9181040G0818 黄海浪 机器学习第四次作业

1. 使用梯度下降法求解（加上正则，解决过拟合问题）

归一化（快速收敛（保证能够收敛））：

$$x = \frac{x - min(x)}{max(x) - min(x)}$$

Cost 函数（方差+正则；不对 theta0 进行惩罚）：

$$cost\ = J = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}{\theta_j}^2$$

梯度推导出有（取$\lambda = 1$，λ值不同效果不一样，越大越平，不能<0；$x_0^{(i)} = 1$）：

$$\frac{\partial J}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}(\theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} - y^{(i)})\, x_0^{(i)}$$

$$\frac{\partial J}{\partial \theta_1} = \frac{1}{m}\left[\sum_{i=1}^{m}[(\theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} - y^{(i)}) x_1^{(i)}]\right] + \frac{\lambda}{m}\theta_1$$

迭代步骤（$\alpha = 0.01$）：

$$\theta_0 = \theta_0 - \alpha\frac{\partial J}{\partial \theta_0}$$

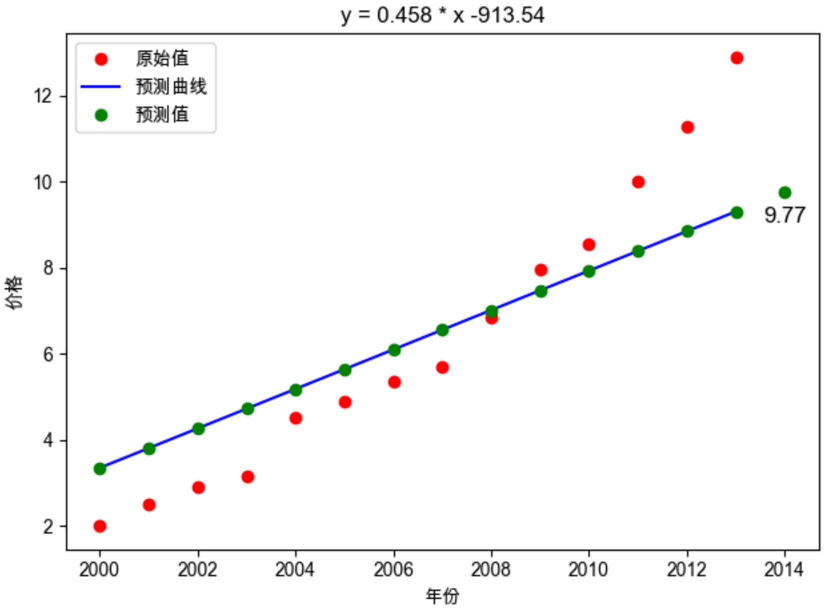
$$\theta_1 = \theta_1 - \alpha\frac{\partial J}{\partial \theta_1}$$

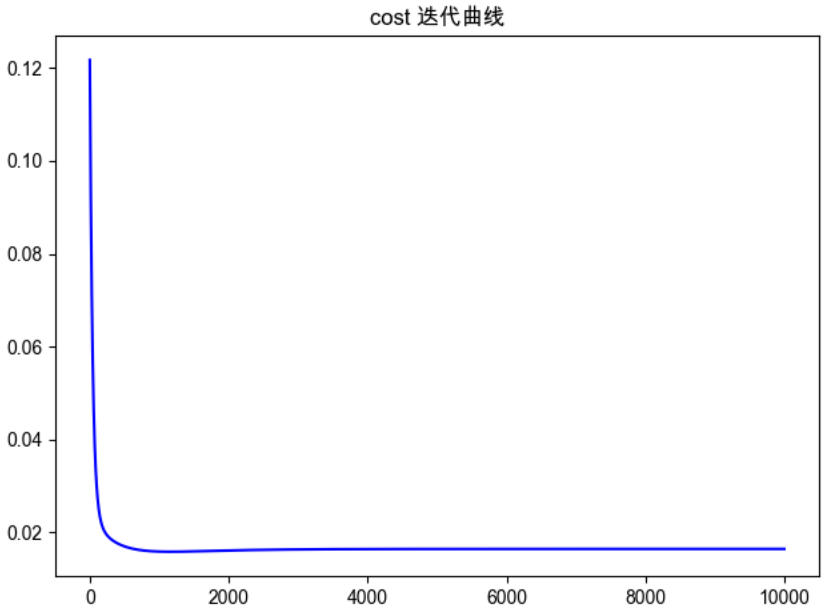
计算得到归一化处理后的

$\theta_{01}\ =\ 0.12372582507195107$ $\theta_{11}\ =\ 0.5467686512334945$

反归一化后得到$\theta_0\ =\ -913.540357498268$ $\theta_1\ =\ 0.45844448449577613$

$$Y =\ \theta_1 X\ +\ \theta_0$$





预测 2014 年房价为 9.766834276225154

2. logistic 回归

2.1 梯度下降实现

由问题可以得到下面公式（其中随机梯度为随机选择部分样品进行更新，公式类似，不过不是 1～m，而是 1～m 的子集）

① 归一化（快速收敛（保证能够收敛））:

$$x = \frac{x - min(x)}{max(x) - min(x)}$$

② 假设：

$$h_\theta(x^{(i)}) = g(\theta^T x^{(i)})$$

其中$g(z) = \frac{1}{1+e^{-z}}$ 、$\theta = [\theta_0; \theta_1; \theta_2]$ 、 $x = [x_0; x_1; x_2]$ “;” 表示列向量

③Cost 函数（类似于概率；不对 theta0 进行惩罚）：

$$cost = J = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} log h_\theta(x^{(i)}) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} {\theta_j}^2$$

④梯度推导出有（取$\lambda = 0$，λ值不同效果不一样，越大越平，不能<0；$x_0^{(i)} = 1$）：

$$\frac{\partial J}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\, x_0^{(i)}$$

$$\frac{\partial J}{\partial \theta_1} = \frac{1}{m}\left[\sum_{i=1}^{m}[(h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}]\right] + \frac{\lambda}{m}\theta_1$$

$$\frac{\partial J}{\partial \theta_2} = \frac{1}{m}\left[\sum_{i=1}^{m}[(h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}]\right] + \frac{\lambda}{m}\theta_2$$

⑤迭代步骤（$\alpha = 0.01$）：

$$\theta_0 = \theta_0 - \alpha\frac{\partial J}{\partial \theta_0}$$

$$\theta_1 = \theta_1 - \alpha\frac{\partial J}{\partial \theta_1}$$

$$\theta_2 = \theta_2 - \alpha\frac{\partial J}{\partial \theta_2}$$

⑥由于归一化后最后求得的θ_i为θ_i'，易得θ_i'并非原始公式的θ_i，需要做变换后才能得到最终求的最后的θ_i，显然此处 y 并不做归一化处理，有：

$$\theta^T x^{(i)} = \theta_0' + \theta_1' x_1'^{(i)} + \theta_2' x_2'^{(i)}$$

其中 $x_1'^{(i)} = \frac{x_1^{(i)} - min(x_1)}{max(x_1) - min(x_1)}$ ，$x_2'^{(i)} = \frac{x_2^{(i)} - min(x_2)}{max(x_2) - min(x_2)}$

继而：

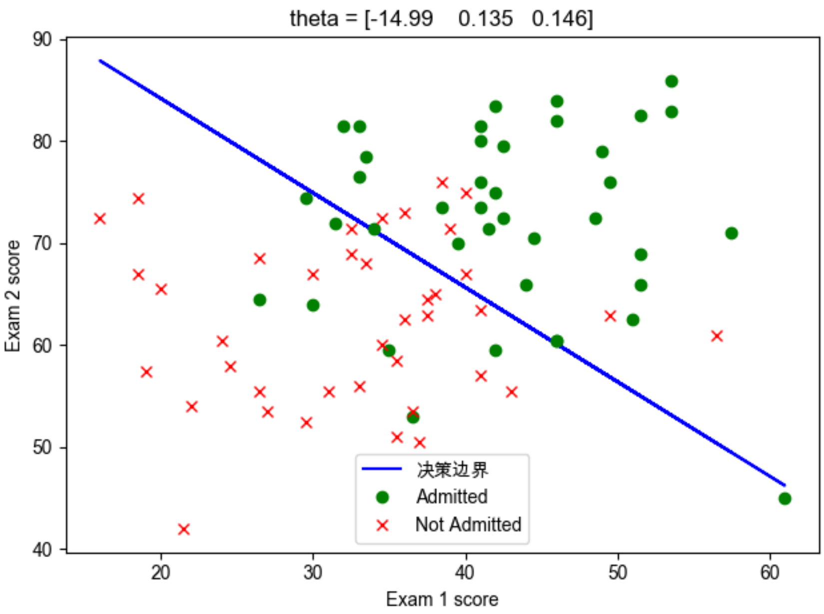
$$\theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} = \theta_0' + \theta_1' x_1'^{(i)} + \theta_2' x_2'^{(i)}$$

$$\theta_0 = \theta_0' + \frac{\theta_1' min(x_1)}{max(x_1) - min(x_1)} + \frac{\theta_2' min(x_2)}{max(x_2) - min(x_2)}$$

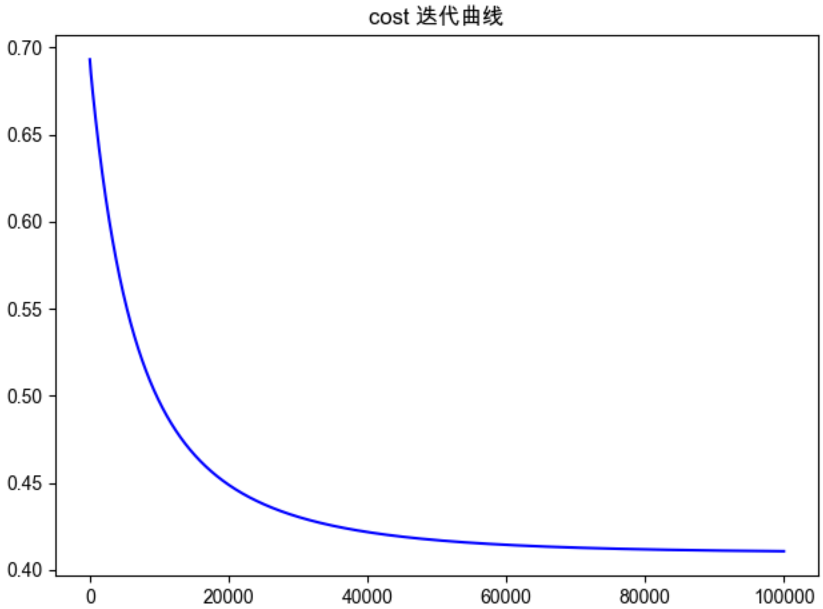
$$\theta_1 = \frac{\theta_1'}{max(x_1) - min(x_1)}$$

$$\theta_2 = \frac{\theta_2'}{max(x_2) - min(x_2)}$$

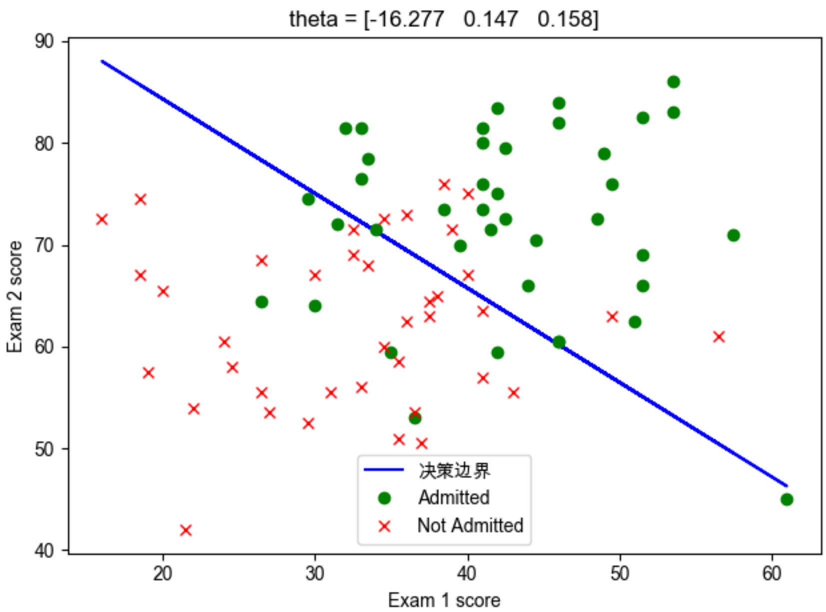
结果：



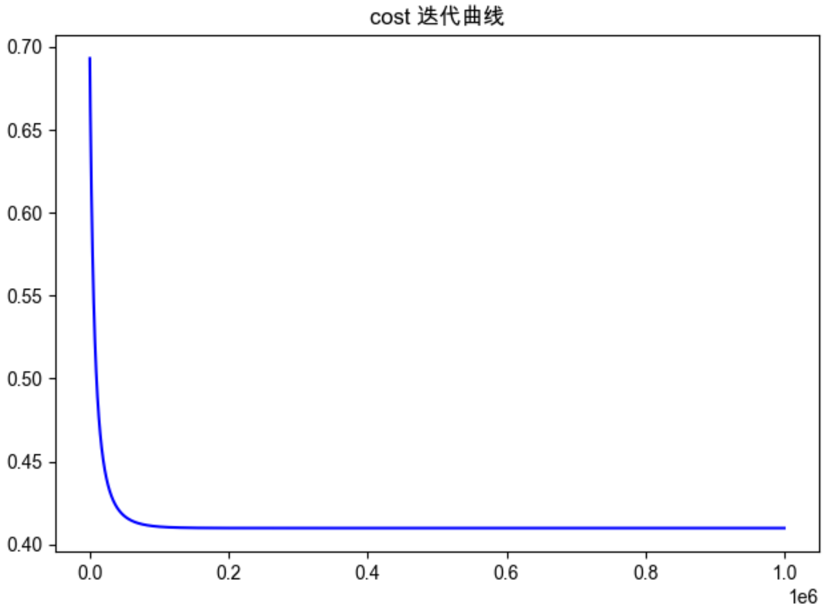
图表 1 迭代 1e5 次结果



图表 2 迭代 1e5 次结果

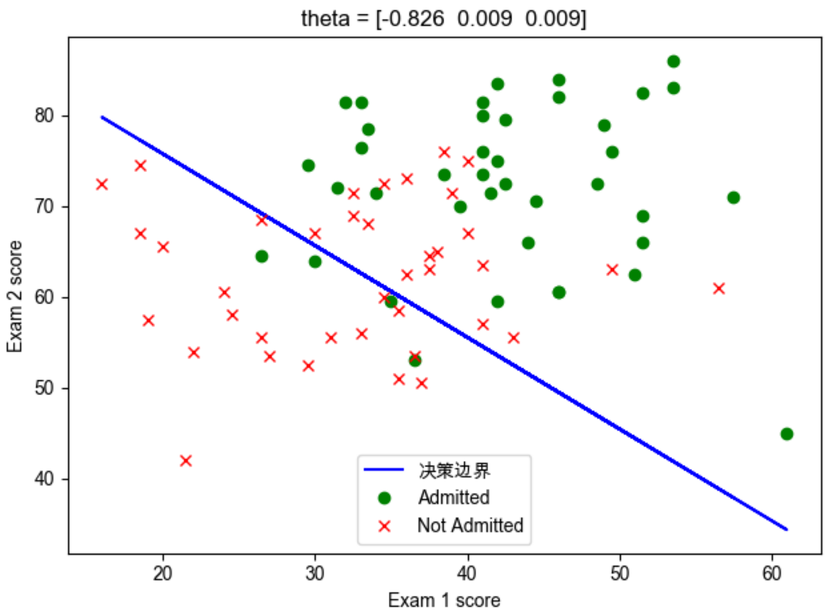


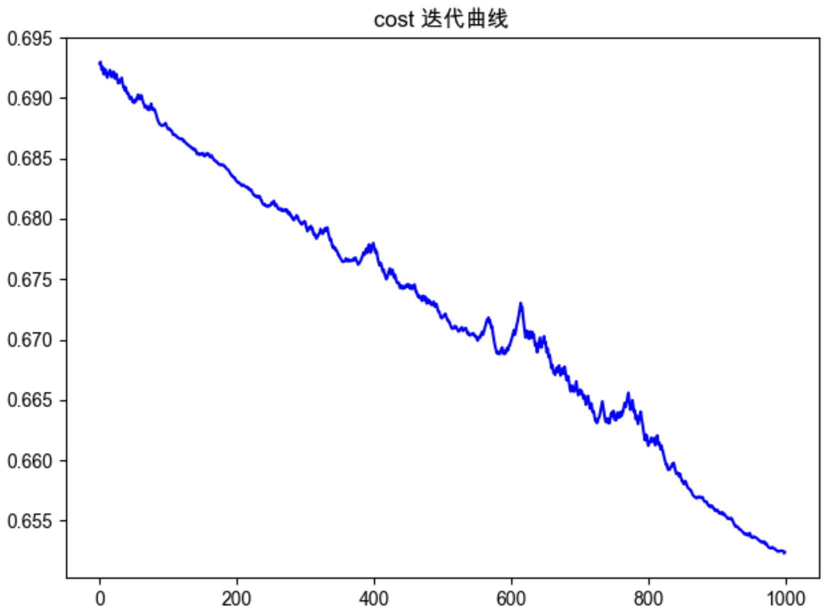
图表 3 迭代 1e6 次结果



图表 4 迭代 1e6 次结果

2.1 随机梯度下降（每次使用 1 个变量更新，共 1000 次）




cost 迭代曲线
0.695
0.690
0.685
0.680
0.675
0.670
0.665
0.660
0.655
0
200
400
600
800
1000

2.2 牛顿法实现

除了迭代步骤不同外，其他与梯度下降法一致，这里不使用正则。即上述 cost 函数和梯度去掉后面+的那一部分。

迭代步骤：

$$\theta_0 = \theta_0 - [\mathrm{Hf}(\theta_0)]^{-1}\frac{\partial J}{\partial \theta_0}$$

$$\theta_1 = \theta_1 - [\mathrm{Hf}(\theta_1)]^{-1}\frac{\partial J}{\partial \theta_1}$$

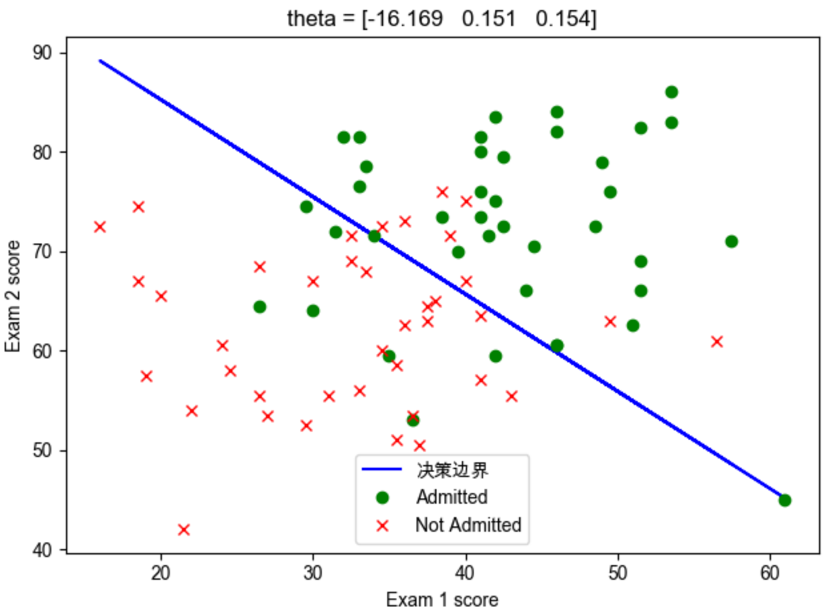
$$\theta_2 = \theta_2 - [\mathrm{Hf}(\theta_2)]^{-1}\frac{\partial J}{\partial \theta_2}$$

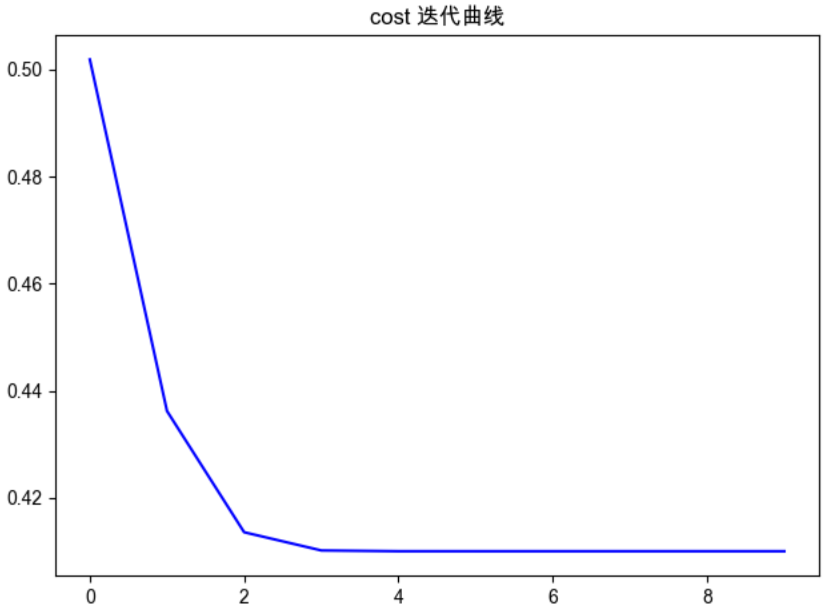
其中

$$\mathrm{H} = \frac{1}{m}\left[\sum_{i=1}^{m}[(h_\theta(x^{(i)}))(1-h_\theta(x^{(i)}))x^{(i)}{x^{(i)}}^{\mathrm{T}}]\right]$$

其中$x^{(i)}$为列向量(3*1)，$x^{(i)}{x^{(i)}}^{\mathrm{T}}$得到 3*3 向量，再点乘$(h_\theta(x^{(i)}))(1-h_\theta(x^{(i)}))$

得到结果如下：




cost 迭代曲线
0.50
0.48
0.46
0.44
0.42
0
2
4
6
8

附录：

1．房价预测梯度下降正则化代码（其中数据 csv 为 x 一行，y 一行）：

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
# @time    : 2020/10/24 12:18
# @author  : lerogo
# @fileName: main.py

import csv

import numpy as np
import matplotlib.pyplot as plt

# 解决 plt 画图中文字体字体
plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']  # macos
# plt.rcParams['font.sans-serif'] = ['KaiTi'] # windows
plt.rcParams['axes.unicode_minus'] = False  # 解决保存图像是负号'-'显示为方块的问
题


# 从 csv 读取数据
def readData():
    csv_reader = csv.reader(open("../data/data.csv"))
    rows = []
    for row in csv_reader:
        rows.append(row)
    x = rows[0]
    y = rows[1]
    for i in range(0, len(x)):
        x[i] = int(str(x[i]).strip())
    for i in range(0, len(y)):
        y[i] = float(str(y[i]).strip())
    return np.array(x), np.array(y)


# 归一化
def normalized(para):
    para = (para - para.min()) / (para.max() - para.min())
    return para


# 归一化 特殊
def normalized_special(para, special):
    special = (special - para.min()) / (para.max() - para.min())
    return special
```

```python
# 反归一化
def reverse_normalized(para, special):
    special = special * (para.max() - para.min()) + para.min()
    return special


# sita[0] sita[1] 代价函数
def get_cost(x, y, sita, numta):
    cost = ((sita[0] + sita[1] * x - y) ** 2).sum()
    cost += numta * ((np.array(sita) ** 2).sum() - sita[0])  # 加上正则
    return cost / 2 / len(x)


# x,y 步长 sita[0]sita[1] 接受的 cost 最小多少  最多迭代次数
def get_gradient(x, y, alpha, sita, accept_cost, max_times, numta):
    m = len(x)  # 多少个量
    dev = [0, 0]  # 梯度
    times = 0  # 迭代次数
    cost = get_cost(x, y, sita, numta)  # 计算第一次 cost
    cost_list = []  # 储存迭代出的 cost
    # 开始迭代
    while cost > accept_cost and times < max_times:
        dev[0] = ((sita[0] + sita[1] * x - y).sum()) / m  # 梯度 sita0
        dev[1] = (((sita[0] + sita[1] * x - y) * x).sum()) / m  # 梯度 sita1
        # 重新计算 sita
        sita[0] -= alpha * dev[0]  # theta[0]不变
        sita[1] = sita[1] * (1 - alpha * numta / m) - alpha * dev[1]  #
theta[1](1-alpha*numta/m)
        cost = get_cost(x, y, sita, numta)  # 重新计算 cost
        cost_list.append(cost)  # 加入 cost_list 方便画图
        times += 1
    return sita, cost_list


if __name__ == '__main__':
    data_x, data_y = readData()
    x = normalized(data_x)
    y = normalized(data_y)
    sita, cost_list = get_gradient(
        x=x,
        y=y,
        alpha=0.01,
        sita=[0, 0],
```

```
        accept_cost=1e-5,
        max_times=1e4,
        numta=1
    )
    print(sita)

    # 计算预测值 老算法
    # predict_y = sita[0] + sita[1] * x
    # predict_y = reverse_normalized(data_y, predict_y)
    #  计算 反归一化后的 sita
    sita2 = [0, 0]
    sita2[1] = sita[1] / (data_x.max() - data_x.min()) * (data_y.max() -
data_y.min())
    sita2[0] = (sita[0] - data_x.min() * sita[1] / (data_x.max() -
data_x.min())) * (
            data_y.max() - data_y.min()) + data_y.min()
    print(sita2)

    predict_2014 = sita2[0] + sita2[1] * 2014
    print(predict_2014)
    predict_y = sita2[0] + sita2[1] * data_x
    # 作出 cost 的迭代曲线
    plt.title("cost 迭代曲线")
    plt.plot([x for x in range(int(1e4))], cost_list, c='blue', label='cost 迭
代曲线')
    plt.show()
    # 作出原始值
    plt.title("y = " + str(round(sita2[1], 3)) + " * x " +
str(round(sita2[0], 2)))
    plt.plot(data_x, data_y, 'o', c='red', label='原始值')
    plt.plot(data_x, predict_y, c='blue', label='预测曲线')
    plt.plot(data_x, predict_y, 'o', c='green', label='预测值')
    plt.plot(2014, predict_2014, 'o', c='green')
    plt.text(2014, predict_2014 - 0.8, "%.2f" % predict_2014, ha='center',
va='bottom', fontsize=12)
    plt.plot()
    plt.xlabel('年份')
    plt.ylabel('价格')
    plt.legend()  # 显示图例
    plt.show()
    exit()
```

2. logistic 回归非随机梯度下降

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
# @time    : 2020/10/24 13:21
# @author  : lerogo
# @fileName: main.py

import csv

import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt

# 解决 plt 画图中文字体字体
plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']  # macos
# plt.rcParams['font.sans-serif'] = ['KaiTi'] # windows
plt.rcParams['axes.unicode_minus'] = False  # 解决保存图像是负号'-'显示为方块的问
题


# 从 data 读取数据
def readData():
    data_x = pd.read_table("../data/ex4x.dat", sep=' ').values
    x1 = np.array(data_x[:, 3])
    x2 = np.array(data_x[:, 6])
    data_y = pd.read_table("../data/ex4y.dat", sep=' ').values
    y = np.array(data_y[:, 3])
    return x1, x2, y


# 归一化
def normalized(para):
    para = (para - para.min()) / (para.max() - para.min())
    return para


# 反归一化 算出需要的 theta
def reverse_normalized_sita(x1: np.ndarray, x2: np.ndarray, ori_sita):
    sita = [0, 0, 0]
    sita[0] = ori_sita[0] - (ori_sita[1] * x1.min()) / (
            x1.max() - x1.min()) - (ori_sita[2] * x2.min()) / (x2.max() -
x2.min())
    sita[1] = ori_sita[1] / (x1.max() - x1.min())
    sita[2] = ori_sita[2] / (x2.max() - x2.min())
```

```python
    return sita


# sita[0] sita[1] 代价函数
def get_cost(x1: np.ndarray, x2: np.ndarray, y: np.ndarray, sita, numta):
    m = len(y)
    hx = h_x(x1, x2, sita)
    cost = -1 / m * ((y * np.log(hx) + (1 - y) * np.log(1 - hx)).sum())
    cost += (numta * ((np.array(sita) ** 2).sum() - sita[0])) / 2 / m  # 加上
正则
    return cost


def h_x(x1: np.ndarray, x2: np.ndarray, sita):
    hx = []
    for i in range(len(x1)):
        hx.append(1 / (1 + math.exp(-(sita[0] + sita[1] * x1[i] + sita[2] *
x2[i]))))
    return np.array(hx)


def get_gradient(x1: np.ndarray, x2: np.ndarray, y: np.ndarray, sita,
numta):
    dev = []
    m = len(y)
    hx = h_x(x1, x2, sita)
    dev.append(
        ((1 / m) * (hx - y).sum())
    )
    dev.append(
        ((1 / m) * ((hx - y) * x1).sum()) + numta / m * sita[1]
    )
    dev.append(
        ((1 / m) * ((hx - y) * x2).sum()) + numta / m * sita[2]
    )
    return np.array(dev)


def get_theta(sita, dev, alpha):
    sita = np.array(sita)
    return sita - alpha * dev


# x,y 步长 sita[0]sita[1] 接受的 cost 最小多少  最多迭代次数
def gen_ans(x1, x2, y, alpha, sita, accept_cost, max_times, numta):
    times = 0  # 迭代次数
```

```python
    cost = get_cost(x1, x2, y, sita, numta)  # 计算第一次 cost
    cost_list = []  # 储存迭代出的 cost
    # 开始迭代
    while cost > accept_cost and times < max_times:
        sita = get_theta(
            sita=sita,
            dev=get_gradient(x1, x2, y, sita, numta),
            alpha=alpha
        )
        cost = get_cost(x1, x2, y, sita, numta)  # 重新计算 cost
        cost_list.append(cost)  # 加入 cost_list 方便画图
        times += 1
    return sita, cost_list


if __name__ == '__main__':
    data_x1, data_x2, data_y = readData()
    # print(data_x1, data_x2, data_y)
    x1 = normalized(data_x1)
    x2 = normalized(data_x2)
    max_times = 1e6
    sita, cost_list = gen_ans(
        x1=x1,
        x2=x2,
        y=data_y,
        alpha=0.01,
        sita=[0, 0, 0],
        accept_cost=1e-5,
        max_times=max_times,
        numta=0
    )
    print(sita)
    sita = reverse_normalized_sita(data_x1, data_x2, sita)
    print(sita)
    plot_y = -(sita[0] + sita[1] * data_x1) / sita[2]

    # 作出 cost 的迭代曲线
    plt.title("cost 迭代曲线")
    plt.plot([x for x in range(int(max_times))], cost_list, c='blue',
label='cost 迭代曲线')
    plt.show()
    # 作出原始值
    plt.title("theta = " + str(np.round(sita, 3)))
    plt.plot(data_x1, plot_y, c='blue', label='决策边界')
    plt.plot(data_x1[:40], data_x2[:40], 'o', c='green', label='Admitted')
```

```python
plt.plot(data_x1[40:], data_x2[40:], 'x', c='red', label='Not Admitted')
plt.plot()
plt.xlabel('Exam 1 score')
plt.ylabel('Exam 2 score')
plt.legend()  # 显示图例
plt.show()
exit()
```

3. logistic 回归**随机**梯度下降

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
# @time    : 2020/10/24 14:33
# @author  : lerogo
# @fileName: main.py

import csv
import random

import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt

# 解决 plt 画图中文字体字体
plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']  # macos
# plt.rcParams['font.sans-serif'] = ['KaiTi'] # windows
plt.rcParams['axes.unicode_minus'] = False  # 解决保存图像是负号'-'显示为方块的问
题


# 从 data 读取数据
def readData():
    data_x = pd.read_table("../data/ex4x.dat", sep=' ').values
    x1 = np.array(data_x[:, 3])
    x2 = np.array(data_x[:, 6])
    data_y = pd.read_table("../data/ex4y.dat", sep=' ').values
    y = np.array(data_y[:, 3])
    return x1, x2, y


# 归一化
def normalized(para):
    para = (para - para.min()) / (para.max() - para.min())
    return para


# 反归一化 算出需要的 theta
def reverse_normalized_sita(x1: np.ndarray, x2: np.ndarray, ori_sita):
    sita = [0, 0, 0]
    sita[0] = ori_sita[0] - (ori_sita[1] * x1.min()) / (
            x1.max() - x1.min()) - (ori_sita[2] * x2.min()) / (x2.max() -
x2.min())
    sita[1] = ori_sita[1] / (x1.max() - x1.min())
```

```
    sita[2] = ori_sita[2] / (x2.max() - x2.min())
    return sita


# sita[0] sita[1] 代价函数
def get_cost(x1: np.ndarray, x2: np.ndarray, y: np.ndarray, sita, numta):
    m = len(y)
    hx = h_x(x1, x2, sita)
    cost = -1 / m * ((y * np.log(hx) + (1 - y) * np.log(1 - hx)).sum())
    cost += (numta * ((np.array(sita) ** 2).sum() - sita[0])) / 2 / m  # 加上
正则
    return cost


def h_x(x1: np.ndarray, x2: np.ndarray, sita):
    hx = []
    for i in range(len(x1)):
        hx.append(1 / (1 + math.exp(-(sita[0] + sita[1] * x1[i] + sita[2] *
x2[i]))))
    return np.array(hx)


def get_gradient(x1: np.ndarray, x2: np.ndarray, y: np.ndarray, sita,
numta):
    dev = []
    m = len(y)
    hx = h_x(x1, x2, sita)
    dev.append(
        ((1 / m) * (hx - y).sum())
    )
    dev.append(
        ((1 / m) * ((hx - y) * x1).sum()) + numta / m * sita[1]
    )
    dev.append(
        ((1 / m) * ((hx - y) * x2).sum()) + numta / m * sita[2]
    )
    return np.array(dev)


def get_theta(sita, dev, alpha):
    sita = np.array(sita)
    return sita - alpha * dev


# x,y 步长 sita[0]sita[1] 接受的 cost 最小多少  最多迭代次数
def gen_ans(x1, x2, y, alpha, sita, accept_cost, max_times, numta):
```

```
    times = 0  # 迭代次数
    cost = get_cost(x1, x2, y, sita, numta)  # 计算第一次 cost
    cost_list = []  # 储存迭代出的 cost
    # 开始迭代
    while cost > accept_cost and times < max_times:
        x_index = [random.randint(0, len(x1)-1) for i in range(0, 1)]
        tmpx1 = [x1[i] for i in x_index]
        tmpx2 = [x2[i] for i in x_index]
        tmpy = [y[i] for i in x_index]
        sita = get_theta(
            sita=sita,
            dev=get_gradient(tmpx1, tmpx2, tmpy, sita, numta),
            alpha=alpha
        )
        cost = get_cost(x1, x2, y, sita, numta)  # 重新计算 cost
        cost_list.append(cost)  # 加入 cost_list 方便画图
        times += 1
    return sita, cost_list


if __name__ == '__main__':
    data_x1, data_x2, data_y = readData()
    # print(data_x1, data_x2, data_y)
    x1 = normalized(data_x1)
    x2 = normalized(data_x2)
    max_times = 1e3
    sita, cost_list = gen_ans(
        x1=x1,
        x2=x2,
        y=data_y,
        alpha=0.01,
        sita=[0, 0, 0],
        accept_cost=1e-5,
        max_times=max_times,
        numta=0
    )
    print(sita)
    sita = reverse_normalized_sita(data_x1, data_x2, sita)
    print(sita)
    plot_y = -(sita[0] + sita[1] * data_x1) / sita[2]

    # 作出 cost 的迭代曲线
    plt.title("cost 迭代曲线")
    plt.plot([x for x in range(int(max_times))], cost_list, c='blue',
label='cost 迭代曲线')
```

```python
plt.show()
# 作出原始值
plt.title("theta = " + str(np.round(sita, 3)))
plt.plot(data_x1, plot_y, c='blue', label='决策边界')
plt.plot(data_x1[:40], data_x2[:40], 'o', c='green', label='Admitted')
plt.plot(data_x1[40:], data_x2[40:], 'x', c='red', label='Not Admitted')
plt.plot()
plt.xlabel('Exam 1 score')
plt.ylabel('Exam 2 score')
plt.legend()  # 显示图例
plt.show()
exit()
```

4. 牛顿法

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
# @time    : 2020/10/24 15:42
# @author  : lerogo
# @fileName: main.py


import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt

# 解决 plt 画图中文字体字体
plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']  # macos
# plt.rcParams['font.sans-serif'] = ['KaiTi'] # windows
plt.rcParams['axes.unicode_minus'] = False  # 解决保存图像是负号'-'显示为方块的问
题


# 从 data 读取数据
def readData():
    data_x = pd.read_table("../data/ex4x.dat", sep=' ').values
    x1 = np.array(data_x[:, 3])
    x2 = np.array(data_x[:, 6])
    data_y = pd.read_table("../data/ex4y.dat", sep=' ').values
    y = np.array(data_y[:, 3])
    return x1, x2, y


# 归一化
def normalized(para):
    para = (para - para.min()) / (para.max() - para.min())
    return para


# 反归一化 算出需要的 theta
def reverse_normalized_sita(x1: np.ndarray, x2: np.ndarray, ori_sita):
    sita = [0, 0, 0]
    sita[0] = ori_sita[0] - (ori_sita[1] * x1.min()) / (
            x1.max() - x1.min()) - (ori_sita[2] * x2.min()) / (x2.max() -
x2.min())
    sita[1] = ori_sita[1] / (x1.max() - x1.min())
    sita[2] = ori_sita[2] / (x2.max() - x2.min())
    return sita
```

```python
# sita[0] sita[1] 代价函数
def get_cost(x1: np.ndarray, x2: np.ndarray, y: np.ndarray, sita):
    m = len(y)
    hx = h_x(x1, x2, sita)
    cost = -1 / m * ((y * np.log(hx) + (1 - y) * np.log(1 - hx)).sum())
    return cost


def h_x(x1: np.ndarray, x2: np.ndarray, sita):
    hx = []
    for i in range(len(x1)):
        hx.append(1 / (1 + math.exp(-(sita[0] + sita[1] * x1[i] + sita[2] *
x2[i]))))
    return np.array(hx)


def get_gradient(x1: np.ndarray, x2: np.ndarray, y: np.ndarray, sita):
    dev = []
    m = len(y)
    hx = h_x(x1, x2, sita)
    dev.append(
        ((1 / m) * (hx - y).sum())
    )
    dev.append(
        ((1 / m) * ((hx - y) * x1).sum())
    )
    dev.append(
        ((1 / m) * ((hx - y) * x2).sum())
    )
    return np.array(dev)


def get_theta(x1, x2, sita, dev):
    H_1 = getH_1(x1, x2, sita)
    sita = np.array(sita)
    tmpSita = [0, 0, 0]
    for i in range(len(dev)):
        tmpSita[i] = sita[i] - np.dot(H_1[i], dev)
    return tmpSita


def getH_1(x1, x2, sita):
    m = len(x1)
    hx = h_x(x1, x2, sita)
```

```python
    the_matrix = []
    for i in range(m):
        tmp = np.mat([1, x1[i], x2[i]])
        tmp = tmp.T * tmp
        the_matrix.append(tmp)
    the_matrix = np.array(the_matrix)
    H = np.zeros((3, 3))
    for i in range(m):
        H = H + (hx[i] * (1 - hx[i]) * (the_matrix[i]))
    H = H / m
    return np.linalg.pinv(H)


# x,y 步长 sita[0]sita[1] 接受的 cost 最小多少  最多迭代次数
def gen_ans(x1, x2, y, sita, accept_cost, max_times):
    times = 0  # 迭代次数
    cost = get_cost(x1, x2, y, sita)  # 计算第一次 cost
    cost_list = []  # 储存迭代出的 cost
    # 开始迭代
    while cost > accept_cost and times < max_times:
        sita = get_theta(
            x1=x1,
            x2=x1,
            sita=sita,
            dev=get_gradient(x1, x2, y, sita),
        )
        cost = get_cost(x1, x2, y, sita)  # 重新计算 cost
        cost_list.append(cost)  # 加入 cost_list 方便画图
        times += 1
    return sita, cost_list


if __name__ == '__main__':
    data_x1, data_x2, data_y = readData()
    # print(data_x1, data_x2, data_y)
    x1 = normalized(data_x1)
    x2 = normalized(data_x2)
    max_times = 10
    sita, cost_list = gen_ans(
        x1=x1,
        x2=x2,
        y=data_y,
        sita=[0, 0, 0],
        accept_cost=1e-5,
```

```
        max_times=max_times,
    )
    print(sita)
    sita = reverse_normalized_sita(data_x1, data_x2, sita)
    print(sita)
    plot_y = -(sita[0] + sita[1] * data_x1) / sita[2]

    # 作出 cost 的迭代曲线
    plt.title("cost 迭代曲线")
    plt.plot([x for x in range(int(max_times))], cost_list, c='blue',
label='cost 迭代曲线')
    plt.show()
    # 作出原始值
    plt.title("theta = " + str(np.round(sita, 3)))
    plt.plot(data_x1, plot_y, c='blue', label='决策边界')
    plt.plot(data_x1[:40], data_x2[:40], 'o', c='green', label='Admitted')
    plt.plot(data_x1[40:], data_x2[40:], 'x', c='red', label='Not Admitted')
    plt.plot()
    plt.xlabel('Exam 1 score')
    plt.ylabel('Exam 2 score')
    plt.legend()  # 显示图例
    plt.show()
    exit()
```