**智能信息系统综合实践**

**实验报告**

|  |  |
| --- | --- |
| **题 目：** | 回归分析 |
| **年 级：** | **2021** |
| **专 业：** | **软件工程** |
| **学 号：** | **2021117405** |
| **姓 名：** | **孙潇桐** |

**目录**

[一、 题目 3](#_Toc160607116)

[1. 题目一 3](#_Toc160607117)

[2. 题目二&三 3](#_Toc160607118)

[二、 解题步骤 3](#_Toc160607119)

[1. 题目一 3](#_Toc160607120)

[1) 第一问 3](#_Toc160607121)

[2) 第二问 12](#_Toc160607122)

[2. 题目二 12](#_Toc160607123)

[3. 题目三 14](#_Toc160607124)

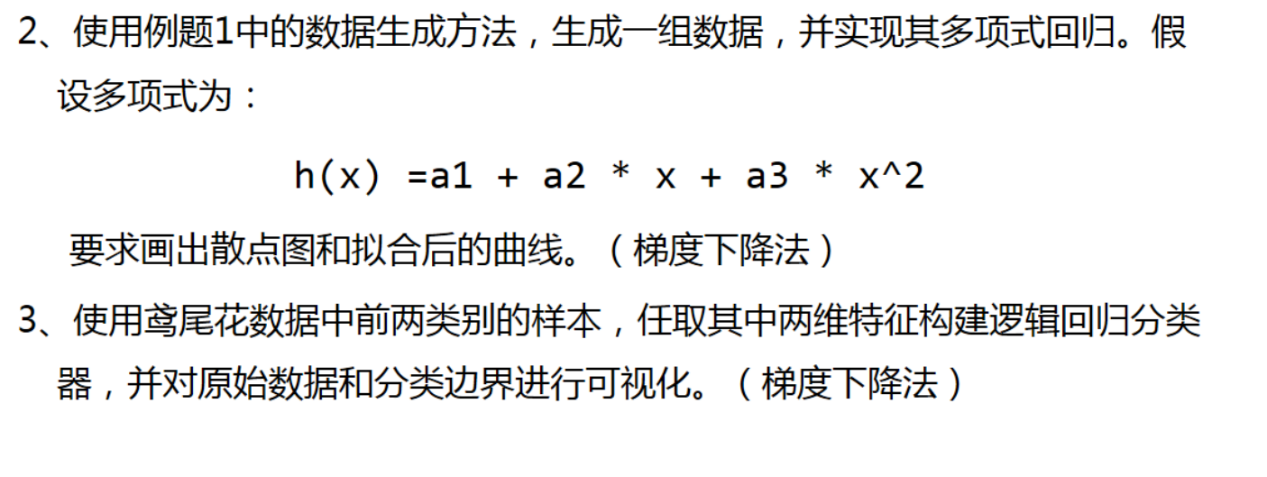
[三、 总结 18](#_Toc160607125)

# 题目

## 题目一



## 题目二&三



# 解题步骤

## 题目一

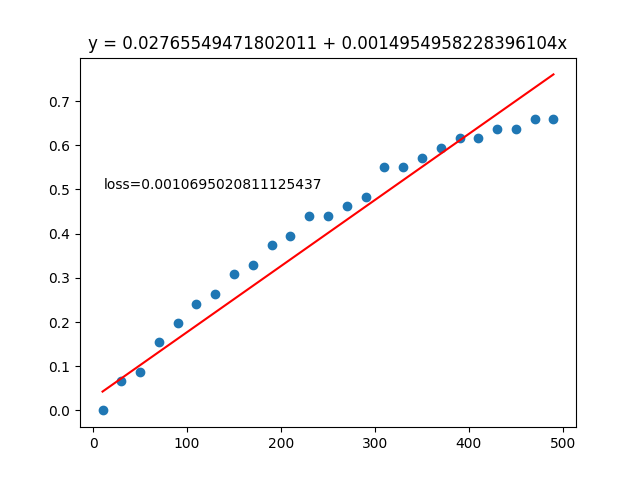
### 第一问

**下面的解题均使用损失函数：**

* + - 1. **线性拟合**

思路：将数据拟合成 ，根据损失函数J得到迭代公式：

结果：



代码：

def linear\_fitting():

    a0 = random.random()

    a1 = random.random()

    alpha = 0.000001

    for \_ in range(30000):

        sum\_a0, sum\_a1 = 0, 0

        for x, y in zip(iter(list\_x), iter(list\_y)):

            cur\_y = a0 + a1 \* x

            sum\_a0 += cur\_y - y

            sum\_a1 += (cur\_y - y) \* x

        a0 -= alpha \* (sum\_a0 / num)

        a1 -= alpha \* (sum\_a1 / num)

        # print(a0, a1, sum\_a0, sum\_a1)

    print(a0, a1)

    res\_y = []

    for x in list\_x:

        res\_y.append(a0 + a1 \* x)

    diff = calc\_diff(res\_y)

    plt.text(11, 0.5, "loss=" + str(diff))

    plt.title(f"y = {a0} + {a1}x")

    plt.plot(list\_x, res\_y, "r")

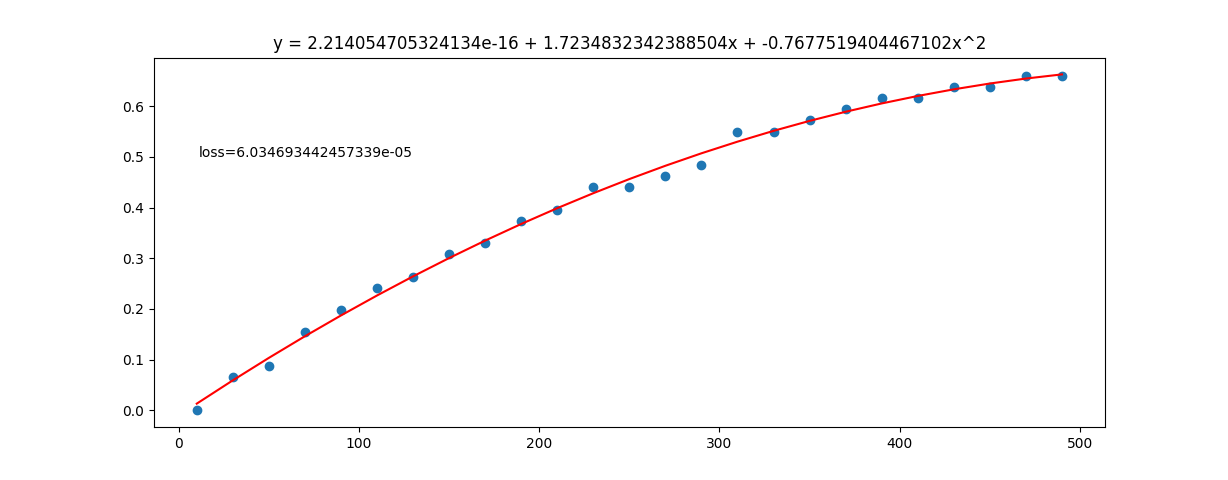
    plt.scatter(list\_x, list\_y)

    plt.show()

* + - 1. **多项式二次拟合**

思路：将数据拟合成 ，使用多项式拟合的原理，根据损失函数J得到迭代公式：

结果：



代码：

def quadratic\_polynomial\_fitting():

    # 转换为多元线性回归 y = a0 + a1 \* x1 + a2 \* x2

    std\_x1, mean\_x1, list\_x1\_std = standardz(np.array(list\_x))

    std\_x2, mean\_x2, list\_x2\_std = standardz(np.array([x \* x for x in list\_x]))

    std\_y, mean\_y, list\_y\_std = standardz(np.array(list\_y))

    a = [random.random(), random.random(), random.random()]

    alpha = 0.03

    for \_ in range(10000):

        sum = [0, 0, 0]

        for x1, x2, y in zip(list\_x1\_std, list\_x2\_std, list\_y\_std):

            cur\_y = a[0] + a[1] \* x1 + a[2] \* x2

            sum[0] += cur\_y - y

            sum[1] += (cur\_y - y) \* x1

            sum[2] += (cur\_y - y) \* x2

        for i, cur\_sum in enumerate(iter(sum)):

            a[i] -= alpha \* (cur\_sum / num)

    print(a)

    res\_y = []

    for x in list\_x:

        y\_std = a[0] + a[1] \* (x - mean\_x1) / std\_x1 + a[2] \* (x \* x - mean\_x2) / std\_x2

        res\_y.append(y\_std \* std\_y + mean\_y)

    plt.title(f"y = {a[0]} + {a[1]}x + {a[2]}x^2")

    diff = calc\_diff(res\_y)

    plt.text(11, 0.5, "loss=" + str(diff))

    plt.plot(list\_x, res\_y, "r")

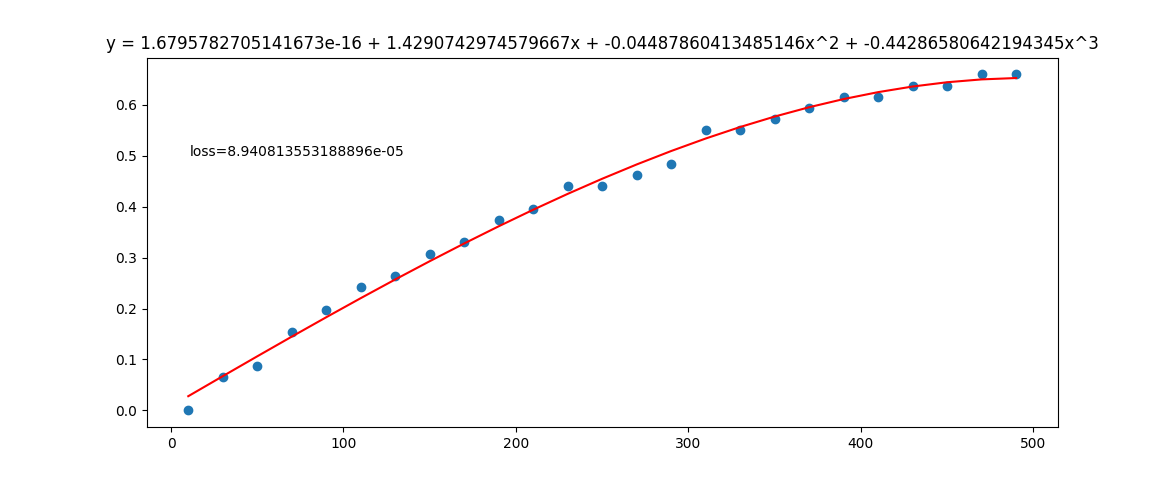
    plt.scatter(list\_x, list\_y)

    plt.show()

* + - 1. **多项式三次拟合**

思路：将数据拟合成 ，根据损失函数J得到迭代公式：

结果：



代码：

def cubic\_polynomial\_fitting():

    # 转换为多元线性回归 y = a0 + a1 \* x1 + a2 \* x2 + a3 \* x3

    std\_x1, mean\_x1, list\_x1\_std = standardz(np.array(list\_x))

    std\_x2, mean\_x2, list\_x2\_std = standardz(np.array([x \* x for x in list\_x]))

    std\_x3, mean\_x3, list\_x3\_std = standardz(np.array([x \* x \* x for x in list\_x]))

    std\_y, mean\_y, list\_y\_std = standardz(np.array(list\_y))

    a = [random.random(), random.random(), random.random(), random.random()]

    alpha = 0.03

    for \_ in range(10000):

        sum = [0, 0, 0, 0]

        for x1, x2, x3, y in zip(list\_x1\_std, list\_x2\_std, list\_x3\_std, list\_y\_std):

            cur\_y = a[0] + a[1] \* x1 + a[2] \* x2 + a[3] \* x3

            sum[0] += cur\_y - y

            sum[1] += (cur\_y - y) \* x1

            sum[2] += (cur\_y - y) \* x2

            sum[3] += (cur\_y - y) \* x3

        for i, cur\_sum in enumerate(iter(sum)):

            a[i] -= alpha \* (cur\_sum / num)

    print(a)

    res\_y = []

    for x in list\_x:

        y\_std = (

            a[0]

            + a[1] \* (x - mean\_x1) / std\_x1

            + a[2] \* (x \* x - mean\_x2) / std\_x2

            + a[3] \* (x \* x \* x - mean\_x3) / std\_x3

        )

        res\_y.append(y\_std \* std\_y + mean\_y)

    plt.title(f"y = {a[0]} + {a[1]}x + {a[2]}x^2 + {a[3]}x^3")

    diff = calc\_diff(res\_y)

    plt.text(11, 0.5, "loss=" + str(diff))

    plt.plot(list\_x, res\_y, "r")

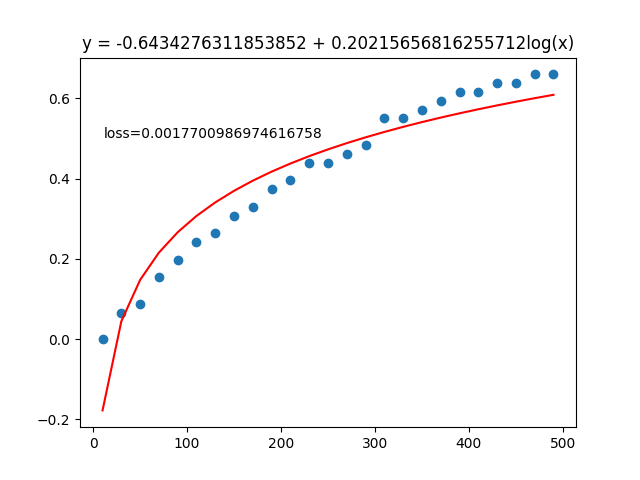
    plt.scatter(list\_x, list\_y)

    plt.show()

* + - 1. **对数函数拟合**

思路：将数据拟合成 ，根据损失函数J得到迭代公式：

结果：



代码：

def logarithmic\_function\_fitting():

    def fun(a0, a1, x):

        return a0 + a1 \* np.log(x)

    a = [random.random(), random.random()]

    alpha = 0.005

    for \_ in range(40000):

        sum = [0] \* 2

        for x, y in zip(list\_x, list\_y):

            cur\_y = fun(a[0], a[1], x)

            sum[0] += cur\_y - y

            sum[1] += (cur\_y - y) \* np.log(x)

        for i, cur\_sum in enumerate(sum):

            a[i] -= alpha \* cur\_sum / num

    print(a)

    res\_y = []

    for x in list\_x:

        res\_y.append(a[0] + a[1] \* np.log(x))

    plt.title(f"y = {a[0]} + {a[1]}log(x)")

    diff = calc\_diff(res\_y)

    plt.text(11, 0.5, "loss=" + str(diff))

    plt.plot(list\_x, res\_y, "r")

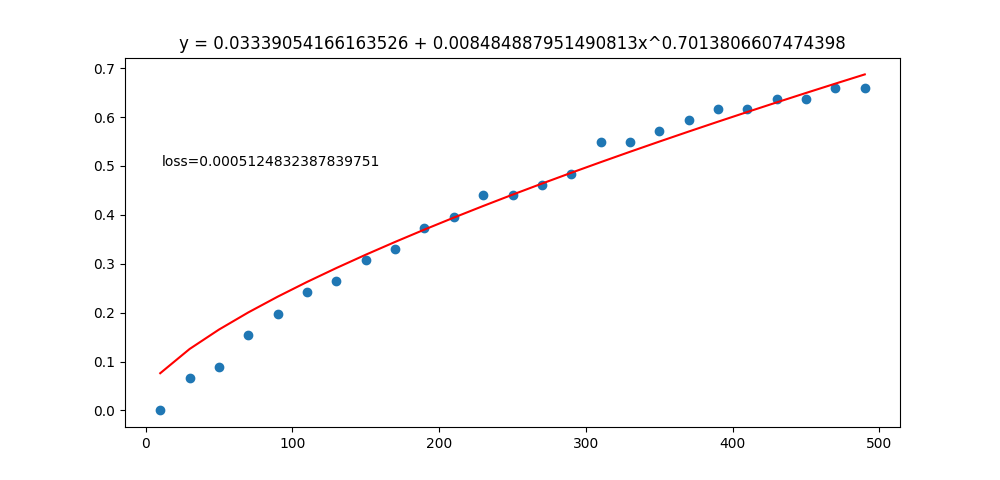
    plt.scatter(list\_x, list\_y)

    plt.show()

* + - 1. **幂函数拟合**

思路：将数据拟合成 ，根据损失函数J得到迭代公式：

结果：



代码：

def power\_function\_fitting():

    def fun(a, x):

        return a[0] + a[1] \* pow(x, a[2])

    a = [random.random(), random.random(), random.random()]

    alpha = 0.00078

    for \_ in range(13000):

        sum = [0] \* 3

        for x, y in zip(list\_x, list\_y):

            cur\_y = fun(a, x)

            sum[0] += cur\_y - y

            sum[1] += (cur\_y - y) \* pow(x, a[2])

            sum[2] += (cur\_y - y) \* a[1] \* pow(x, a[2]) \* log(x)

        for i, cur\_sum in enumerate(iter(sum)):

            a[i] -= alpha \* (cur\_sum / num)

    print(a)

    res\_y = []

    for x in list\_x:

        # y\_std = fun(a, (x - mean\_x) / std\_x)

        # res\_y.append(y\_std \* std\_y + mean\_y)

        res\_y.append(fun(a, x))

    diff = calc\_diff(res\_y)

    plt.text(11, 0.5, "loss=" + str(diff))

    plt.title(f"y = {a[0]} + {a[1]}x^{a[2]}")

    plt.plot(list\_x, res\_y, "r")

    plt.scatter(list\_x, list\_y)

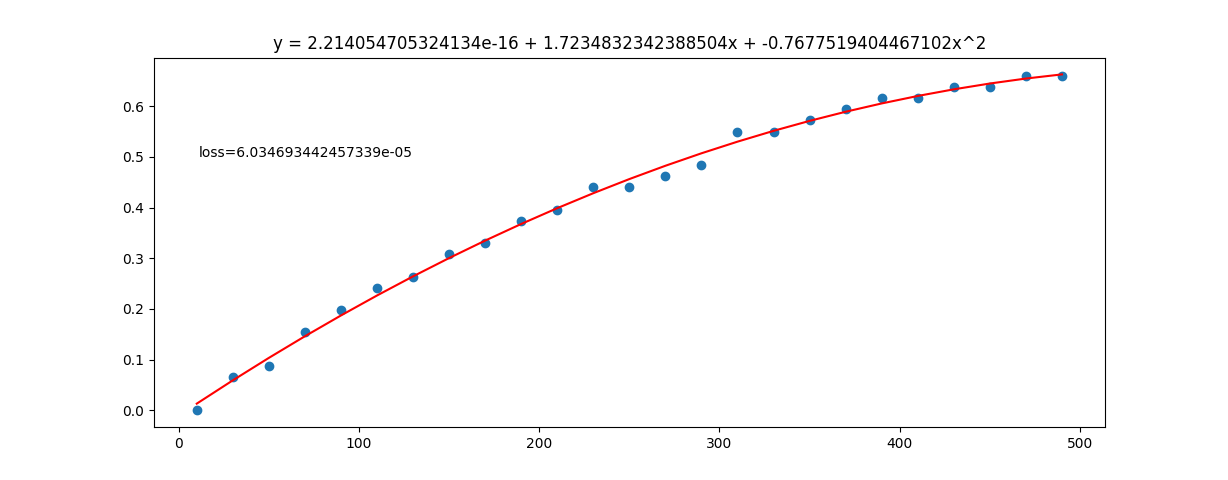
    plt.show()

* + - 1. **第一问总结**

我把误差平方和计算的结果都输出在图上了，根据我的比较，拟合情况最好的是多项式二次拟合

### 第二问

根据第一问的结果，拟合结果最好的是多项式二次拟合 。拟合曲线如下：

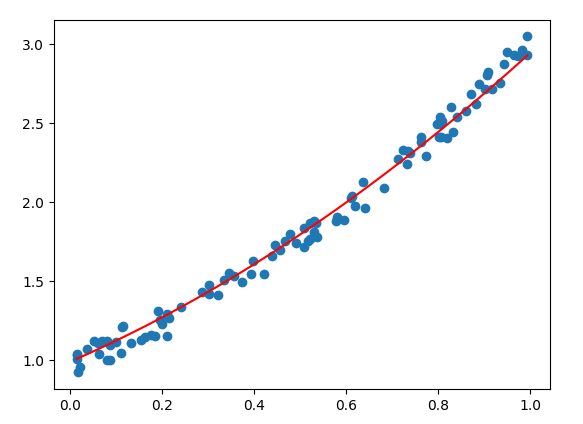


求加速度可以对计算出的函数求导，然后带入200就可以得到结果：我的结果是：0.862412118791859

## 题目二

思路：将数据拟合成 ，使用多项式拟合的原理，根据损失函数J得到迭代公式：

结果：



代码：

import random

import numpy as np

import matplotlib.pyplot as plt

num = 100

a = [random.random(), random.random(), random.random()]

alpha = 0.0015

list\_x = np.zeros(num)

list\_y = np.zeros(num)

for i in range(num):

    list\_x[i] = random.random()

    list\_y[i] = random.random() \* 0.2 - 0.1 + list\_x[i] \* list\_x[i] + list\_x[i] + 1

# 排序

sorted\_indices = sorted(range(len(list\_x)), key=lambda k: list\_x[k])

list\_x = [list\_x[i] for i in sorted\_indices]

list\_y = [list\_y[i] for i in sorted\_indices]

# y = a0 + a1 \* x + a\_2 \* x \* x

for \_ in range(10000):

    sum = [0.0] \* len(a)

    for x, y in zip(list\_x, list\_y):

        cur\_y = a[0] + a[1] \* x + a[2] \* x \* x

        sum[0] += cur\_y - y

        sum[1] += (cur\_y - y) \* x

        sum[2] += (cur\_y - y) \* x \* x

    for i, cur\_sum in enumerate(sum):

        a[i] -= alpha \* (cur\_sum / num)

res\_y = np.zeros(num)

for i, x in enumerate(list\_x):

    res\_y[i] = a[0] + a[1] \* x + a[2] \* x \* x

plt.plot(list\_x, res\_y, "r")

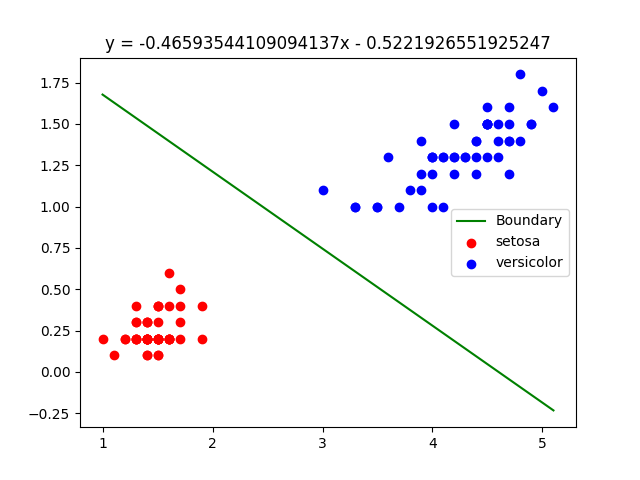
plt.scatter(list\_x, list\_y)

plt.show()

## 题目三

思路：本体是一个二分类问题，使用激活函数sigmoid 来对数据进行分类。我在本题中使用了四个属性的在引入了激活函数之后通过 就能得到一个样本的类别（将类别分为0和1）。对于这个函数使用多项式拟合的原理，根据损失函数J得到迭代公式：

结果：

  
代码：

import csv

import random

import numpy as np

import matplotlib.pyplot as plt

from tqdm import tqdm

mp = dict()

mp["setosa"] = []

mp["versicolor"] = []

def standardize(list):

    mean = np.mean(list)

    std = np.std(list)

    return std, mean, (list - mean) / std

# 采用前两个参数

with open("data/iris.csv", "r", newline="") as file:

    csvreader = csv.reader(file)

    next(csvreader)

    for row in csvreader:

        mp[row[5]].append(np.array(row[3:5], dtype=float))

list\_s = np.array(mp["setosa"])

list\_v = np.array(mp["versicolor"])

list\_s\_with\_tag = np.concatenate((list\_s, np.ones((list\_s.shape[0], 1))), axis=1)

list\_v\_with\_tag = np.concatenate((list\_v, np.zeros((list\_v.shape[0], 1))), axis=1)

list\_with\_tag = np.concatenate((list\_s\_with\_tag, list\_v\_with\_tag), axis=0)

s\_len\_list = list\_s[:, 0]

s\_width\_list = list\_s[:, 1]

v\_len\_list = list\_v[:, 0]

v\_width\_list = list\_v[:, 1]

std\_x1, mean\_x1, list\_with\_tag[:, 0] = standardize(list\_with\_tag[:, 0])

std\_x2, mean\_x2, list\_with\_tag[:, 1] = standardize(list\_with\_tag[:, 1])

def sigmoid(z):

    return 1 / (1 + np.exp(-z))

def dsigmoid(z, x):

    return x \* sigmoid(z) \* (1 - x \* sigmoid(z))

# y = sigmoid(a0 + a1 \* x1 + a2 \* x2)

a = [random.random(), -random.random(), -random.random()]

num = list\_with\_tag.shape[0]

alpha = 0.01

for \_ in tqdm(range(5000)):

    sum = [0.0] \* len(a)

    for x1, x2, tag in list\_with\_tag:

        inner = a[0] + a[1] \* x1 + a[2] \* x2

        cur\_tag = sigmoid(inner)

        diff = cur\_tag - tag

        sum[0] += diff \* dsigmoid(inner, 1)

        sum[1] += diff \* dsigmoid(inner, x1)

        sum[2] += diff \* dsigmoid(inner, x2)

    for i, cur\_sum in enumerate(sum):

        a[i] -= alpha \* (cur\_sum / num)

print(a)

minn = np.min(list\_with\_tag[:, 0])

maxn = np.max(list\_with\_tag[:, 0])

list\_x = np.linspace(minn, maxn, num)

list\_y = (-(a[0] + a[1] \* list\_x) / a[2]) \* std\_x2 + mean\_x2

list\_x = list\_x \* std\_x1 + mean\_x1

plt.plot(list\_x, list\_y, color="g", label="Boundary")

plt.scatter(s\_len\_list, s\_width\_list, color="r", label="setosa")

plt.scatter(v\_len\_list, v\_width\_list, color="b", label="versicolor")

plt.legend()

k = -(a[1] \* std\_x2) / (a[2] \* std\_x1)

b = -a[0] \* std\_x2 / a[2] + mean\_x2 - mean\_x1 \* a[1] \* std\_x2 / (a[2] \* std\_x1)

plt.title(f"y = {k}x " + ("+" if b > 0 else "-") + f" {abs(b)}")

plt.show()

# 总结

在每个拟合方法的实验中，我都在上面详细介绍了思路和算法，并提供了相应的代码实现。通过调整参数和迭代次数，得到了最终的拟合结果，并通过损失函数评估了拟合效果。在题目一中，我们特别强调了多项式二次拟合在拟合效果上的优越性，并据此得出了结论。在题目二和题目三中，我们延伸了回归分析的应用领域，分别探讨了对复杂数据集的拟合和二分类问题的解决方案。

在这次的实验中，我遇到最多的问题就是梯度下降时不收敛或者收敛比较慢的问题，我通过使用标准化解决了这些问题。同时我也学会了在原始空间和标准化空间之间映射，这样就能得到在原始数据空间中的函数图像。

综合而言，通过本次实验，我们不仅掌握了回归分析的基本理论和方法，还通过实践加深了对相关概念的理解。同时，也学会了如何利用Python编程语言实现算法并进行数据分析。