

数据挖掘实验报告

学院: 软件学院

专业: 软件工程

|  |  |
| --- | --- |
| *作者:*  周治平 | *指导教师:*  陶乾 |
|  |  |
| *学号:*  202030483178 | *班级:*  20级软件工程1班 |

2020-10-24

# 线性回归与LSTM

**摘要**

线性回归是利用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。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。利用线性回归，我们可以预测一组特定数据（如GDP、石油价格和股票价格）是否在一段时期内增长或下降。在本次实验中，我们将使用线性回归预测波士顿房价，进一步理解该算法的原理，并通过可视化工具展示和分析数据集。另外，我们还将搭建一个简单的LSTM网络，实现对股市收盘价格的预测。

## **介绍**

1. 学习数据集的可视化和分析；
2. 进一步掌握线性回归的原理；
3. 熟悉LSTM的相关知识。

## **方法和理论**

1线性回归模型

1.1模型定义定义

本实验采用的是线性模型，其定义为



其中为权重矩阵，为自变量，Y为数据标签，b为偏置量。

1.2定义损失函数



1.3求解梯度下降函数





1.4全梯度下降

输入：



输出：

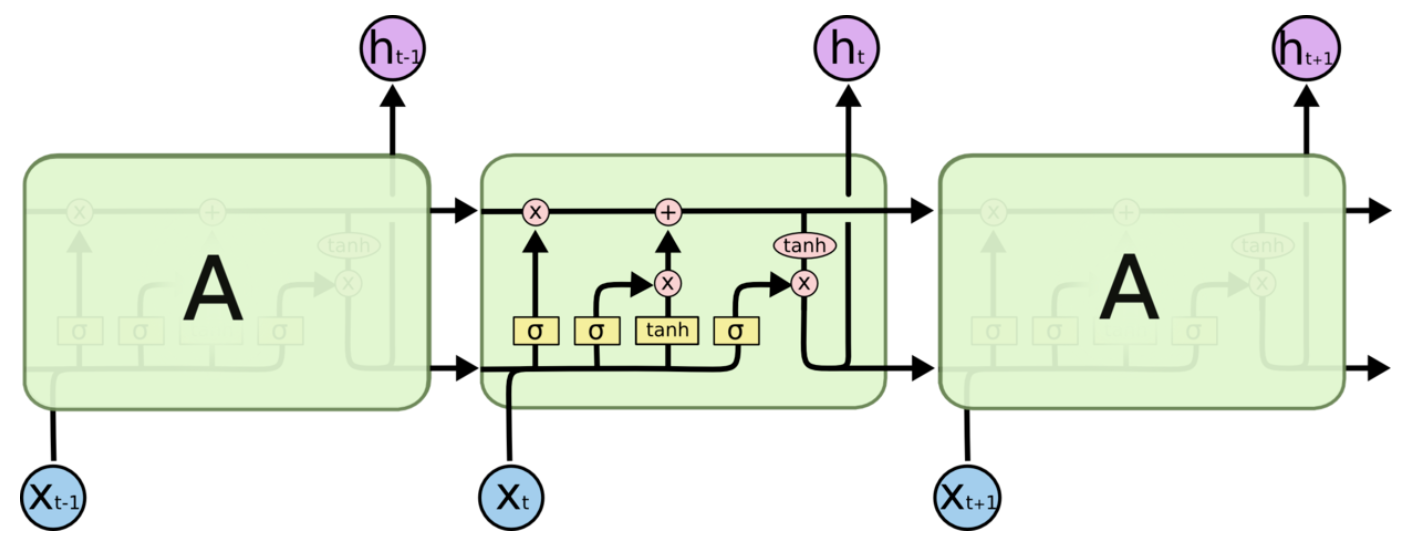


2 LSTM模型

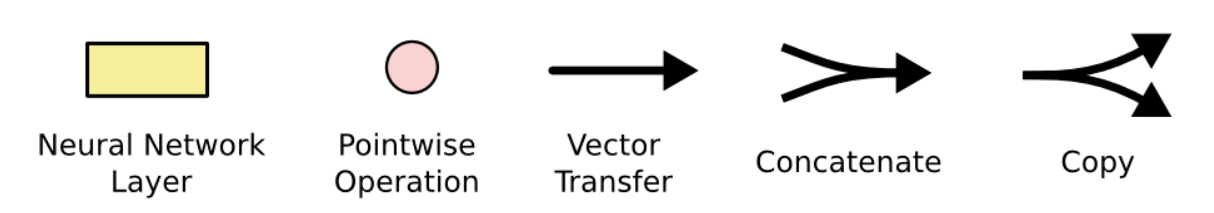
2.1模型介绍

Long Short Term 网络—— 一般就叫做 LSTM ——是一种 RNN 特殊的类型，可以学习长期依赖信息。LSTM 由[Hochreiter & Schmidhuber (1997)](http://deeplearning.cs.cmu.edu/pdfs/Hochreiter97_lstm.pdf" \t "_blank)提出，并在近期被[Alex Graves](https://scholar.google.com/citations?user=DaFHynwAAAAJ&hl=en" \t "_blank)进行了改良和推广。在很多问题，LSTM 都取得相当巨大的成功，并得到了广泛的使用。  
 LSTM 通过刻意的设计来避免长期依赖问题。记住长期的信息在实践中是 LSTM 的默认行为，而非需要付出很大代价才能获得的能力。

2.2模型标准结构



（Figure. 1.LSTM标准结构）

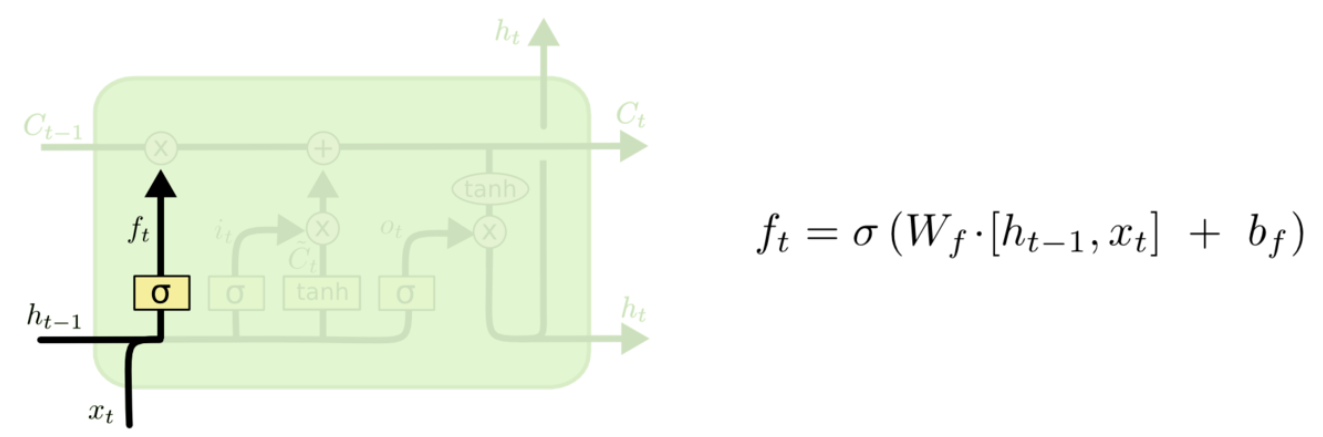


（Figure. 2.LSTM图标解释）

2.3模型解释

2.3.1决定丢弃信息

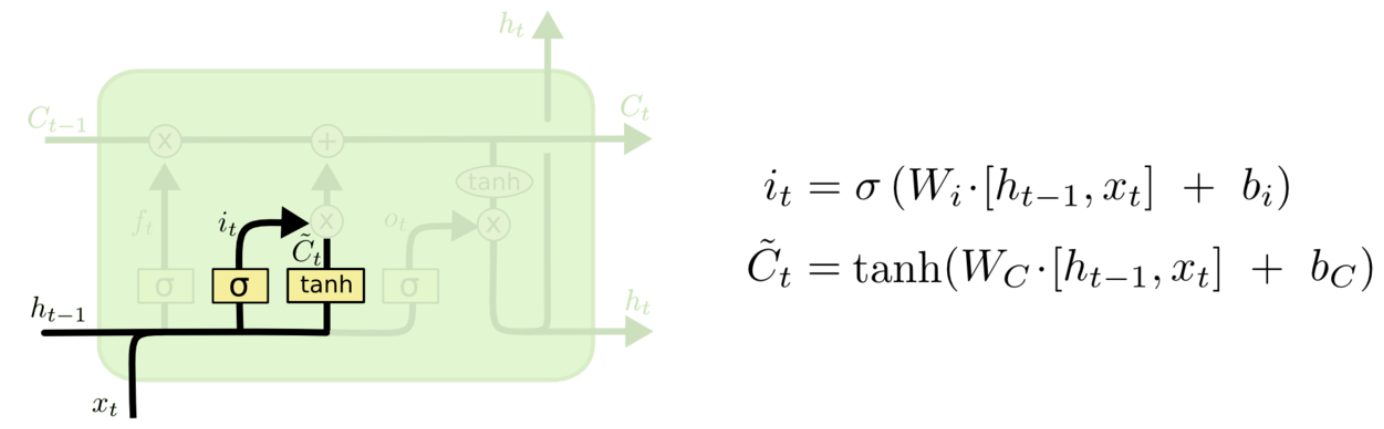
LSTM 中的第一步是决定我们会从细胞状态中丢弃什么信息。这个决定通过一个称为忘记门层完成。该门会读取h\_{t-1}和x\_t，输出一个在 0 到 1 之间的数值给每个在细胞状态C\_{t-1}中的数字。1 表示“完全保留”，0 表示“完全舍弃”。



（Figure. 3.决定丢弃信息结构）

2.3.2确定更新的信息

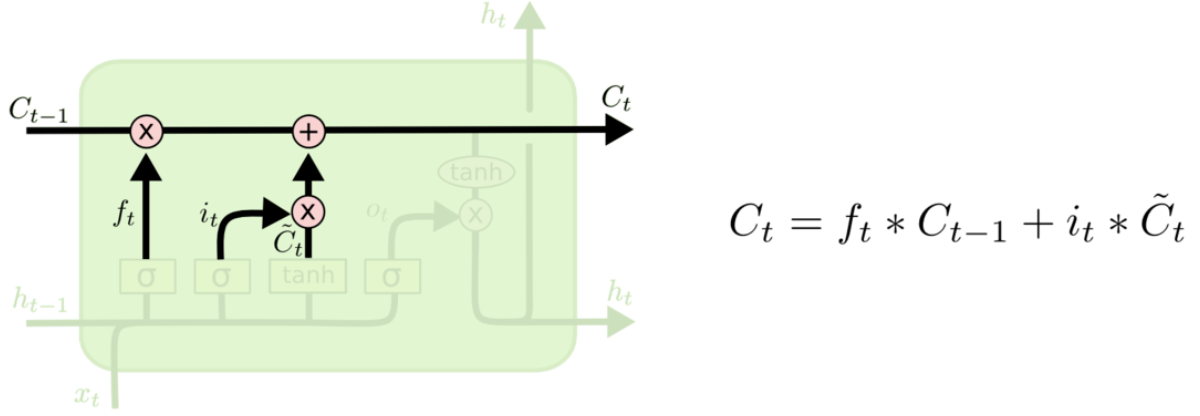
下一步是确定什么样的新信息被存放在细胞状态中。这里包含两个部分。第一，sigmoid 层称 “输入门层” 决定什么值我们将要更新。然后，一个 tanh 层创建一个新的候选值向量，\tilda{C}\_t，会被加入到状态中。



（Figure. 4.确定更新的信息结构）

2.3.3更新细胞状态

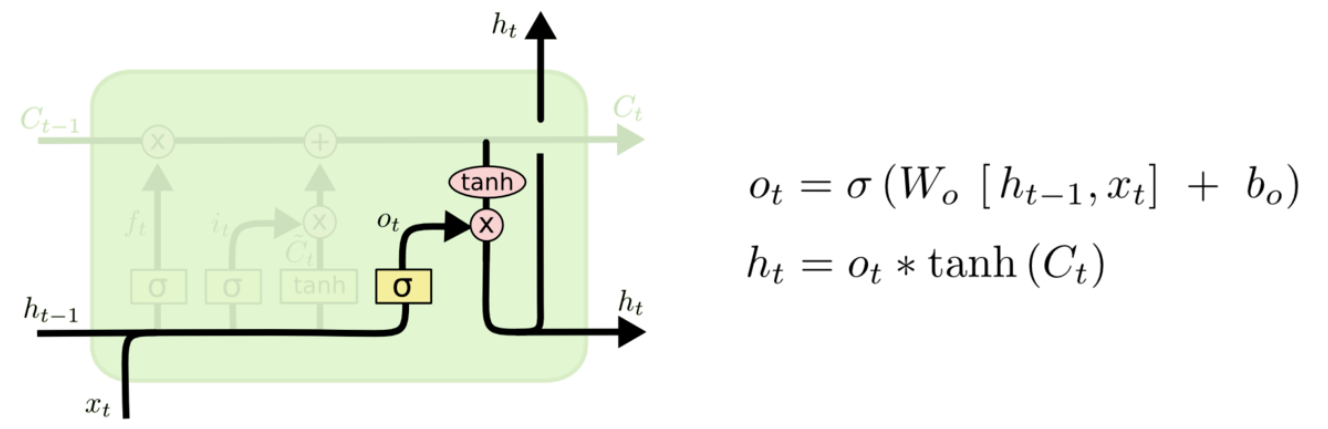
现在是更新旧细胞状态的时间，C\_{t-1}更新为C\_t。前面的步骤已经决定了将会做什么，我们现在就是实际去完成。把旧状态与f\_t相乘，丢弃掉确定需要丢弃的信息。接着加上i\_t \* \tilda{C}\_t。这就是新的候选值，根据我们决定更新每个状态的程度进行变化。



（Figure. 5.更新细胞状态结构）

2.3.4输出信息

最终，我们需要确定输出什么值。这个输出将会基于我们的细胞状态，但是也是一个过滤后的版本。首先，我们运行一个 sigmoid 层来确定细胞状态的哪个部分将输出出去。接着，我们把细胞状态通过 tanh 进行处理（得到一个在 -1 到 1 之间的值）并将它和 sigmoid 门的输出相乘，最终我们仅仅会输出我们确定输出的那部分。



（Figure. 6.输出信息结构）

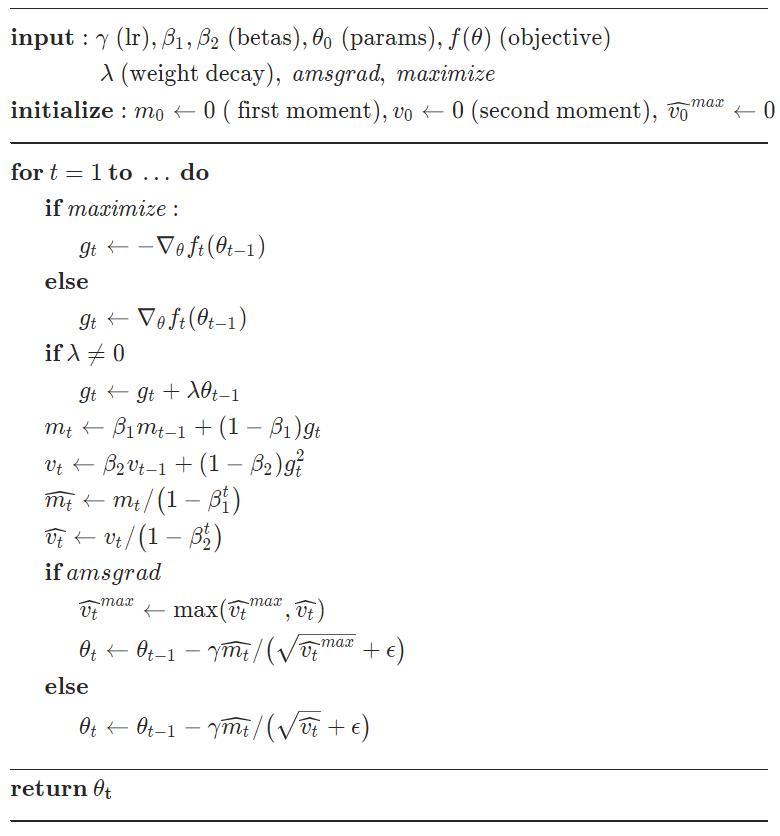
2.3.5模型公式



2.4定义损失函数



2.5定义参数优化器(使用Adam优化算法)

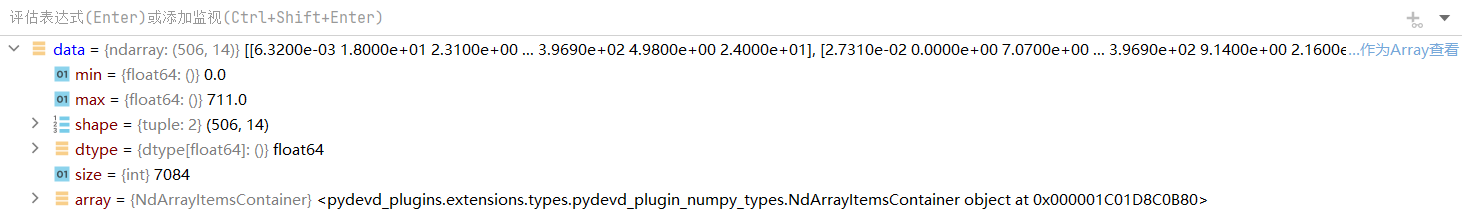


## **实验**

### 数据集

1房价数据集

1.1数据内容



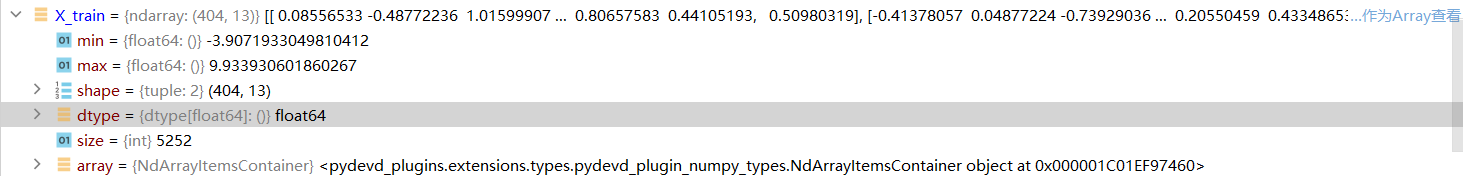
（Figure. 6.房价数据集信息）

1.2数据数量

由Figure.7.可知，房价数据集中有506个样本，每个样本有14列数据（包含13个数据特征以及1个数据标签），因此数据数量一共为7084个。

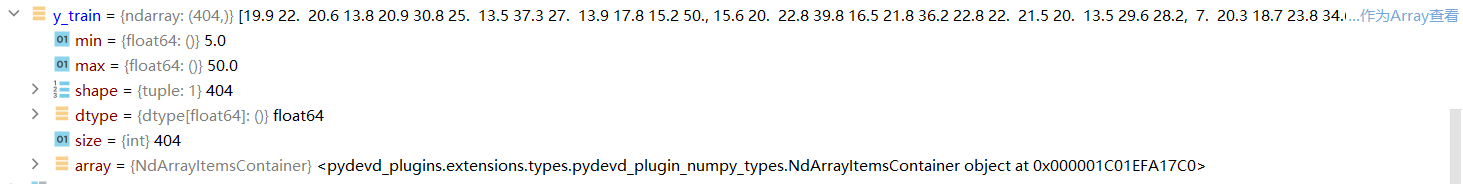
1.3训练数据集

1.3.1数据特征



（Figure. 7.房价训练集数据特征信息）

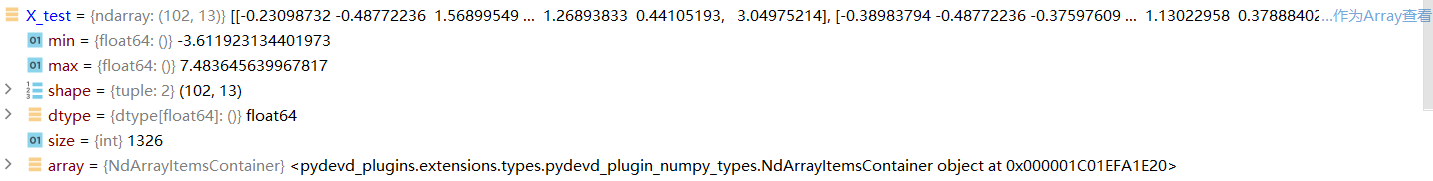
1.3.2数据标签



（Figure. 8.房价训练集数据标签信息）

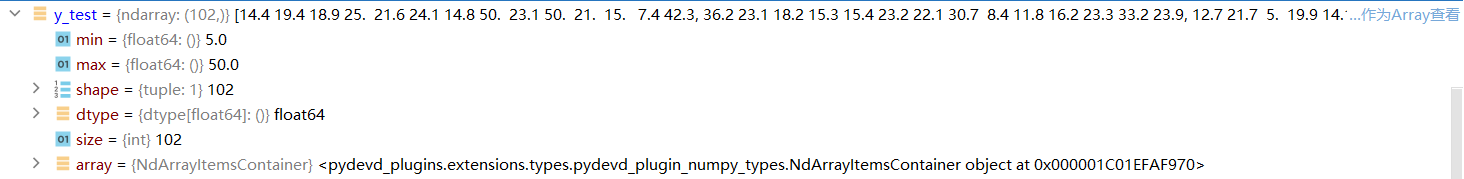
1.4验证数据集

1.4.1数据特征



（Figure. 9.房价测试集数据特征信息）

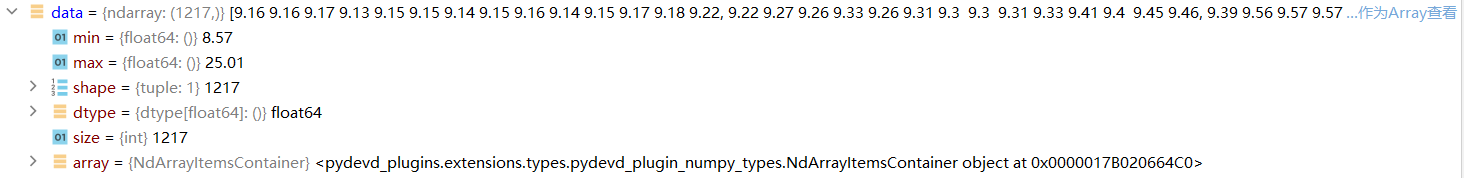
1.4.2数据标签



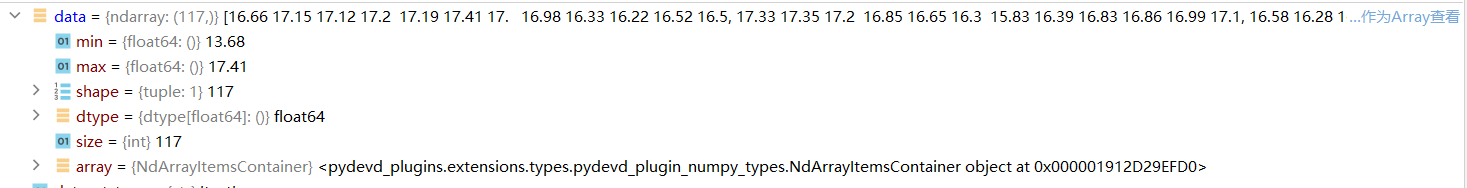
（Figure. 10.房价测试集数据标签信息）

2股票数据集

2.1数据内容



（Figure. 11.股票训练数据集信息）



（Figure. 12.股票测试数据集信息）

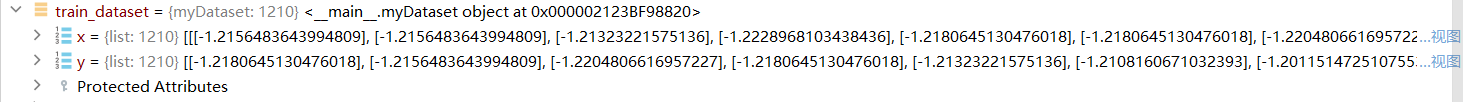
2.2数据数量

由Figure.11.可知，训练数据集中，股票数据集中有1217个样本，其中用每7天的样本数据预测第8天的样本数据，余数6个样本剔除。

由Figure.12.可知，测试数据集中，股票数据集中有117个样本，其中用每7天的样本数据预测第8天的样本数据，余数5个样本剔除。

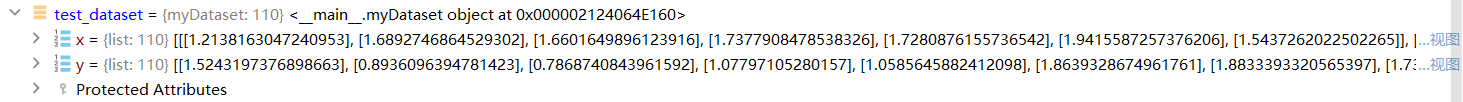
因此数据集总数为1217+117=1334个样本。

2.3训练数据集



（Figure. 13.股票训练数据集信息）

2.4测试数据集



（Figure. 14.股票测试数据集信息）

### 实现

1线性回归模型

1.1模型初始化

1.1.1初始化模型参数

*# 初始化模型参数*def initialize\_params(feature\_num):  
 *#初始化w与b* w=np.random.rand(feature\_num,1)  
 b=0  
 return w,b

1.1.2定义损失函数以及梯度求解函数

*#定义损失函数以及梯度求解函数*def forward(X, y, w, b):  
 *#定义loss函数  
 #w的梯度以及b的梯度* num\_train=X.shape[0]  
 y\_hat=np.dot(X,w)+b  
 loss=np.sum((y\_hat-y)\*\*2)/num\_train  
 dw=np.dot(X.T,(y\_hat-y))/num\_train  
 db=np.sum((y\_hat-y))/num\_train  
 return y\_hat,loss,dw,db

1.1.3定义MSE函数

*# 定义MSE函数*def MSE(y\_test, y\_pred):  
 return np.sum(np.square(y\_pred - y\_test)) / y\_pred.shape[0]

1.1.4定义R系数函数

*# 定义R系数函数*def r2\_score(y\_test, y\_pred):  
 *# 测试集标签均值* y\_avg = np.mean(y\_test)  
 *# 总离差平方和* ss\_tot = np.sum((y\_test - y\_avg) \*\* 2)  
 *# 残差平方和* ss\_res = np.sum((y\_test - y\_pred) \*\* 2)  
 *# R计算* r2 = 1 - (ss\_res / ss\_tot)  
 return r2

1.2模型训练

1.2.1定义模型训练函数

*# 定义线性回归模型的训练过程*def my\_linear\_regression(X, y, learning\_rate, epochs):  
 *#定义模型训练过程* loss\_his=[]*#保存训练误差* w,b=initialize\_params(X.shape[1])*#初始化模型参数  
  
 #开始训练* for i in range(epochs):  
 y\_hat, loss, dw, db=forward(X,y,w,b)  
 *#参数优化* w+=-learning\_rate\*dw  
 b+=-learning\_rate\*db  
  
 loss\_his.append(loss)  
  
 if i%100==0:  
 print(' epoch %d loss %f' %(i,loss))  
  
 return loss\_his,w,b

1.2.2数据导入以及预处理

*# 读数据  
# delimiter分隔符*data = np.loadtxt('./boston\_house\_price.csv', float, delimiter=",", skiprows=1)  
*#X为前13列数据，y为最后一列数据*X, y = data[:, :13], data[:, 13]  
*# Z-score归一化*for i in range(X.shape[1]):  
 X[:, i] = (X[:, i] - np.mean(X[:, i])) / np.std(X[:, i])  
*# 划分训练集、测试集，将数据集的0.2作为测试集*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)  
*# 将训练集、测试集改为列向量的形式*y\_train = y\_train.reshape((-1, 1))  
y\_test = y\_test.reshape((-1, 1))

1.2.3模型训练与绘图

*# 线性回归模型训练，获取数据*loss\_his, w, b = my\_linear\_regression(X\_train, y\_train, 0.01, 5000)  
*# 打印loss曲线*plt.plot(range(len(loss\_his)), loss\_his, linewidth=1, linestyle="solid", label="train loss")  
plt.show()  
*# 打印训练后得到的模型参数*print("w:", w, "\nb", b)

1.3模型预测

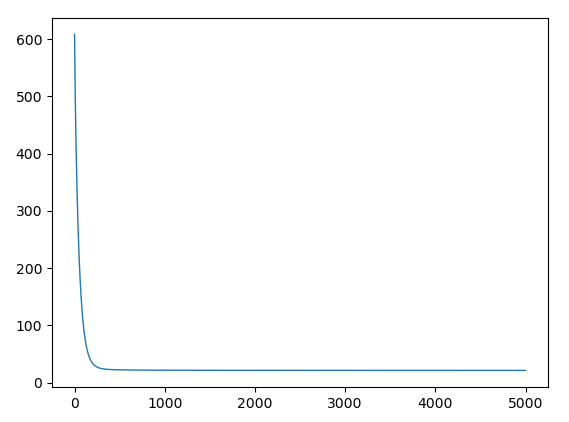
*# 在测试集上预测*y\_pred = np.dot(X\_test, w) + b  
*# 计算测试集的MSE*print("测试集的MSE: {:.2f}".format(MSE(y\_test, y\_pred)))  
*# 计算测试集的R方系数*print("测试集的R2: {:.2f}".format(r2\_score(y\_test, y\_pred)))

1.4预测结果

TABLE I

SIMULATION PARAMETERS

|  |  |
| --- | --- |
| w\_1 | -1.12294266 |
| w\_2 | 0.91670903 |
| w\_3 | 0.30477001 |
| w\_4 | 0.91090737 |
| w\_5 | -1.98387802 |
| w\_6 | 2.70041635 |
| w\_7 | 0.2254662 |
| w\_8 | -2.95812527 |
| w\_9 | 2.78625958 |
| w\_10 | -1.94122577 |
| w\_11 | -2.11902642 |
| w\_12 | 0.76956183 |
| w\_13 | -4.50404499 |
| b | 22.595546259950762 |
| 测试集MSE | 26.77 |
| 测试集R^2 | 0.64 |



（Figure. 15.线性回归拟合结果）

2 LSTM模型回归

2.1模型初始化

2.1.1实验参数设置

*# 实验参数设置*time\_step = 7 *# 用前七天的数据预测第八天*hidden\_size = 4 *# 隐藏层维度*lstm\_layers = 1 *# 网络层数*batch\_size = 64 *# 每一批次训练多少个样例*input\_size = 1 *# 输入层维度*output\_size = 1 *# 输出层维度*lr = 0.05 *# 学习率*

2.1.2定义数据导入类

class myDataset(Dataset):  
 *#导入数据* def \_\_init\_\_(self, x, y):  
 self.x = x  
 self.y = y  
   
 *#获取相应坐标的数据* def \_\_getitem\_\_(self, index):  
 return torch.Tensor(self.x[index]), torch.Tensor(self.y[index])  
   
 *#返回X数据的长度* def \_\_len\_\_(self):  
 return len(self.x)

2.1.3定义LSTM模型

class LSTM(nn.Module):  
 def \_\_init\_\_(self, input\_size, output\_size, hidden\_size, device):  
 super(LSTM, self).\_\_init\_\_()  
  
 *#定义输入数据集维度，输出数据集维度，隐藏层数据集维度，模型运行设备* self.input\_size=input\_size  
 self.ouput\_size=output\_size  
 self.hidden\_size=hidden\_size  
 self.device=device  
  
 def \_one(a,b):  
 *#先类型转换，将list ,numpy转化为tensor  
 # torch.nn.Parameter()  
 # 理解为类型转换函数，将一个不可训练的类型Tensor转换成可以训练的类型parameter，  
 # 并将这个parameter绑定到这个module里面，  
 # 经过类型转换这个 self.v 变成了模型的一部分，  
 # 成为模型中根据训练可以改动的参数* return nn.Parameter(torch.FloatTensor(a,b).to(self.device))  
  
 def \_three():  
 return (\_one(input\_size,hidden\_size),\_one(hidden\_size,hidden\_size),nn.Parameter(torch.zeros(hidden\_size).to(self.device)))  
  
 *#初始化参数* self.W\_xi, self.W\_hi, self.b\_i = \_three()*#输入门参数* self.W\_xf, self.W\_hf, self.b\_f = \_three()*#遗忘门参数* self.W\_xo, self.W\_ho, self.b\_o = \_three()*#隐藏层输出参数* self.W\_xc, self.W\_hc, self.b\_c = \_three()*#候选的细胞参数  
  
 #初始化输出层参数* self.W\_hq=\_one(hidden\_size,output\_size)  
 self.b\_q=nn.Parameter(torch.zeros(output\_size).to(self.device))  
  
 *#创建梯度* self.params=[self.W\_xi,self.W\_hi,self.b\_i,self.W\_xf, self.W\_hf, self.b\_f,self.W\_xo, self.W\_ho, self.b\_o,self.W\_xc, self.W\_hc, self.b\_c,self.W\_hq,self.b\_q]  
  
 for param in self.params:  
 if param.dim()==2:  
 *#使初始化的参数呈正太分布* nn.init.xavier\_normal(param)  
  
 def init\_lstm\_state(self, batch\_size):  
 *#初始化输入参数* return (torch.zeros((batch\_size,self.hidden\_size),device=self.device),  
 torch.zeros((batch\_size,self.hidden\_size),device=self.device))  
   
 *#用于返回预测结果* def forward(self, seq):  
 *#初始化输入参数* (H,C)=self.init\_lstm\_state(seq.shape[0])  
 for step in range(seq.shape[1]):  
 X=seq[:,step,:]  
 *#模型计算  
 #@ 和 \* 代表矩阵的两种相乘方式：  
 # @表示常规的数学上定义的矩阵相乘；  
 # \*表示两个矩阵对应位置处的两个元素相乘  
 #LSTM神经网络正向传播计算* I = torch.sigmoid((X@self.W\_xi)+(H@self.W\_hi)+self.b\_i)  
 F = torch.sigmoid((X @ self.W\_xf) + (H @ self.W\_hf) + self.b\_f)  
 O = torch.sigmoid((X @ self.W\_xo) + (H @ self.W\_ho) + self.b\_o)  
 *#matmul类似于矩阵相乘* C\_tilda = torch.tanh(torch.matmul(X.float(),self.W\_xc)+torch.matmul(H.float(),self.W\_hc)+self.b\_c)  
 C=F\*C+I\*C\_tilda  
 H=O\*torch.tanh(C)  
 *#最终预测结果* Y=(H@self.W\_hq)+self.b\_q  
 return Y,(H,C)

2.1.4初始化训练参数

*# 设定训练轮数*num\_epochs = 50  
device = torch.device('cuda') if torch.cuda.is\_available() else torch.device('cpu')  
hist = np.zeros(num\_epochs)*#用于记录模型每一次迭代的预测误差*

2.1.5初始化损失函数以及参数优化器

*# 定义优化器和损失函数*optimiser = torch.optim.Adam(model.parameters(), lr=lr) *# 使用Adam优化算法*loss\_func = torch.nn.MSELoss(reduction='mean') *# 使用均方差作为损失函数*

2.2模型训练

2.2.1划分数据特征以及数据标签

def read\_dataset(dataset\_type):  
 assert dataset\_type == "train" or dataset\_type == "test"  
 df = pd.read\_csv('stock\_market\_price\_{}.csv'.format(dataset\_type)) *# 读入股票数据* data = np.array(df['close']) *# 获取收盘价序列* data = data[::-1] *# 反转，使数据按照日期先后顺序排列* normalize\_data = (data - np.mean(data)) / np.std(data) *# 标准化* normalize\_data = normalize\_data[:, np.newaxis] *# 增加维度* X, y = [], []  
 for i in range(len(normalize\_data) - time\_step):*#每七天数据预测第八天数据* \_x = normalize\_data[i:i + time\_step]  
 \_y = normalize\_data[i + time\_step]  
 X.append(\_x.tolist())  
 y.append(\_y.tolist())  
 *# plt.figure()  
 # plt.plot(data)  
 # plt.show() # 以折线图展示data* return X, y

2.2.2数据导入

X\_train, y\_train = read\_dataset('train')  
X\_test, y\_test = read\_dataset('test')  
train\_dataset = myDataset(X\_train, y\_train)  
test\_dataset = myDataset(X\_test, y\_test)  
train\_loader = DataLoader(train\_dataset, batch\_size, shuffle=True)  
test\_loader = DataLoader(test\_dataset, 1)

2.2.3模型训练以及绘图

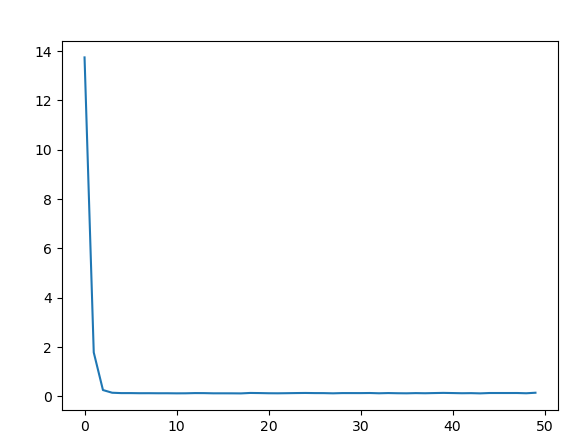
model = LSTM(input\_size, output\_size, hidden\_size, device)

for epoch in range(num\_epochs):  
 epoch\_loss = 0  
 for i, data in enumerate(train\_loader):  
 X, y = data  
 pred\_y, \_ = model(X.to(device))*#预测结果* loss = loss\_func(pred\_y, y.to(device))*#获取损失* optimiser.zero\_grad()*#将梯度归0* loss.backward()*#计算梯度* optimiser.step()*#参数优化* epoch\_loss += loss.item()*#统计损失* print("Epoch ", epoch, "MSE: ", epoch\_loss)  
 hist[epoch] = epoch\_loss*#保存误差*plt.plot(hist)*#绘图每一次迭代的损失*plt.show()*#展示绘图结果*

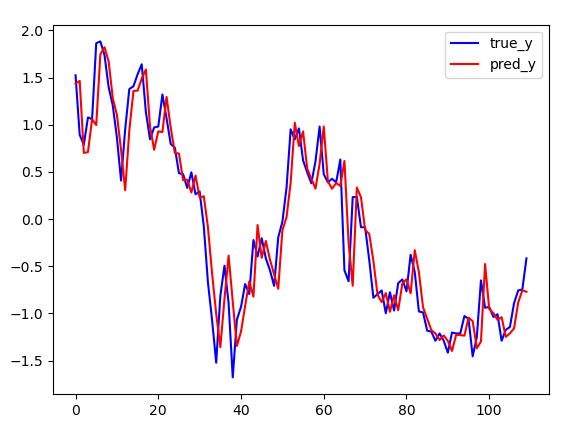
2.3模型预测

*# 测试*model.eval()*#模型预测*result = []*#保存预测结果*for i, data in enumerate(test\_loader):  
 X, y = data  
 pred\_y, \_ = model(X.to(device))  
 result.append(pred\_y.item())  
  
plt.plot(range(len(y\_test)), y\_test, label="true\_y", color="blue")*#蓝线表示真实值*plt.plot(range(len(result)), result, label="pred\_y", color="red")*#红线表示预测值*plt.legend(loc='best')  
plt.show()

2.4预测结果



（Figure. 16.LSTM模型预测误差随迭代次数变化图）

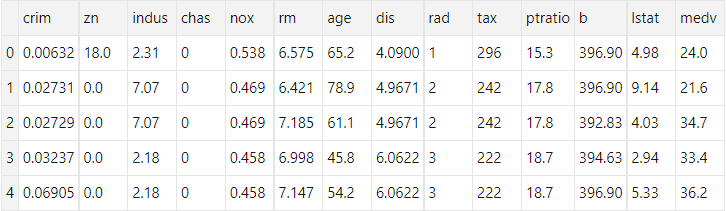


（Figure. 17.LSTM模型预测结果图）

3数据可视化

3.1导入数据集

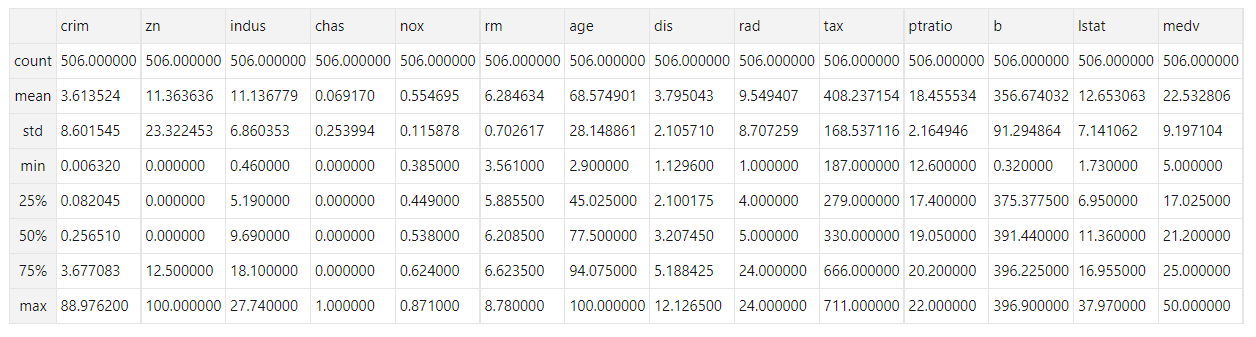
*# 导入Python的数据处理库pandas, 相当于python里的excel*  
**import** pandas **as** pd  
  
*#导入python高级数据可视化库seaborn*  
**import** seaborn **as** sns  
  
*# 导入python绘图matplotlib*  
**import** matplotlib.pyplot **as** plt  
  
**"""**  
**各个字段的含义：**  
**CRIM     犯罪率**  
**ZN       住宅用地所占比例**  
**INDUS    城镇中非商业用地所占比例**  
**CHAS     是否处于查尔斯河边**  
**NOX      一氧化碳浓度**  
**RM       住宅房间数**  
**AGE      1940年以前建成的业主自住单位的占比**  
**DIS      距离波士顿5个商业中心的加权平均距离**  
**RAD      距离高速公路的便利指数**  
**TAX      不动产权税**  
**PTRATIO  学生/教师比例**  
**B        黑人比例**  
**LSTAT    低收入阶层占比**  
**MEDV     房价中位数**  
**"""**  
  
*# 读取数据集*  
df = pd.read\_csv(**'./boston\_house\_price.csv'**, encoding=**'utf-8'**)  
*# 展示前五行*  
df.head()



（Figure. 18.数据集中前5行数据）

3.2统计信息

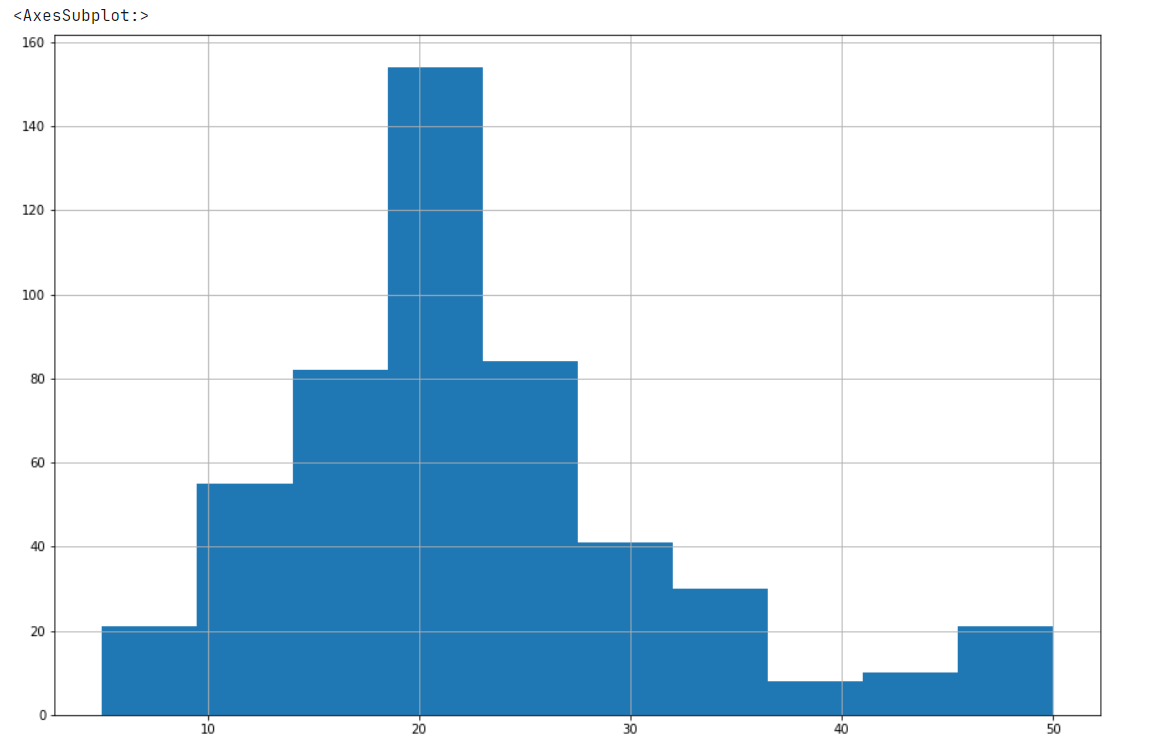
*#统计信息*  
df.describe()



（Figure. 19.房价信息统计）

3.3绘制房价直方图

*#绘制房价的直方图*  
df[**'medv'**].hist()

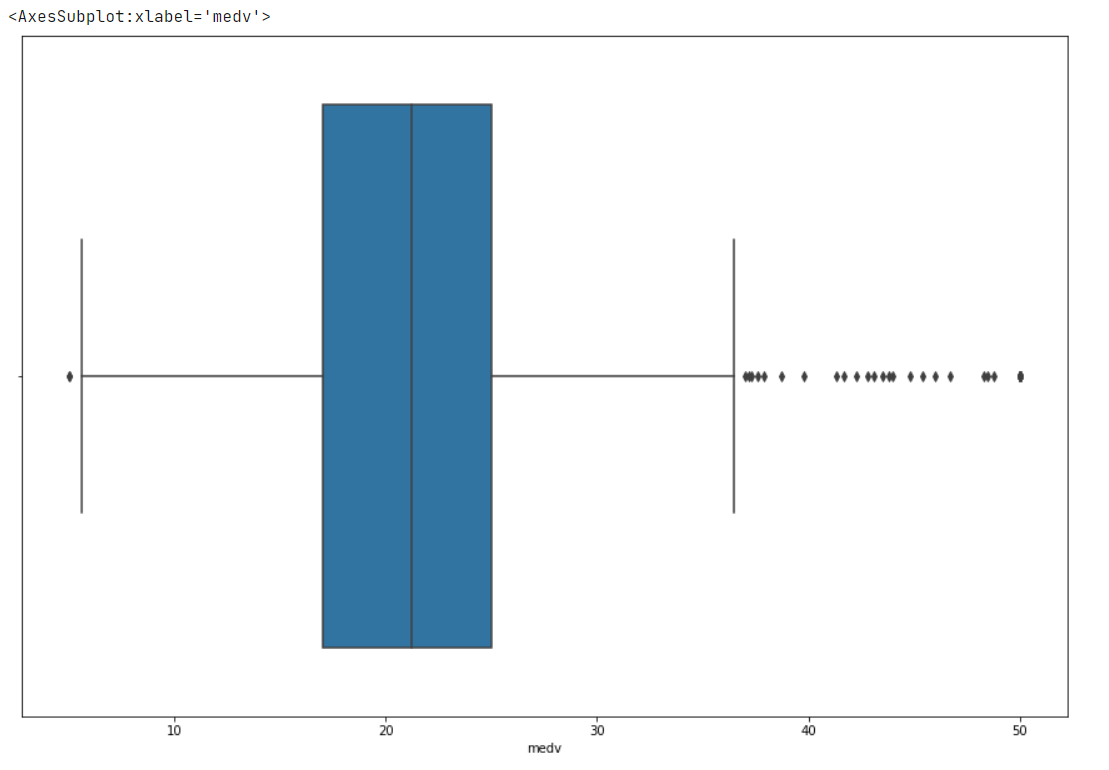


（Figure. 20.房价直方图）

3.4绘制房价的箱线图

*#绘制房价的箱线图*

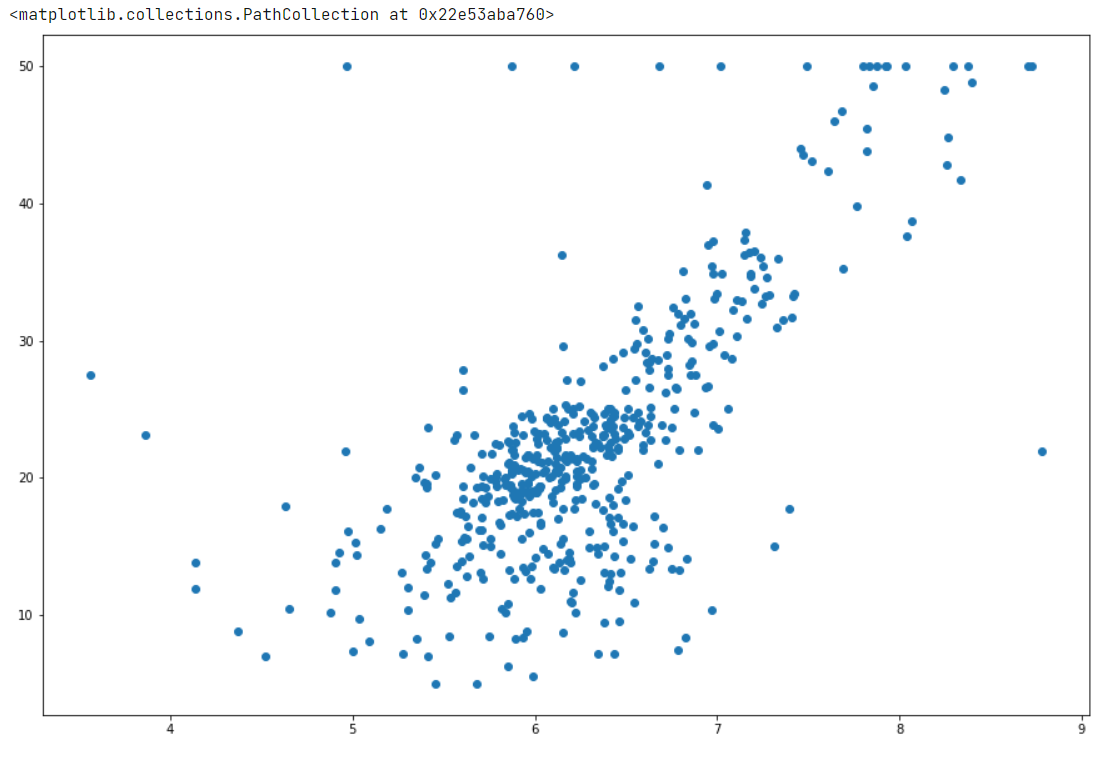
sns.boxplot(df[**'medv'**])



（Figure. 21.房价箱线图）

3.5绘制房间数的散点图

*#绘制房间数的散点图*  
plt.scatter(df[**'rm'**],df[**'medv'**])



（Figure. 22.房价散点图）

3.6筛除房间数的异常点的统计信息

**def** box\_plot\_outliers(df,s):  
    q1,q3=df[s].quantile(0.25),df[s].quantile(0.75)  
    iqr=q3-q1  
    low,up=q1-1.5\*iqr,q3+1.5\*iqr  
    df=df[(df[s]>up)|(df[s]<low)]  
    **return** df  
  
df\_filter=box\_plot\_outliers(df,**'rm'**)  
df\_filter.mean()

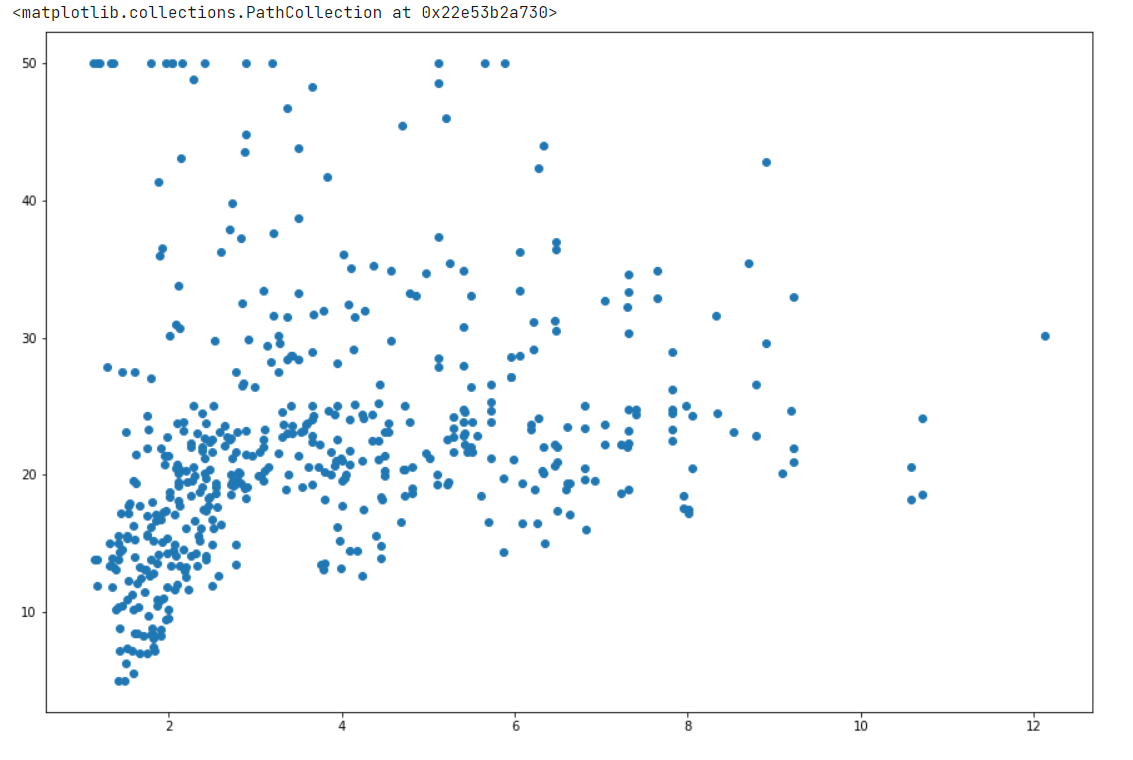
TABLE II

房间异常点统计信息

|  |  |
| --- | --- |
| crim | 5.994001 |
| zn | 15.400000 |
| indus | 9.715667 |
| chas | 0.133333 |
| nox | 0.556303 |
| rm | 7.100533 |
| age | 74.393333 |
| dis | 3.033943 |
| rad | 10.700000 |
| tax | 397.166667 |
| ptratio | 17.163333 |
| b | 352.374667 |
| lstat | 10.098333 |
| medv | 37.420000 |
| dtype | float64 |

3.7输出距离和房价的散点图

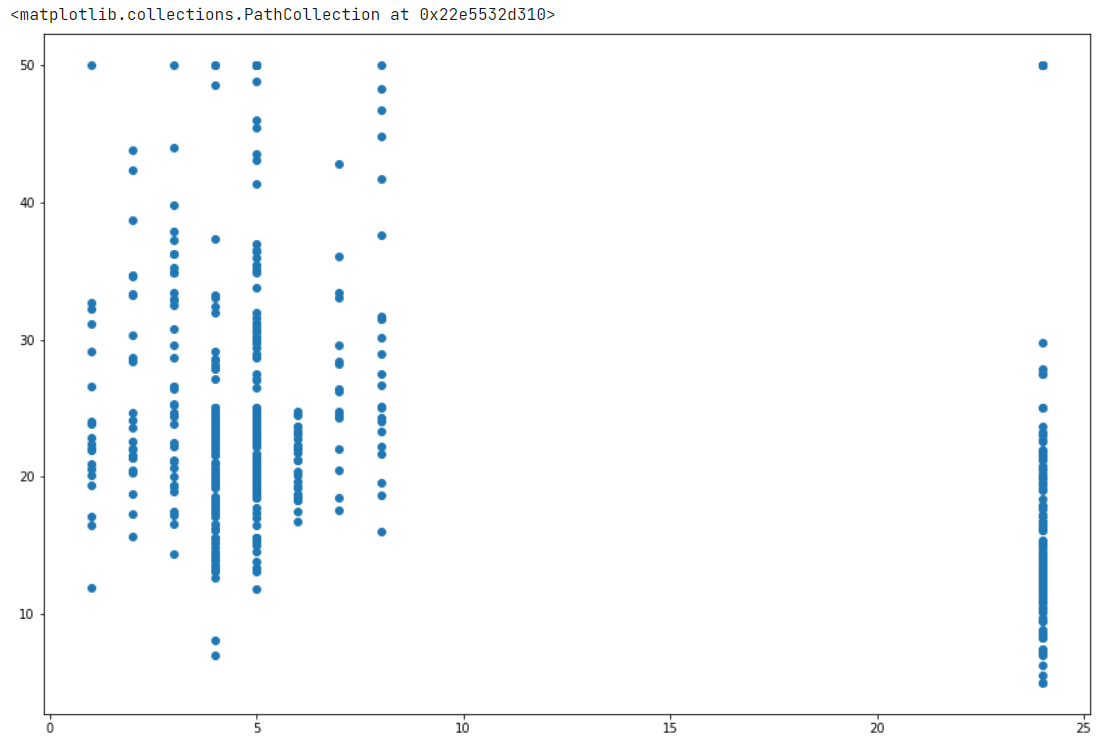
*#输出距离和房价的散点图*  
plt.scatter(df[**'dis'**],df[**'medv'**])



（Figure. 23.距离与房价的散点图）

3.8输出rad和房价的散点图

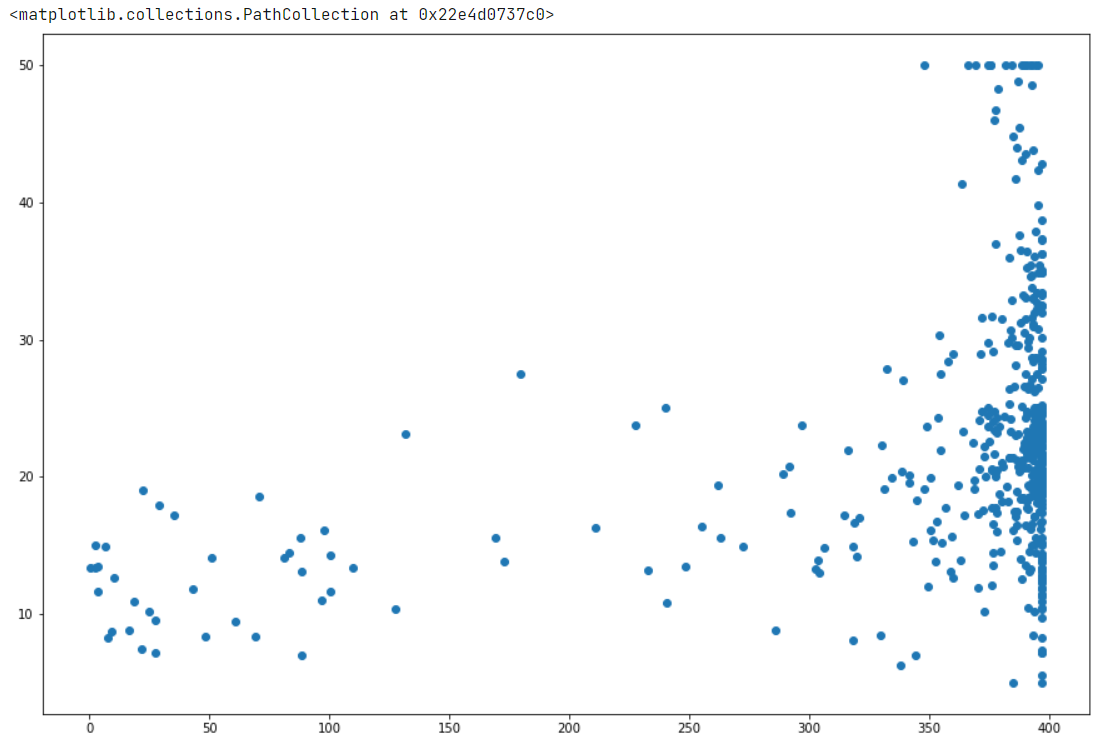
*#输出rad和房价的散点图*  
plt.scatter(df[**'rad'**],df[**'medv'**])



（Figure. 24.rad与房价的散点图）

3.9输出B和房价的散点图

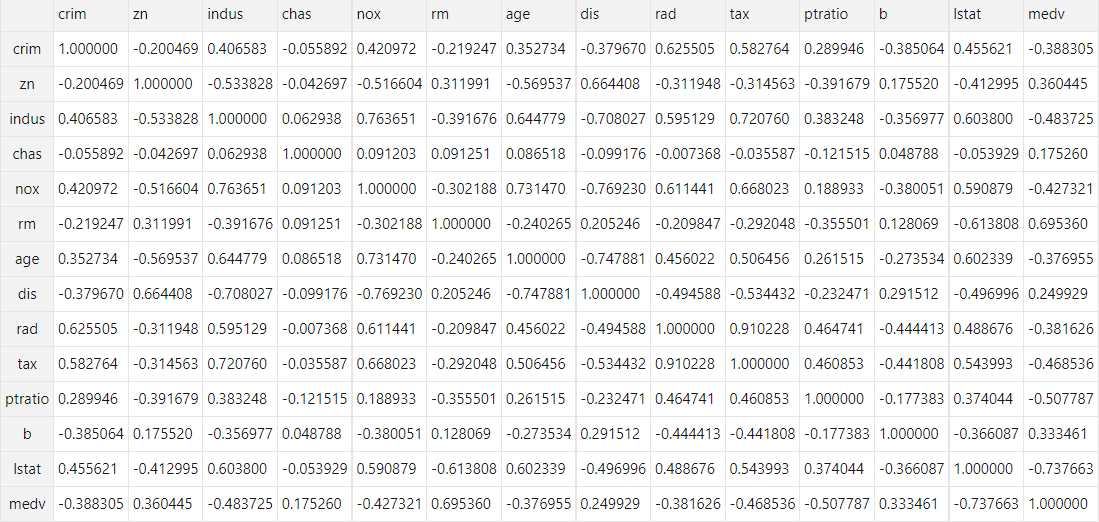
*#输出B和房价的散点图*  
plt.scatter(df[**'b'**],df[**'medv'**])



（Figure. 25.B与房价的散点图）

3.10输出两个变量之间的相关系数

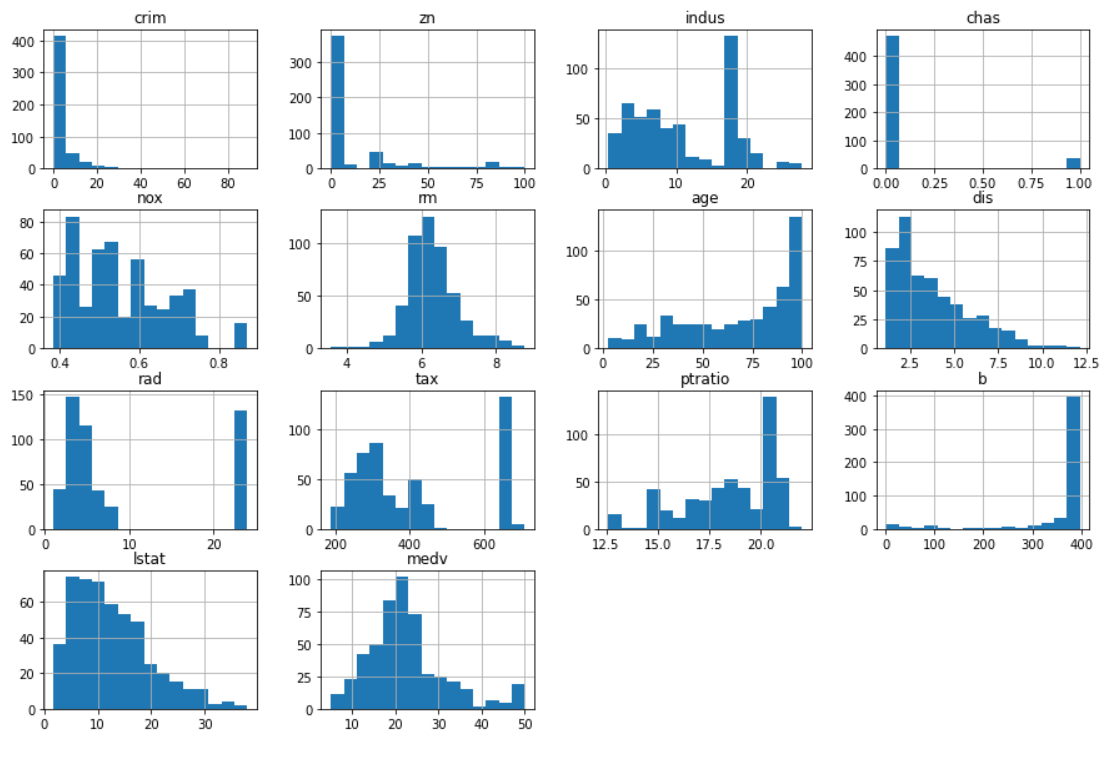
*#两个变量之间的相关系数*  
df.corr()



（Figure. 26.两个变量之间的相关系数）

3.11绘制直方图

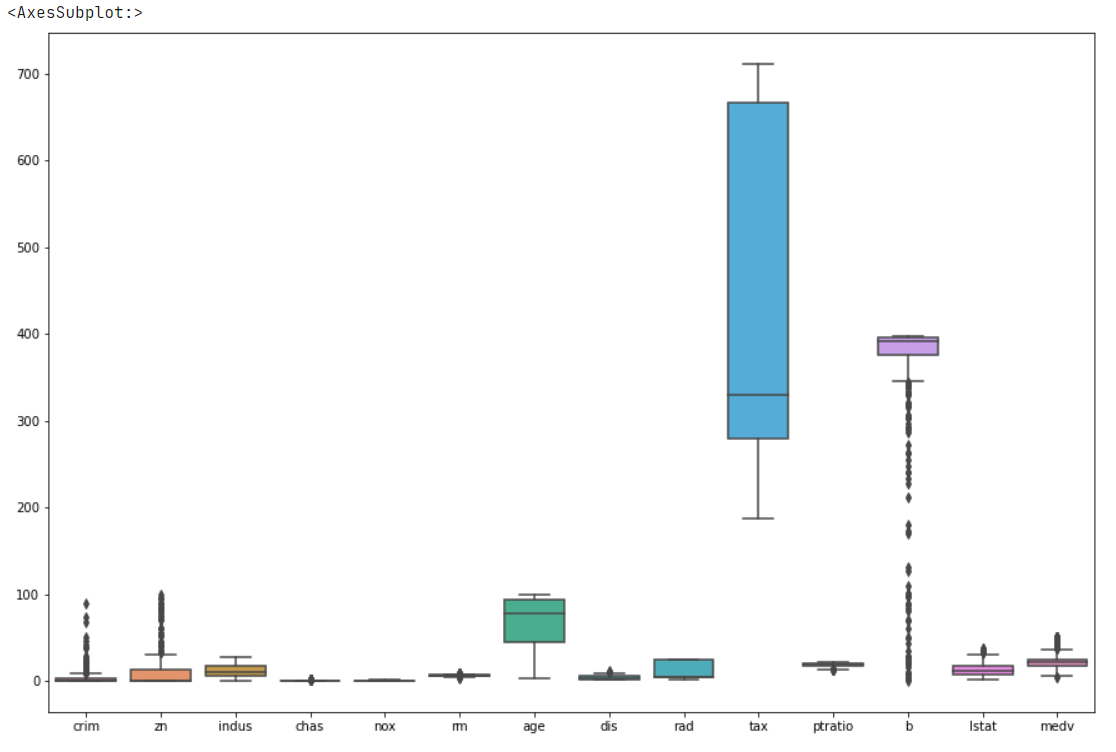
*#设置绘图大小*  
plt.style.use({**'figure.figsize'**:(15,10)})  
*#绘制直方图*  
df.hist(bins=15)



（Figure. 27.各变量的直方图）

3.12绘制箱线图

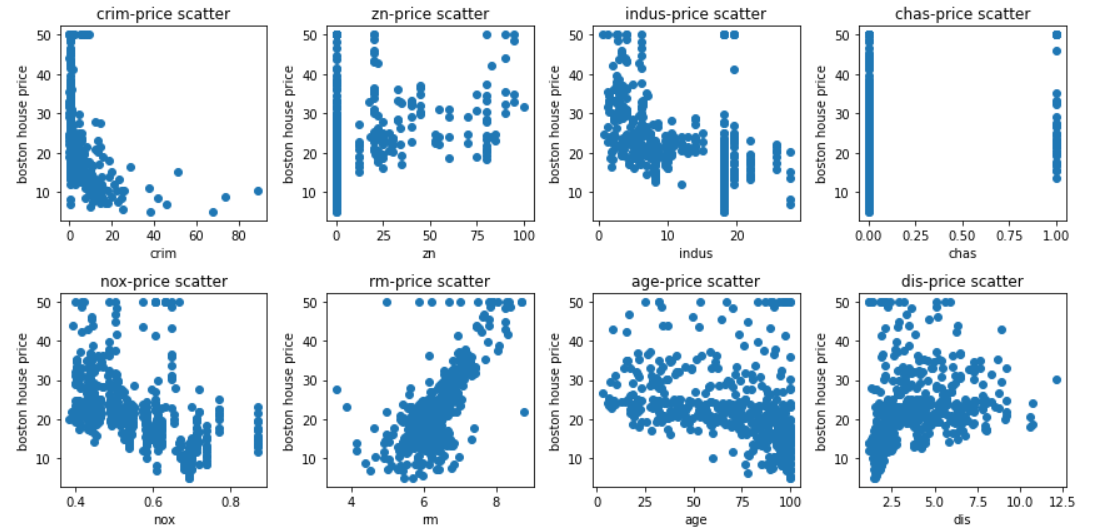
*#绘制箱线图*  
sns.boxplot(data=df)

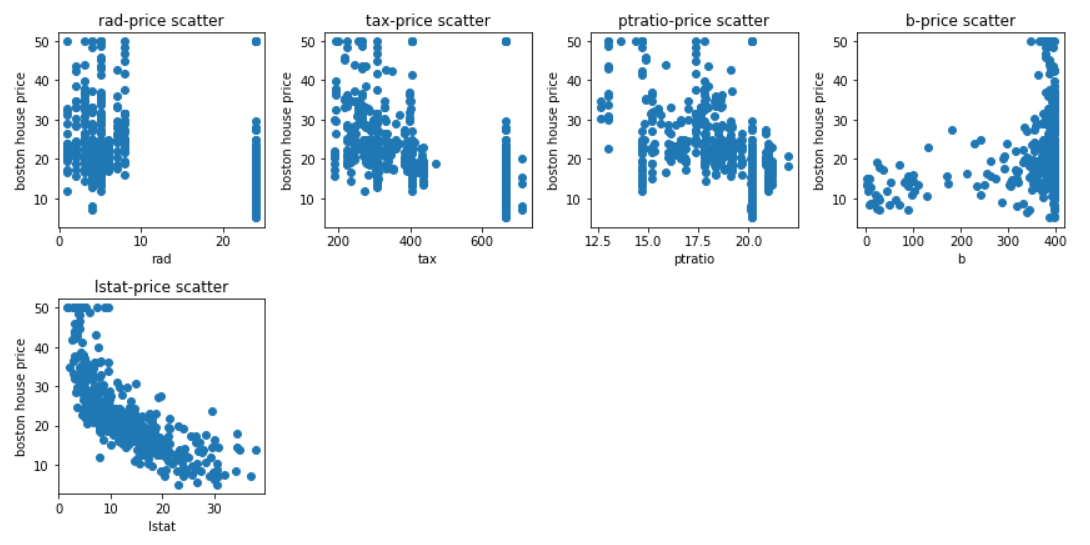


（Figure. 28.各变量的箱线图）

3.13绘制散点图

*#绘制散点图*  
plt.figure(figsize=(12,12))  
**for** i **in** range(13):  
    plt.subplot(4,4,(i+1))  
    plt.scatter(df.iloc[:,i],df[**'medv'**])  
    plt.title(**'{}-price scatter'**.format(df.columns[i]))  
    plt.xlabel(df.columns[i])  
    plt.ylabel(**'boston house price'**)  
  
plt.tight\_layout()





（Figure. 29.各变量的散点图）

## **结论**

在本次实验中，主要是实现了线性回归算法以及LSTM算法以及数据可视化。

在线性回归算法中，其主要步骤是先导入数据，划分训练集与验证集，然后定义线性模型，由于模型中有14个特征（13个特征+1个偏置量），因此定义权重系数有14个，然后定义损失函数，然后通过全批量梯度下降训练的方式进行训练，进行参数优化，最后训练结果输出与绘图以及预测就完成了。

此外还有LSTM算法，是一种 RNN 特殊的类型，其主要通过一系列输入门，遗忘门，输出门组成的神经元网络，通过长期循环对数据进行训练，以达到参数迭代优化的效果，其主要步骤是先定义每一个门中的转化函数，以及结果输出函数，然后定义损失函数，最后通过Adam算法对参数进行优化，最终通过迭代获取优化后的模型参数，然后对模型进行预测就完成了。

在数据可视化中，主要学会了对变量的统计，单个变量的直方图，箱线图的绘制，以及多个变量或整个数据集的直方图，箱线图以及散点图的绘制，然后还有变量数据的异常点筛除，相关系数矩阵的计算等。在数据可视化中充当着不可或缺的角色。

对于本次实验而言，主要有代码参考，加上线性回归算法结构比较简单，其算法模型比较容易理解，而LSTM需要结合其网络结构以及代码才能进一步深入地理解其算法内核，以及数据可视化的操作，但更多是了解如何去进行代码实现，这与在课本上仅学习到的公式是相差很大的。此外还学习到了许多机器学习的库，让自己对这些库中的函数有了更进一步的了解与运用。总体而言本次实验的收获是挺大的。

## **拓展实验**

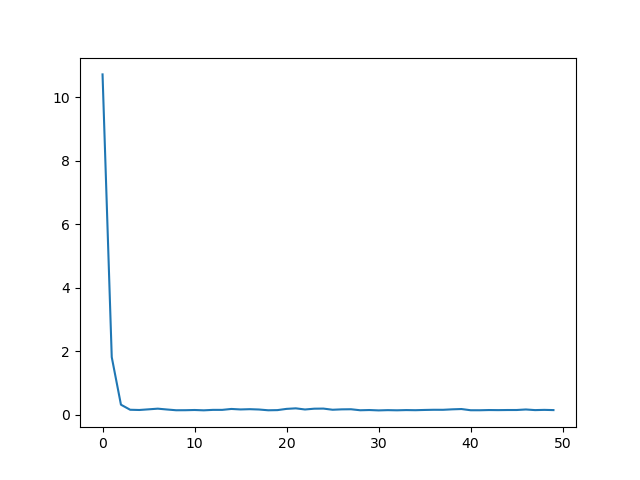
1你可以使用两层的LSTM预测股票收盘价的变化吗？

1.1架构实现

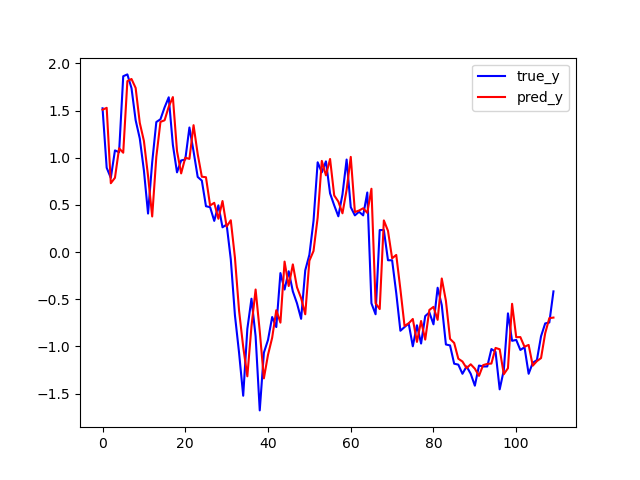
1. class LSTM\_Model(nn.Module):
2. def \_\_init\_\_(self,input\_size, output\_size, hidden\_size, device):
3. super(LSTM\_Model, self).\_\_init\_\_()
4. self.lstm1=LSTM(input\_size, output\_size, hidden\_size, device)
5. self.lstm2 =LSTM(output\_size, output\_size, hidden\_size, device)
6. self.device=device
7. def forward(self,seq):
8. *#print(seq)*
9. x, \_ =self.lstm1(seq)
10. *#print(x)*
11. size=x.shape[0]
12. x=x.resize(size,1,1)
13. *#print(x)*
14. return self.lstm2(x)

具体实现详见 LSTM2.py

1.2实验结果



（Figure. 30.两层LSTM模型预测误差随迭代次数变化图）



（Figure. 31.两层LSTM模型预测结果图）

1.3实验结论

由实验结果可知，采用两层LSTM模型与单层LSTM模型预测结果相似，因此可见单层的LSTM模型的拟合效果已经较佳，如果LSTM模型层数过多，容易出现过拟合现象。总而言之，两层LSTM模型预测股票收盘价的变化是可行的。

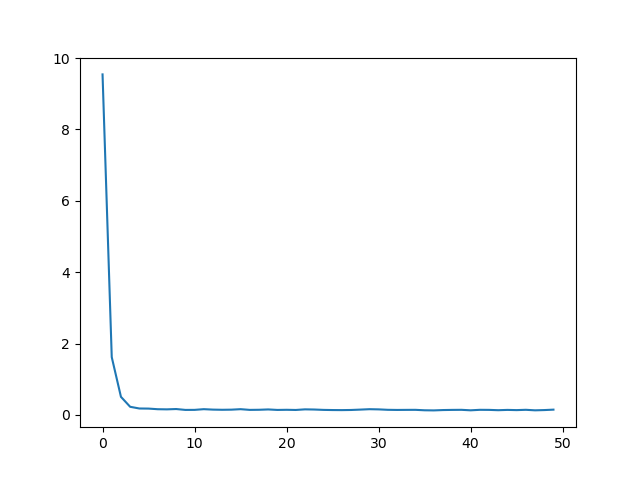
2你可以利用前几天的收盘价、开盘价、交易额等数据预测下一天的收盘价吗？

2.1架构实现（采用7天的收盘价、开盘价、交易额预测第8天的收盘价）

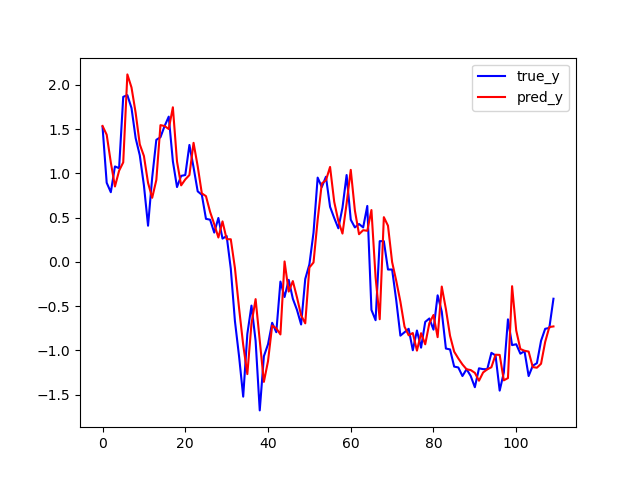
1. def read\_dataset(dataset\_type):
2. assert dataset\_type == "train" or dataset\_type == "test"
3. df = pd.read\_csv('stock\_market\_price\_{}.csv'.format(dataset\_type))  *# 读入股票数据*
4. data = np.array(df['close'])  *# 获取收盘价序列*
5. data2=np.array(df['open'])
6. data3=np.array(df['amount'])
7. *#print(data)*
8. data = data[::-1]  *# 反转，使数据按照日期先后顺序排列*
9. data2=data2[::-1]
10. data3=data3[::-1]
11. normalize\_data1 = (data - np.mean(data)) / np.std(data)  *# 标准化*
12. normalize\_data1 = normalize\_data1[:, np.newaxis]  *# 增加维度*
13. normalize\_data2 = (data2 - np.mean(data2)) / np.std(data2)  *# 标准化*
14. normalize\_data2 = normalize\_data2[:, np.newaxis]  *# 增加维度*
15. normalize\_data3 = (data3 - np.mean(data3)) / np.std(data3)  *# 标准化*
16. normalize\_data3 = normalize\_data3[:, np.newaxis]  *# 增加维度*
17. normalize\_data=np.c\_[normalize\_data1,normalize\_data2,normalize\_data3]
18. *#print(normalize\_data)*
19. X,y = [],[]
20. for i in range(len(normalize\_data) - time\_step):*#每七天数据预测第八天数据*
21. \_x = normalize\_data[i:i + time\_step,:]
22. \_y = normalize\_data1[i + time\_step]
23. X.append(\_x.tolist())
24. y.append(\_y.tolist())
25. *# plt.figure()*
26. *# plt.plot(data)*
27. *# plt.show() # 以折线图展示data*
28. return X, y

具体实现详见LTSM3.py

2.2实验结果



（Figure. 32.多个特征下LSTM模型预测误差随迭代次数变化图）



（Figure. 33.多个特征下LSTM模型预测结果图）

2.3实验结论

由实验结果可知，多个特征下LSTM模型预测效果比单个收盘价特征下的预测效果要差，分析原因可能是因为其他特征与单个特征相比，其他特征可能充当了噪声数据的角色，导致预测效果下降，但总体来说预测效果还是较好的。因此利用前几天的收盘价、开盘价、交易额等数据预测下一天的收盘价是可行的。