2 Detail of linear regression

2.1 Introduction

2.1.1 About This Experiment

This experiment mainly uses basic Python code and the simplest data to reproduce how a linear regression algorithm iterates and fits the existing data distribution step by step.

The experiment mainly used Numpy module and Matplotlib module.Numpy for calculation, Matplotlib for drawing.

2.1.2 Objectives

The main purpose of this experiment is as follows.

- Familiar with basic Python statements
- Master the implementation steps of linear regression

2.2 Experiment Code

2.2.1 Data preparation

10 data were randomly set, and the data were in a linear relationship.

The data is converted to array format so that it can be computed directly when multiplication and addition are used.

Code:

```
#Import the required modules, numpy for calculation, and Matplotlib for drawing
import numpy as np
import matplotlib.pyplot as plt
#This code is for jupyter Notebook only
%matplotlib inline

# define data, and change list to array
x = [3,21,22,34,54,34,55,67,89,99]
x = np.array(x)
y = [1,10,14,34,44,36,22,67,79,90]
y = np.array(y)

#Show the effect of a scatter plot
plt.scatter(x,y)
```

Output:

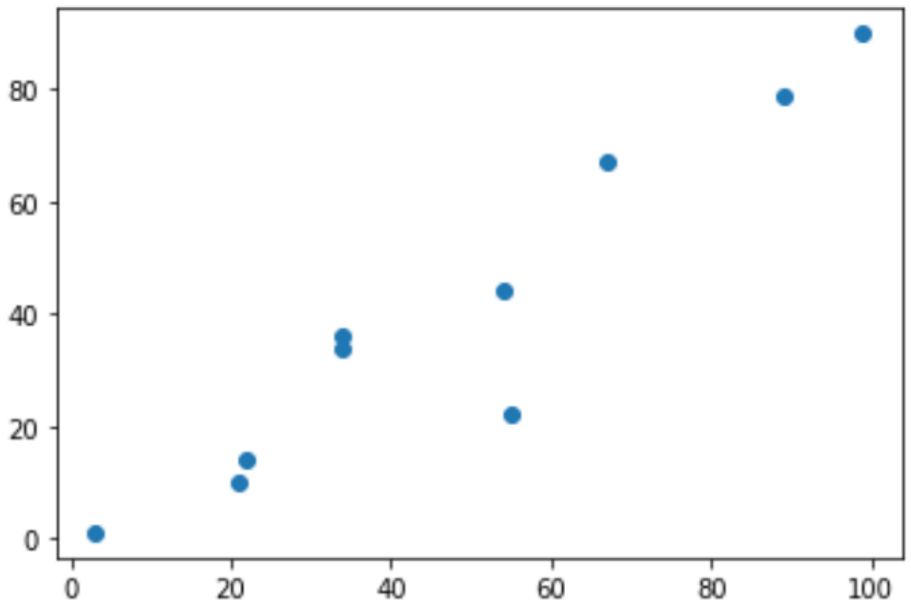

Figure 2-1 Scatter Plot

2.2.2 Define related functions

Model function: Defines a linear regression model wx+b.

Loss function: loss function of Mean square error.

Optimization function: gradient descent method to find partial derivatives of w and b.

Code:

```
#The basic linear regression model is wx+ b, and since this is a two-dimensional space, the model is ax+ b

def model(a, b, x):
    return a*x + b

#The most commonly used loss function of linear regression model is the loss function of mean variance difference
def loss_function(a, b, x, y):
    num = len(x)
    prediction=model(a,b,x)
    return (0.5/num) * (np.square(prediction-y)).sum()

#The optimization function mainly USES partial derivatives to update two parameters a and b
def optimize(a,b,x,y):
    num = len(x)
    prediction = model(a,b,x)
    #Update the values of A and B by finding the partial derivatives of the loss function on a and b
    da = (1.0/num) * ((prediction -y)*x).sum()
    db = (1.0/num) * ((prediction -y).sum())
```

```
    a = a - Lr*da
    b = b - Lr*db
    return a, b

#iterated function, return a and b
def iterate(a,b,x,y,times):
    for i in range(times):
        a,b = optimize(a,b,x,y)
    return a,b
```

2.2.3 Start the iteration

Step 1 Initialization and Iterative optimization model

Code:

```
#Initialize parameters and display
a = np.random.rand(1)
print(a)
b = np.random.rand(1)
print(b)
Lr = 1e-4

#For the first iteration, the parameter values, losses, and visualization after the iteration are displayed
a,b = iterate(a,b,x,y,1)
prediction=model(a,b,x)
loss = loss_function(a, b, x, y)
print(a,b,loss)
plt.scatter(x,y)
plt.plot(x,prediction)
```

Output:

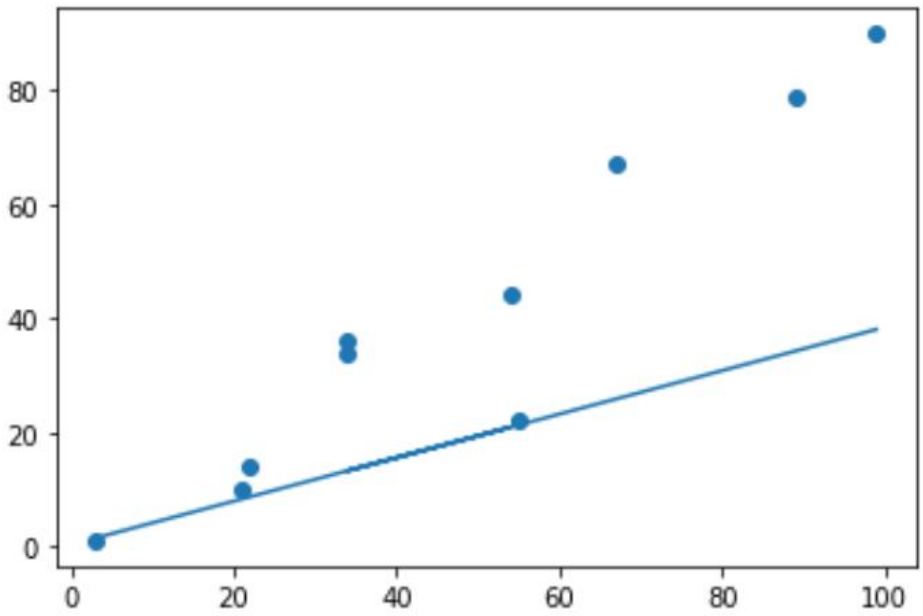

Figure 2-2 Iterate 1 time

Step 2 In the second iteration, the parameter values, loss values and visualization effects after the iteration are displayed

Code:

```
a,b = iterate(a,b,x,y,2)
prediction=model(a,b,x)
loss = loss_function(a, b, x, y)
print(a,b,loss)
plt.scatter(x,y)
plt.plot(x,prediction)
```

Output:

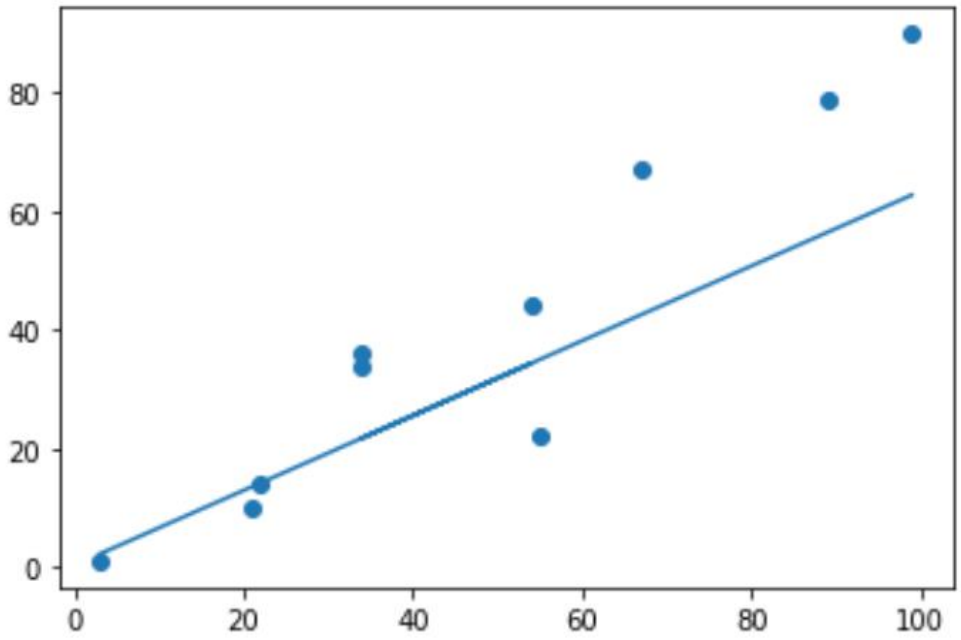

Figure 2-3 Iterate 2 times

Step 3 The third iteration shows the parameter values, loss values and visualization after iteration

Code:

```
a,b = iterate(a,b,x,y,3)
prediction=model(a,b,x)
loss = loss_function(a, b, x, y)
print(a,b,loss)
plt.scatter(x,y)
plt.plot(x,prediction)
```

Output:

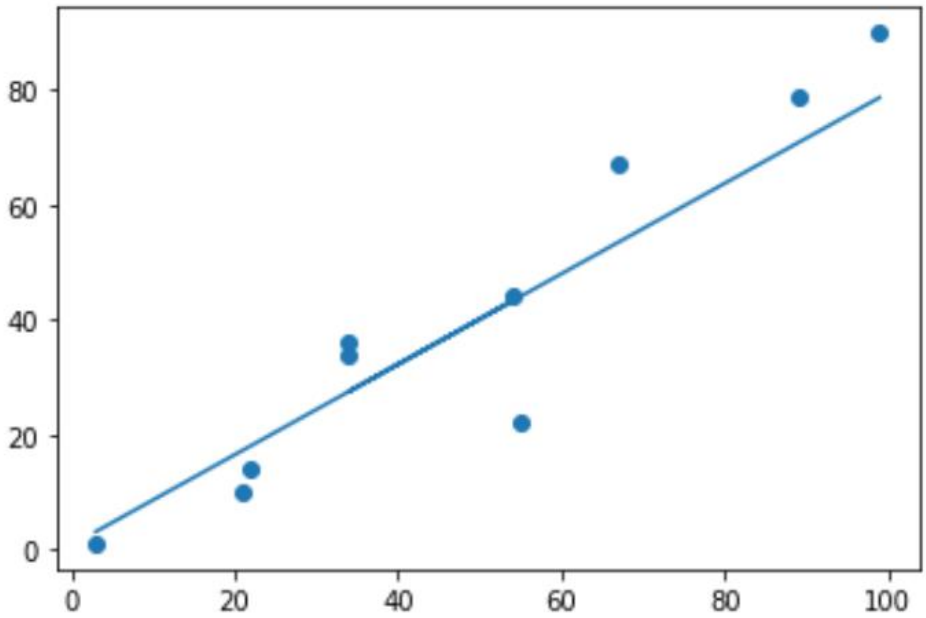

Figure 2-4 Iterate 3 times

Step 4 In the fourth iteration, parameter values, loss values and visualization effects are displayed

Code:

```
a,b = iterate(a,b,x,y,4)
prediction=model(a,b,x)
loss = loss_function(a, b, x, y)
print(a,b,loss)
plt.scatter(x,y)
plt.plot(x,prediction)
```

Output:

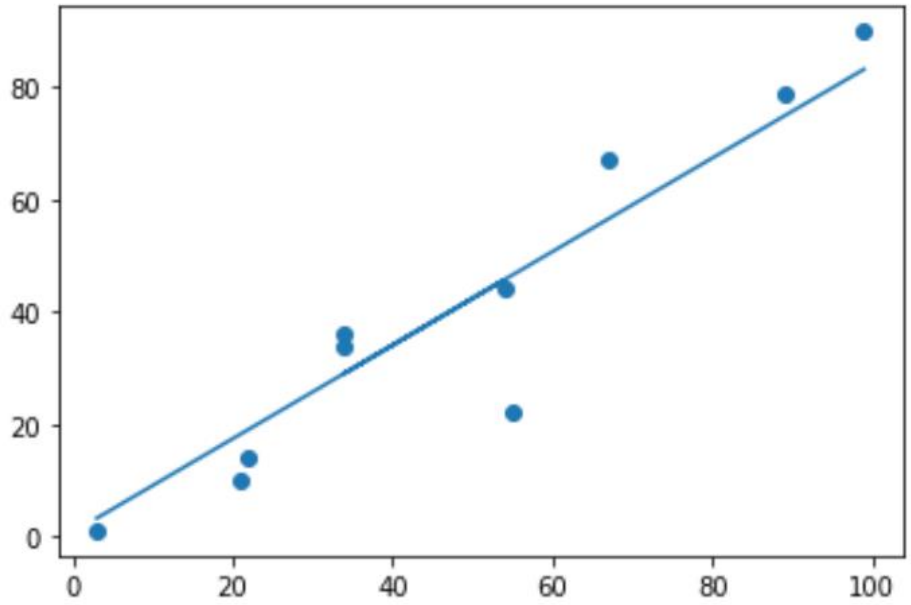

Figure 2-5 Iterate 4 times

Step 5 The fifth iteration shows the parameter value, loss value and visualization effect after iteration

Code:

```
a,b = iterate(a,b,x,y,5)
prediction=model(a,b,x)
loss = loss_function(a, b, x, y)
print(a,b,loss)
plt.scatter(x,y)
plt.plot(x,prediction)
```

Output:

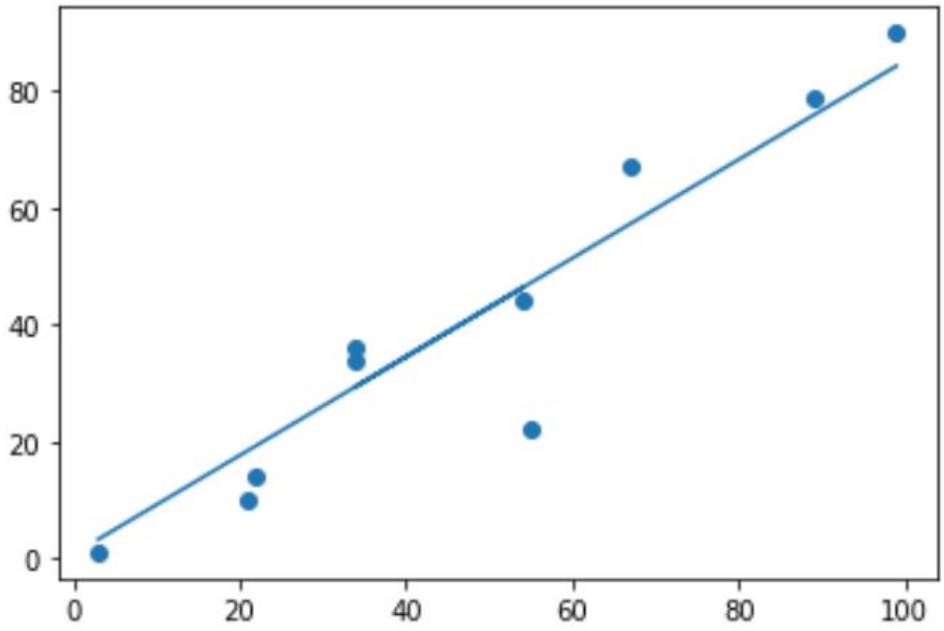

Figure 2-6 Iterate 5 times

Step 6 The 10000th iteration, showing the parameter values, losses and visualization after iteration

Code:

```
a,b = iterate(a,b,x,y,10000)
prediction=model(a,b,x)
loss = loss_function(a, b, x, y)
print(a,b,loss)
plt.scatter(x,y)
plt.plot(x,prediction)
```

Output:

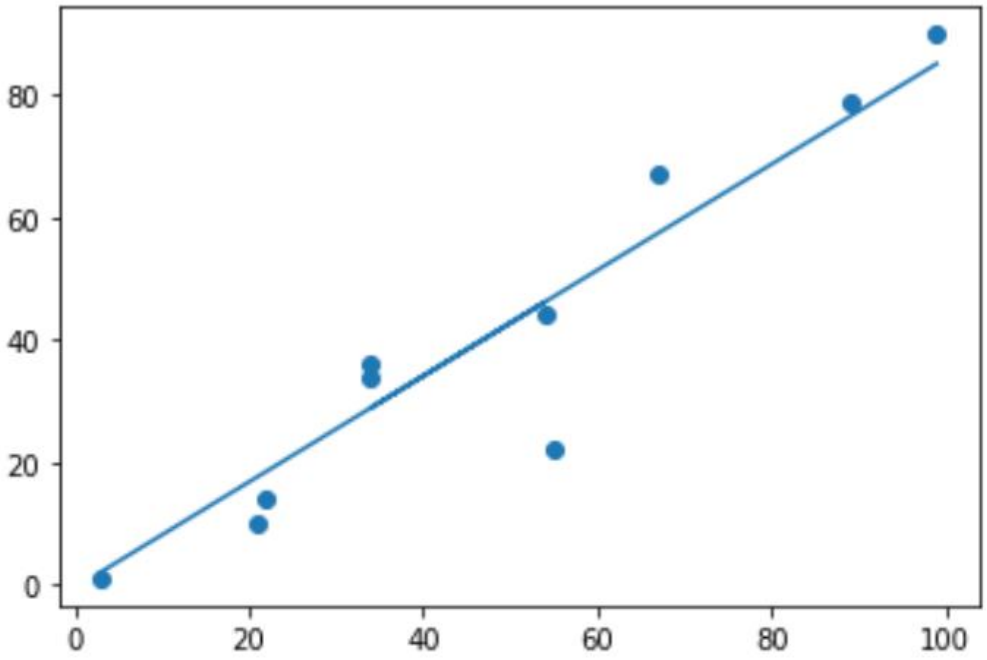

Figure 2-7 Iterate 10000 times

2.3 Thinking and practice

2.3.1 Question 1

Try to modify the original data yourself, Think about it: Does the loss value have to go to zero?

2.3.2 Question 2

Modify the values of Lr, Think: What is the role of the Lr parameter?