Lab3-基于牛顿法的逻辑回归求解

吴欣怡 PB21051111

2024 年 1 月 11 日

1 问题介绍

逻辑回归是一种用于二分类问题的机器学习算法，通过学习输入特征与二元目标变量之间的关系来进行预测。本实验使用牛顿法实现逻辑回归，通过训练数据集来拟合模型参数，进而对测试集进行预测，评估模型性能。

2 算法原理

逻辑回归的目标是找到一个能够最大化似然函数的参数，通过最小化损失函数来实现。牛顿法是一种迭代优化算法，通过利用目标函数的二阶导数信息来更新模型参数，更快地收敛到最优解。在逻辑回归中，牛顿法的迭代步骤包括计算梯度、海森矩阵、利用线搜索更新模型参数。

1. 逻辑回归

   逻辑回归的目标是找到一个能够最大化似然函数（或最小化对数损失函数）的参数，从而实现二分类。模型的预测值通过 sigmoid 函数进行转换，确保预测的概率在 [0, 1] 的范围内，表示样本属于正类的概率。逻辑回归模型的表达式如下：

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

其中，$h_\theta(x)$ 是预测的概率，θ 是模型参数，x 是输入特征向量。

2. 牛顿法

牛顿法是一种迭代优化算法，用于求解无约束优化问题。其基本思想是利用目标函数的一阶导数和二阶导数信息来迭代地更新参数，以更快地收敛到最优解。

1) 牛顿法迭代步骤

给定目标函数 $f(\theta)$，牛顿法的迭代步骤如下：

1. ** 计算梯度（一阶导数）：**

$$\nabla f(\theta) = \frac{\partial f(\theta)}{\partial \theta}$$

2. ** 计算海森矩阵（二阶导数）：**

$$H(\theta) = \frac{\partial^2 f(\theta)}{\partial \theta \partial \theta^T}$$

3. ** 更新参数：**

$$\theta_{\text{new}} = \theta_{\text{old}} - H(\theta_{\text{old}})^{-1} \nabla f(\theta_{\text{old}})$$

其中，$\nabla f(\theta)$ 是梯度，$H(\theta)$ 是海森矩阵。

3. 回溯线搜索

回溯线搜索是一种确定合适步长的方法，确保在更新参数时能够满足足够的下降条件。在牛顿法中，回溯线搜索的步骤如下：

当然可以！以下是回溯线搜索的算法步骤，满足指定条件：

1. 选择初始步长 α_0，衰减率 γ 且 $\gamma \in (0,1)$，以及 c 且 $c \in (0,1)$。

2. 找到最小的正数 $t \geq 0$，使得以下条件满足：

$$f(x_k + \alpha_0 \gamma^t d_k) \leq f(x_k) + c\alpha_0 \gamma^t \nabla f(x_k)^T d_k$$

其中，$f(x_k)$ 是目标函数值，$\nabla f(x_k)$ 是梯度，d_k 是搜索方向。

3. 令 $\alpha_k = \alpha_0 \gamma^t$ 并更新参数 $x_{k+1} = x_k + \alpha_k d_k$。

4. 逻辑回归中的牛顿法在逻辑回归中，目标函数为对数损失函数：

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

其中，m 是样本数，n 是特征数，$h_\theta(x)$ 是逻辑回归的假设函数。

梯度计算：

$$\nabla J(\theta) = \frac{1}{m} X^T (\sigma(X\theta) - y) + \frac{\lambda}{m} \theta$$

其中，$\sigma(z)$ 是 sigmoid 函数，X 是包含偏置项的输入特征矩阵。

海森矩阵计算：

$$H(\theta) = \frac{1}{m} X^T W X + \frac{2\lambda}{m} I$$

其中，W 是对角矩阵，对角元素为：

$$W_{ii} = \sigma(x^{(i)}\theta) \cdot (1 - \sigma(x^{(i)}\theta))$$

回溯线搜索：

在回溯线搜索中，选择初始步长 α_0 和衰减率 γ，并找到最小的正数 t 满足以下条件：

$$J(\theta_{\text{new}}) \leq J(\theta_{\text{old}}) + \alpha_0 \gamma^t \nabla J(\theta_{\text{old}})^T (\theta_{\text{new}} - \theta_{\text{old}})$$

更新参数时，令 $\alpha_k = \alpha_0 \gamma^t$：

$$\theta_{\text{new}} = \theta_{\text{old}} - H(\theta_{\text{old}})^{-1}\nabla J(\theta_{\text{old}})$$

并更新模型参数：

$$\theta_{\text{new}} = \theta_{\text{old}} + \alpha_k(H(\theta_{\text{old}})^{-1}\nabla J(\theta_{\text{old}}))$$

其中，α_k 是经过回溯线搜索得到的步长。

3 编译环境及使用方法

- **编译环境：** Python 3.11, NumPy, Pandas,Matplotlib 等常用库
- **使用方法：**
  1. 导入必要的库和类
  2. 读取数据集（特征和标签）
  3. 创建 LogisticRegressionNewton 类的实例，设置参数；‘
  4. 在 LogisticRegressionNewton 类的实现中，采用 sigmoid 激活函数，将数据实数映射到 [0, 1] 范围内；用 hession() 函数计算 Hessian 矩阵；用 gradient() 计算梯度。在牛顿法的迭代过程中，依次执行：计算梯度，海森矩阵，更新方向向量，通过回溯线性搜索确定合适的步长，计算损失函数，即逻辑回归的负对数似然加上正则化项，判断迭代是否收敛，即小于阈值则判定为收敛，若不收敛则重复以上步骤。采用的加速技巧有：矩阵运算的向量化：使用 NumPy 进行矩阵运算，能够利用底层优化提高运算效率；算法中采用早停策略，减少计算开销；正则化：在损失函数中加入了正则化项，有助于防止过拟合。
  5. 调用 `train()` 方法训练模型
  6. 评估模型性能并可视化参数和迭代过程的关系。

4 数据集说明

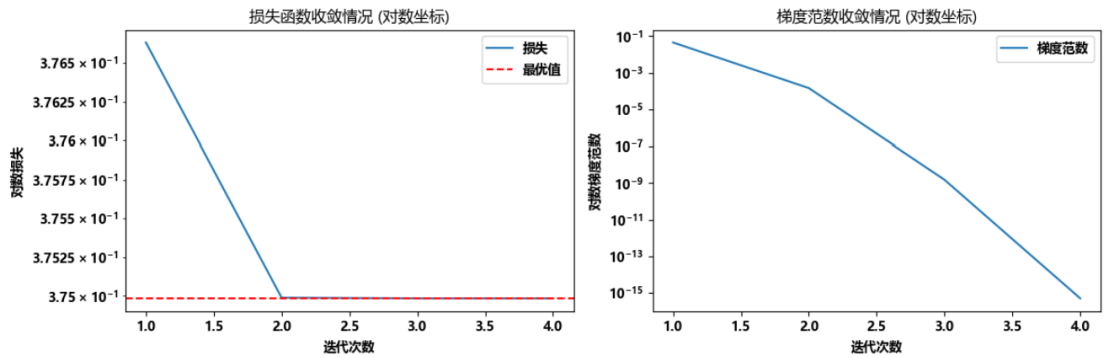
本实验使用了数据集 a9a_features.csv 和 a9a_labels.csv，其中 a9a_features.csv 包含特征信息，a9a_labels.csv 包含相应的标签信息。处理数据集，使得类别平衡，用于逻辑回归模型的训练和测试。采用的加速技巧有：

5 测试结果

在训练过程中，模型通过牛顿法迭代更新参数，最终收敛于某个解，迭代结束。

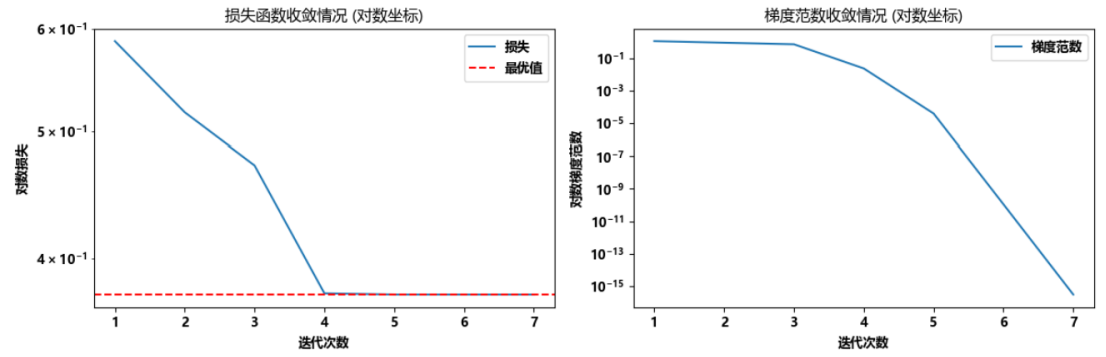
线搜索下降速率：0.1，线搜索衰减率：0.2

```
在 4 次迭代中收敛.
线搜索中的下降速率:0.1
线搜索的衰减率:0.2
```



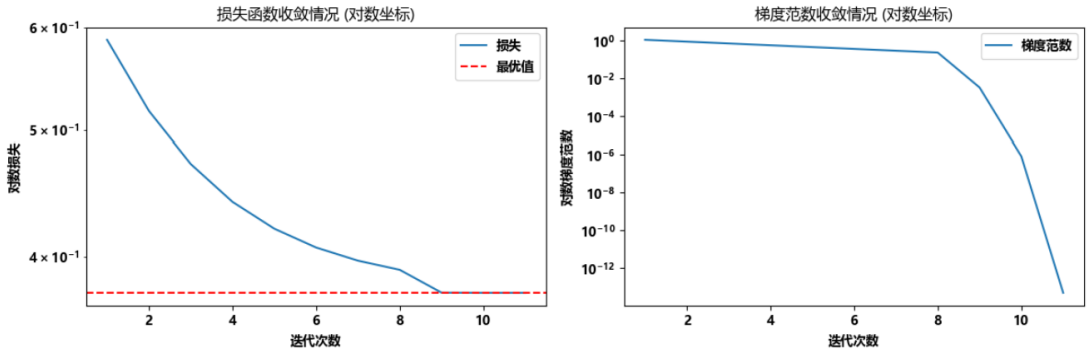
线搜索下降速率：0.6，线搜索衰减率：0.2

```
在 7 次迭代中收敛.
线搜索中的下降速率:0.6
线搜索的衰减率:0.2
```



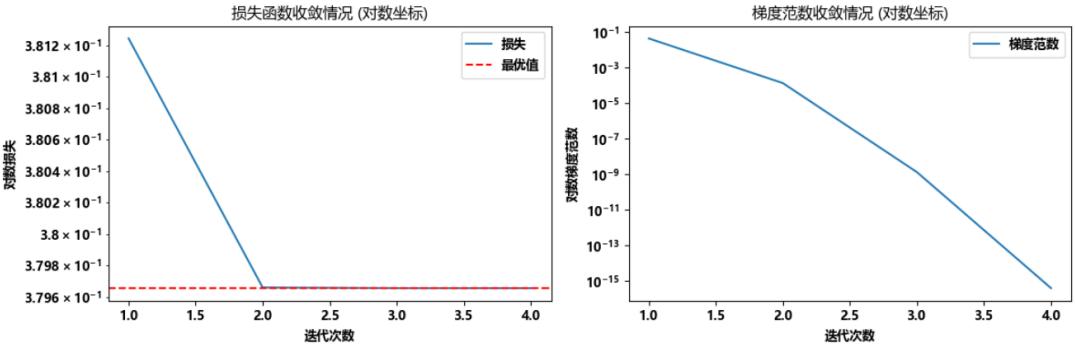
线搜索下降速率：0.8，线搜索衰减率：0.2

```
在 11 次迭代中收敛.
线搜索中的下降速率:0.8
线搜索的衰减率:0.2
```



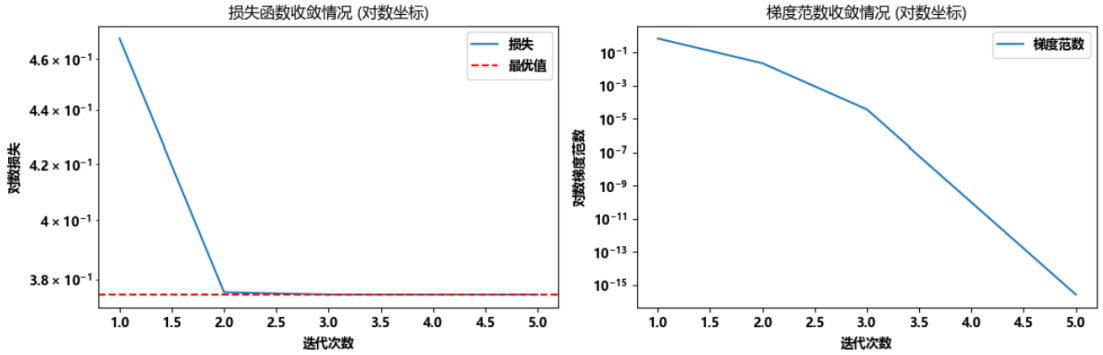
线搜索下降速率：0.1，线搜索衰减率：0.5

```
在 4 次迭代中收敛.
线搜索中的下降速率:0.1
线搜索的衰减率:0.5
```



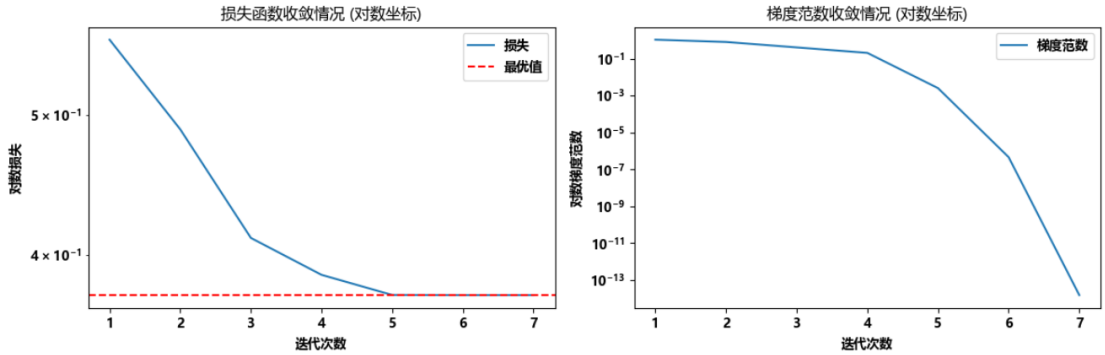
线搜索下降速率：0.6，线搜索衰减率：0.5

```
在 5 次迭代中收敛.
线搜索中的下降速率:0.6
线搜索的衰减率:0.5
```



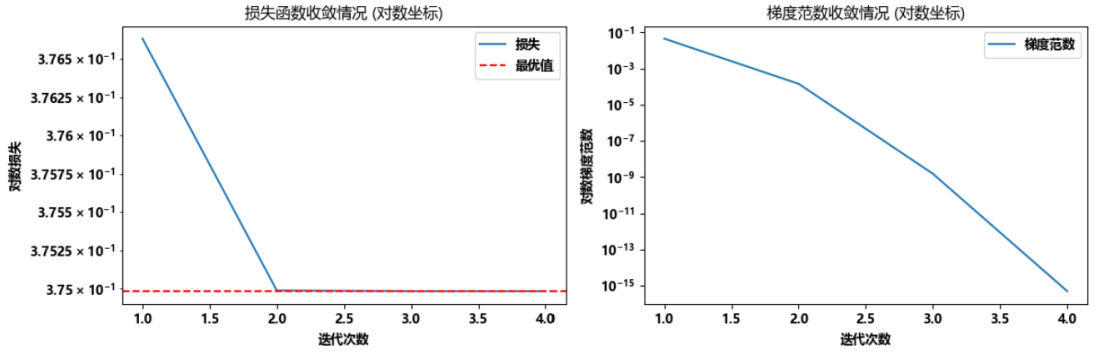
线搜索下降速率：0.8，线搜索衰减率：0.5

```
在 7 次迭代中收敛.
线搜索中的下降速率:0.8
线搜索的衰减率:0.5
```



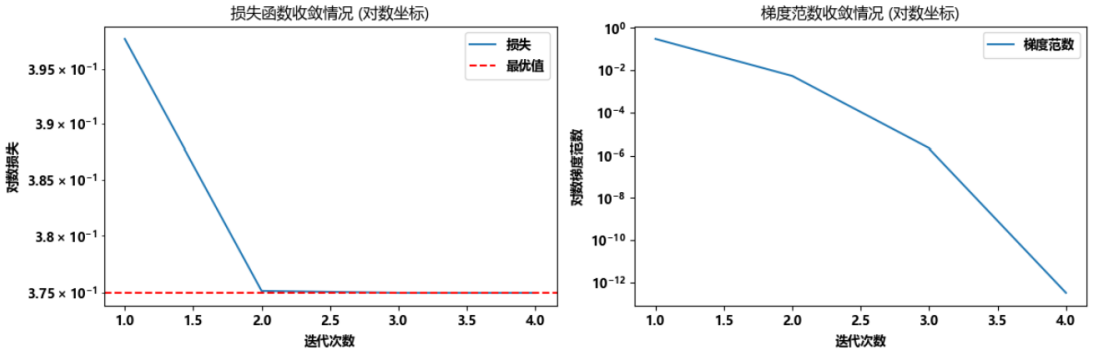
线搜索下降速率：0.1，线搜索衰减率：0.8

```
在 4 次迭代中收敛.
线搜索中的下降速率:0.1
线搜索的衰减率:0.9
```



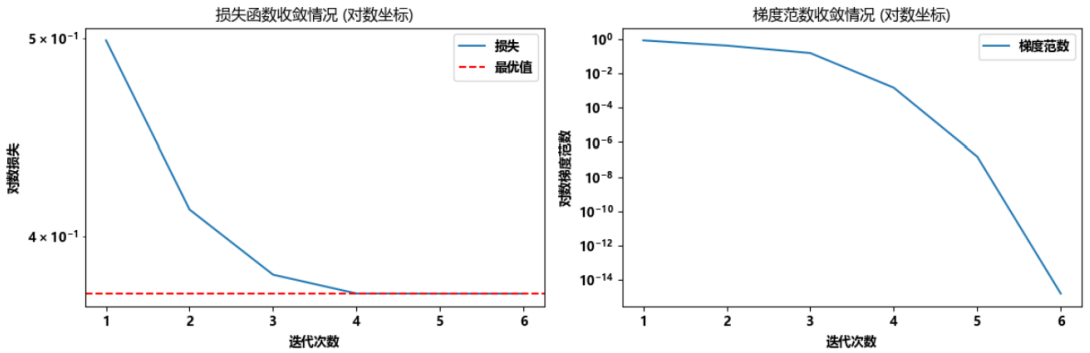
线搜索下降速率：0.6，线搜索衰减率：0.8

```
在 4 次迭代中收敛.
线搜索中的下降速率:0.6
线搜索的衰减率:0.9
```



线搜索下降速率：0.8，线搜索衰减率：0.8

```
在 6 次迭代中收敛.
线搜索中的下降速率:0.8
线搜索的衰减率:0.9
```



6 分析与总结

- 实验结果表明，牛顿法实现的逻辑回归模型在给定数据集上表现良好，收敛迅速。
- 收敛信息的可视化图表显示，损失值和梯度范数在迭代过程中逐渐减小，证明了模型的有效性。
- 根据九组参数达到收敛的迭代次数可以看出：较小的线搜索下降速率和较大的线搜索衰减率在这组数据上的收敛效果最好，牛顿法（梯度范数及损失函数）的收敛最快。一般来说，较小的线搜索下降速率会导致搜索过程变得缓慢，但可能提高稳定性。较大的线搜索下降速率可能加快搜索，但可能导致不稳定性。较小的线搜索衰减率可能导致步长减小过快，而较大的线搜索衰减率则可能导致步长减小不足。

A Computer Code

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 plt.rc("font",family='MicroSoft␣YaHei',weight="bold")
```

```
5
6  class LogisticRegressionNewton():
7      def __init__(self, X, y, alpha, beta, regularization=1.0,
           max_iter=1000, tol=1e-6):
8          # 初始化函数，包括添加偏置项，设定参数等
9          self.X = np.insert(X, 0, values=1.0, axis=1)
10         self.y = y.reshape(-1, 1)
11         self.m, self.n = self.X.shape
12         self.regularization = regularization
13         self.max_iter = max_iter
14         self.tol = tol
15         self.theta = np.zeros((self.n, 1))
16         self.loss = self.calculate_loss(self.theta)
17         self.alpha = alpha
18         self.beta = beta
19         self.loss_history = []
20         self.gradient_norm_history = []
21
22     # sigmoid激活函数
23     def sigmoid(self, z):
24         return 1.0 / (1.0 + np.exp(-z))
25
26     # 计算Hessian矩阵
27     def hessian(self):
28         diag_elements = self.sigmoid(self.X.dot(self.theta)) *
               self.sigmoid(-self.X.dot(self.theta))
29         W = np.diag(diag_elements.flatten())
30         hessian = (self.X.T @ W @ self.X) / self.m + 2 * self.
               regularization * np.identity(self.n)
31         return hessian
32
33     # 计算梯度
34     def gradient(self):
35         return self.X.T @ (self.sigmoid(self.X.dot(self.theta))
               - self.y) / self.m + 2 * self.regularization * self.
               theta
36
37     # 回溯线性搜索
```

```
38      def backtracking_line_search(self, update, alpha, beta):
39          t = 1.0
40          while True:
41              new_theta = self.theta + t * update
42              new_loss = self.calculate_loss(new_theta)
43              if new_loss < self.loss + alpha * t * self.gradient
                    ().T @ update:
44                  break
45              t *= beta
46          return t
47
48      # 牛顿法迭代步骤
49      def newton_step(self, alpha, beta):
50          update = -np.linalg.inv(self.hessian()) @ self.gradient
                ()
51          step_size = self.backtracking_line_search(update, alpha,
                beta)
52          self.theta += step_size * update
53          self.loss = self.calculate_loss(self.theta)
54
55          # 记录收敛信息
56          self.loss_history.append(self.loss)
57          gradient_norm = np.linalg.norm(self.gradient())
58          self.gradient_norm_history.append(gradient_norm)
59
60          return step_size
61
62      # 计算损失函数
63      def calculate_loss(self, theta):
64          H = self.sigmoid(self.X.dot(theta))
65          loss = -np.sum(self.y * np.log(H) + (1 - self.y) * np.
                log(1 - H)) / self.m
66          return loss + self.regularization * np.sum(theta[1:]**2)
                # 添加正则化项
67
68      # 训练模型
69      def train(self):
70          for iteration in range(self.max_iter):
```

```
71              old_theta = np.copy(self.theta)
72              step_size = self.newton_step(self.alpha, self.beta)
73              if np.linalg.norm(self.theta - old_theta) < self.tol
                    :
74                  print(f"在 {iteration+1} 次迭代中收敛.")
75                  break
76
77      # 预测函数
78      def predict(self, X):
79          X_with_bias = np.insert(X, 0, values=1.0, axis=1)
80          probabilities = self.sigmoid(X_with_bias.dot(self.theta)
                )
81          predictions = np.sign(probabilities - 0.3)  # 设置阈值为
                0.5
82          return predictions
83
84      # 绘制收敛信息图
85      def plot_convergence(self, iterations):
86          plt.figure(figsize=(12, 4))
87
88          plt.subplot(1, 2, 1)
89          plt.semilogy(iterations, self.loss_history, label='损失'
                )
90          plt.axhline(y=self.calculate_loss(self.theta), color='r'
                , linestyle='—', label='最优值')
91          plt.xlabel('迭代次数')
92          plt.ylabel('对数损失')
93          plt.legend()
94          plt.title('损失函数收敛情况 (对数坐标)')
95
96          plt.subplot(1, 2, 2)
97          plt.semilogy(iterations, self.gradient_norm_history,
                label='梯度范数')
98          plt.xlabel('迭代次数')
99          plt.ylabel('对数梯度范数')
100         plt.legend()
101         plt.title('梯度范数收敛情况 (对数坐标)')
102         plt.tight_layout()
```

```
103             plt.show()
104
105 if __name__ == "__main__":
106     # 读取数据
107     features_data = pd.read_csv('a9a_features.csv', header=None)
108     labels_data = pd.read_csv('a9a_labels.csv', header=None)
109     X = features_data.T.values
110     y = labels_data.values
111
112     # 数据处理，随机抽样保持类别平衡
113     class_0_indices = np.where(y == 1)[0]
114     class_1_indices = np.where(y == -1)[0]
115     min_samples = min(len(class_0_indices), len(class_1_indices)
            )
116     balanced_class_0_indices = np.random.choice(class_0_indices,
            min_samples, replace=False)
117     balanced_class_1_indices = np.random.choice(class_1_indices,
            min_samples, replace=False)
118     balanced_indices = np.concatenate([balanced_class_0_indices,
            balanced_class_1_indices])
119     np.random.shuffle(balanced_indices)
120     X_balanced=X[balanced_indices]
121     y_balanced=y[balanced_indices]
122
123     # 创建模型实例
124     model = LogisticRegressionNewton(X_balanced, y_balanced,
            0.1, 0.5)
125
126     # 训练模型
127     model.train()
128     print(f'线搜索中的下降速率:{0.1}')
129     print(f'线搜索的衰减率:{0.5}')
130
131     # 绘制收敛信息图
132     iterations = np.arange(1, len(model.loss_history) + 1)
133     model.plot_convergence(iterations)
```