**《机器学习基础》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | |  | | | **姓名** |  |
| **实验题目** | **对数几率回归算法实践** | | | | | |
| **实验时间** | **2020/11/28** | | **实验地点** | **DS3 303** | | |
| **实验成绩** |  | | **实验性质** | **□验证性 ☑设计性 □综合性** | | |
| 教师评价：  **□**算法/实验过程正确； **□**源程序/实验内容提交 **□**程序结构/实验步骤合理；  **□**实验结果正确； **□**语法、语义正确； **□**报告规范；  其他：  评价教师签名： | | | | | | |
| 一、实验目的  掌握线性模型、对率回归算法原理。 | | | | | | |
| 二、实验项目内容  1. 理解对率回归算法原理。 2. 编程实现对数几率回归算法。 3. 将算法应用于西瓜数据集、鸢尾花数据集分类问题。 | | | | | | |
| 三、实验过程或算法（源程序）  **1. 对率回归模型构建**  **（1）sigmoid函数的定义为**  分类边界定义为  则逻辑回归模型函数构建为  使用均方差作为代价函数，得  使用对数极大似然法，对代价函数进行变换得损失函数  问题转换为对J(θ)求最小值，求得参数θ，即得到训练的模型。  **（2）使用梯度下降法对损失函数进行优化**  令J(θ)对θi求偏导，得  最终化简得  所以梯度下降的迭代过程为  **2. 基于上述模型推导，在Python中实现模型**  **（1）sigmoid函数的定义为**   1. # sigmoid函数的定义 2. **def** sigmoid(z): 3. **return** 1.0 / (1.0 + np.exp(-z))   **（2）损失函数J的定义为**   1. # 损失函数J的定义 2. **def** J(theta, array\_x, array\_y, my\_lambda=0): 3. m, n = array\_x.shape  # X的尺寸 4. h = sigmoid(np.dot(array\_x, theta))  # 调用sigmoid函数求h 5. J = (-1.0 / m) \* (np.dot(np.log(h).T, array\_y) + np.dot(np.log(1 - h).T, 1 - array\_y)) + (my\_lambda / (2.0 \* m)) \* np.sum(np.square(theta[1:])) 6. **if** np.isnan(J[0]): 7. **return** np.inf  # J为空的特殊情况返回无穷大 8. **return** J.flatten()[0]  # 否则返回第一项的值即为求出的损失函数值   **（3）使用梯度下降法对损失函数进行最优化**   1. # 梯度下降法优化损失函数的核心算法 2. **def** gradient(array\_x, array\_y, test\_X, test\_Y, alpha, epsilon, max\_loop): 3. m = len(array\_x) 4. n = 3 5. # 矩阵尺寸的定义 6. theta = np.zeros((n, 1)) 7. cost = J(theta, array\_x, array\_y)  # 求当前损失函数的值 8. costs = [cost] 9. thetas = [theta] 10. accuracy = 0.0 11. accuracies = [accuracy] 12. my\_Lambda = float(0)  # 需将拉姆达定义为浮点数，否则除法运算为0 13. count = 0 14. **while** count < max\_loop:  # 对每次循环求损失，并用测试集求预测准确率 15. h = sigmoid(np.dot(array\_x, theta)) 16. theta = theta - alpha \* ((1.0 / m) \* np.dot(array\_x.T, (h - array\_y)) + (my\_Lambda / m) \* np.r\_[[[0]], theta[1:]]) 17. thetas.append(theta) 18. cost = J(theta, array\_x, array\_y, int(my\_Lambda)) 19. costs.append(cost) 21. testPredict = predict(test\_X, theta) 22. accuracy = accuracy\_rate(test\_Y, testPredict) 23. accuracies.append(accuracy) 25. **if** abs(costs[-1] - costs[-2]) < epsilon: 26. **break** 27. count += 1 28. **return** thetas, costs, count, accuracies   **（4）使用上述模型解决二分类问题**  使用如上的对率回归算法进行二分类问题的解决。对输入的数据集，分为数据x和标签y（数据的归类）。首先训练模型得到参数后，将线性回归模型带入，求得预测值，然后利用单位阶跃函数，转化为0-1值，完成二分类。然后评估预测的正确率。  本次实验中我使用了西瓜数据集3a进行对率回归算法二分类模型的训练和评估。  其中的训练集、测试集数据划分，我使用了自助法的取样方法，原因是因为西瓜数据集3a中的样本量比较少，如果进行随机划分的话可能出现欠拟合的问题。  **（5）使用上述模型解决多分类问题**  基于如上二分类问题的思想，使用一对其余的方法进行多分类问题的求解。本次实验中我使用的是鸢尾花数据集进行多分类模型的训练和评估，是将数据集进行三次预处理，分别训练出三个二分类器，再将这三个二分类器的输出进行投票选择，得出最终的三分类结果。  鸢尾花数据集的样本数量相对校对，在进行训练集与测试集的划分的时候，我使用了随机划分的方法，按照8:2的比例进行分层抽样，最后形成训练集与数据集，训练的效果比较准确。  鸢尾花数据集需要进行数据的预处理，因为鸢尾花数据集的标签值不是数字，通过将标签转换为0和1的数字，形成可用的训练集与测试集。  **（6）模型代码以及二分类、多分类代码如下：**  **模型：logistic.py**   1. **import** numpy as np  4. # sigmoid函数的定义 5. **def** sigmoid(z): 6. **return** 1.0 / (1.0 + np.exp(-z))  9. # 损失函数J的定义 10. **def** J(theta, array\_x, array\_y, my\_lambda=0): 11. m, n = array\_x.shape  # X的尺寸 12. h = sigmoid(np.dot(array\_x, theta))  # 调用sigmoid函数求h 13. J = (-1.0 / m) \* (np.dot(np.log(h).T, array\_y) + np.dot(np.log(1 - h).T, 1 - array\_y)) + (my\_lambda / (2.0 \* m)) \* np.sum(np.square(theta[1:])) 14. **if** np.isnan(J[0]): 15. **return** np.inf  # J为空的特殊情况返回无穷大 16. **return** J.flatten()[0]  # 否则返回第一项的值即为求出的损失函数值  19. # 预测根据X对其标记进行预测 20. **def** predict(array\_x, theta): 21. predictResult = sigmoid(np.dot(array\_x, theta)) 22. predictResult[predictResult >= 0.5] = 1  # 根据sigmoid函数定义与性质，大于0.5即预测为1 23. predictResult[predictResult < 0.5] = 0  # 否则预测为0 24. **return** predictResult  27. # 对标记值与预测值进行比较，求出预测准确率 28. **def** accuracy\_rate(array\_y, pre\_y): 29. n = len(pre\_y) 30. accNum = 0.0 31. **for** i **in** range(n): 32. **if** pre\_y[i] == array\_y[i]: 33. accNum = accNum + 1 34. **return** accNum / float(n)  37. # 梯度下降法优化损失函数的核心算法 38. **def** gradient(array\_x, array\_y, test\_X, test\_Y, alpha, epsilon, max\_loop): 39. m = len(array\_x) 40. n = 3 41. # 矩阵尺寸的定义 42. theta = np.zeros((n, 1)) 43. cost = J(theta, array\_x, array\_y)  # 求当前损失函数的值 44. costs = [cost] 45. thetas = [theta] 46. accuracy = 0.0 47. accuracies = [accuracy] 48. my\_Lambda = float(0)  # 需将拉姆达定义为浮点数，否则除法运算为0 49. count = 0 50. **while** count < max\_loop:  # 对每次循环求损失，并用测试集求预测准确率 51. h = sigmoid(np.dot(array\_x, theta)) 52. theta = theta - alpha \* ((1.0 / m) \* np.dot(array\_x.T, (h - array\_y)) + (my\_Lambda / m) \* np.r\_[[[0]], theta[1:]]) 53. thetas.append(theta) 54. cost = J(theta, array\_x, array\_y, int(my\_Lambda)) 55. costs.append(cost) 57. testPredict = predict(test\_X, theta) 58. accuracy = accuracy\_rate(test\_Y, testPredict) 59. accuracies.append(accuracy) 61. **if** abs(costs[-1] - costs[-2]) < epsilon: 62. **break** 63. count += 1 64. **return** thetas, costs, count, accuracies  67. # 多分类时无需在每个迭代中求预测准确率，故另外定义一个梯度下降函数 68. **def** gradient2(array\_x, array\_y, alpha, epsilon, max\_loop): 69. m = len(array\_x) 70. n = 5 71. # 初始化模型参数，n个特征对应n个参数 72. theta = np.zeros((n, 1)) 73. cost = J(theta, array\_x, array\_y)  # 当前代价 74. costs = [cost] 75. thetas = [theta] 76. my\_lambda = float(0) 77. count = 0 78. **while** count < max\_loop: 79. h = sigmoid(np.dot(array\_x, theta)) 80. theta = theta - alpha \* ((1.0 / m) \* np.dot(array\_x.T, (h - array\_y)) + (my\_lambda / m) \* np.r\_[[[0]], theta[1:]]) 81. thetas.append(theta) 82. cost = J(theta, array\_x, array\_y, int(my\_lambda)) 83. costs.append(cost) 84. **if** abs(costs[-1] - costs[-2]) < epsilon: 85. **break** 86. count += 1 87. **return** thetas, costs, count   **二分类：watermelon\_logistic.py:**   1. **import** numpy as np 2. **import** pandas as pd 3. **import** matplotlib.pyplot as plt 4. **import** Logistic  7. **class** WatermelonLogistic: 8. data = [] 9. dataX = [] 10. dataY = [] 11. data\_train\_X = [] 12. data\_train\_Y = [] 13. data\_test\_X = [] 14. data\_test\_Y = [] 16. **def** \_\_init\_\_(self): 17. self.data\_get() 18. self.data\_split() 19. self.model\_train(0.05, 0.00000001, 100000) 21. **def** data\_get(self): 22. self.data = np.array(pd.read\_csv('data/watermelon\_3a.csv')) 23. self.dataX = self.data[:, np.arange(1, 3)] 24. self.dataY = np.transpose([self.data[:, 3]]) 26. **def** data\_split(self): 27. # 由于数据集很小，所以使用自助法，通过随机抽样的方式划分训练集与测试集 28. # 通过产生的随机数获得抽取样本的序号 29. bootstrapping = [] 30. **for** i **in** range(len(self.data)): 31. bootstrapping.append(np.floor(np.random.random() \* len(self.data))) 32. # 通过序号获得原始数据集中的数据 33. D\_1 = []  # 训练集 34. **for** i **in** range(len(self.data)): 35. D\_1.append(self.data[int(bootstrapping[i])]) 37. l = []  # 用l存储a中b的每一行的索引位置 38. **for** i **in** range(len(np.array(D\_1))): 39. **for** j **in** range(len(self.data)): 40. **if** list(self.data[j]) == list(np.array(D\_1)[i]):  # op.eq比较两个list，相同返回Ture 41. l.append(j) 42. # delete函数删除数据集中对应行 43. D\_2 = np.delete(self.data, l, axis=0) 44. self.data\_train\_X = np.array(D\_1)[:, np.arange(1, 3)] 45. self.data\_train\_Y = np.transpose([np.array(D\_1)[:, 3]]) 46. self.data\_test\_X = np.array(D\_2)[:, np.arange(1, 3)] 47. self.data\_test\_Y = np.transpose([np.array(D\_2)[:, 3]]) 49. **def** model\_train(self, alpha, epsilon, maxloop): 50. # 训练模型 51. m = len(self.data\_train\_X) 52. X = np.concatenate((np.ones((m, 1)), self.data\_train\_X), axis=1) 53. Y = self.data\_train\_Y 54. testX = np.concatenate((np.ones((len(self.data\_test\_X), 1)), self.data\_test\_X), axis=1) 55. testY = self.data\_test\_Y 56. thetas, costs, iterationCount, accuracy = Logistic.gradient(X, Y, testX, testY, alpha, epsilon, maxloop) 57. plt.rcParams['font.sans-serif'] = ['SimHei'] 58. plt.rcParams['axes.unicode\_minus'] = False 60. # 绘制分类结果 61. **for** i **in** range(len(self.data\_train\_X)): 62. x = np.concatenate((np.ones((len(self.data\_train\_X), 1)), self.data\_train\_X), axis=1)[i] 63. **if** self.data\_train\_Y[i] == 1: 64. plt.scatter(x[1], x[2], marker='\*', color='blue', s=50) 65. **else**: 66. plt.scatter(x[1], x[2], marker='o', color='green', s=50) 68. **for** i **in** range(len(self.data\_test\_X)): 69. x = np.concatenate((np.ones((len(self.data\_test\_X), 1)), self.data\_test\_X), axis=1)[i] 70. **if** self.data\_test\_Y[i] == 1: 71. plt.scatter(x[1], x[2], marker='\*', color='red', s=50) 72. **else**: 73. plt.scatter(x[1], x[2], marker='o', color='orange', s=50) 75. hSpots = np.linspace(X[:, 1].min(), X[:, 1].max(), 100) 76. theta0, theta1, theta2 = thetas[-1] 78. vSpots = -(theta0 + theta1 \* hSpots) / theta2 79. plt.plot(hSpots, vSpots, color='red', linewidth=.5) 80. plt.xlabel(r'$x\_1$') 81. plt.ylabel(r'$x\_2$') 82. plt.title('分类结果图示') 83. plt.show() 85. # 绘制代价随着迭代次数的变化情况 86. plt.plot(range(len(costs)), costs) 87. plt.xlabel(u'迭代次数') 88. plt.ylabel(u'代价J') 89. plt.title("代价随迭代次数的变化") 91. # 绘制各预测参数theta随迭代次数变化 92. thetasFig, ax = plt.subplots(len(thetas[0])) 93. thetas = np.asarray(thetas) 94. **for** idx, sp **in** enumerate(ax): 95. thetaList = thetas[:, idx] 96. sp.plot(range(len(thetaList)), thetaList) 97. sp.set\_xlabel('Number of iteration') 98. sp.set\_ylabel(r'$\theta\_%d$' % idx) 99. plt.title('各模型参数随迭代次数的变化') 100. plt.show() 102. # 绘制分类准确率随迭代次数的变化 103. plt.plot(range(len(accuracy)), accuracy) 104. plt.title('分类准确率随迭代次数的变化') 105. plt.xlabel(u'迭代次数') 106. plt.ylabel(u'测试集上的预测准确率')  109. **if** \_\_name\_\_ == '\_\_main\_\_': 110. test = WatermelonLogistic() 111. plt.show()   **多分类：iris\_logistic.py:**   1. **import** numpy as np 2. **import** pandas as pd 3. **import** matplotlib.pyplot as plt 4. **import** Logistic  7. **class** IrisLogistic: 8. test\_X = [] 9. test\_Y = [] 11. train\_X = [] 12. train\_Y1 = [] 14. **def** \_\_init\_\_(self): 15. self.data\_get() 16. self.loop\_train()  19. **def** data\_get(self): 20. data = np.array(pd.read\_csv('data/iris.csv')) 22. data1 = data[np.where(data[:, 5] == 'Iris-setosa')] 23. data2 = data[np.where(data[:, 5] == 'Iris-versicolor')] 24. data3 = data[np.where(data[:, 5] == 'Iris-virginica')] 26. r1 = np.random.permutation(data1) 27. test1 = r1[np.arange(0, 10), :] 28. test1\_X = test1[:, np.arange(1, 5)] 29. test1\_Y = np.transpose([test1[:, 5]]) 30. train1 = r1[np.arange(10, 50), :] 31. train1\_X = train1[:, np.arange(1, 5)] 32. train1\_Y = np.transpose([train1[:, 5]]) 34. r2 = np.random.permutation(data2) 35. test2 = r2[np.arange(0, 10), :] 36. test2\_X = test2[:, np.arange(1, 5)] 37. test2\_Y = np.transpose([test2[:, 5]]) 38. train2 = r2[np.arange(10, 50), :] 39. train2\_X = train2[:, np.arange(1, 5)] 40. train2\_Y = np.transpose([train2[:, 5]]) 42. r3 = np.random.permutation(data3) 43. test3 = r3[np.arange(0, 10), :] 44. test3\_X = test3[:, np.arange(1, 5)] 45. test3\_Y = np.transpose([test3[:, 5]]) 46. train3 = r3[np.arange(10, 50), :] 47. train3\_X = train3[:, np.arange(1, 5)] 48. train3\_Y = np.transpose([train3[:, 5]]) 50. self.train\_X = np.concatenate((train1\_X, train2\_X, train3\_X), axis=0) 51. self.train\_Y = np.concatenate((train1\_Y, train2\_Y, train3\_Y), axis=0) 52. self.test\_X = np.concatenate((test1\_X, test2\_X, test3\_X), axis=0) 53. self.test\_Y = np.concatenate((test1\_Y, test2\_Y, test3\_Y), axis=0) 55. **def** model\_train(self, alpha, epsilon, maxloop): 56. # 训练三个模型 57. m1 = len(self.train\_X) 58. X = np.concatenate((np.ones((m1, 1)), self.train\_X), axis=1).astype(np.float64) 59. Y = self.train\_Y 60. Y\_temp1 = Y.copy() 61. Y\_temp2 = Y.copy() 62. Y\_temp3 = Y.copy()  65. Y\_temp1[self.train\_Y != 'Iris-setosa'] = np.float64(0) 66. Y\_temp1[self.train\_Y == 'Iris-setosa'] = np.float64(1) 67. Y1 = Y\_temp1.astype(np.float64) 68. thetas1, costs1, iterationCount1 = Logistic.gradient2(X, Y1,  alpha, epsilon, maxloop)  71. Y\_temp2[self.train\_Y == 'Iris-setosa'] = np.float64(0) 72. Y\_temp2[self.train\_Y == 'Iris-virginica'] = np.float64(0) 73. Y\_temp2[self.train\_Y == 'Iris-versicolor'] = np.float64(1) 74. Y2 = Y\_temp2.astype(np.float64) 75. thetas2, costs2, iterationCount2 = Logistic.gradient2(X, Y2, alpha, epsilon, maxloop)  78. Y\_temp3[self.train\_Y == 'Iris-setosa'] = np.float64(0) 79. Y\_temp3[self.train\_Y == 'Iris-versicolor'] = np.float64(0) 80. Y\_temp3[self.train\_Y == 'Iris-virginica'] = np.float64(1) 81. Y3 = Y\_temp3.astype(np.float64) 82. thetas3, costs3, iterationCount3 = Logistic.gradient2(X, Y3, alpha, epsilon, maxloop) 84. **return** thetas1, thetas2, thetas3 86. **def** loop\_train(self): 87. accuracies = [] 88. **for** i **in** range(1, 100): 89. accu = self.vote(int(i)) 90. accuracies.append(accu) 92. plt.rcParams['font.sans-serif'] = ['SimHei'] 93. plt.rcParams['axes.unicode\_minus'] = False 94. plt.plot(accuracies) 95. plt.title('Iris数据集分类准确率随迭代次数的变化情况') 96. plt.xlabel('迭代次数') 97. plt.ylabel('分类准确率') 98. plt.show()   102. **def** vote(self, looptime): 103. thetas1, thetas2, thetas3 = self.model\_train(0.05, 0.00000001, looptime) 104. X = self.test\_X.copy() 105. X = np.concatenate((np.ones((30, 1)), X), axis=1).astype(np.float64) 106. vote1 = Logistic.sigmoid(np.dot(X, thetas1[len(thetas1)-1])) 107. vote2 = Logistic.sigmoid(np.dot(X, thetas2[len(thetas2)-1])) 108. vote3 = Logistic.sigmoid(np.dot(X, thetas3[len(thetas3)-1])) 109. result = [] 110. **for** i **in** range(0, 30): 111. max\_vote = max(vote1[i], vote2[i], vote3[i]) 112. **if** max\_vote == vote1[i]: 113. result.append(1) 114. **elif** max\_vote == vote2[i]: 115. result.append(2) 116. **else**: 117. result.append(3) 118. realistic = [] 119. **for** j **in** range(0, 30): 120. **if** self.test\_Y[j] == 'Iris-setosa': 121. realistic.append(1) 122. **elif** self.test\_Y[j] == 'Iris-versicolor': 123. realistic.append(2) 124. **else**: 125. realistic.append(3) 127. hit = 0 128. **for** k **in** range(0, 30): 129. **if** result[k] == realistic[k]: 130. hit += 1 132. accuracy = 1.0 \* hit / 30 133. # print("测试集上的预测结果为") 134. # print(result) 135. # 136. # print("真实的分类为") 137. # print(realistic) 138. # 139. # print("准确率为") 140. # print(accuracy) 141. **return** accuracy  144. **if** \_\_name\_\_ == '\_\_main\_\_': 145. test = IrisLogistic() | | | | | | |
| 四、实验结果及分析  1. 基于西瓜数据集3a的二分类问题结果  对西瓜数据集进行测试，指定迭代次数100000次，学习率α=0.05，使用自助法划分测试集与训练集。得到的结果如下：    可以从命令行输出中看出训练出的模型的准确率为66.7%与模型的参数。  结果可视化：   * 1. 决策边界     其中绿色圆点和蓝色五角星代表训练集的坏瓜和好瓜，黄 色圆点和红色五角星代表测试集的坏瓜和好瓜。   * 1. 代价随迭代次数的变化      * 1. 各个参数随迭代次数的变化      * 1. 分类准确率随迭代次数的变化     2. 基于鸢尾花数据集的多分类问题结果  鸢尾花数据集共有3种分类标签，4个属性。因此需要首先对分类标签进行0-1映射。通过将Iris-setosa表示当前分类器将Iris-setosa作为正例，其他标签作为反例，其余两个分类器以此类推。  使用训练集训练完三个分类器之后，将测试集分别带入三个分类器，若只有一个分类器预测为正例，则该测试样例属于该分类器对应的正例类，否则得到各自的置信概率，选择置信概率较大的分类器对应的类作为分到的类。     * 1. 仅迭代100次，使用数据可视化技术查看多分类准确率随迭代次数的变化情况。     此时分类的准确率为：    可以看出仅迭代100次的情况下，分类的准确率达到73.3%   * 1. 加大迭代次数，再次运行查看结果。   将迭代次数加大为1000次    可以看出迭代次数加大到1000次的情况下，分类的准确率提高到了93.3% | | | | | | |
| 五、实验总结与体会，亮点分析  总结体会：  本次实验中的核心思想是利用对率回归算法进行分类问题的求解，此思想利用广义线性模型，巧妙地将分类问题转化为了回归问题，使得问题更加简单化，本次算法模型建立与编程实现的过程使我受益匪浅。  本次实验的亮点主要是：  进行了详细的可视化，代码结构简洁清楚，算法正确。 | | | | | | |