18, :]

print(train\_each\_month[0, 0, :48])

test\_data = test\_data.iloc[:, 2:]

print(test\_data.shape)

print(test\_data.head())

test\_data.replace("NR", 0 , inplace=True)

1. **准备训练集，测试集数据**

train\_rows = 471 \* 12

train\_x = np.zeros((train\_rows, 18 \* 9), dtype=np.float64)

train\_y = np.zeros((train\_rows, 1), dtype=np.float64)

for i in range(train\_each\_month.shape[0]):

for j in range(471):

train\_x[i \* 471 + j] = train\_each\_month[i, :, j : j + 9].flatten()

train\_y[i \* 471 + j] = train\_each\_month[i, 9, j + 9]

print("train\_x[2, : 10] :", train\_x[2, : 10])

print("train\_y[2:5] :", train\_y[2:5])

train\_bias = np.ones((train\_x.shape[0], 1))

train\_x = np.concatenate((train\_bias, train\_x), axis=1)

test\_rows = int(test\_data.shape[0] / 18)

test\_x = np.zeros((test\_rows, 18 \* 9), dtype=np.float64)

test\_y = np.zeros((test\_rows, 1), dtype=np.float64)

for i in range(test\_x.shape[0]):

test\_x[i] = test\_data.iloc[i \* 18 : i \* 18 + 18, :].values.flatten()

print("test\_x[0, : 10] :", test\_x[0, : 10])

test\_bias = np.ones((test\_x.shape[0], 1))

test\_x = np.concatenate((test\_bias, test\_x), axis=1)

1. **定义训练函数**

*# 梯度下降算法*

def GD(train\_x, train\_y, weight, lr, iteration, lambdaL2):

list\_loss = []

for i in range(iteration):

predict = np.dot(train\_x, weight)

loss\_bref = predict - train\_y

loss = np.mean(loss\_bref \*\* 2)

list\_loss.append(loss)

# f(x) = (a\*x + b)^2

# f(x)' = 2(a\*x + b) \* a

grad = np.dot(train\_x.T, loss\_bref) / train\_x.shape[0] + lambdaL2 \* weight

weight -= lr \* grad

return weight, list\_loss

*# 自适应梯度下降算法*

def ada\_GD(train\_x, train\_y, weight, lr, iteration, lambdaL2):

s\_grad = np.zeros((train\_x.shape[1], 1), dtype=np.float64)

list\_loss = []

for i in range(iteration):

predict = np.dot(train\_x, weight)

loss\_bref = predict - train\_y

loss = np.mean(loss\_bref \*\* 2)

list\_loss.append(loss)

grad = np.dot(train\_x.T, loss\_bref) / train\_x.shape[0] + lambdaL2 \* weight

s\_grad += grad \*\* 2

ada = np.sqrt(s\_grad + 1e-8)

weight -= lr \* grad / ada

return weight, list\_loss

*# 随机梯度下降算法*

def SGD(train\_x, train\_y, weight, lr, iteration, lambdaL2, batch\_size):

list\_loss = []

for i in range(iteration):

predict = np.dot(train\_x, weight)

loss\_bref = predict - train\_y

loss = np.mean(loss\_bref \*\* 2)

list\_loss.append(loss)

rand = np.random.randint(0, train\_x.shape[0], size=batch\_size)

grad = np.dot(train\_x[rand].T, loss\_bref[rand]) / train\_x.shape[0] + lambdaL2 \* weight

weight -= lr \* grad

return weight, list\_loss

1. **训练**

*## hyperparameter*

lr = 0.001

iteration = 20000

lambdaL2 = 0

batch\_size = 64

*## hyperparameter*

weight = np.zeros((train\_x.shape[1], 1))

*# 梯度下降loss较大，超过np.float64存储范围，暂时不进行运算*

*# weight\_gd, list\_loss\_gd = GD(train\_x, train\_y, weight, lr, iteration, lambdaL2)*

weight\_ada\_gd, list\_loss\_ada\_gd = ada\_GD(train\_x, train\_y, weight, lr, iteration, lambdaL2)

weight\_sgd, list\_loss\_sgd = SGD(train\_x, train\_y, weight, lr, iteration, lambdaL2, batch\_size)

plt.figure(figsize=(20, 12))

plt.title("Train\_Process")

plt.xlabel("iteration")

plt.ylabel("loss")

# plt.plot(list\_loss\_gd, label="gd")

plt.plot(list\_loss\_ada\_gd, label="ada\_gd", linewidth=10)

plt.plot(list\_loss\_sgd, label="sgd", linewidth=1)

plt.legend()

plt.show()

1. **测试并保存数据**

# y\_gd = np.dot(test\_x, weight\_gd)

y\_ada\_gd = np.dot(test\_x, weight\_ada\_gd)

y\_sgd = np.dot(test\_x, weight\_sgd)

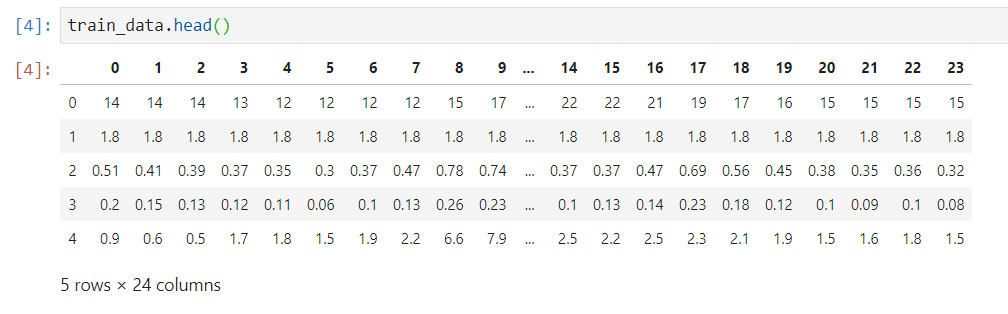
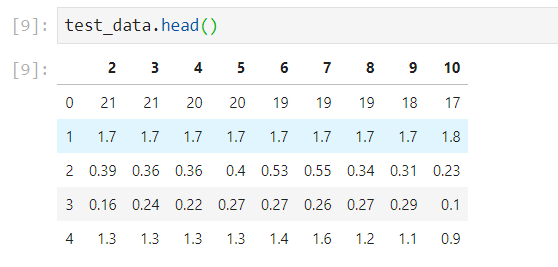
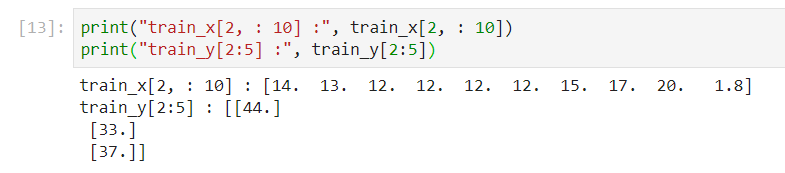
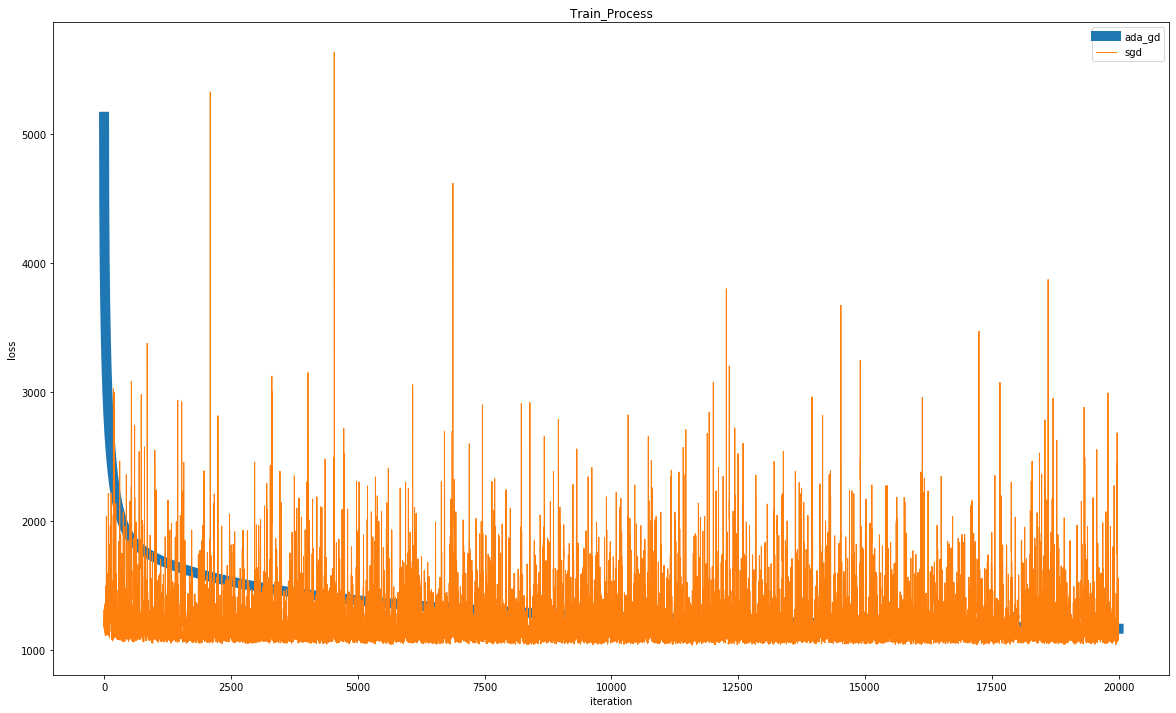
sample\_submission.pop("value")

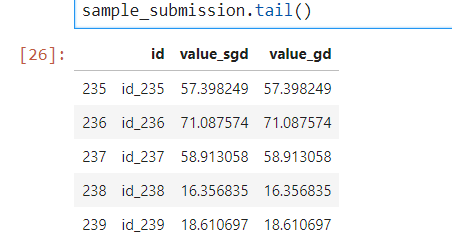
sample\_submission.insert(1, "value\_gd", y\_ada\_gd)

sample\_submission.insert(1, "value\_sgd", y\_sgd)

sample\_submission.tail()

sample\_submission.to\_csv("./dataset/sampleSubmission.csv")

1. **结果分析**
2. **查看训练数据表表头部分**
3. **查看测试数据表表头部分**
4. **检查训练数据是否正确处理**
5. **画出训练过程的loss变化图像(sgd和ada\_gd)** 
6. **查看sample\_submission是否已正确储存**

****

1. **调试报告**
2. **所遇到的问题**

看完实验报告的简述之后，依然感到无从下手。原因是对数据的处理有许多的疑惑。一开始准备使用list来完成数据的处理，网上的教程几乎都是用list进行处理的，但想到既然是.csv数据，使用pandas试试吧。用numpy处理数据，用pandas储存数据，先处理成每个月的数据，再提取每十个小时的数据为一组，大概是这么个思路。可能是第一次自己处理数据吧，以前都只进行过分割操作。但既然是list可以解决的问题，那numpy && pandas肯定是没问题的。

在写gd的时候遇到了loss爆炸的问题，之前用框架的时候没感觉。所以就先把那段代码注释了，而sgd和ada\_gd没出现这个问题。

遇到了shape不一致的问题，用了pdb进行调试，问题得以解决。

1. **经验和体会**

第一次自己写gd、sgd、ada\_gd，之前都是直接用的框架，感觉如果是数据集比较小的话，自己写的可能比使用框架(理解着写的)还要直接简单明了一些。

感觉这门课是实战性很强的一门课程，对于新手来说有些吃力，不过网上的参考挺多的，希望自己能坚持下来吧。