哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称： 机器学习

课程类型：选修

实验题目： 实现k-means聚类方法和混合高斯模型

学号：1173710204

姓名： 陈东鑫

1. 实验目的

实现一个k-means算法和混合高斯模型，并且用EM算法估计模型中的参数。二、实验要求及实验环境

测试：

用高斯分布产生k个高斯分布的数据（不同均值和方差）（其中参数自己设定）。

（1）用k-means聚类，测试效果；

（2）用混合高斯模型和你实现的EM算法估计参数，看看每次迭代后似然值变化情况，考察EM算法是否可以获得正确的结果（与你设定的结果比较）。

应用：可以UCI上找一个简单问题数据，用你实现的GMM进行聚类。

实验环境：

操作系统：Windows10

语言：python3

编程环境：jupyter notebook

三、设计思想（本程序中的用到的主要算法及数据结构）

1．算法原理

GMM模型

混合高斯模型，即多个单高斯模型的叠加

涉及到了k-means算法和EM算法

k-means算法：

1．从样本点中随机选择k个点作为初始的样本中心点。点集记为K0

2．对于所有的样本点，在k个样本中心点中找出与其欧式距离最短的样本中心点，并将该点加入到对应的集合Si（其中，i ∈ [0, k））中。

3．对于每个点集S，求出其中心点作为新的样本中心点。记为Kt（其中t代表第t次迭代）

4．if Kt ≠ Kt-1重复2，3步，else return Kt，S

为使k-means算法有一个比较好的结果，对k-means算法进行了改进。对每一次运行完k-means算法之后的结果，即样本中心点集和k个分类后的样本点集。对于每个分类集合，计算集合中所有点与集合的中心点的平均欧氏距离，将这k个平均欧氏距离再求其平均值，记为E，作为评判结果好坏的标准。多次（如，100次）运行k-means算法，找到E的值最小的那一次，作为k-means算法的最终结果。

EM算法（Expectation-maximization algorithm，即最大期望算法）：

是在概率模型中寻找参数最大似然估计或者最大后验估计的算法，其中概率模型依赖于无法观测的隐性变量。

最大期望算法经过两个步骤交替进行计算：

           第一步是计算期望（E），利用对隐藏变量的现有估计值，计算其最大似然估计值；

           第二步是最大化（M），最大化在E步上求得的最大似然值来计算参数的值。M步上找到的参数估计值被用于下一个E步计算中，这个过程不断交替进行。

其中,x的小标代表第i个特征，上标代表第j个样本

其矩阵表示为

同时考虑其凹凸性，在loss函数上加上符号，故其loss函数应为

为了计算方便，对右式的每个加数取对数

作为这次学习的loss函数。

再计算其梯度

在确定其学习率lr之后，即可进行梯度下降算法更新参数向量

接下来关注共轭梯度法的算法原理：

共轭梯度法的原型函数为这样的一个线性方程组：

在这一次实验中，，，故可得到b。但此时的b并不是Y，两者之间存在一个等式关系，该等式为：

为了做下一步的处理，先对Y进行一个近似处理，近似系数为like（0.5 < like < 1），ones为形如Y的一个全0的矩阵：

由此可得：

故共轭梯度法所使用的对象为：

2. 算法的实现

1）生成数据

使用np.random.multivariate\_normal(mean, cov, size)函数生成二维高斯分布数据集。mean代表均值，cov代表协方差矩阵，size代表样本点个数。其中包含两个二维高斯分布的数据，训练集中，其中一个分布的均值为[0.25, 0.25]，协方差矩阵为[[0.017, 0], [0, 0.017]]，样本点个数为500。另一个分布的均值为[0.75, 0.75]，协方差矩阵为[[0.017, 0], [0, 0.017]]，样本点个数为500。

协方差矩阵为对角阵即满足朴素贝叶斯假设。

测试集也遵循一样的分布，将所得到的数据进行归一化处理。得到样本点的分布如下图所示：

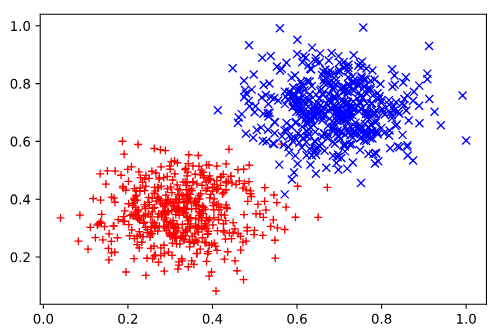


图 1训练数据集

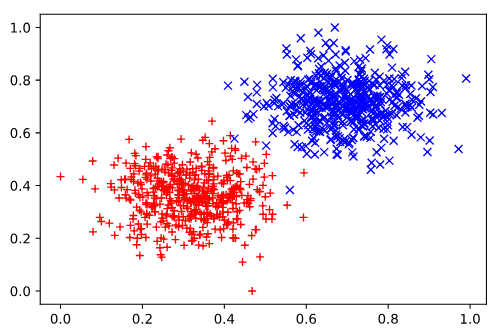


图 2测试数据集

生成数据所需函数的python代码如下

1. # 生成数据函数
2. # mean平均值，cov协方差矩阵，M样本点个数，为偶数
3. # 返回值的形状解释:
4. # data.shape = (3, M)
5. # X.shape = (2, M)
6. # Y.shape = (1, M)
7. # data1.shape = (3, M/2)
8. # data2.shape = (3, M/2)
9. **def** genData(mean1, cov1, mean2, cov2, M):
10. # 训练集
11. X1 = np.random.multivariate\_normal(mean1, cov1, int(M / 2)).T
12. X2 = np.random.multivariate\_normal(mean2, cov2, int(M / 2)).T
13. Y1 = np.zeros\_like(X1[0]).reshape((1, int(M / 2)))
14. Y2 = np.ones\_like(X2[0]).reshape((1, int(M / 2)))
15. X = np.c\_[X1, X2]
16. Y = np.c\_[Y1, Y2]
18. # 测试集
19. XT1 = np.random.multivariate\_normal(mean1, cov1, int(M / 2)).T
20. XT2 = np.random.multivariate\_normal(mean2, cov2, int(M / 2)).T
21. YT1 = np.zeros\_like(XT1[0]).reshape((1, int(M / 2)))
22. YT2 = np.ones\_like(XT2[0]).reshape((1, int(M / 2)))
23. XT = np.c\_[XT1, XT2]
24. YT = np.c\_[YT1, YT2]
26. # 归一化
27. theMax = np.max(np.c\_[X, XT], axis=1).reshape((2, 1))
28. theMin = np.min(np.c\_[X, XT], axis=1).reshape((2, 1))
29. X = X - theMin
30. XT = XT - theMin
31. X = X / (theMax - theMin)
32. XT = XT / (theMax - theMin)
34. # 训练集
35. data1 = np.r\_[X[:, 0:int(M / 2)], Y1]
36. data2 = np.r\_[X[:, int(M / 2):], Y2]
37. data = np.c\_[data1, data2]
39. # 测试集
40. dataT1 = np.r\_[XT[:, 0:int(M / 2)], YT1]
41. dataT2 = np.r\_[XT[:, int(M / 2):], YT2]
42. dataT = np.c\_[dataT1, dataT2]
44. **return** data, X, Y, data1, data2, dataT, XT, YT, dataT1, dataT2

生成数据的代码如下：

1. # 生成数据
2. mean1 = [0.25, 0.25]
3. cov1 = 0.017 \* np.eye(2)
5. # 不满足朴素贝叶斯假设
6. # cov1[0][1] = cov1[0][0] \* (-0.9)
7. # cov1[1][0] = cov1[0][1]
9. mean2 = [0.75, 0.75]
10. cov2 = cov1
11. M = 1000
12. data, X, Y, data1, data2, dataT, XT, YT, dataT1, dataT2 = genData(
13. mean1, cov1, mean2, cov2, M)

2）训练模型

由算法原理处的论述，算法实现所需的函数代码如下

1. # 初始化W向量
2. # W.shape = (N, 1)
3. **def** param(N):
4. **if** N < 3 **or** N % 2 != 1:
5. **print**('the invaild N')
6. **return**
7. W = np.ones((N, 1))
8. **return** W

11. # 初始化用于计算的X（特征数的不同会有不同的X）
12. # X.shape = (N, M)
13. **def** initX(X, N, M):
14. temp = X
15. n = int((N - 3) / 2)
16. **for** i **in** range(n):
17. X = np.r\_[X, temp\*\*(i + 2)]
18. X = np.r\_[np.ones((1, M)), X]
19. **return** X

22. # 多项式函数
23. # W是参数，X是样本，N是特征数与W中元素个数相等
24. # N应为大于等于3的奇数
25. # res.shape = (1, M)
26. **def** H(W, X):
27. res = np.dot(W.T, X)
28. **return** res

31. # 加正则项的多项式函数
32. **def** RegH(W, X, lamda, M):
33. reg = lamda / (2 \* M) \* np.dot(W.T, W)
34. res = np.dot(W.T, X) + reg \* np.ones((1, M))
35. **return** res

38. # sigmod函数
39. # sigmod.shape = (1, M)
40. **def** G(z):
41. sigmod = np.exp(-1 \* z) + np.ones\_like(z)
42. sigmod = 1 / sigmod
43. **return** sigmod

46. # loss函数
47. # type(sum) = <class 'numpy.float64'>
48. **def** loss(W, X, Y, M):
49. sigmod = G(H(W, X))
50. sum1 = np.log(sigmod) \* Y
51. sum2 = np.log(np.ones\_like(sigmod) - sigmod) \* (np.ones\_like(Y) - Y)
52. temp = sum1 + sum2
53. sum = np.sum(temp)
54. sum = -1 \* sum
55. sum = sum / M
57. **return** sum

60. # 梯度
61. **def** grad(X\_, Y, sigmod, M):
62. dz = sigmod - Y
63. grad = np.dot(X\_, dz.T)
64. grad = grad / M
65. **return** grad

68. # 加正则项的梯度
69. **def** RegGrad(X\_, Y, sigmod, M, lamda, W):
70. dz = sigmod - Y
71. grad = np.dot(X\_, dz.T) + lamda \* W
72. grad = grad / M
73. **return** grad

76. #共轭梯度法
77. **def** conGrad(X, Y, W, N, like):
78. # 对Y做一个近似以满足共轭梯度法的要求
79. Y = like \* Y + (1 - like) \* (np.ones\_like(Y) - Y)
80. Y = -1 \* np.log(1 / Y - np.ones\_like(Y))
82. b = np.dot(X, Y.T)
83. X = np.dot(X, X.T)
85. b = b.reshape((b.size, 1))
86. x = W
87. p = b - np.dot(X.T, W)
88. r = p
90. **for** k **in** range(N):
91. **if** (r == np.zeros\_like(r)).all():
92. **break**
93. alpha = ((np.dot(r.T, r)) / np.dot(np.dot(X.T, p).T, p))[0][0]
94. x = x + alpha \* p
95. temp = r
96. r = r - alpha \* np.dot(X.T, p)
97. belta = np.dot(r.T, r)[0][0] / np.dot(temp.T, temp)[0][0]
98. p = r + belta \* p
99. **return** x

其中，

param(N)：用来初始化W向量（列向量），N是W向量的行数，也代表特征数，要求N是大于等于3的奇数。

initX(X, N, M)：初始化用于计算的X，根据N特征数的不同，生成不同的X矩阵，（比如当N为5时，则在X原有的（）基础上加入项）。最终返回矩阵X\_，是一个的矩阵

H(W, X)：多项式函数，返回（的矩阵），作为sigmod函数的输入，是sigmod函数中e的负指数。

RegH(W, X, lamda, M)：加入了L2正则项的多项式函数，返回（的矩阵）。

G(z)：sigmod函数，其中z为H()或regH()的返回值（取决于选用的方法）。计算并返回sigmod（的矩阵）。

loss(W, X, Y, M)：计算并返回

grad(X\_, Y, sigmod, M)：计算并返回之。

RegGrad(X\_, Y, sigmod, M, lamda, W)：计算并返回之

conGrad(X, Y, W, N, like)：进行共轭梯度法的迭代，返回结果值为所求的参数。。

然后通过这样的函数调用实现梯度下降法，带正则项的梯度下降法同理：

1. # init
2. N = 11
3. W0 = param(N)
4. X\_ = initX(X, N, M)
5. XT\_ = initX(XT, N, M)
7. # epoch
8. epoch = 800
9. # learning rate
10. lr = 0.05
11. newLoss = loss(W0, X\_, Y, M)
12. oldLoss = newLoss
13. sigmod = G(H(W0, X\_))
15. **for** item **in** range(epoch):
16. W0 = W0 - lr \* grad(X\_, Y, sigmod, M)
17. oldLoss = newLoss
18. newLoss = loss(W0, X\_, Y, M)
19. t = oldLoss - newLoss
20. **if** t < 0:
21. lr /= 1.5

共轭梯度法：

1. like = 0.999
2. N = 11
3. W2 = param(N)
4. X\_ = initX(X, N, M)
5. XT\_ = initX(XT, N, M)
6. W2 = conGrad(X\_, Y, W2, N, like)

为了读取UCI数据，生成数据函数重新写了一个，可从txt文件中读取数据，生成80%训练集和20%测试集：

1. **def** readFile(filename):
2. f = open(filename, 'r', encoding='utf-8')
3. **return** f.readlines()

6. # M为训练集样本点个数
7. # O为维度数
8. **def** genDataFromFile(filename):
9. context = readFile(filename)
11. **for** i **in** range(len(context)):
12. context[i] = context[i].replace('\n', '')
13. context[i] = context[i].replace(' ', '')
14. context[i] = (context[i]).split(',')
15. O = len(context[0]) - 1
16. M = len(context)
18. **for** i **in** range(M):
19. **for** j **in** range(O + 1):
20. **if** j != O:
21. context[i][j] = float(context[i][j])
22. **else**:
23. context[i][j] = int(context[i][j])
25. data = np.array(context).reshape((M, O + 1))
26. data = np.random.permutation(data)
28. # 归一化
29. dataMax = np.max(data, axis=0)
30. dataMin = np.min(data, axis=0)
31. temp = dataMax - dataMin
32. data = data - dataMin
33. data = data / temp
35. # data, dataT
36. M\_ = M
37. M = int(M \* 0.8)
38. MT = M\_ - M
39. **print**("M =", M)
40. **print**("MT =", MT)
41. dataT = data.copy()[M:, :]
42. data = data.copy()[0:M, :]
44. # data1, data2
45. flag1 = False
46. flag2 = False
47. **for** i **in** range(M):
48. **if** data[i][O] == 0:
49. **if** flag1:
50. data1 = np.c\_[data1, data[:][i].reshape((O + 1, 1))]
51. **else**:
52. data1 = data[:][i].reshape((O + 1, 1))
53. flag1 = True
54. **if** data[i][O] == 1:
55. **if** flag2:
56. data2 = np.c\_[data2, data[:][i].reshape((O + 1, 1))]
57. **else**:
58. data2 = data[:][i].reshape((O + 1, 1))
59. flag2 = True
61. # dataT1, dataT2
62. flag1 = False
63. flag2 = False
64. **for** i **in** range(MT):
65. **if** dataT[i][O] == 0:
66. **if** flag1:
67. dataT1 = np.c\_[dataT1, dataT[:][i].reshape((O + 1, 1))]
68. **else**:
69. dataT1 = dataT[:][i].reshape((O + 1, 1))
70. flag1 = True
71. **if** dataT[i][O] == 1:
72. **if** flag2:
73. dataT2 = np.c\_[dataT2, dataT[:][i].reshape((O + 1, 1))]
74. **else**:
75. dataT2 = dataT[:][i].reshape((O + 1, 1))
76. flag2 = True
77. data = data.T
78. dataT=dataT.T
80. # X, Y, XT, YT
81. X = data[0:O][:].copy()
82. Y = data[O][:].copy().reshape((1, M))
83. XT = dataT[0:O][:].copy()
84. YT = dataT[O][:].copy().reshape((1, MT))
86. **return** M, O, data, data1, data2, X, Y, MT, dataT, dataT1, dataT2, XT, YT

四、实验结果与分析

实验使用的参数为，初始W为全1的列向量，N为11即特征数为11，M为1000即有1000个样本点。

当数据分布满足朴素贝叶斯假设时，结果如下:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 训练 | 测试 | 测试集上的正确率 |
| 梯度下降法 |  |  | 98.90% |
| 带L2正则项的梯度下降法 |  |  | 99.10% |
| 共轭梯度法 |  |  | 99.40% |

表 1数据分布满足朴素贝叶斯假设时结果一览

当数据分布不满足朴素贝叶斯假设时，，由于本次实验生成的两个分布之间相对位置的关系，当协方差矩阵控制分布的两个维度之间为正相关时，训练集上正确率变差，当协方差矩阵控制分布的两个维度之间为负相关时，训练集上正确率变高。从样本点在图上的分布来看，产生这一结果的原因在于，正相关时两个分类混杂的区域变大，该区域内的样本点无法在低维空间做到很好的划分，负相关时则正好相反。

正相关时，协方差矩阵为[[0.017, 0.0068], [0.0068, 0.017 ]]结果如下：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 训练 | 测试 | 测试集上的正确率 |
| 梯度下降法 |  |  | 96.70% |
| 带L2正则项的梯度下降法 |  |  | 97.30% |
| 共轭梯度法 |  |  | 98.30% |

表 2正相关的协方差矩阵结果一览

负相关时，协方差矩阵为[[ 0.017, -0.0068], [-0.0068, 0.017 ]]，结果如下：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 训练 | 测试 | 测试集上的正确率 |
| 梯度下降法 |  |  | 99.70% |
| 带L2正则项的梯度下降法 |  |  | 99.90% |
| 共轭梯度法 |  |  | 100.00% |

表 3负相关的协方差矩阵结果一览

可见，共轭梯度法生成的决策面是过拟合

使用在UCI上得到的数据集Blood Transfusion Service Center Data Set运行算法，当N=25时，得到的结果如下：

|  |  |
| --- | --- |
| 算法 | 正确率 |
| 梯度下降法 | 78.67% |
| 带L2正则项的梯度下降法 | 78.67% |
| 共轭梯度法 | 81.33% |

五、结论

1）正确率

梯度下降法 < 带L2正则项的梯度下降法 < 共轭梯度法

2）关于是否满足朴素贝叶斯假设的讨论。

前述：当数据分布不满足朴素贝叶斯假设时，，由于本次实验生成的两个分布之间相对位置的关系，当协方差矩阵控制分布的两个维度之间为正相关时，训练集上正确率变差，当协方差矩阵控制分布的两个维度之间为负相关时，训练集上正确率变高。从样本点在图上的分布来看，产生这一结果的原因，在数据分布图上看来，正相关时两个分类混杂的区域变大，该区域内的样本点无法在低维空间做到很好的划分，负相关时则正好相反。

由此可知，是否满足朴素贝叶斯假设对结果会产生一定影响，具体影响的方向要视实际样本的分布情况而定

六、参考文献

七、附录：源代码（带注释）

1. **import** numpy as np
2. **import** math
3. **import** matplotlib.pyplot as plt
4. **import** time
6. # 生成数据函数
7. # mean平均值，cov协方差矩阵，M样本点个数，为偶数
8. # 返回值的形状解释:
9. # data.shape = (3, M)
10. # X.shape = (2, M)
11. # Y.shape = (1, M)
12. # data1.shape = (3, M/2)
13. # data2.shape = (3, M/2)
14. **def** genData(mean1, cov1, mean2, cov2, M):
15. # 训练集
16. X1 = np.random.multivariate\_normal(mean1, cov1, int(M / 2)).T
17. X2 = np.random.multivariate\_normal(mean2, cov2, int(M / 2)).T
18. Y1 = np.zeros\_like(X1[0]).reshape((1, int(M / 2)))
19. Y2 = np.ones\_like(X2[0]).reshape((1, int(M / 2)))
20. X = np.c\_[X1, X2]
21. Y = np.c\_[Y1, Y2]
23. # 测试集
24. XT1 = np.random.multivariate\_normal(mean1, cov1, int(M / 2)).T
25. XT2 = np.random.multivariate\_normal(mean2, cov2, int(M / 2)).T
26. YT1 = np.zeros\_like(XT1[0]).reshape((1, int(M / 2)))
27. YT2 = np.ones\_like(XT2[0]).reshape((1, int(M / 2)))
28. XT = np.c\_[XT1, XT2]
29. YT = np.c\_[YT1, YT2]
31. # 归一化
32. theMax = np.max(np.c\_[X, XT], axis=1).reshape((2, 1))
33. theMin = np.min(np.c\_[X, XT], axis=1).reshape((2, 1))
34. X = X - theMin
35. XT = XT - theMin
36. X = X / (theMax - theMin)
37. XT = XT / (theMax - theMin)
39. # 训练集
40. data1 = np.r\_[X[:, 0:int(M / 2)], Y1]
41. data2 = np.r\_[X[:, int(M / 2):], Y2]
42. data = np.c\_[data1, data2]
44. # 测试集
45. dataT1 = np.r\_[XT[:, 0:int(M / 2)], YT1]
46. dataT2 = np.r\_[XT[:, int(M / 2):], YT2]
47. dataT = np.c\_[dataT1, dataT2]
49. **print**("data.shape:", data.shape)
50. **print**("X.shape:", X.shape)
51. **print**("Y.shape:", Y.shape)
52. **return** data, X, Y, data1, data2, dataT, XT, YT, dataT1, dataT2

55. # 初始化W向量
56. # W.shape = (N, 1)
57. **def** param(N):
58. **if** N < 3 **or** N % 2 != 1:
59. **print**('the invaild N')
60. **return**
61. W = np.ones((N, 1))
62. **return** W

65. # 初始化用于计算的X（特征数的不同会有不同的X）
66. # X.shape = (N, M)
67. **def** initX(X, N, M):
68. temp = X
69. n = int((N - 3) / 2)
70. **for** i **in** range(n):
71. X = np.r\_[X, temp\*\*(i + 2)]
72. X = np.r\_[np.ones((1, M)), X]
73. **return** X

76. # 多项式函数
77. # W是参数，X是样本，N是特征数与W中元素个数相等
78. # N应为大于等于3的奇数
79. # res.shape = (1, M)
80. **def** H(W, X):
81. res = np.dot(W.T, X)
82. **return** res

85. # 加正则项的多项式函数
86. **def** RegH(W, X, lamda, M):
87. reg = lamda / (2 \* M) \* np.dot(W.T, W)
88. res = np.dot(W.T, X) + reg \* np.ones((1, M))
89. **return** res

92. # sigmod函数
93. # sigmod.shape = (1, M)
94. **def** G(z):
95. sigmod = np.exp(-1 \* z) + np.ones\_like(z)
96. sigmod = 1 / sigmod
97. **return** sigmod

100. # loss函数
101. # type(sum) = <class 'numpy.float64'>
102. **def** loss(W, X, Y, M):
103. sigmod = G(H(W, X))
104. sum1 = np.log(sigmod) \* Y
105. sum2 = np.log(np.ones\_like(sigmod) - sigmod) \* (np.ones\_like(Y) - Y)
106. temp = sum1 + sum2
107. sum = np.sum(temp)
108. sum = -1 \* sum
109. sum = sum / M
111. **return** sum

114. # 梯度
115. **def** grad(X\_, Y, sigmod, M):
116. dz = sigmod - Y
117. grad = np.dot(X\_, dz.T)
118. grad = grad / M
119. **return** grad

122. # 加正则项的梯度
123. **def** RegGrad(X\_, Y, sigmod, M, lamda, W):
124. dz = sigmod - Y
125. grad = np.dot(X\_, dz.T) + lamda \* W
126. grad = grad / M
127. **return** grad

130. #共轭梯度法
131. **def** conGrad(X, Y, W, N, like):
132. # 对Y做一个近似以满足共轭梯度法的要求
133. Y = like \* Y + (1 - like) \* (np.ones\_like(Y) - Y)
134. Y = -1 \* np.log(1 / Y - np.ones\_like(Y))
136. b = np.dot(X, Y.T)
137. X = np.dot(X, X.T)
139. b = b.reshape((b.size, 1))
140. x = W
141. p = b - np.dot(X.T, W)
142. r = p
144. **for** k **in** range(N):
145. **if** (r == np.zeros\_like(r)).all():
146. **break**
147. alpha = ((np.dot(r.T, r)) / np.dot(np.dot(X.T, p).T, p))[0][0]
148. x = x + alpha \* p
149. temp = r
150. r = r - alpha \* np.dot(X.T, p)
151. belta = np.dot(r.T, r)[0][0] / np.dot(temp.T, temp)[0][0]
152. p = r + belta \* p
153. **return** x
155. # 生成数据
156. mean1 = [0.25, 0.25]
157. cov1 = 0.017 \* np.eye(2)
159. # 不满足朴素贝叶斯假设
160. cov1[0][1] = cov1[0][0] \* (-0.4)
161. cov1[1][0] = cov1[0][1]
162. **print**(cov1)
164. mean2 = [0.75, 0.75]
165. cov2 = cov1
166. M = 1000
167. data, X, Y, data1, data2, dataT, XT, YT, dataT1, dataT2 = genData(
168. mean1, cov1, mean2, cov2, M)
170. # 画出样本点
171. plt.figure()
172. x1, y1 = data1[0:2, :]
173. x2, y2 = data2[0:2, :]
174. plt.plot(x1, y1, '+', color='red')
175. plt.plot(x2, y2, 'x', color='blue')
176. plt.show()
178. #画出测试集
179. plt.figure()
180. x1, y1 = dataT1[0:2, :]
181. x2, y2 = dataT2[0:2, :]
182. plt.plot(x1, y1, '+', color='red')
183. plt.plot(x2, y2, 'x', color='blue')
184. plt.show()
186. # init
187. N = 11
188. W0 = param(N)
189. X\_ = initX(X, N, M)
190. XT\_ = initX(XT, N, M)
192. # epoch
193. epoch = 800
194. # learning rate
195. lr = 0.05
196. newLoss = loss(W0, X\_, Y, M)
197. oldLoss = newLoss
198. sigmod = G(H(W0, X\_))
200. **for** item **in** range(epoch):
201. W0 = W0 - lr \* grad(X\_, Y, sigmod, M)
202. oldLoss = newLoss
203. newLoss = loss(W0, X\_, Y, M)
204. t = oldLoss - newLoss
205. **if** t < 0:
206. lr /= 1.5
207. # print('第', item, '次迭代:平方损失函数为', newLoss, '与上次相差', t)
209. # 画出结果
210. plt.figure()
211. x1, y1 = data1[0:2, :]
212. x2, y2 = data2[0:2, :]
213. plt.plot(x1, y1, '+', color='red')
214. plt.plot(x2, y2, 'x', color='blue')
216. x3 = np.arange(-2, 2, 0.01)
217. y3 = np.arange(-2, 2, 0.01)
218. x3, y3 = np.meshgrid(x3, y3)
219. z3 = W0[0] \* np.ones\_like(x3)
220. **for** i **in** range(1, N, 2):
221. temp1 = W0[i] \* np.power(x3, (i + 1) / 2)
222. temp2 = W0[i + 1] \* np.power(y3, (i + 1) / 2)
223. z3 += temp1 + temp2
224. plt.contour(x3, y3, z3, 0)
225. plt.xlim((-0.05, 1.05))
226. plt.ylim((-0.05, 1.05))
227. plt.show()
229. # 测试结果
230. plt.figure()
231. x1, y1 = dataT1[0:2, :]
232. x2, y2 = dataT2[0:2, :]
233. plt.plot(x1, y1, '+', color='red')
234. plt.plot(x2, y2, 'x', color='blue')
236. x3 = np.arange(-2, 2, 0.01)
237. y3 = np.arange(-2, 2, 0.01)
238. x3, y3 = np.meshgrid(x3, y3)
239. z3 = W0[0] \* np.ones\_like(x3)
240. **for** i **in** range(1, N, 2):
241. temp1 = W0[i] \* np.power(x3, (i + 1) / 2)
242. temp2 = W0[i + 1] \* np.power(y3, (i + 1) / 2)
243. z3 += temp1 + temp2
244. plt.contour(x3, y3, z3, 0)
245. plt.xlim((-0.05, 1.05))
246. plt.ylim((-0.05, 1.05))
247. plt.show()
249. res = G(H(W0, XT\_))
251. # print(res>0.5)
252. # print(res.shape)
254. res[res > 0.5] = 1
255. res[res <= 0.5] = 0
257. temp = res - YT
258. temp = np.abs(temp)
259. sum = np.sum(temp)
260. **print**(sum)
262. **print**("正确率为：%.2f%%" % (100 - 100 \* sum / res.size))
264. W1 = param(N)
265. lamda = 10
266. # epoch
267. epoch = 800
268. # learning rate
269. lr = 0.05
270. newLoss = loss(W1, X\_, Y, M)
271. oldLoss = newLoss
272. sigmod = G(RegH(W1, X\_, lamda, M))
274. **for** item **in** range(epoch):
275. W1 = W1 - lr \* RegGrad(X\_, Y, sigmod, M, lamda, W1)
276. oldLoss = newLoss
277. newLoss = loss(W1, X\_, Y, M)
278. t = oldLoss - newLoss
279. **if** t <= 0:
280. lr /= 1.5
281. # print('第', item, '次迭代:平方损失函数为', newLoss, '与上次相差', t)
283. # 画出结果
284. plt.figure()
285. x1, y1 = data1[0:2, :]
286. x2, y2 = data2[0:2, :]
287. plt.plot(x1, y1, '+', color='red')
288. plt.plot(x2, y2, 'x', color='blue')
290. x3 = np.arange(-2, 2, 0.01)
291. y3 = np.arange(-2, 2, 0.01)
292. x3, y3 = np.meshgrid(x3, y3)
293. z3 = W1[0] \* np.ones\_like(x3)
294. **for** i **in** range(1, N, 2):
295. temp1 = W1[i] \* np.power(x3, (i + 1) / 2)
296. temp2 = W1[i + 1] \* np.power(y3, (i + 1) / 2)
297. z3 += temp1 + temp2
298. plt.contour(x3, y3, z3, 0)
299. plt.xlim((-0.05, 1.05))
300. plt.ylim((-0.05, 1.05))
301. plt.show()
303. # 测试结果
304. plt.figure()
305. x1, y1 = dataT1[0:2, :]
306. x2, y2 = dataT2[0:2, :]
307. plt.plot(x1, y1, '+', color='red')
308. plt.plot(x2, y2, 'x', color='blue')
310. x3 = np.arange(-2, 2, 0.01)
311. y3 = np.arange(-2, 2, 0.01)
312. x3, y3 = np.meshgrid(x3, y3)
313. z3 = W1[0] \* np.ones\_like(x3)
314. **for** i **in** range(1, N, 2):
315. temp1 = W1[i] \* np.power(x3, (i + 1) / 2)
316. temp2 = W1[i + 1] \* np.power(y3, (i + 1) / 2)
317. z3 += temp1 + temp2
318. plt.contour(x3, y3, z3, 0)
319. plt.xlim((-0.05, 1.05))
320. plt.ylim((-0.05, 1.05))
321. plt.show()
323. res = G(H(W1, XT\_))
324. res[res > 0.5] = 1
325. res[res <= 0.5] = 0
327. temp = res - YT
328. temp = np.abs(temp)
329. sum = np.sum(temp)
330. **print**(sum)
331. **print**("正确率为：%.2f%%" % (100 - 100 \* sum / res.size))
333. # init
334. like = 0.999
335. N = 11
336. W2 = param(N)
337. X\_ = initX(X, N, M)
338. XT\_ = initX(XT, N, M)
339. W2 = conGrad(X\_, Y, W2, N, like)
341. # 画出结果
342. plt.figure()
343. x1, y1 = data1[0:2, :]
344. x2, y2 = data2[0:2, :]
345. plt.plot(x1, y1, '+', color='red')
346. plt.plot(x2, y2, 'x', color='blue')
348. x3 = np.arange(-2, 2, 0.01)
349. y3 = np.arange(-2, 2, 0.01)
350. x3, y3 = np.meshgrid(x3, y3)
351. z3 = W2[0] \* np.ones\_like(x3)
352. **for** i **in** range(1, N, 2):
353. temp1 = W2[i] \* np.power(x3, (i + 1) / 2)
354. temp2 = W2[i + 1] \* np.power(y3, (i + 1) / 2)
355. z3 += temp1 + temp2
356. plt.contour(x3, y3, z3, 0)
357. plt.xlim((-0.05, 1.05))
358. plt.ylim((-0.05, 1.05))
359. plt.show()
361. # 测试结果
362. plt.figure()
363. x1, y1 = dataT1[0:2, :]
364. x2, y2 = dataT2[0:2, :]
365. plt.plot(x1, y1, '+', color='red')
366. plt.plot(x2, y2, 'x', color='blue')
368. x3 = np.arange(-2, 2, 0.01)
369. y3 = np.arange(-2, 2, 0.01)
370. x3, y3 = np.meshgrid(x3, y3)
371. z3 = W2[0] \* np.ones\_like(x3)
372. **for** i **in** range(1, N, 2):
373. temp1 = W2[i] \* np.power(x3, (i + 1) / 2)
374. temp2 = W2[i + 1] \* np.power(y3, (i + 1) / 2)
375. z3 += temp1 + temp2
376. plt.contour(x3, y3, z3, 0)
377. plt.xlim((-0.05, 1.05))
378. plt.ylim((-0.05, 1.05))
379. plt.show()
381. res = G(H(W2, XT\_))
382. res[res > 0.5] = 1
383. res[res <= 0.5] = 0
384. temp = res - YT
385. temp = np.abs(temp)
386. sum = np.sum(temp)
387. **print**(sum)
388. **print**("正确率为：%.2f%%" % (100 - 100 \* sum / res.size))