20337270_钟海财_实验

中山大学计算机学院

本科生实验报告

(2021学年春季学期)

课程名称：Artificial Intelligence

教学班级	**20级软工+网安**	**专业（方向）**	**软件工程**
学号	**20337270**	**姓名**	**钟海财**

一、 实验题目

实验任务1 KNN

在给定文本数据集完成文本情感分类训练，在测试集完成测试， 计算准确率。

实验任务2 回归

在给定文本数据集完成房价预测回归训练，画出训练loss曲 线图、数据可视化图、预测函数图。

实验任务3 分类

在给定文本数据集完成大学生录取预测分类训练，画出训练 loss曲线图、数据可视化图。（数据集课上发布）

二、 实验内容

1. 算法原理

1.KNN算法

一个样本与数据集中的k个样本最相似， 如果这k个样本中的大多数属于某一个类别， 则该样本也属于这个类别。也就是说，该方法在确定分类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。KNN方法在类别决策时，只与极少量的相邻样本有关。

2.单层感知机

由于M-P神经元模型参数需要事先设定好，为了能够自适应学习出所需要的参数，有研究人员就提出了单层感知机(Perceptron)

• 感知机的基本公式为：y x = $sign$(w x + b)

• sign为符号函数，当自变量为正数时取值为1，否则取值为0

3.损失函数

• 作用：为了衡量网络表现是否良好，并为之后的网络参数优化提供指导。

• 常见的用在分类任务上的损失函数：

• 均方误差(MSE): • 交叉熵

4.梯度下降

• 梯度定义：梯度是一个向量，表示某一函数在该点出的方向导数沿着该方向取得最大值。

• 也就是说该点处沿着梯度的方向变化最快，变化率最大

• 沿着梯度方向容易找到函数最大值

• 沿着梯度方向的反方向，容易找到函数最小值

• 梯度下降的一般公式为：θ = θ − $\eta\Delta L$

• 其中， η是学习率，ΔL是梯度， θ是参数

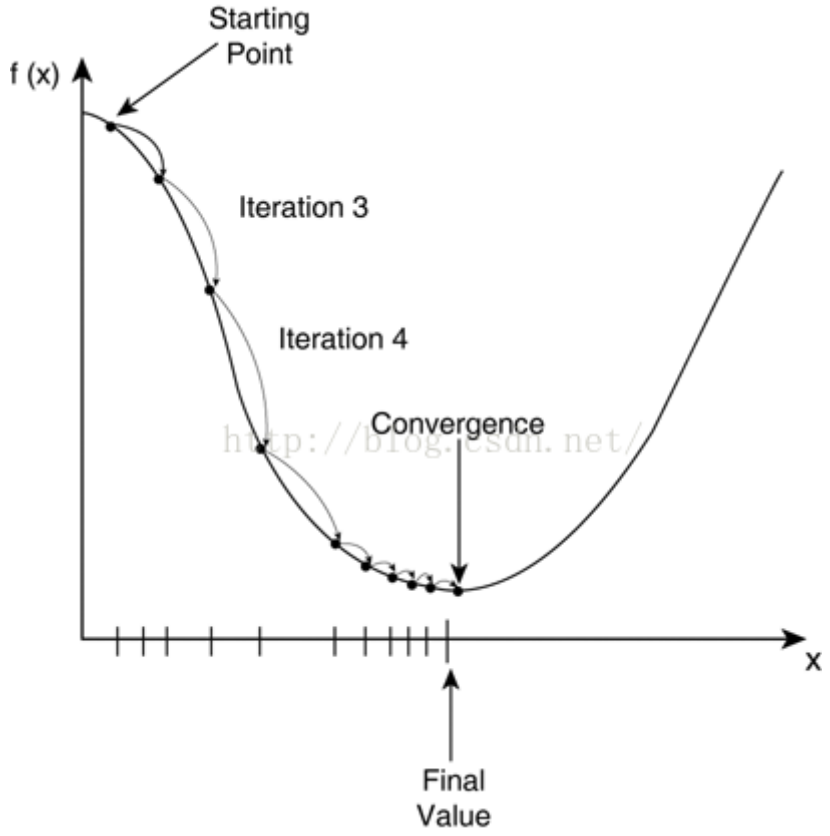
3.梯度下降法

最大似然估计的目标是求，似然函数 $L(w)$ （即，所有样本出现的总概率)最大时，对应的参数 w 的组合，而我们希望构造一个**代价函数（Cost Function）**来衡量我们在**某组参数**下预估的结果和实际结果的差距，**当代价函数值最小的时候，相应的参数w就是我们希望的最优解，**

即求 $l(w) = -L(w)$ 的最小值。

梯度下降算法是调整参数 w 的组合，使代价函数，取最小值，的最基本方法之一。

从直观上理解就是，我们在碗状结构的凸函数上，取一个初始值，然后挪动这个值，一步步靠近最低点，的过程，如下图所示：



因此我们知道，运用梯度下降算法求解，**要求代价函数（Cost Function）是凸函数**（为碗状），因为凸函数具有良好的性质（对于凸函数来说局部最小值点即为全局最小值点），这个性质使我们，一般会将非凸函数，转换为凸函数，进行求解。

而极大似然法得到的函数，为非凸函数，因此要对，最大似然函数，采用对数变换，转换为，**对数似然函数**：

$$log(l(w)) = -\sum_{i=1}^{m}[y_i log h_w(x^{(i)}) + (1-y_i) log(1-h_w(x^{(i)})] \text{；}$$

又因为log(l(w))是，对所有样本求得的对数似然函数，而代价函数希望可以描述一个样本，所以需要对log(l(w))取平均值，我们定义**逻辑回归的代价函数**为：

$$J(w) = -\frac{1}{m}\sum_{i=1}^{m}[y_i log h_w(x^{(i)}) + (1-y_i) log(1-h_w(x^{(i)})]$$

从代价函数的直观表达上来看，当 $y_i = 1，\ h_w(x^{(i)}) = 1$ 时(预测类别和真实类别相同)，$J(w \mid x^{(i)}) = 0$；当 $y_i = 1，\ h_w(x^{(i)}) \to 0$ 时(预测类别和真实类别相反)，$J(w \mid x^{(i)}) \to \infty$（注意对数函数前有个负号）。这意味着，当预测结果和真实结果越接近时，预测产生的代价越小，当预测结果和真实结果完全相反时，预测会产生很大的惩罚。该理论同样适用于 $y^{(i)} = 0$ 的情况。代价函数（*Cost Function*）也被称为，损失函数、目标函数。

在求解时，我们为了找到最小值点，将分为两个步骤操作：

1. 找到下降速度最快的方向(导函数/偏导方向)；
2. 朝这个方向迈进一小步，再重复步骤1、2，直至最低点。

针对步骤1，通过 $J(w)$ 对 w_j 的一阶导数来找下降方向 g

$$
\begin{aligned}
g &= \frac{\partial J(w)}{\partial w} \\
&= \sum_{i=1}^{m} \frac{y^{(i)}}{h_w(x^{(i)})} h_w(x^{(i)})(1-h_w(x^{(i)}))(-x_j^{(i)}) + (1-y^{(i)}) \frac{1}{1-h_w(x^{(i)})} h_w(x^{(i)})(1-h_w(x^{(i)}))x_j^{(i)} \\
&= \sum_{i=1}^{m} (y^{(i)} - h_w(x^{(i)}))x_j^{(i)}
\end{aligned}
$$

针对步骤2，以迭代的方式来实现，迭代方式为 $w_j^{(k+1)} = w_j^{(k)} - \alpha g$ ，α表示步长,k为迭代次数，则梯度下降法的迭代表达式为：

$$
w_j = w_j + \alpha \sum_{i=1}^{m} (y^{(i)} - h_w(x^{(i)}))x_j^{(i)}
$$

对单个样本 $y^{(i)}$ 来说，参数 w_j 的表达式为：

$$
w_j = w_j + \alpha (y^{(i)} - h_w(x^{(i)}))x_j^{(i)}
$$

2. 关键代码展示（带注释）

设计思路：

1.回归：

仿照给出的参考代码test.py，先获得原始数据并将其画出数据可视化图，然后对原始数据进行处理：添加bias，再对其进行归一化。

然后将数据分为两个部分：X（包含[bias, x1, x2]）和 y，再将X，y，w转化为数组。

然后使用梯度下降法学习单层感知机的参数。得到模型收敛的参数后，画出预测函数图和训练loss曲线图。

2.分类：

类似回归的步骤，仿照给出的参考代码test.py，先获得原始数据并将其画出数据可视化图，然后对原始数据进行处理：添加bias，再对其进行归一化（这时不用再对y进行归一化，因为y只有两个值：0或1）。

然后将数据分为两个部分：X（包含[bias, x1, x2]）和 y，再将X，y，w转化为数组。

然后使用梯度下降法学习单层感知机的参数。得到模型收敛的参数后，根据预测函数f = x2 * g[2, 0] + x1 * g[1, 0] + g[0, 0]令f=0.5时便得到了一条有关x1和x2的直线: x2 = (0.5 - x1 * g[1, 0] - g[0, 0]) / g[2, 0] ,然后画出该直线（分类曲线）和数据可视化图（两者在同一张图上）。

最后画出训练loss曲线图，并计算分类准确率。

代码：

1.KNN

```
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np
import heapq


def main():
    x1 = []     # 句子
    y1 = []     # 标签
    all_y1 = [0, 0, 0, 0, 0, 0, 0]

    train_data = []
    for line in open("Classification/train.txt", "r"):  # 读入train.txt到列表clause中
        data = line[:]
        str_list = data.split()     # 字符串转化成列表
        train_data.append(str_list)  # 将每一行文件加入到列表clause中
    for i in range(1, len(train_data)):
        list_words = train_data[i][3:]
        clause = " ".join(list_words)
        x1.append(clause)
        y1.append(train_data[i][1])
        all_y1[int(train_data[i][1])] += 1
    # print("train_data  ", all_y1)

    test_data = []
    for line2 in open("Classification/test.txt", "r"):  # 读取测试集
        data = line2[:]
        str_list = data.split()
        test_data.append(str_list)

    # 词袋模型，转化为词向量
    countvec = CountVectorizer(token_pattern='[\w]{1,}')  # 26,38.1%
    countvec1 = countvec.fit_transform(x1).toarray()
    # print(countvec1)

    k1 = 26
    right = 0   # 记录正确的分类结果的数目
    for i in range(1, len(test_data)):  # 对测试集中的每个句子进行分类
        list_words = test_data[i][3:]
        clause = " ".join(list_words)
        s_seg_vec = countvec.transform([clause]).toarray()  # 将测试的句子转化为词向量
        # print(s_seg_vec)
        kdd = []    # 用于记录距离
        for t in range(len(countvec1)):
            dist = np.linalg.norm(countvec1[t] - s_seg_vec)
            # 求该句子的词向量与每个测试集句子的词向量欧式距离
            heapq.heappush(kdd, [dist, y1[t]])
        num = [0, 0, 0, 0, 0, 0, 0]     # 记录最近的k个点中每种点的数量
        for k3 in range(k1):
            min = heapq.heappop(kdd)
            num[int(min[1])] += 1
        key = 1
        max_pi = 0
        for t3 in range(1, 7):  # 选出k个点中数量最多的点
            if num[t3] * all_y1[t3] > max_pi:
                max_pi = num[t3] * all_y1[t3]
                key = t3  # 获得最大概率对应的情绪的序号
```

```
        if key == int(test_data[i][1]):    # 如果与正确结果相同
            right += 1
        test_data[i].append(key)     # 在测试的句子末尾加入该分类结果，方便观察
        print(test_data[i])
    print("正确率为：  ", right / (len(test_data) - 1) * 100, "%", sep="")  # 输出正确率


if __name__ == '__main__':
    main()
```

2.回归

```
# import imp
import numpy as np
import pandas as pd
from pathlib import Path
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号

# 获得数据
path = Path(__file__).parent / 'data/regress_data2.csv'
data = pd.read_csv(str(path), dtype=np.double)
data2 = pd.read_csv(str(path))

# 画原始数据可视化图
fig1 = plt.figure()
ax = fig1.add_subplot(projection='3d')
ax.scatter(data['面积'], data['房间数'], data['价格'])
ax.set_xlabel('面积')
ax.set_ylabel('房间数')
ax.set_zlabel('价格')
plt.suptitle("数据可视化图")
plt.show()

#计算均方误差
def computeCost(X, y, w):
    inner = np.power((np.matmul(X, w) - y), 2)
    return np.sum(inner) / (2 * X.shape[0])


# 添加bias
data.insert(0, 'Ones', 1) # 添加b的系数，因为b前面的系数为1，所以可以把1直接放在x里面
'''
例如 x = [3, 4], k = [0.1, 0.2], b = 0.01
kx+b = [0.1, 0.2] * [3, 4] + 0.01= [0.01, 0.1, 0.2] * [1, 3, 4] = [b, k0, k1] * [1, x1, x2] = b + k0*x0 + k1*x1 = b + kx
为了方便运算，把b放到k里面的第一项，b和x无关，所以在x前面放个1
'''
# 数据归一化
x1max = data['面积'].max()
x1min = data['面积'].min()
x2max = data['房间数'].max()
x2min = data['房间数'].min()
ymax = data['价格'].max()
ymin = data['价格'].min()
for i in range(data.shape[0]):  # 归一化
    data['面积'][i] = (data['面积'][i]-x1min) / (x1max - x1min)
    data['房间数'][i] = (data['房间数'][i] - x2min) / (x2max - x2min)
```

```
    data['价格'][i] = (data['价格'][i] - ymin) / (ymax - ymin)
'''
获取数据
'''
cols = data.shape[1]
X = data.iloc[:, :cols-1]    # 去掉y
y = data.iloc[:, cols-1:]    # 去掉x1,x2
'''
为了方便矩阵运算，设置X，y，w的shape
'''
X = np.array(X.values, dtype=np.double)  # (97, 2)
y = np.array(y.values, dtype=np.double).reshape(-1, 1) # (97, 1)
w = np.array([0,0,0], dtype=np.double).reshape(-1, 1) # (2, 1)


# 梯度下降
def batch_gradientDescent(X, y, w, alpha, iters):
    temp = np.zeros(w.shape, dtype=np.double)
    parameters = w.shape[0]
    cost = []
    for i in range(iters):
        error = np.matmul(X, w) - y
        for j in range(parameters):
            term = np.multiply(error, X[:, j:j+1])
            temp[j, 0] = w[j, 0] - ((alpha / len(X)) * np.sum(term))
        w = temp
        cost.append(computeCost(X, y, w))
    return w, cost

# 设置学习率和迭代次数
alpha = 0.01
iters = 10000
# 神经网络利用梯度下降学习参数
g, cost = batch_gradientDescent(X, y, w, alpha, iters)

# 画出预测函数图
x1 = np.linspace(0, 1, data.shape[0])
x2 = np.linspace(0, 1, data.shape[0])
x1, x2 = np.meshgrid(x1, x2)
f = x2 * g[2, 0] + x1 * g[1, 0] + g[0, 0]
for i in range(data.shape[0]):  # 数据复原
    x1[i] = x1[i] * (x1max - x1min) + x1min
    x2[i] = x2[i] * (x2max - x2min) + x2min
    f[i] = f[i] * (ymax - ymin) + ymin

ax1 = plt.axes(projection='3d')
ax1.scatter(data2['面积'],data2['房间数'], data2['价格'], label='训练数据')
ax1.plot_surface(x1,x2,f,rstride=1, cstride=1, cmap=plt.cm.coolwarm, alpha=0.5)
ax1.set_xlabel('面积')
ax1.set_ylabel('房间数')
ax1.set_zlabel('价格')
plt.suptitle("预测函数图")
plt.show()

# 画loss曲线图
fig, ax2 = plt.subplots(figsize=(8, 6))
ax2.plot(np.arange(iters), cost, 'r')
ax2.set_xlabel('迭代次数', fontsize=18)
ax2.set_ylabel('代价', rotation=0, fontsize=18)
ax2.set_title('误差和训练Epoch数', fontsize=18)
plt.show()
```

3.分类

```
# import imp
import numpy as np
import pandas as pd
from pathlib import Path
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号

# 获得数据
path = Path(__file__).parent / 'classification_data.txt'
data = pd.read_csv(str(path), header=None, names=['特征一', '特征二', '录取结果'],
dtype=np.double)
data2 = pd.read_csv(str(path), header=None, names=['特征一', '特征二', '录取结果'],
dtype=np.double)
colors = []
for i in range(data.shape[0]):
    t1 = 'g' if data['录取结果'][i] == 1 else 'r'
    colors.append(t1)
'''
画原始数据曲线图
'''
fig1 = plt.figure()
ax = fig1.add_subplot()
ax.scatter(data['特征一'], data['特征二'], c=colors)
ax.set_xlabel('特征一')
ax.set_ylabel('特征二')
plt.suptitle("数据可视化图")
plt.show()


# 计算均方误差
def computeCost(X, y, w):
    inner = np.power((np.matmul(X, w) - y), 2)
    return np.sum(inner) / (2 * X.shape[0])


# 添加bias
data.insert(0, 'Ones', 1)  # 添加b的系数，因为b前面的系数为1，所以可以把1直接放在x里面
'''
例如 x = [3, 4] , k = [0.1, 0.2], b = 0.01
kx+b = [0.1, 0.2] * [3, 4] + 0.01= [0.01, 0.1, 0.2] * [1, 3, 4] = [b, k0, k1] * [1,
x1, x2] = b + k0*x0 + k1*x1 = b + kx
为了方便运算，把b放到k里面的第一项，b和x无关，所以在x前面放个1
'''
# 归一化
x1max = data['特征一'].max()
x1min = data['特征一'].min()
x2max = data['特征二'].max()
x2min = data['特征二'].min()
for i in range(data.shape[0]):  # 归一化
    data['特征一'][i] = (data['特征一'][i] - x1min) / (x1max - x1min)
    data['特征二'][i] = (data['特征二'][i] - x2min) / (x2max - x2min)
    # data['录取结果'][i] = (data['录取结果'][i] - ymin) / (ymax - ymin)

# 获取数据
cols = data.shape[1]
X = data.iloc[:, :cols - 1]  # 去掉y
```

```
y = data.iloc[:, cols - 1:]  # 去掉x1,x2

# 为了方便矩阵运算，设置X，y，w的shape
X = np.array(X.values, dtype=np.float)  # (97, 2)
y = np.array(y.values, dtype=np.float).reshape(-1, 1)  # (97, 1)
w = np.array([0, 0, 0], dtype=np.float).reshape(-1, 1)  # (2, 1)


# 梯度下降
def batch_gradientDescent(X, y, w, alpha, iters):
    temp = np.zeros(w.shape, dtype=np.float)
    parameters = w.shape[0]
    cost = []
    for i in range(iters):
        error = np.matmul(X, w) - y
        for j in range(parameters):
            term = np.multiply(error, X[:, j:j + 1])
            temp[j, 0] = w[j, 0] - ((alpha / len(X)) * np.sum(term))
        w = temp
        cost.append(computeCost(X, y, w))
    return w, cost


# 设置学习率和迭代次数
alpha = 0.01
iters = 1500
# 神经网络利用梯度下降学习参数
g, cost = batch_gradientDescent(X, y, w, alpha, iters)

# 画分界线
x1 = np.linspace(0, 1, 100)
x2 = np.linspace(0, 1, 100)
x1, x2 = np.meshgrid(x1, x2)
f = x2 * g[2, 0] + x1 * g[1, 0] + g[0, 0]
# x_2 = (f - x_1*g[1,0] - g[0,0])/g[2,0]
x3 = (0.5 - x1 * g[1, 0] - g[0, 0]) / g[2, 0]
for i in range(100):
    x1[i] = x1[i] * (x1max - x1min) + x1min
    x2[i] = x2[i] * (x2max - x2min) + x2min
    x3[i] = x3[i] * (x2max - x2min) + x2min

# 画出分界线
plt.scatter(x1, x3,  c='b')     # 该句画出了分界线
plt.scatter(data2['特征一'], data2['特征二'], c=np.array(colors))
plt.xlabel('特征一')
plt.ylabel('特征二')
plt.suptitle("数据可视化图 + 分界线")
plt.show()

# 画loss曲线图
fig, ax2 = plt.subplots(figsize=(8, 6))
ax2.plot(np.arange(iters), cost, 'r')
ax2.set_xlabel('迭代次数', fontsize=18)
ax2.set_ylabel('代价', rotation=0, fontsize=18)
ax2.set_title('误差和训练Epoch数', fontsize=18)
plt.show()

# 计算准确率
x1 = np.array(data.iloc[:, 1:2], dtype=np.double).reshape(-1, 1)
x2 = np.array(data.iloc[:, 2:3], dtype=np.double).reshape(-1, 1)
f = x2 * g[2, 0] + x1 * g[1, 0] + g[0, 0]
```

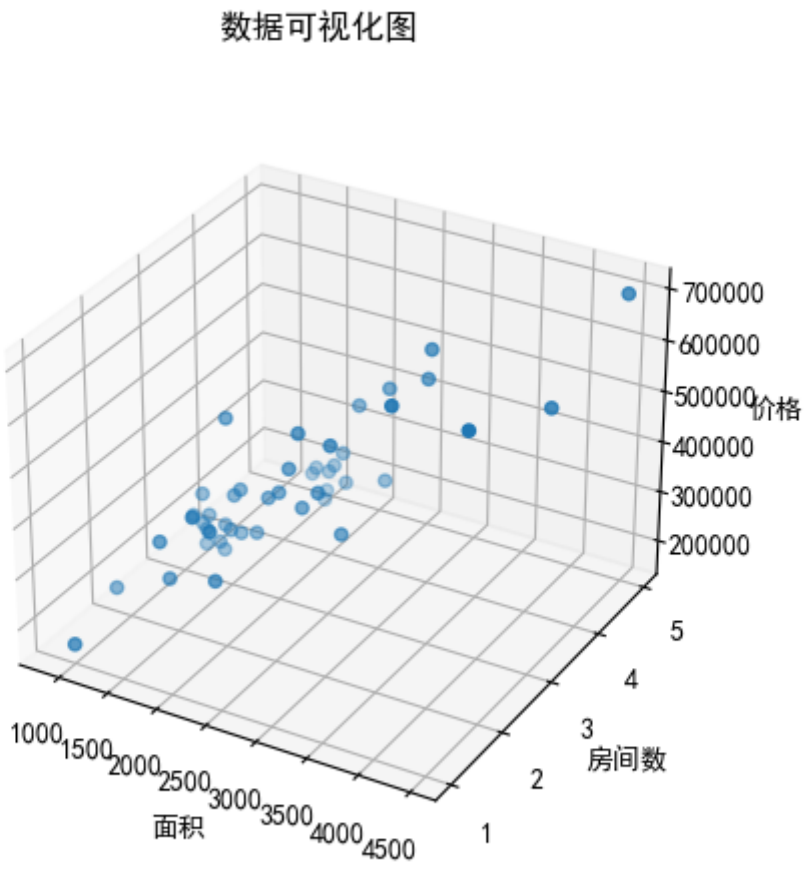
```
y = data.iloc[:, 3:]
y = np.array(y, dtype=np.float).reshape(-1, 1)
right = 0
for i in range(f.shape[0]):
    if f[i] > 0.5 and y[i] == 1:
        right += 1
    elif f[i] <= 0.5 and y[i] == 0:
        right += 1
print("模型收敛后的分类准确率: ", right, "%")
```

三、 实验结果及分析

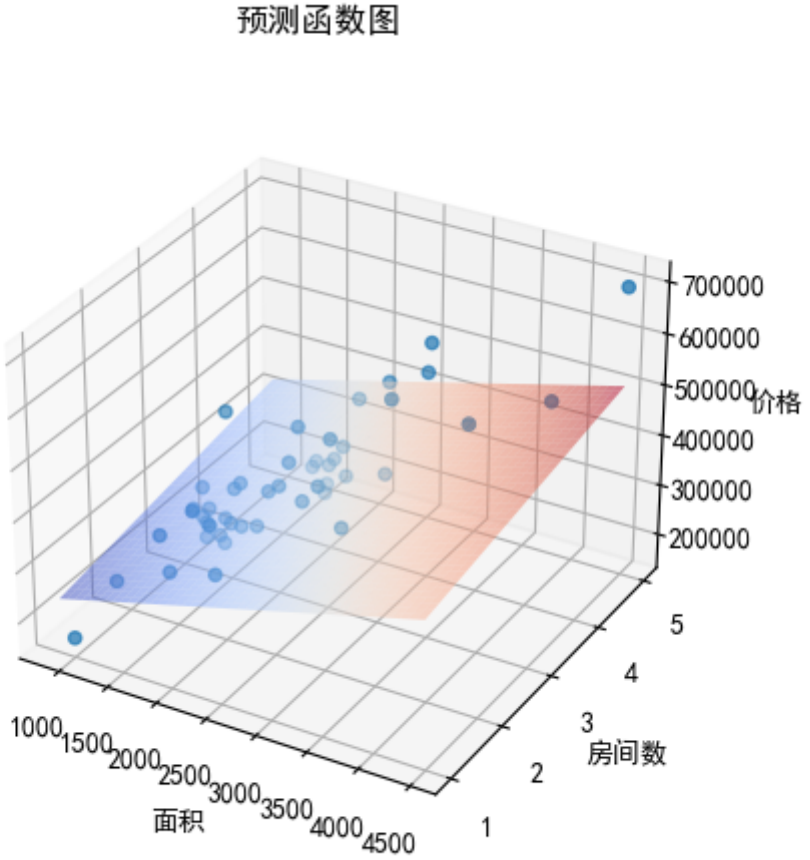
1.回归

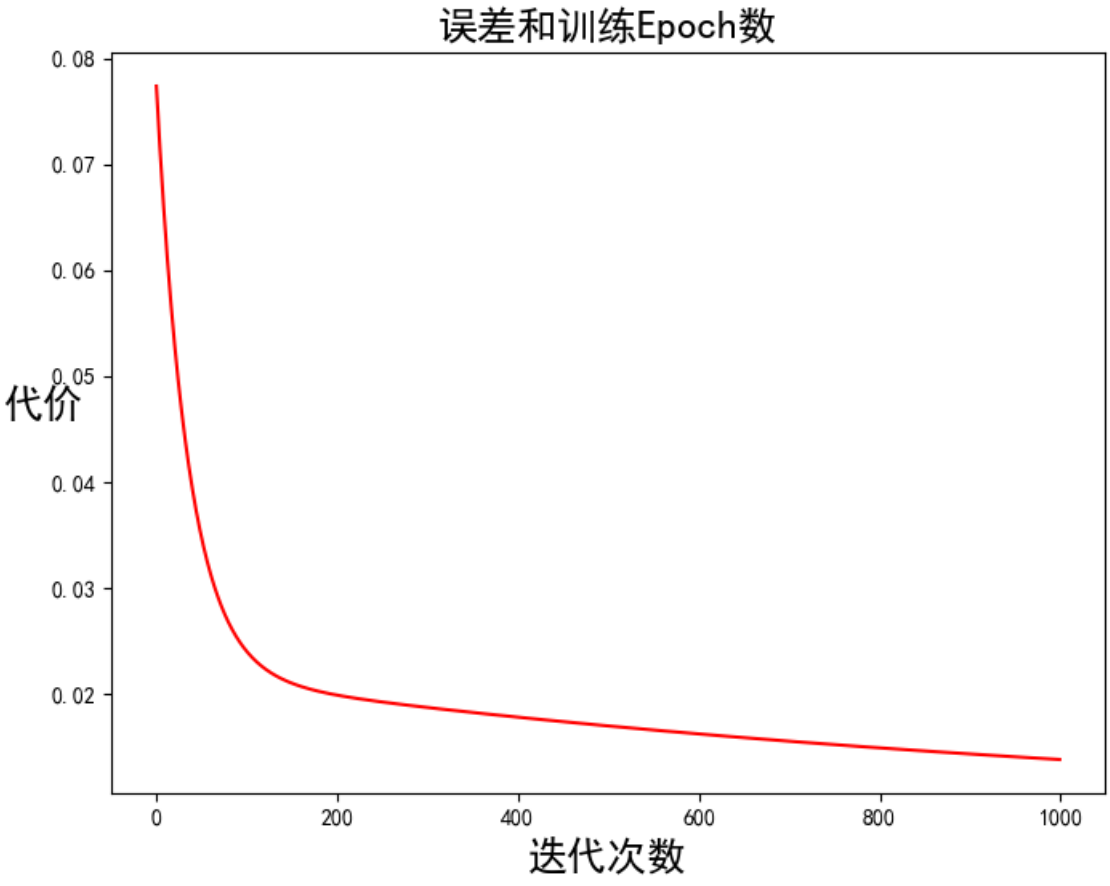
1. 实验结果展示示例（可图可表可文字，尽量可视化）

数据可视化图:



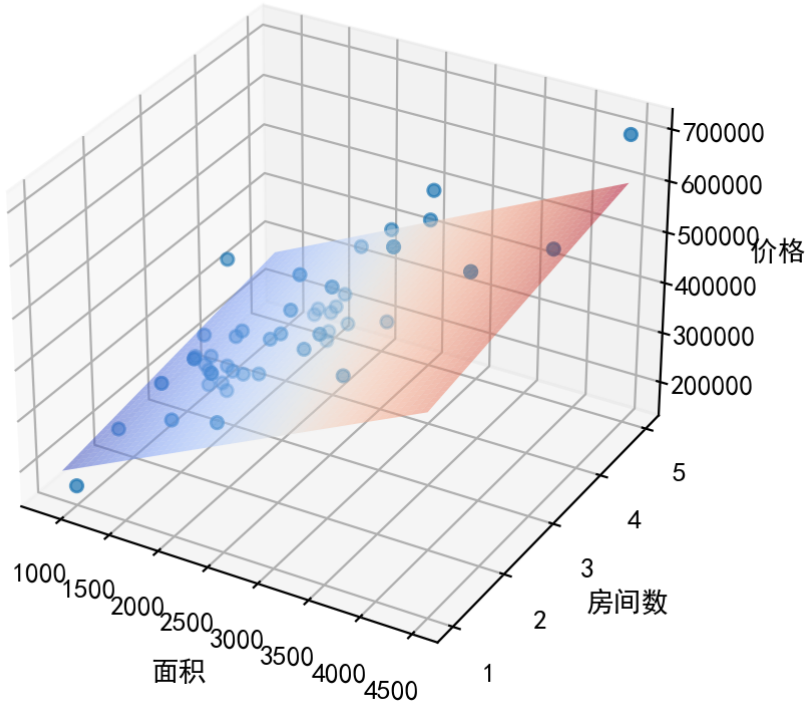
迭代次数为1000时:

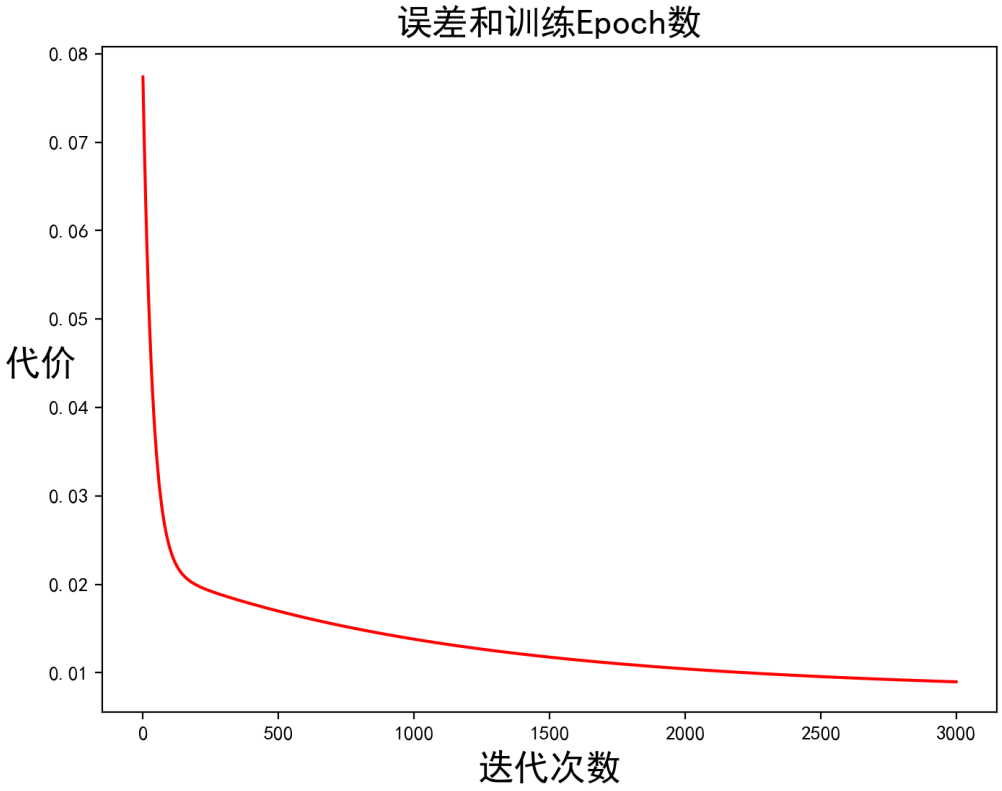

预测函数图
700000
600000
500000
400000
300000
200000
价格
1000
1500
2000
2500
3000
3500
4000
4500
面积
1
2
3
4
5
房间数


误差和训练Epoch数
0.08
0.07
0.06
0.05
0.04
0.03
0.02
代价
0
200
400
600
800
1000
迭代次数

迭代次数为3000时：

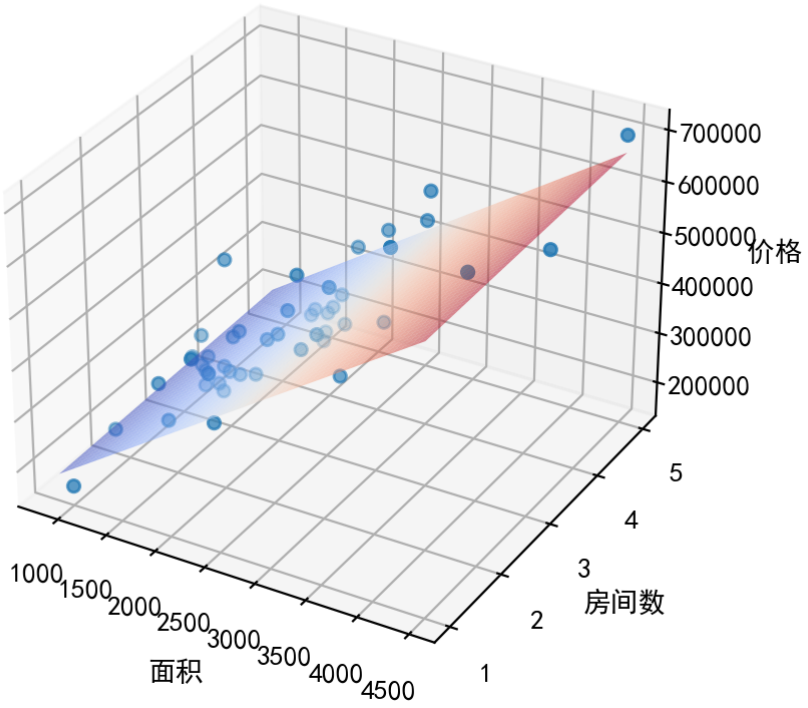
预测函数图

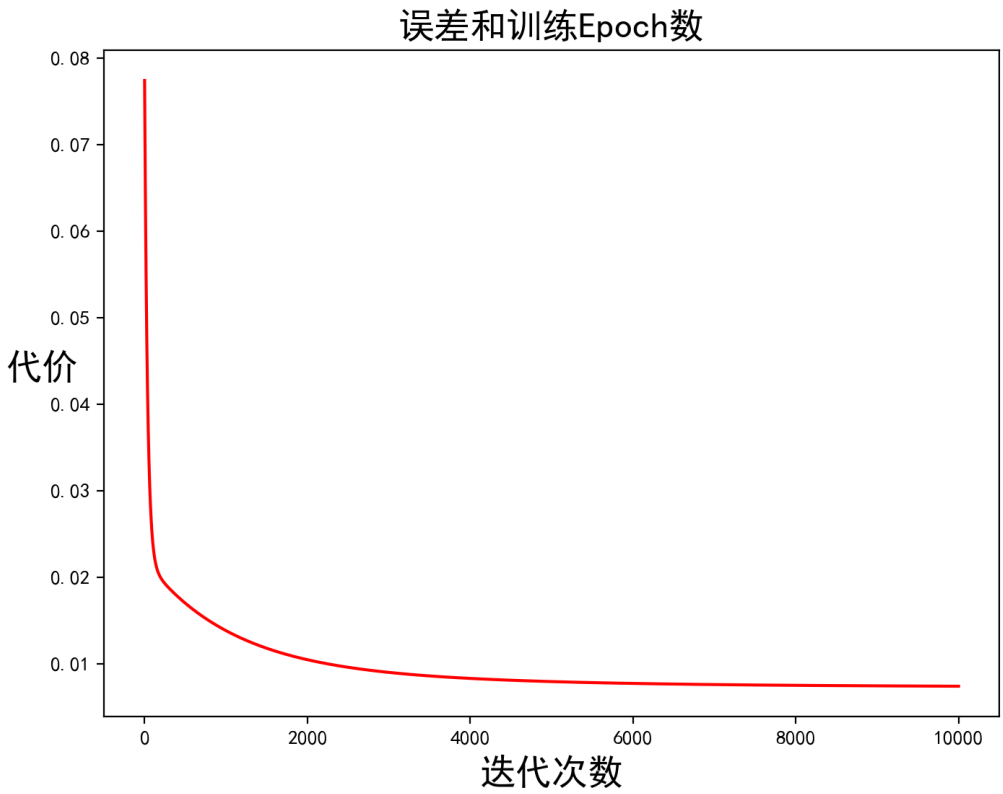




迭代次数为10000时：

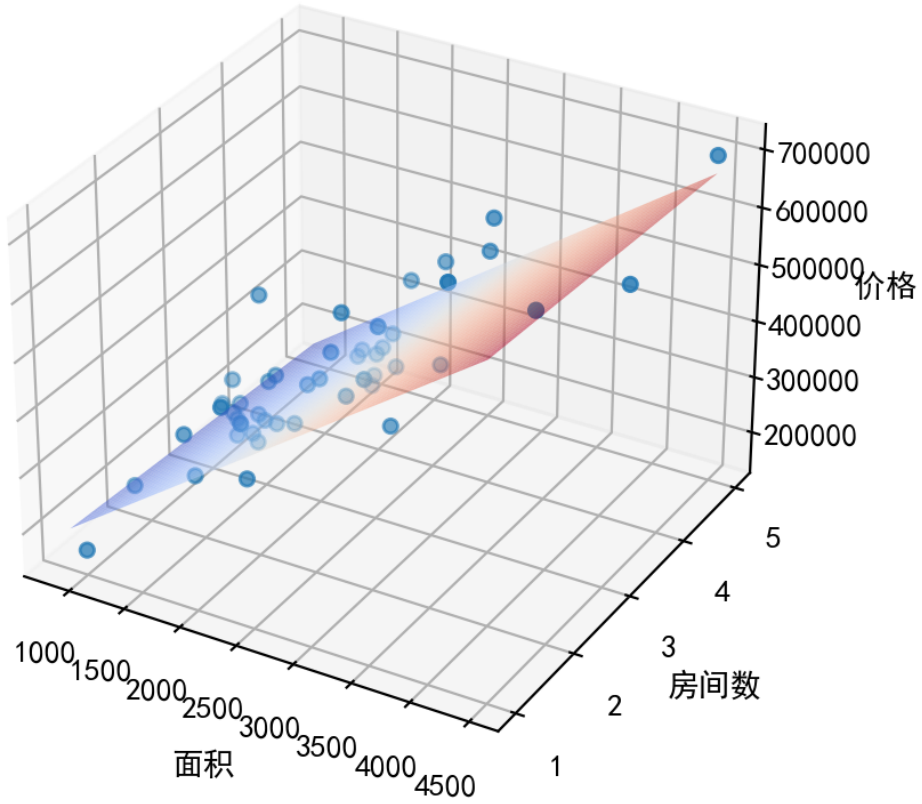
预测函数图

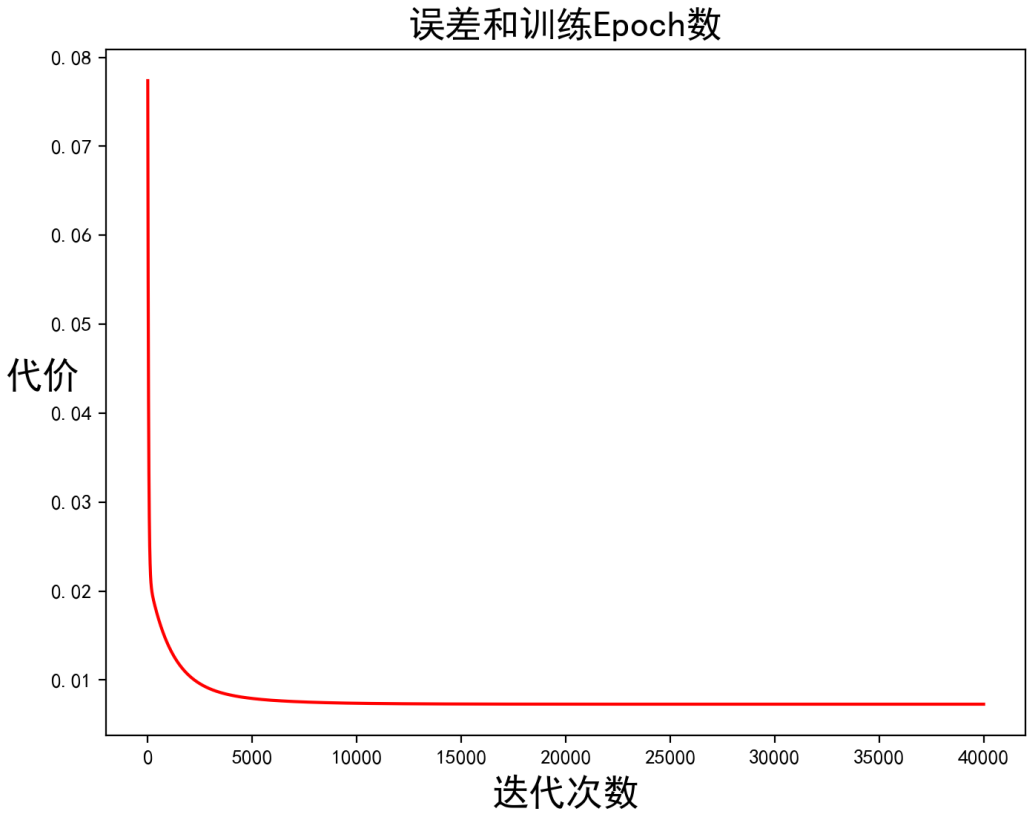




迭代次数为40000时：

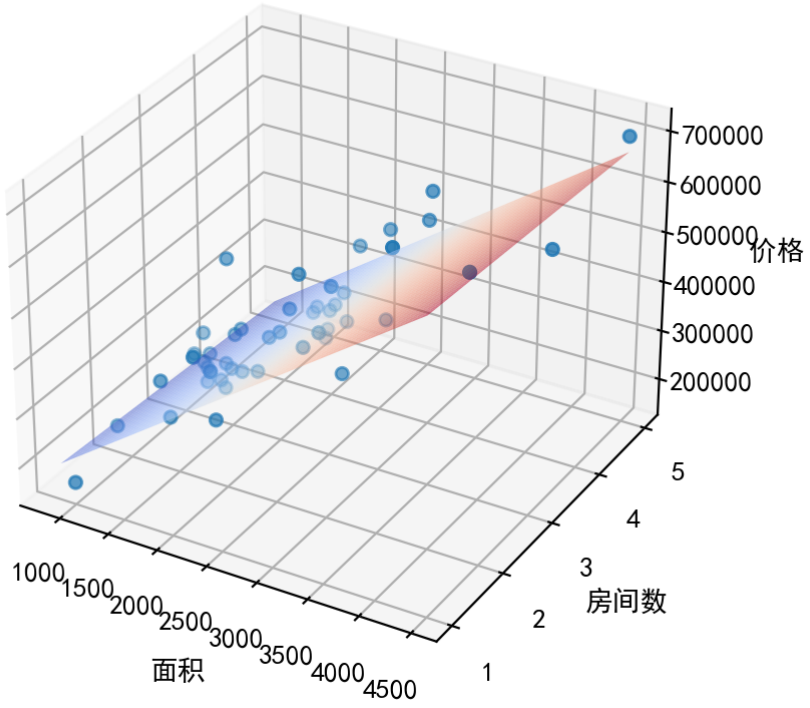
预测函数图



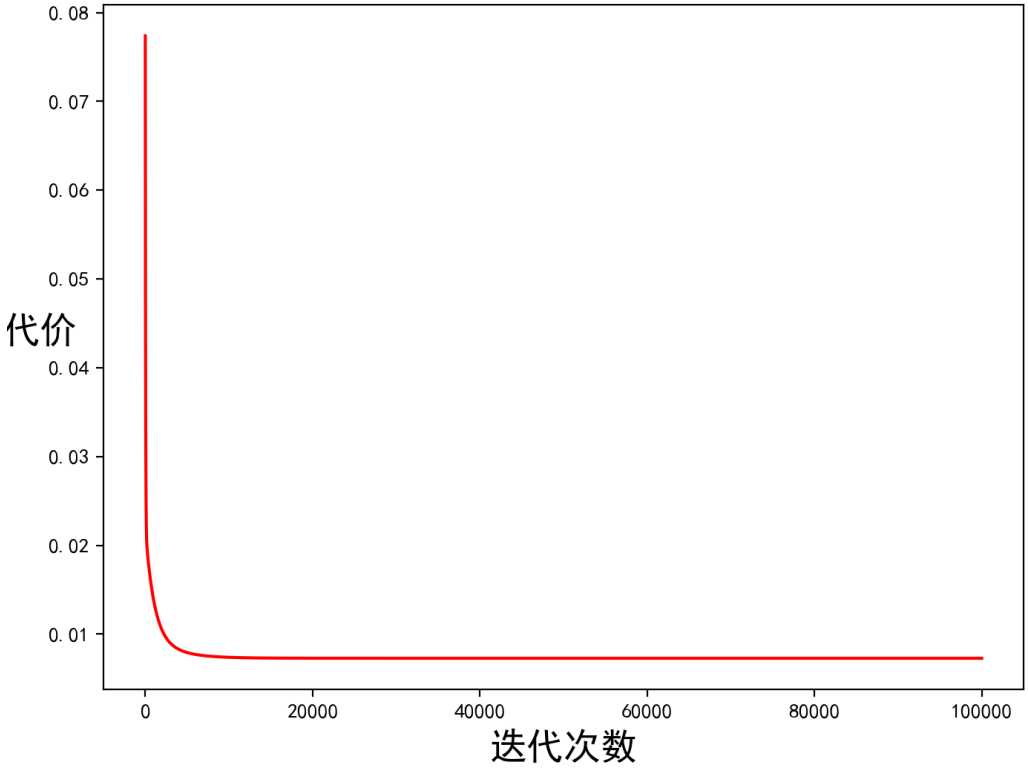


迭代次数为100000时：

预测函数图



误差和训练Epoch数



2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

评测指标：代价值 和 模型状态

分析：

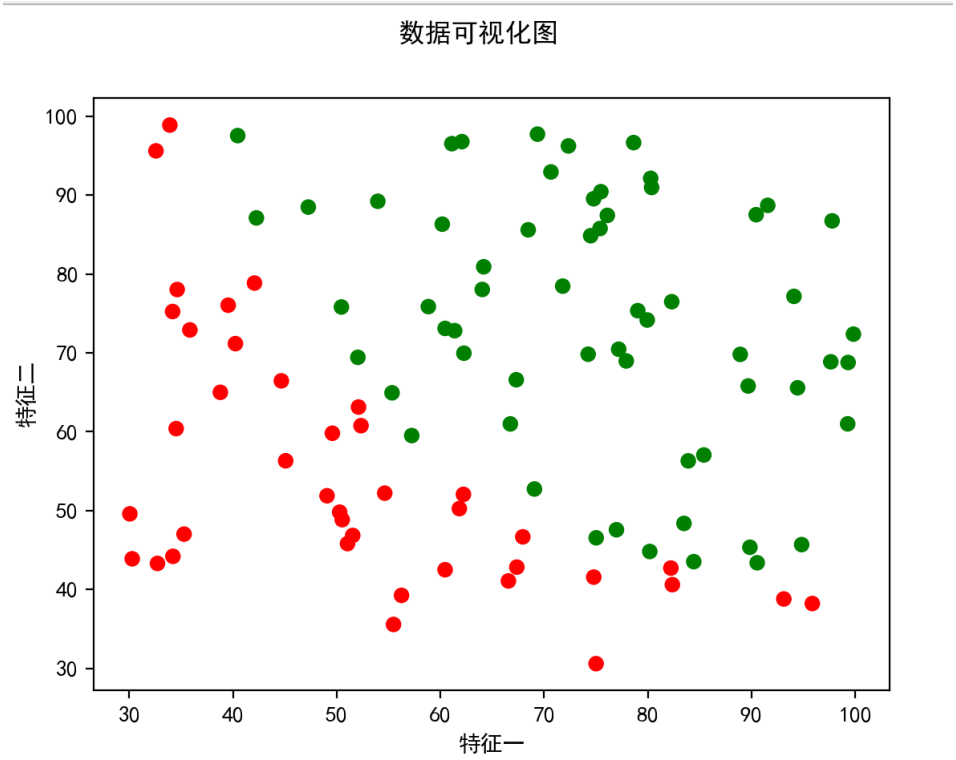
随着迭代次数的增加，代价值整体上逐渐减小，最终趋于收敛，不为0，但还存在一定波动，因为可能存在过拟合的情况。

随着迭代次数的增加，模型拟合状态逐渐提升，最终保持稳定。

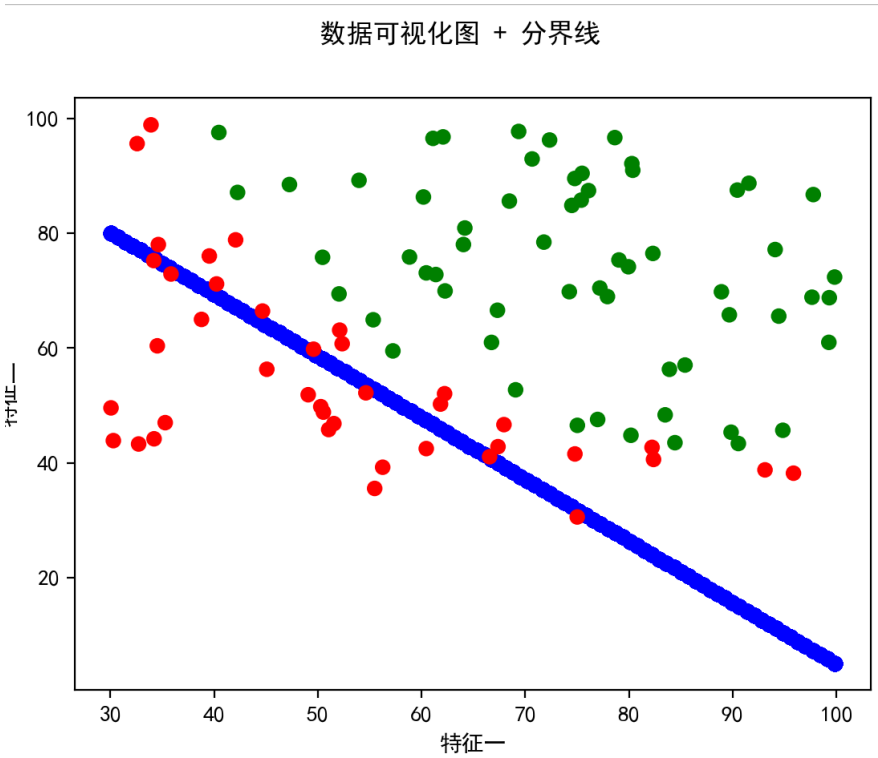
2.分类

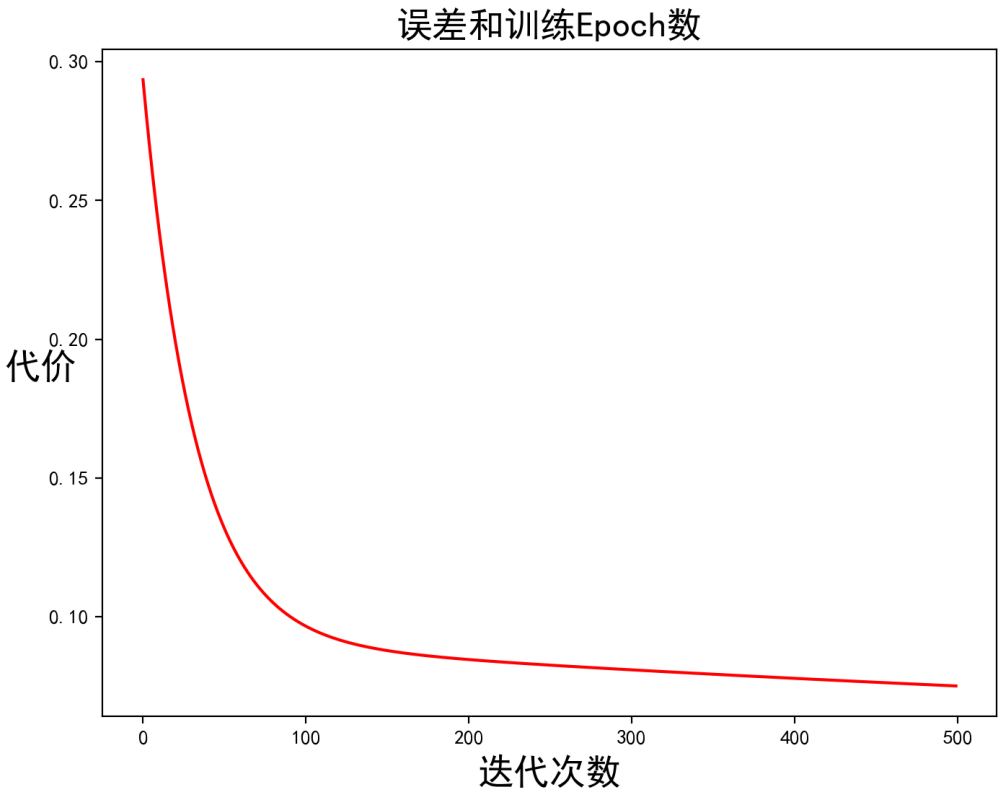
1. 实验结果展示示例（可图可表可文字，尽量可视化）

数据可视化：



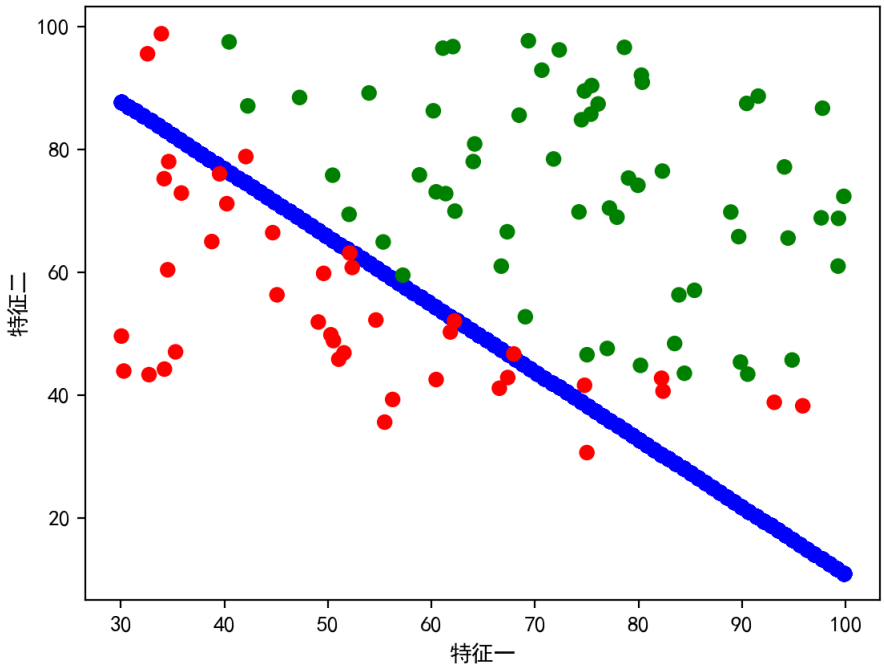
迭代次数为500时（正确率：80%）：

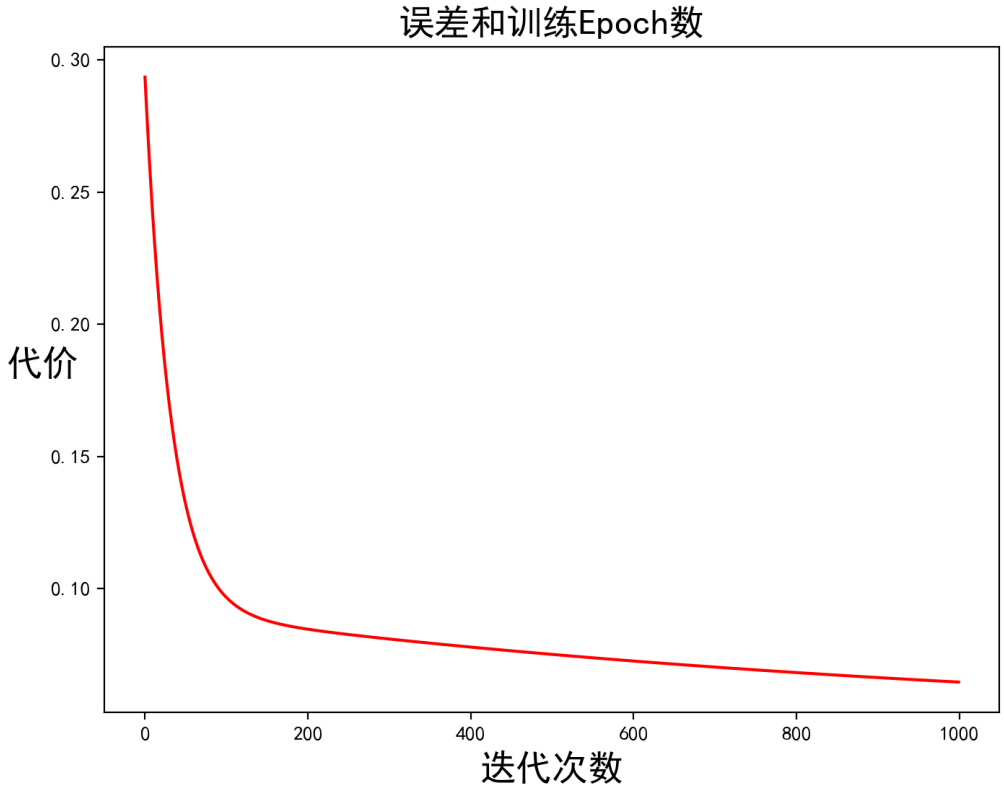




迭代次数为1000时（正确率：90%）：

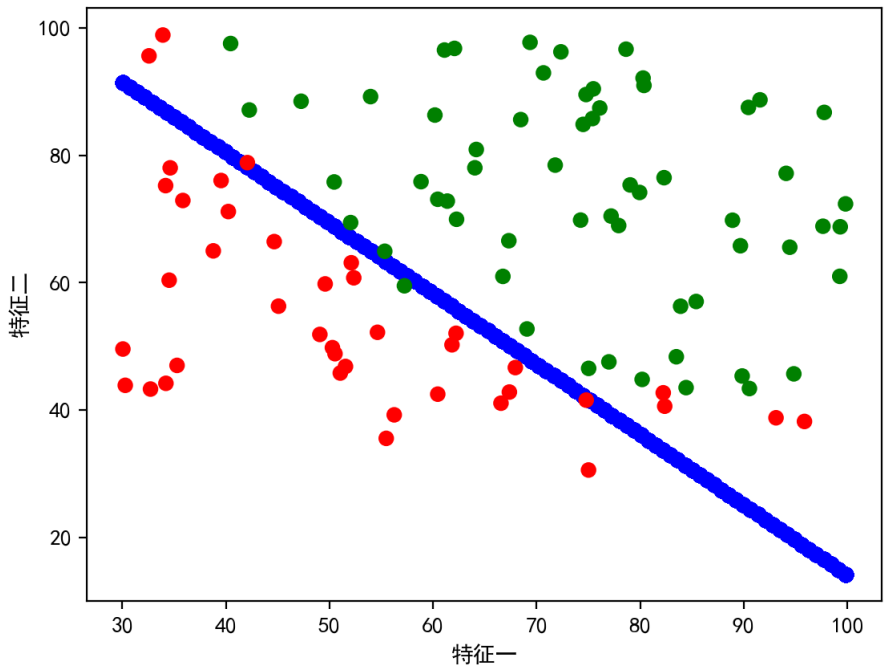
数据可视化图 + 分界线

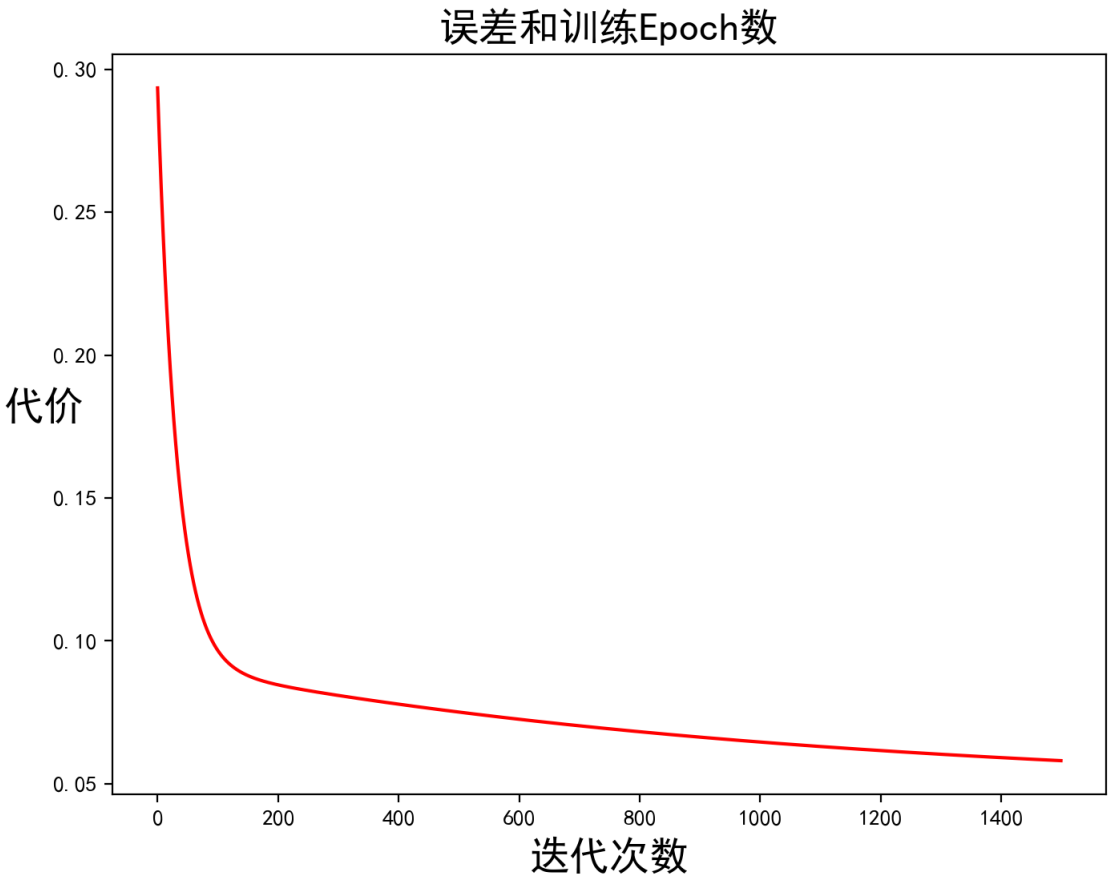




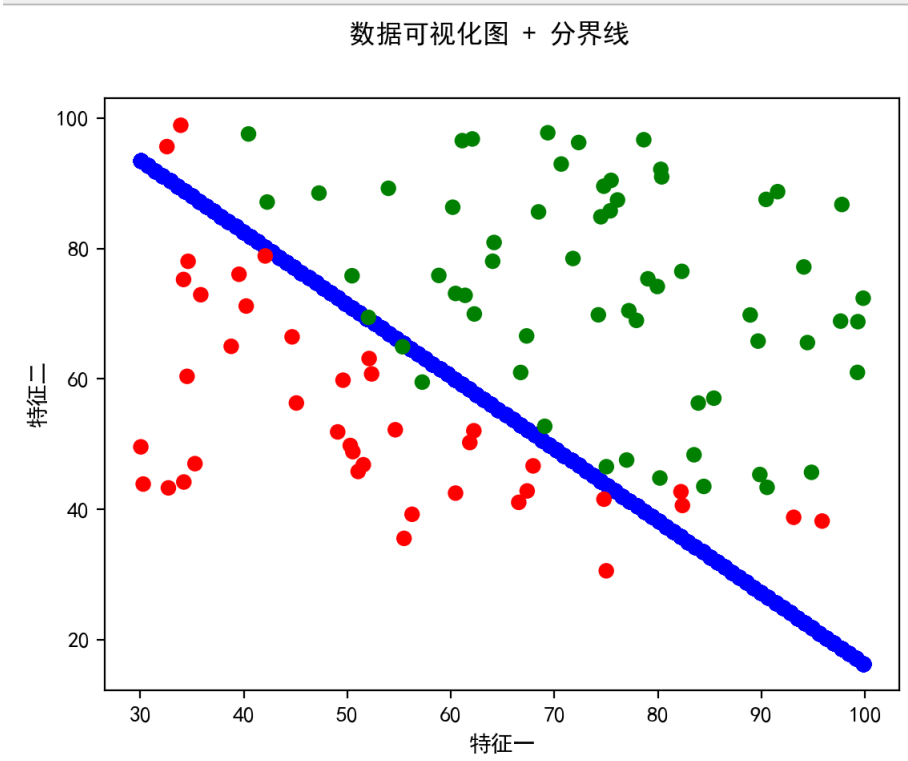
迭代次数为1500时（正确率：92%）：

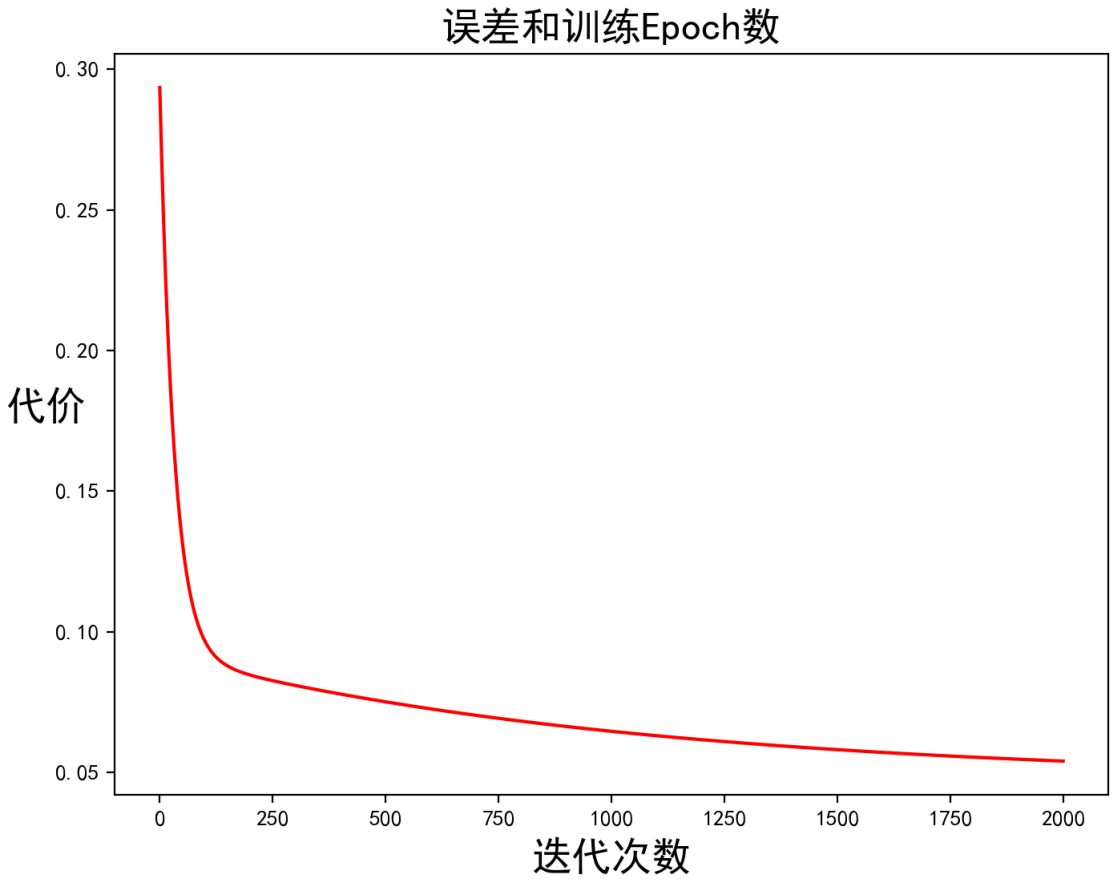
数据可视化图 + 分界线



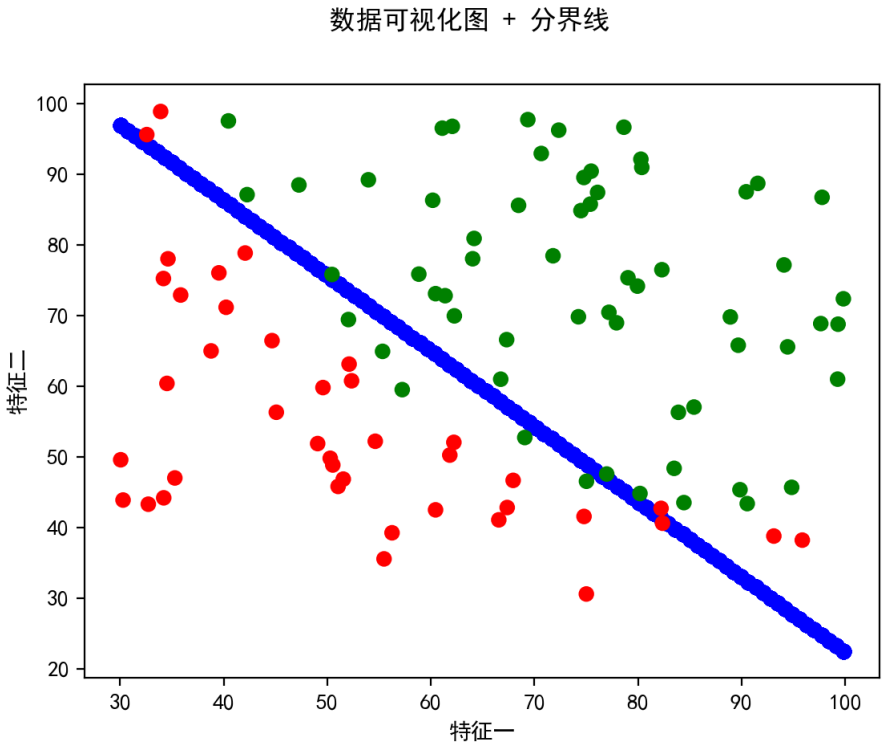


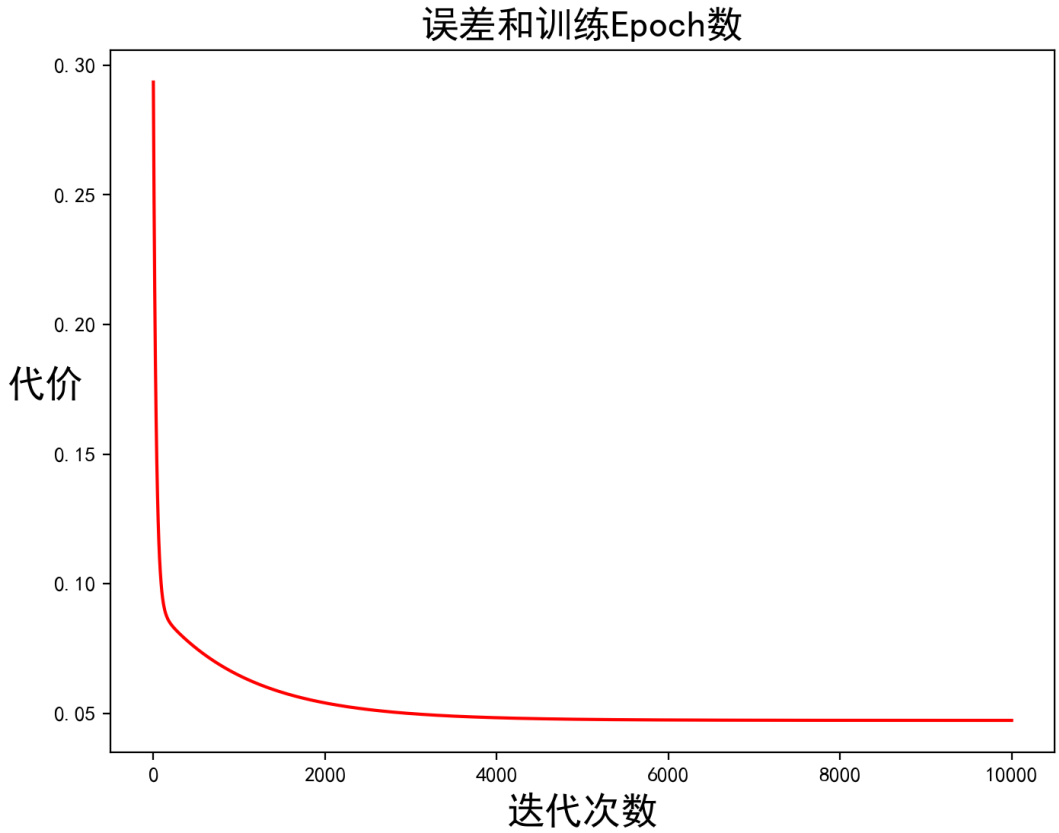
迭代次数为2000时（正确率：92%）：





迭代次数为10000时（正确率：90%）：





2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

评测指标：代价，模型收敛后的分类准确率。

分析：

随着迭代次数的增加，代价值整体上逐渐减小，最终趋于收敛，不为0，但还存在一定波动，因为可能存在过拟合的情况。

同样的，随着迭代次数的增加，模型收敛后的分类准确率逐渐提升（从80%提升至92%），最终稳定在90%（偶尔存在1%的波动）。

四、 思考题

本次实验无思考题。

五、 参考资料

1. 实验文档：ML-1，2，3.pdf
2. 课本：《人工智能》(第三版) 清华大学出版社
3. 逻辑回归2（极大似然与梯度下降） - 知乎 (zhihu.com)
4. 梯度下降求解逻辑回归2（代码编写以及三种梯度下降对比）*wangbowj123*的博客-*CSDN*博客逻辑回归梯度下降代码