《模式识别与机器学习A》实验报告

实验题目： 逻辑回归

学 号： 2021112845

姓 名： 张智雄

**实验报告内容**

1. **实验目的**

理解逻辑回归模型，掌握逻辑回归模型的参数估计算法。

1. **实验内容**

实现两种损失函数的参数估计（1.无惩罚项；2.加入对参数的惩罚），可以采用梯度下降、共轭梯度或者牛顿法等。

1. 可以手工生成两个分别类别数据（可以用高斯分布），验证你的算法。考察类条件分布不满足朴素贝叶斯假设，会得到什么样的结果。
2. 逻辑回归有广泛的用处，例如广告预测。可以到UCI网站上，找一实际数据加以测试。
3. **实验环境**

Windows11; Anaconda+python3.11; VS Code

1. **实验过程、结果及分析（包括代码截图、运行结果截图及必要的理论支撑等）**

**4.1 算法理论支撑**

4.1.1 逻辑回归

二项逻辑回归可用于一些简单的二分类问题，不妨设是问题的输入，是输出。则逻辑回归假设服从伯努利分布。具体模型为

其中，为模型的参数，。应用时，根据两个条件概率值的大小将实例分到概率值较大的一类。

和线性回归不同的是，逻辑回归通过函数引入了非线性因素，将实数域内的值约束到的取值范围内，从而可以较为地轻松处理二分类问题。

图形用户界面, 图表, 直方图

描述已自动生成

图1 函数

令观测函数为，于是，可以整合上述逻辑回归的概率模型得到输入集合的最大似然估计为，取对数，乘将最大值问题转化为最小值问题。

是凸的、可微的，因此可以运用梯度下降法、牛顿法及其他方法对此问题进行求解此优化问题。

4.1.2 梯度下降

梯度下降法(Gradient Descent)的基本思想是通过迭代地更新模型参数，对上述进行求导，得到其梯度各方向为

将其代入原式，最终可以化简得到。

而后更新参数，重复上述过程，即可逐步向局部最优解逼近。

4.1.3 牛顿法

牛顿法(Newton's method)则是利用目标函数的一阶导数（梯度）和二阶导数（海森矩阵）信息在每一次迭代中近似目标函数的局部曲线来进行参数更新，从而逐步逼近最优解。

应用到本问题中，上述二阶导数的泰勒展开式为

函数的极值点在时取得，于是上式可以化简为。的具体求解同上述梯度求值，而则需要对的Hessian矩阵进行求解，对于函数，其Hessian矩阵为

如果函数二阶连续可导，且，则Hessian矩阵为对称矩阵，求导顺序不影响结果，因而可以变换为如下形式：

而后更新参数，重复迭代，相较于梯度下降法，牛顿法能够更加快速的收敛到极值点。

**ALGORITHM 1** Newton's method (牛顿法)

1：**input** 目标优化函数，迭代次数，参数初值；

2：；

3：**while** **do**

4： ；

5： ；

6：

7： ；

9：**end while**

10：**return** //返回最优参数；

**4.2 实验设计**

4.2.1 随机数据生成

朴素贝叶斯算法假设特征之间是独立的，本实验中通过给定均值和协方差矩阵，使用np.random.multivariate\_normal方法生成服从多元高斯分布的数据，通过协方差矩阵是否为对角矩阵来控制特征间的独立性，即是否满足朴素贝叶斯假设。

设置正例样本标签值设为1，反例样本标签值为0，正反例样本数量相同。

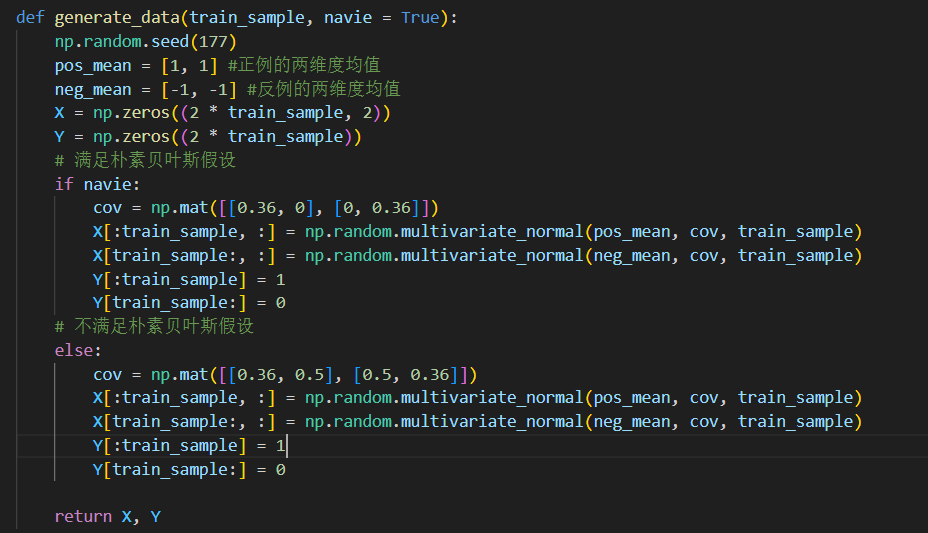


图2 数据生成代码截图

4.2.2 损失函数

根据公式计算，加入正则项后，损失函数变为，同时通过np.clip方法对预测值进行截断避免溢出。

图形用户界面, 文本

描述已自动生成

图 3 损失函数代码截图

4.2.3 梯度下降法

根据上述公式，加入正则项则原式子变为（当时退化），由此根据一定学习率迭代更新计算即可。

文本

描述已自动生成

图4 梯度下降法代码截图

4.2.4 牛顿法

梯度的计算同上，二阶Hessian矩阵通过计算，加入正则项即为。求逆则是通过np.linalg.inv方法。

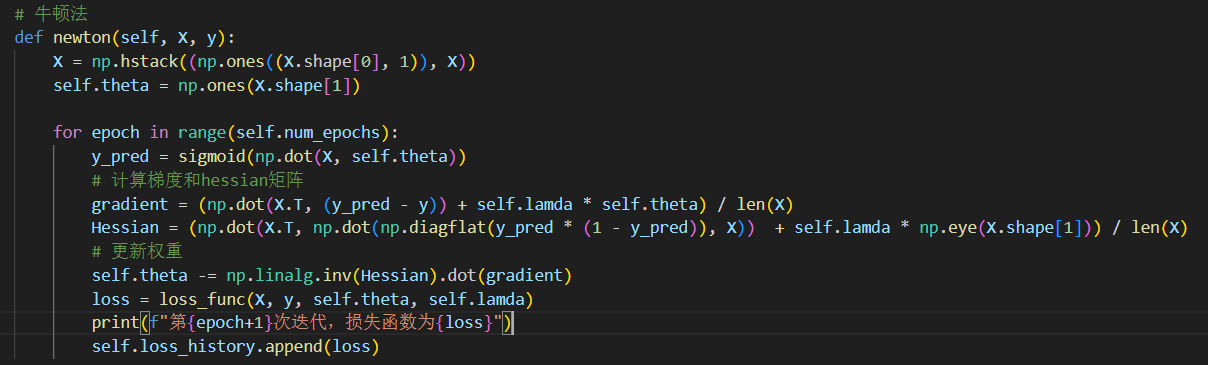


图5 牛顿法代码截图

**4.3 实验结果及分析**

4.3.1 生成数据结果

设置样本量大小为1000，分别设置协方差矩阵为（对应满足朴素贝叶斯假设）和（对应不满足朴素贝叶斯假设），按8:2的比例随机划分为训练集和测试集。

不加入正则项，分别在梯度下降法和牛顿法下进行测试，结果如下：

1. 满足朴素贝叶斯假设条件下，设置梯度下降学习率为0.1，两种方法在500轮迭代后正确率均达到97.5%，但牛顿法收敛速度远远快于梯度下降法。

图表, 散点图

描述已自动生成

图6 满足朴素贝叶斯假设回归结果

1. 不满足朴素贝叶斯假设条件下，设置梯度下降学习率为0.1，在500轮迭代后梯度下降正确率为95.5%，牛顿法正确率为94.5%，较之满足假设的情况下均有一定程度的能力减弱（当特征维度更高时减弱更明显）。

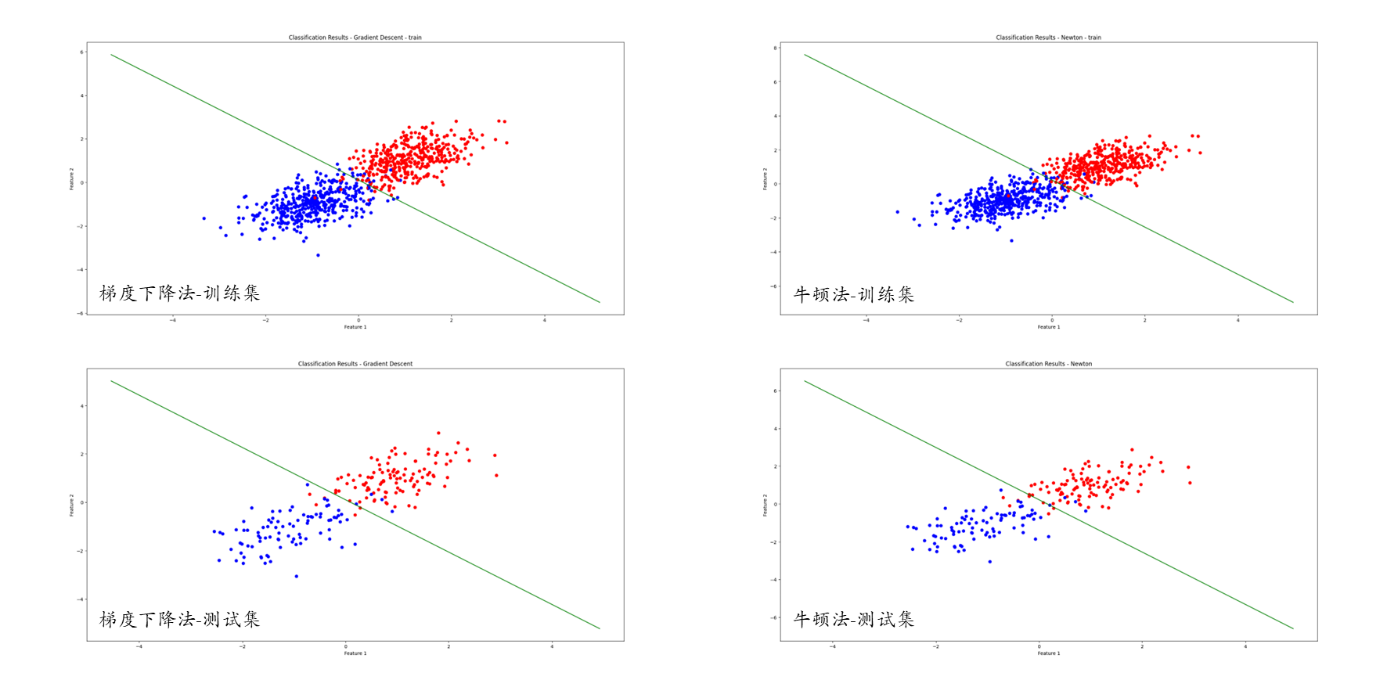


图7 不满足朴素贝叶斯假设回归结果

同时，观察损失函数下降曲线可以发现，使用牛顿法的拟合速度远远高于梯度下降法。

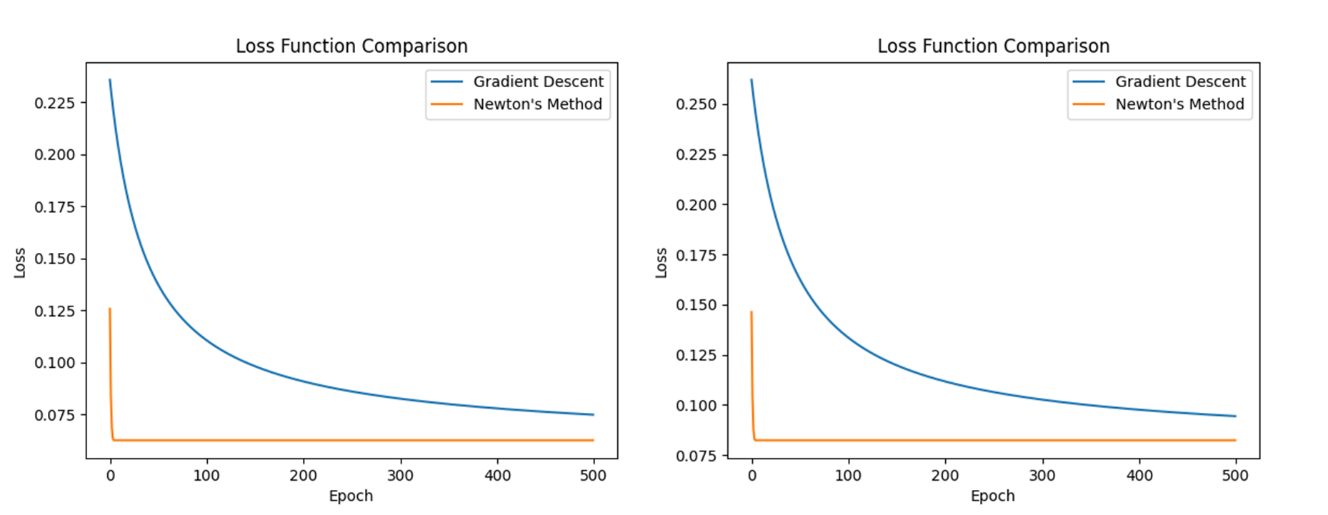


图8 损失函数对比（左为满足假设，右为不满足假设）

而后修改正则项系数，在满足朴素贝叶斯假设的条件下，设置样本量为100，分别在梯度下降法和牛顿法下进行测试，结果如下：

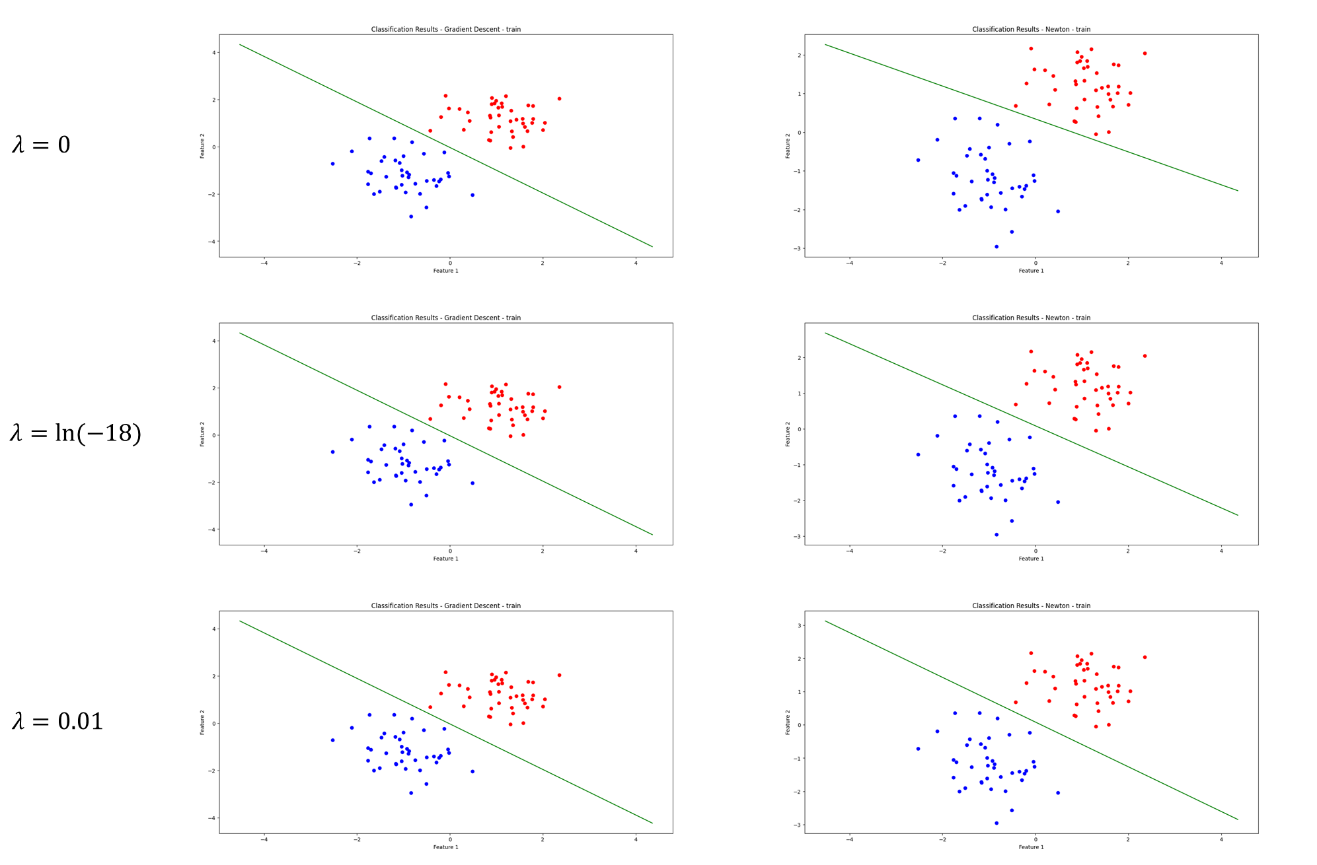


图9 正则项变化下结果对比（满足朴素贝叶斯假设条件）

可以看到，在相同的数据下，回归的正确率没有变化，但牛顿法的决策面随的改变有一定程度的偏移，直观角度可以看到加入正则惩罚项后决策面更为合理；而梯度下降法可能是由于还未收敛的缘故，几乎无变化。

在低维条件下，是否有正则惩罚项对其影响也不大，但当减小训练集时，所得到的判别函数确实存在过拟合现象，加入正则项可以预防此现象的发生。

4.3.2 UCI-Iris数据集结果

选用UCI网站上的Iris数据集，鸢尾花数据集(Iris dataset)包含了来自三个不同种类(Iris setosa, Iris versicolor, Iris virginica)的鸢尾花(Iris)的观测数据，每个种类有50个样本。每个样本由包括花萼长度(sepal length)、花萼宽度(sepal width)、花瓣长度(petal length)和花瓣宽度(petal width)四个特征描述。

实验保留了Iris-setosa和Iris-versicolor两个类别的100组数据进行二项逻辑回归。首先按照8:2的比例对数据进行训练集和测试集的划分，而后分别使用梯度下降法和牛顿法对训练集进行近似。

实验观察损失函数下降的曲线发现，选取学习率为0.001时，梯度下降的效果较为明显。而牛顿法使用了二阶导数的信息来确定参数更新的方向和步长，可以根据损失函数的曲率来自适应地调整步长，因此不需要学习率来控制收敛速度。

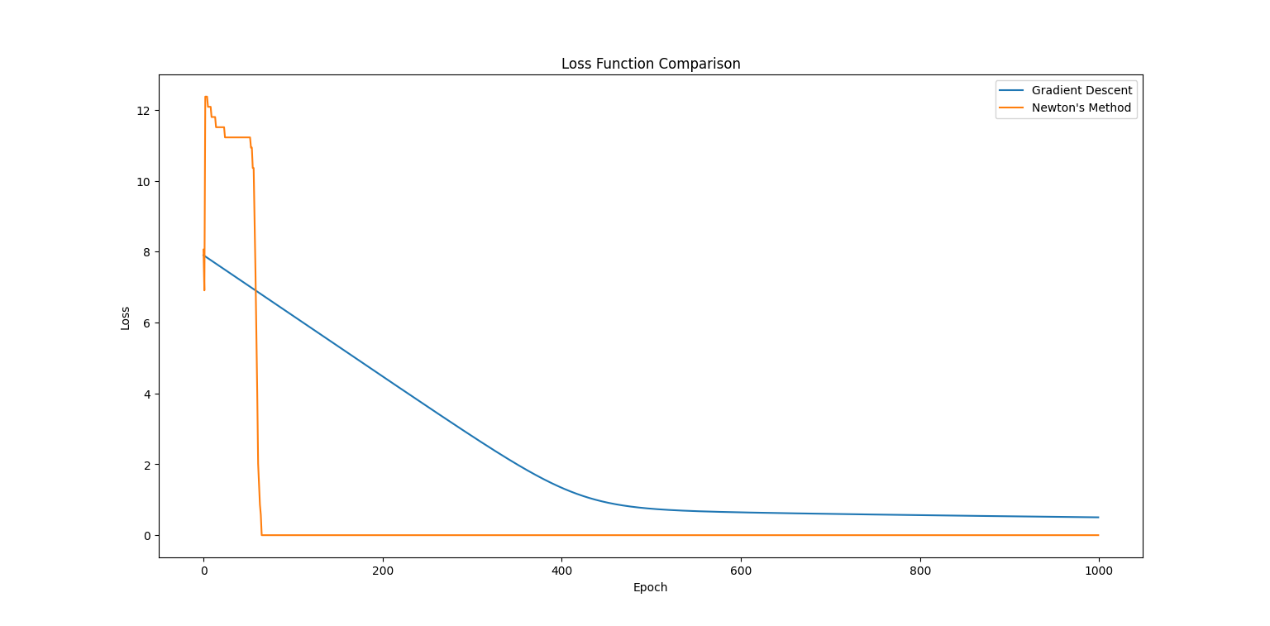


图10 损失函数随迭代次数的变化（蓝线为梯度下降，橙色为牛顿法）

从图像中显而易见可以看到，牛顿法收敛的速度相较于梯度下降更快，在迭代次数为1000时，梯度下降的损失函数仍大于牛顿法。

观察曲线还发现，使用牛顿法初期还存在不稳定的情况，这可能是由于初始参数选择的不好的缘故。而其损失函数变化曲线呈阶梯式的原因可能是数据集较小，参数已经接近最优解，梯度变得非常小。

观察损失函数分析使用牛顿法可能存在过拟合现象，加入正则项后效果未明显改进，由于特征具有4维直观观察也较为困难。

观察分类的具体结果（选取第1、2个特征进行可视化，由于具有多个特征，图中未画出决策面），发现分类结果基本一致，决策面直观上差距不大，符合两种方法均对梯度方向进行收敛的论断。

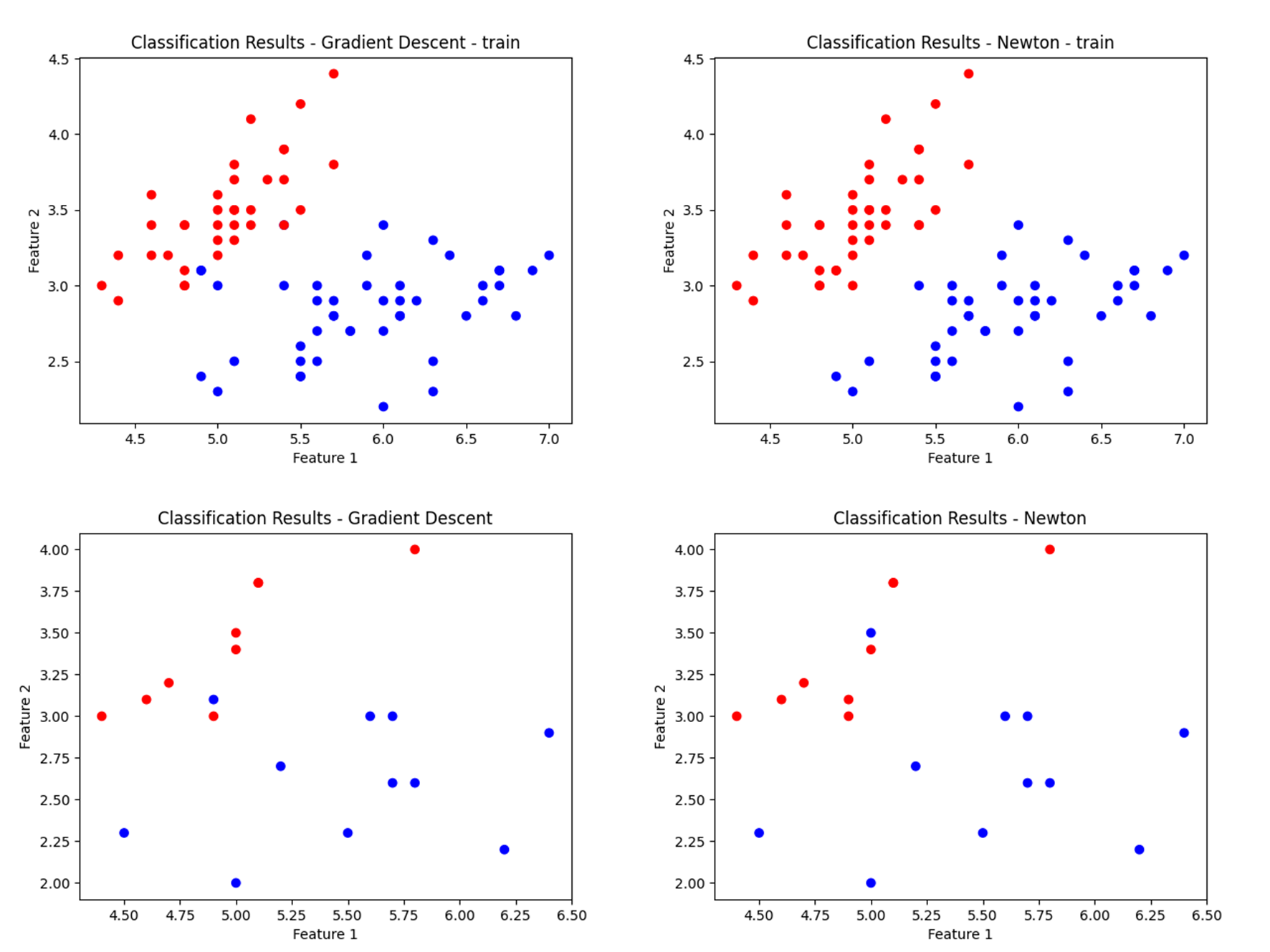


图11 两种方法逻辑回归结果（）

4.3.2 UCI-Dry\_Beans数据集结果

Dry-Beans数据集是Kaggle网站上的关于菜豆的一个数据集，包括菜豆的区域，周长，长轴长、短轴长等指标，在本节中选取SEKER和BARBUNYA两个品种（3349组）。由于数据量太多，此问题中牛顿法求逆较为困难，因而只采用梯度下降法进行分析。

由于有15个特征，全部选择进行逻辑回归得到结果，为便于可视化同时具有较好的回归效果，选择MajorAxisLength和MinorAxisLength两项指标对菜豆进行二项逻辑回归分析，同样选择20%的数据作为测试集。

设置学习率为0.01，迭代次数1000，观察最后分类结果，正确率为93.73%，在图中也能较为直观的找到决策面，具体预测结果如下：

图表, 散点图

描述已自动生成

图12 菜豆数据集逻辑回归结果（梯度下降法）

同时观察损失函数变化曲线，可以发现出现明显振荡现象，原因有可能是学习率为0.01的步长较大，函数在极值点附近反复逼近。

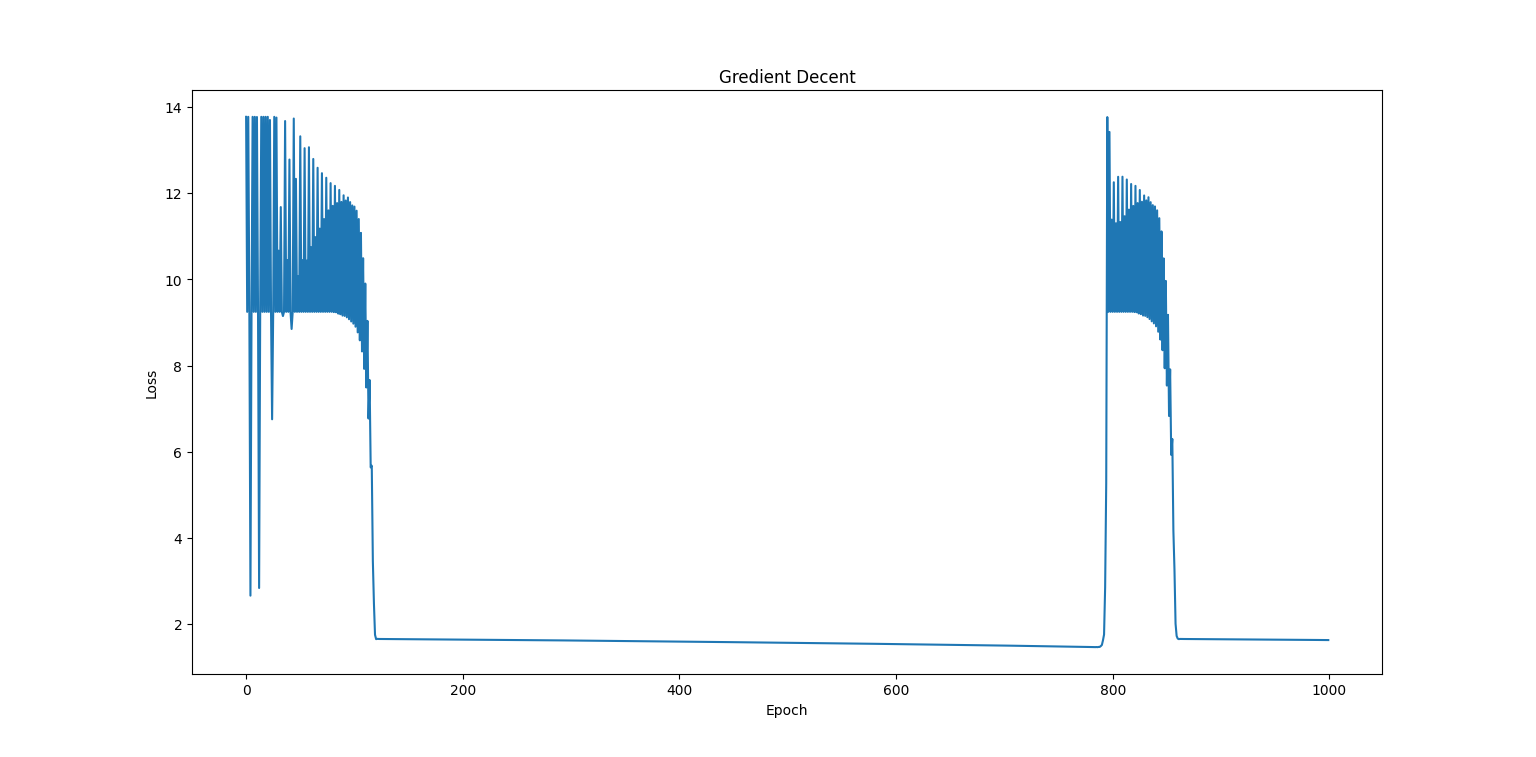


图13 损失函数下降曲线

1. **实验结论**

牛顿法使用了二阶导数的信息来确定参数更新的方向和步长，可以根据损失函数的曲率来自适应地调整步长，因此不需要学习率来控制收敛速度。但由于使用了二阶导数的信息，对数据样本的依赖更强，因而鲁棒性弱于梯度法。

牛顿法相比梯度下降法，每次的时间开销和空间占用会更大但是迭代收敛的速度较快。但是牛顿法的计算过程中涉及求黑塞矩阵的逆，如果矩阵奇异，则牛顿法不再适用。但使用np.linalg.pinv求伪逆作为替代在一定场合下不适用，可以考虑使用拟牛顿法，如BFGS算法来解决此问题。

sigmoid函数可能会发生溢出，主要是当或时，会发生溢出，可以选用np.clip方法约束限制。

逻辑回归的前提是数据满足朴素贝叶斯假设：特征的各维独立、类条件分布满足高斯分布、各类别协方差矩阵相等(与类别无关)；当数据不满足朴素贝叶斯假设时，分类效果可能会变差。

加入惩罚项在本实验中没有显著作用，原因可能是因为梯度下降法不容易收敛，同时数据量较少，在多维特征做逻辑回归时效果可能更为明显。

1. **完整实验代码**

regression.py

|  |
| --- |
| 1. import pandas as pd 2. import numpy as np 3. import matplotlib.pyplot as plt 4. import math 5. from sklearn.model\_selection import train\_test\_split 6. def sigmoid(x): 7. limit\_x = np.clip(x, -500, 500) 8. return 1 / (1 + np.exp(-limit\_x)) 9. def loss\_func(X, y, theta, lamda): 10. m = len(y) 11. epsilon = 1e-10 12. y\_pred = sigmoid(np.dot(X, theta)) 13. y\_pred = np.clip(y\_pred, epsilon, 1 - epsilon) 14. loss = -np.sum(y \* np.log(y\_pred) + (1 - y) \* np.log(1 - y\_pred)) / m + lamda \* np.sum((theta) \*\* 2) / (2 \* m) 15. return loss 16. *# 定义二项逻辑回归模型* 17. class LogisticRegression: 18. def \_\_init\_\_(self, learning\_rate, num\_epochs, lamda=0): 19. self.learning\_rate = learning\_rate 20. self.num\_epochs = num\_epochs 21. self.theta = None 22. self.loss\_history = [] 23. self.lamda = lamda 24. *# 梯度下降* 25. def gredient\_decent(self, X, y): 26. X = np.hstack((np.ones((X.shape[0], 1)), X)) 27. self.theta = np.ones(X.shape[1]) 28. for epoch in range(self.num\_epochs): 29. y\_pred = sigmoid(np.dot(X, self.theta)) 30. *# 计算梯度* 31. gradient = (np.dot(X.T, (y\_pred - y)) + self.lamda \* self.theta) / len(X) 32. *# 更新权重* 33. self.theta -= self.learning\_rate \* gradient 34. loss = loss\_func(X, y, self.theta, self.lamda) 35. print(f"第{epoch+1}次迭代，损失函数为{loss}") 36. self.loss\_history.append(loss) 37. *# 牛顿法* 38. def newton(self, X, y): 39. X = np.hstack((np.ones((X.shape[0], 1)), X)) 40. self.theta = np.ones(X.shape[1]) 41. for epoch in range(self.num\_epochs): 42. y\_pred = sigmoid(np.dot(X, self.theta)) 43. *# 计算梯度和hessian矩阵* 44. gradient = (np.dot(X.T, (y\_pred - y)) + self.lamda \* self.theta) / len(X) 45. Hessian = (np.dot(X.T, np.dot(np.diagflat(y\_pred \* (1 - y\_pred)), X))  + self.lamda \* np.eye(X.shape[1])) / len(X) 46. *# 更新权重* 47. self.theta -= np.linalg.pinv(Hessian).dot(gradient) 48. loss = loss\_func(X, y, self.theta, self.lamda) 49. print(f"第{epoch+1}次迭代，损失函数为{loss}") 50. self.loss\_history.append(loss) 51. def predict(self, X): 52. X = np.hstack((np.ones((X.shape[0], 1)), X)) 53. y\_pred = sigmoid(np.dot(X, self.theta)) 54. y\_pred = np.round(y\_pred) 55. return y\_pred 56. def draw\_loss(model, str): 57. plt.plot(model.loss\_history) 58. plt.xlabel('Epoch') 59. plt.ylabel('Loss') 60. plt.title(str) 61. plt.show() 62. def draw\_result(X, y, theta, str): 63. plt.scatter(X[:, 0], X[:, 1], c=y, cmap='bwr') 64. plot\_x = np.array([min(X[:, 0]) - 2, max(X[:, 0]) + 2]) 65. plot\_y = - (theta[0] + theta[1] \* plot\_x) / theta[2] 66. plt.plot(plot\_x, plot\_y, label='Decision Boundary', color='green') 67. plt.xlabel('Feature 1') 68. plt.ylabel('Feature 2') 69. plt.title(f'Classification Results - {str}') 70. plt.show() 71. def draw\_compare(model\_1, model\_2): 72. plt.plot(model\_1.loss\_history, label="Gradient Descent") 73. plt.plot(model\_2.loss\_history, label="Newton's Method") 74. plt.xlabel('Epoch') 75. plt.ylabel('Loss') 76. plt.title('Loss Function Comparison') 77. plt.legend() 78. plt.show() 79. def generate\_data(train\_sample, navie = True): 80. np.random.seed(177) 81. pos\_mean = [1, 1] *#正例的两维度均值* 82. neg\_mean = [-1, -1] *#反例的两维度均值* 83. X = np.zeros((2 \* train\_sample, 2)) 84. Y = np.zeros((2 \* train\_sample)) 85. *# 满足朴素贝叶斯假设* 86. if navie: 87. cov = np.mat([[0.49, 0], [0, 0.49]]) 88. X[:train\_sample, :] = np.random.multivariate\_normal(pos\_mean, cov, train\_sample) 89. X[train\_sample:, :] = np.random.multivariate\_normal(neg\_mean, cov, train\_sample) 90. Y[:train\_sample] = 1 91. Y[train\_sample:] = 0 92. *# 不满足朴素贝叶斯假设* 93. else: 94. cov = np.mat([[0.49, 0.25], [0.25, 0.49]]) 95. X[:train\_sample, :] = np.random.multivariate\_normal(pos\_mean, cov, train\_sample) 96. X[train\_sample:, :] = np.random.multivariate\_normal(neg\_mean, cov, train\_sample) 97. Y[:train\_sample] = 1 98. Y[train\_sample:] = 0 99. return X, Y 100. *# 自己生成的数据集进行二项分布回归* 101. def main(): 102. train\_sample = 500 103. X, Y = generate\_data(train\_sample, navie = True) 104. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=177) 105. print("梯度下降法") 106. print("======================================") 107. model\_1 = LogisticRegression(learning\_rate=0.1, num\_epochs=1000, lamda=math.exp(-18)) 108. model\_1.gredient\_decent(X\_train, y\_train) 109. draw\_loss(model\_1,'Gredient Decent') 110. *# 在测试集上验证正确率* 111. y\_pred\_test\_1 = model\_1.predict(X\_test) 112. accuracy\_1 = np.mean(y\_pred\_test\_1 == y\_test) 113. print(f"梯度下降法准确率为{accuracy\_1 \* 100}%") 114. draw\_result(X\_train, y\_train, model\_1.theta, 'Gradient Descent - train') 115. draw\_result(X\_test, y\_test, model\_1.theta, 'Gradient Descent') 116. *# ============================================================= #* 117. print("牛顿法") 118. print("======================================") 119. model\_2 = LogisticRegression(learning\_rate=0.001, num\_epochs=1000, lamda=math.exp(-18)) 120. model\_2.newton(X\_train, y\_train) 121. draw\_loss(model\_2, 'NewTons method') 122. *# 在测试集上验证正确率* 123. y\_pred\_test\_2 = model\_2.predict(X\_test) 124. accuracy\_2 = np.mean(y\_pred\_test\_2 == y\_test) 125. print(f"牛顿法准确率为{accuracy\_2 \* 100}%") 126. draw\_result(X\_train, y\_train, model\_2.theta, 'Newton - train') 127. draw\_result(X\_test, y\_test, model\_2.theta, 'Newton') 128. *# 对比* 129. draw\_compare(model\_1, model\_2) 130. *# dry-beans数据集上的二项回归（梯度下降）* 131. def dry\_beans(): 132. data = pd.read\_excel('./Dry\_Bean\_Dataset.xlsx') 133. data["Class"] = data["Class"].map({"SEKER": 1, "BARBUNYA": 0}) 134. data = np.array(data[:3349]) 135. characters = data[:, 2:4].astype(float) 136. labels = data[:, -1].astype(float) 137. X\_train, X\_test, y\_train, y\_test = train\_test\_split(characters, labels, test\_size=0.2, random\_state=10) 138. print("梯度下降法") 139. print("======================================") 140. model\_1 = LogisticRegression(learning\_rate=0.01, num\_epochs=1000, lamda=0.01) 141. model\_1.gredient\_decent(X\_train, y\_train) 142. draw\_loss(model\_1,'Gredient Decent') 143. *# 在测试集上验证正确率* 144. y\_pred\_test\_1 = model\_1.predict(X\_test) 145. accuracy\_1 = np.mean(y\_pred\_test\_1 == y\_test) 146. print(f"梯度下降法准确率为{accuracy\_1 \* 100}%") 147. draw\_result(X\_train, y\_train, model\_1.theta, 'Gradient Descent - train') 148. draw\_result(X\_test, y\_test, model\_1.theta, 'Gradient Descent') 149. *# iris数据集上的二项回归（包含牛顿法和梯度下降法）* 150. def iris(): 151. data = pd.read\_csv('./Iris.data') 152. data["Species"] = data["Species"].map({"Iris-setosa": 1, "Iris-versicolor": 0}) 153. data = np.array(data[:100].values) 154. characters = data[:, :-1].astype(float) 155. labels = data[:, -1].astype(float) 156. X\_train, X\_test, y\_train, y\_test = train\_test\_split(characters, labels, test\_size=0.2, random\_state=10) 157. print("梯度下降法") 158. print("======================================") 159. model\_1 = LogisticRegression(learning\_rate=0.001, num\_epochs=1000) 160. model\_1.gredient\_decent(X\_train, y\_train) 161. *# 在测试集上验证正确率* 162. y\_pred\_test\_1 = model\_1.predict(X\_test) 163. accuracy\_1 = np.mean(y\_pred\_test\_1 == y\_test) 164. print(f"梯度下降法准确率为{accuracy\_1 \* 100}%") 165. draw\_result(X\_train, model\_1.predict(X\_train), model\_1.theta, 'Gradient Descent - train') 166. draw\_result(X\_test, y\_pred\_test\_1, model\_1.theta, 'Gradient Descent') 167. *# ============================================================= #* 168. print("牛顿法") 169. print("======================================") 170. *# 牛顿法lr无意义，方便写程序就没单独创个类* 171. model\_2 = LogisticRegression(learning\_rate=0.001, num\_epochs=100, lamda=0) 172. model\_2.newton(X\_train, y\_train) 173. print('\n') 174. *# 在测试集上验证正确率* 175. y\_pred\_test\_2 = model\_2.predict(X\_test) 176. accuracy\_2 = np.mean(y\_pred\_test\_2 == y\_test) 177. print(f"牛顿法准确率为{accuracy\_2 \* 100}%") 178. draw\_result(X\_train,  model\_2.predict(X\_train), model\_2.theta, 'Newton - train') 179. draw\_result(X\_test, y\_pred\_test\_2, model\_2.theta, 'Newton') 180. *# 对比* 181. draw\_compare(model\_1, model\_2) 182. if \_\_name\_\_ == '\_\_main\_\_': 183. iris() 184. dry\_beans() 185. main() |

1. **参考文献**

[1] 李航. 统计学习方法[M]. 清华大学出版社, 2012.

[2] 周志华. 机器学习[M]. 清华大学出版社, 2016.

[3] 模式识别——牛顿法实现逻辑回归https://zhuanlan.zhihu.com/p/63305895

[4] 机器学习(超详细讲解)：利用Logistic Regression(逻辑回归)实现多分类https:// blog.csdn.net/qq\_44350242/article/details/112372860