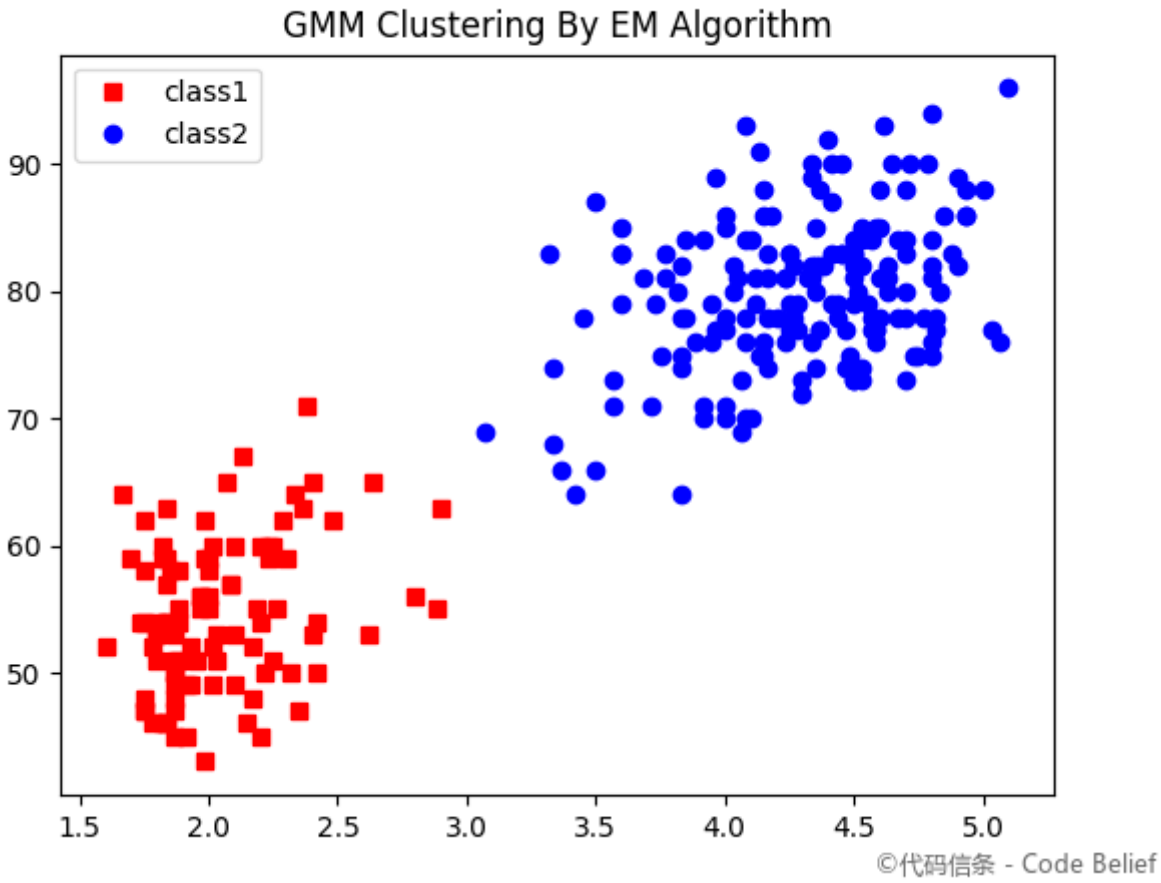
数据库管理系统索引技术概述 (http://www.codebelief.com/article/2018/04/dbms-index technique-overview/)



高斯混合模型 EM 算法的 Python 实现

2017年11月24日 (http://www.codebelief.com/article/2017/11/gmm-em-algorithm-implementation-by-python/) Wray Zheng (http://www.codebelief.com/article/author/wrayzheng/) 1,537

高斯混合模型简介

文章目录

首先简单介绍一下，高斯混合模型（GMM, Gaussian Mixture Model）是多个高斯模型的线性叠加，高斯混合模型的概率分布可以表示如下：

$$P(x) = \sum_{k=1}^{K} \alpha_k \phi(x; \mu_k, \Sigma_k)$$

其中，K 表示模型的个数，α_k 是第 k 个模型的系数，表示出现该模型的概率，$\phi(x; \mu_k, \Sigma_k)$ 是第 k 个高斯模型的概率分布。

这里讨论的是多个随机变量的情况，即多元高斯分布，因此高斯分布中的参数不再是方差 σ_k，而是协方差矩阵 Σ_k 。

我们的目标是给定一堆没有标签的样本和模型个数 K，以此求得混合模型的参数，然后就可以用这个模型来对样本进行聚类。

GMM 的 EM 算法

我们知道，EM 算法是通过不断迭代来求得最佳参数的。在执行该算法之前，需要先给出一个初始化的模型参数。

我们让每个模型的 μ 为随机值，Σ 为单位矩阵，α 为 $\frac{1}{K}$ 即每个模型初始时都是等概率出现的。

EM 算法可以分为 E 步和 M 步。

E 步

直观理解就是我们已经知道了样本 x_i，那么它是由哪个模型产生的呢？我们这里求的就是：样本 x_i 来自于第 k 个模型的概率，我们把这个概率称为模型 k 对样本 x_i 的“责任”，也叫“响应度”，记作 γ_{ik} ，计算公式如下：

$$\gamma_{ik} = \frac{\alpha_k \phi(x_i; \mu_k, \Sigma_k)}{\sum_{k=1}^{K} \alpha_i \phi(x_i; \mu_k, \Sigma_k)}$$

Normalize to 0-1!

M 步

根据样本和当前 γ 矩阵重新估计参数，注意这里 x 为列向量：

note: r_ik is related to sample i and model k. when compute mean, not only choose sampels belong to model k, instead choose all samples, but give a prob to it.

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} x_i$$

$$\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} (x_i - \mu_k)(x_i - \mu_k)^T$$

$$\alpha_k = \frac{N_k}{N}$$

Python 实现

在给出代码前，先作一些说明。

- 在对样本应用高斯混合模型的 EM 算法前，需要先进行数据预处理，即把所有样本值都缩放到 0 和 1 之间。
- 初始化模型参数时，要确保任意两个模型之间参数没有完全相同，否则迭代到最后，两个模型的参数也将完全相同，相当于一个模型。
- 模型的个数必须大于 1。当 K 等于 1 时相当于将样本聚成一类，没有任何意义。

gmm.py 文件：

```
# -*- coding: utf-8 -*-
# ----------------------------------------------------
# Copyright (c) 2017, Wray Zheng. All Rights Reserved.
# Distributed under the BSD License.
# ----------------------------------------------------

import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

DEBUG = True

########################################################
# 调试输出函数
# 由全局变量 DEBUG 控制输出
########################################################
def debug(*args, **kwargs):
    global DEBUG
    if DEBUG:
        print(*args, **kwargs)


########################################################
# 第 k 个模型的高斯分布密度函数
# 每 i 行表示第 i 个样本在各模型中的出现概率
# 返回一维列表
########################################################
def phi(Y, mu_k, cov_k):
    norm = multivariate_normal(mean=mu_k, cov=cov_k)
    return norm.pdf(Y)


########################################################
# E 步：计算每个模型对样本的响应度
# Y 为样本矩阵，每个样本一行，只有一个特征时为列向量
# mu 为均值多维数组，每行表示一个样本各个特征的均值
# cov 为协方差矩阵的数组，alpha 为模型响应度数组
########################################################
def getExpectation(Y, mu, cov, alpha):
    # 样本数
    N = Y.shape[0]
    # 模型数
    K = alpha.shape[0]

    # 为避免使用单个高斯模型或样本，导致返回结果的类型不一致
    # 因此要求样本数和模型个数必须大于1
    assert N > 1, "There must be more than one sample!"
    assert K > 1, "There must be more than one gaussian model!"

    # 响应度矩阵，行对应样本，列对应响应度
    gamma = np.mat(np.zeros((N, K)))

    # 计算各模型中所有样本出现的概率，行对应样本，列对应模型
    prob = np.zeros((N, K))
```

```
    for k in range(K):
        prob[:, k] = phi(Y, mu[k], cov[k])
    prob = np.mat(prob)

    # 计算每个模型对每个样本的响应度
    for k in range(K):
        gamma[:, k] = alpha[k] * prob[:, k]
    for i in range(N):
        gamma[i, :] /= np.sum(gamma[i, :])
    return gamma


########################################################
# M 步：迭代模型参数
# Y 为样本矩阵，gamma 为响应度矩阵
########################################################
def maximize(Y, gamma):
    # 样本数和特征数
    N, D = Y.shape
    # 模型数
    K = gamma.shape[1]

    #初始化参数值
    mu = np.zeros((K, D))
    cov = []
    alpha = np.zeros(K)

    # 更新每个模型的参数
    for k in range(K):
        # 第 k 个模型对所有样本的响应度之和
        Nk = np.sum(gamma[:, k])
        # 更新 mu
        # 对每个特征求均值
        for d in range(D):
            mu[k, d] = np.sum(np.multiply(gamma[:, k], Y[:, d])) / Nk
        # 更新 cov
        cov_k = np.mat(np.zeros((D, D)))
        for i in range(N):
            cov_k += gamma[i, k] * (Y[i] - mu[k]).T * (Y[i] - mu[k]) / Nk
        cov.append(cov_k)
        # 更新 alpha
        alpha[k] = Nk / N
    cov = np.array(cov)
    return mu, cov, alpha


########################################################
# 数据预处理
# 将所有数据都缩放到 0 和 1 之间
########################################################
def scale_data(Y):
    # 对每一维特征分别进行缩放
    for i in range(Y.shape[1]):
        max_ = Y[:, i].max()
        min_ = Y[:, i].min()
```

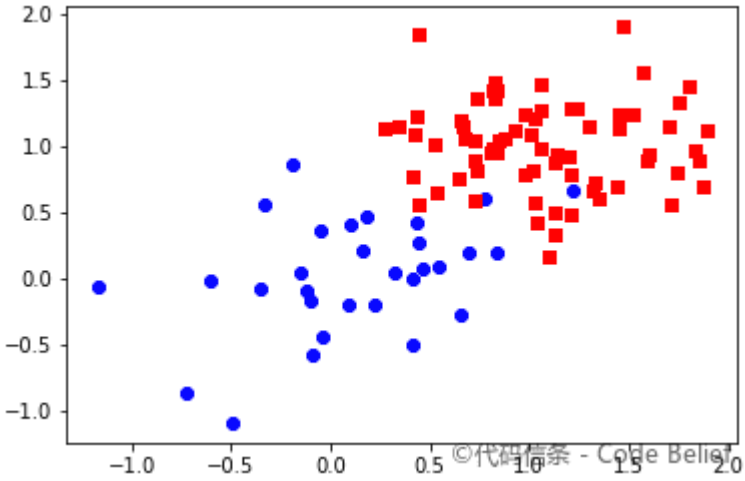
```
        Y[:, i] = (Y[:, i] - min_) / (max_ - min_)
    debug("Data scaled.")
    return Y


########################################################
# 初始化模型参数
# shape 是表示样本规模的二元组，(样本数，特征数)
# K 表示模型个数
########################################################
def init_params(shape, K):
    N, D = shape
    mu = np.random.rand(K, D)
    cov = np.array([np.eye(D)] * K)
    alpha = np.array([1.0 / K] * K)
    debug("Parameters initialized.")
    debug("mu:", mu, "cov:", cov, "alpha:", alpha, sep="\n")
    return mu, cov, alpha


########################################################
# 高斯混合模型 EM 算法
# 给定样本矩阵 Y，计算模型参数
# K 为模型个数
# times 为迭代次数
########################################################
def GMM_EM(Y, K, times):
    Y = scale_data(Y)
    mu, cov, alpha = init_params(Y.shape, K)
    for i in range(times):
        gamma = getExpectation(Y, mu, cov, alpha)
        mu, cov, alpha = maximize(Y, gamma)
    debug("{sep} Result {sep}".format(sep="-" * 20))
    debug("mu:", mu, "cov:", cov, "alpha:", alpha, sep="\n")
    return mu, cov, alpha
```

main.py 文件：

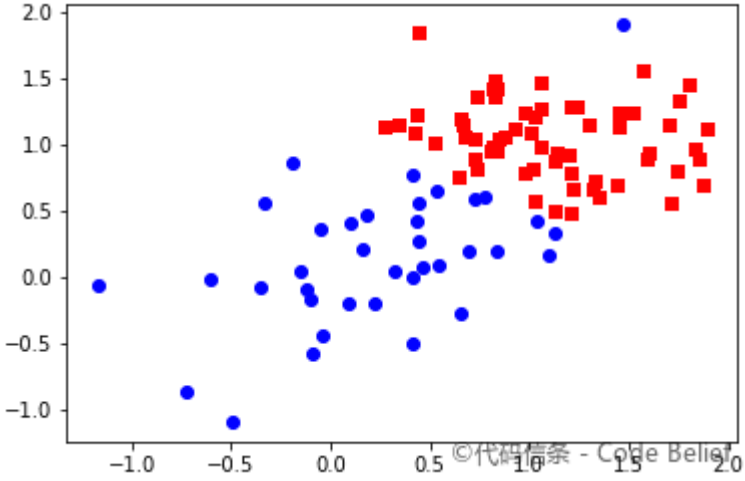
```
# -*- coding: utf-8 -*-
# ----------------------------------------------------
# Copyright (c) 2017, Wray Zheng. All Rights Reserved.
# Distributed under the BSD License.
# ----------------------------------------------------

import matplotlib.pyplot as plt
from gmm import *

# 设置调试模式
DEBUG = True

# 载入数据
Y = np.loadtxt("gmm.data")
matY = np.matrix(Y, copy=True)

# 模型个数，即聚类的类别个数
K = 2

# 计算 GMM 模型参数
mu, cov, alpha = GMM_EM(matY, K, 100)

# 根据 GMM 模型，对样本数据进行聚类，一个模型对应一个类别
N = Y.shape[0]
# 求当前模型参数下，各模型对样本的响应度矩阵
gamma = getExpectation(matY, mu, cov, alpha)
# 对每个样本，求响应度最大的模型下标，作为其类别标识
category = gamma.argmax(axis=1).flatten().tolist()[0]
# 将每个样本放入对应类别的列表中
class1 = np.array([Y[i] for i in range(N) if category[i] == 0])
class2 = np.array([Y[i] for i in range(N) if category[i] == 1])

# 绘制聚类结果
plt.plot(class1[:, 0], class1[:, 1], 'rs', label="class1")
plt.plot(class2[:, 0], class2[:, 1], 'bo', label="class2")
plt.legend(loc="best")
plt.title("GMM Clustering By EM Algorithm")
plt.show()
```

测试结果

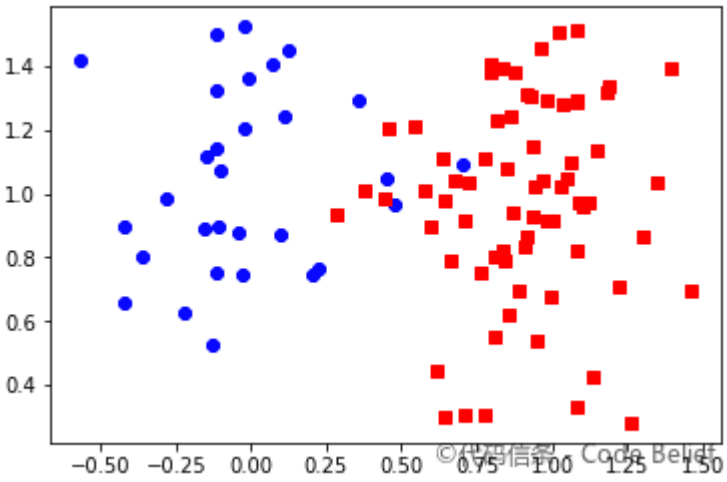
样本 1 原始类别：



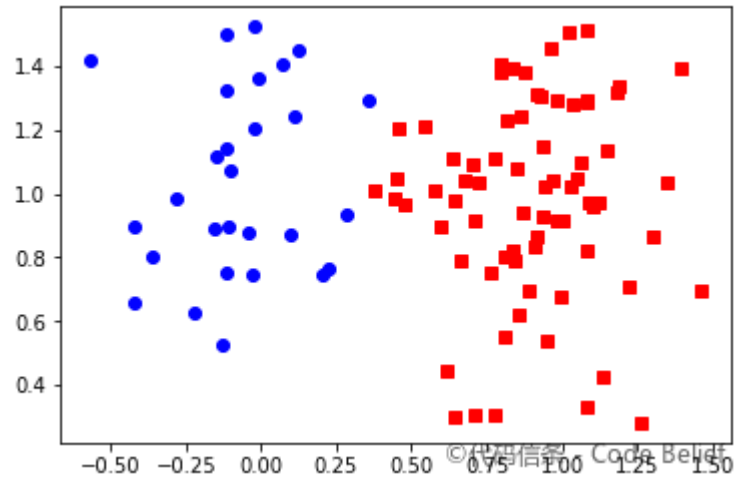
样本 1 聚类结果：



样本 2 原始类别：



样本 2 聚类结果：



测试数据

完整代码和数据放在了 Github 上，可 点此访问 (https://github.com/wrayzheng/gmm-em-clustering) 。

或者通过下列代码生成测试数据：

```
import numpy as np
import matplotlib.pyplot as plt

cov1 = np.mat("0.3 0;0 0.1")
cov2 = np.mat("0.2 0;0 0.3")
mu1 = np.array([0, 1])
mu2 = np.array([2, 1])

sample = np.zeros((100, 2))
sample[:30, :] = np.random.multivariate_normal(mean=mu1, cov=cov1, size=30)
sample[30:, :] = np.random.multivariate_normal(mean=mu2, cov=cov2, size=70)
np.savetxt("sample.data", sample)

plt.plot(sample[:30, 0], sample[:30, 1], "bo")
plt.plot(sample[30:, 0], sample[30:, 1], "rs")
plt.show()
```

参考资料

- **统计学习方法**，李航
- **Pattern Recognition and Machine Learning**，Christopher M Bishop

---

作者：Wray Zheng (http://www.codebelief.com)
原文：http://www.codebelief.com/article/2017/11/gmm-em-algorithm-implementation-by-python/ (http://www.codebelief.com/article/2017/11/gmm-em-algorithm-implementation-by-python/)

相关文章

- Logistic 回归原理分析与 Python 实现 (http://www.codebelief.com/article/2017/11/logistic-regression-analysis-and-python-realization/)
- Python 数据可视化：matplotlib 库学习笔记 (http://www.codebelief.com/article/2017/04/python-data-visualization-matplotlib-learning/)
- 机器学习：scikit-learn 实现手写数字识别 (http://www.codebelief.com/article/2017/05/python-machine-learning-scikit-learn-to-recognize-handwritten-digits/)
- matplotlib 演示最小二乘法拟合过程 (http://www.codebelief.com/article/2017/04/matplotlib-demonstrate-least-square-regression-process/)
- Python 数据分析：numpy 多维数组 ndarray (http://www.codebelief.com/article/2017/03/python-data-analysis-numpy-ndarray/)
- [译]Python 受推崇的 super！ (http://www.codebelief.com/article/2017/03/python-super-considered-super/)

Python (http://www.codebelief.com/tag/python/)、机器学习 (http://www.codebelief.com/tag/machine-learning/)

← 自动机：DFA转化为正则表达式 (HTTP://WWW.CODEBELIEF.COM/ARTICLE/2017/11/AUTOMATA-CONVERT-DFA-TO-REGULAR-EXPRESSION/)

TOMCAT WEB 应用绑定域名的几种方式 → (HTTP://WWW.CODEBELIEF.COM/ARTICLE/2017/12/WAYS-FOR-TOMCAT-WEB-APP-TO-BIND-DOMAINS/)

发表评论

电子邮件地址不会被公开。 必填项已用*标注

评论

姓名 *

电子邮件 *

站点

发表评论

文章搜索

在此输入并回车

近期文章

- 局部最优未必是全局最优 (http://www.codebelief.com/article/2018/05/local-optimum-is-not-necessarily-global-optimum/)
- Java 多线程下载器的设计与实现 (http://www.codebelief.com/article/2018/04/java-multithread-downloader-design-and-implementation/)
- 数据库管理系统索引技术概述 (http://www.codebelief.com/article/2018/04/dbms-index-technique-overview/)
- 彻底理解二分查找及其边界情况 (http://www.codebelief.com/article/2018/04/completely-understand-binary-search-and-its-boundary-cases/)
- Java logging 模块的使用 (http://www.codebelief.com/article/2018/04/usage-of-java-logging-module/)
- 数据库视图的概念、定义与使用 (http://www.codebelief.com/article/2018/03/database-view-concept-define-and-usage/)
- 理解数据库的内连接、外连接和交叉连接 (http://www.codebelief.com/article/2018/03/understand-database-inner-join-outer-join-and-cross-join/)
- Java Swing 编写数据库增删改查 GUI 程序 (http://www.codebelief.com/article/2018/03/java-swing-code-a-mysql-crud-gui-program/)
- Java 是如何利用接口避免函数回调的 (http://www.codebelief.com/article/2018/02/java-how-to-use-interface-to-avoid-function-callback/)
- Java 多线程的竞争条件、互斥和同步 (http://www.codebelief.com/article/2018/02/java-multi-thread-race-condition-mutual-exclusion-and-synchronization/)

文章归档

标签

数据结构 (3) (http://www.codebelief.com/tag/data-structure/)

Python标准库 (3) (http://www.codebelief.com/tag/python-standard-library/)

正则表达式 (3) (http://www.codebelief.com/tag/regular-expression/)

Vim (3) (http://www.codebelief.com/tag/vim/) Qt (3) (http://www.codebelief.com/tag/qt/)

机器学习 (3) (http://www.codebelief.com/tag/machine-learning/)

Web (3) (http://www.codebelief.com/tag/web/) 自动机 (3) (http://www.codebelief.com/tag/automata/)

Node.js (2) (http://www.codebelief.com/tag/node-js/)

计算机网络 (2) (http://www.codebelief.com/tag/network/)

matplotlib (2) (http://www.codebelief.com/tag/matplotlib/)

Jupyter Notebook (2) (http://www.codebelief.com/tag/jupyter-notebook/)

感想 (2) (http://www.codebelief.com/tag/sentiment/) Swing (2) (http://www.codebelief.com/tag/swing/)

Maven (2) (http://www.codebelief.com/tag/maven/)

软件工程 (2) (http://www.codebelief.com/tag/software-engineering/)

并发 (2) (http://www.codebelief.com/tag/concurrency/) Git (2) (http://www.codebelief.com/tag/git/)

Awt (1) (http://www.codebelief.com/tag/awt/) sed (1) (http://www.codebelief.com/tag/sed/)

shell (1) (http://www.codebelief.com/tag/shell/) Ubuntu (1) (http://www.codebelief.com/tag/ubuntu/)

VPS (1) (http://www.codebelief.com/tag/vps/) Boost (1) (http://www.codebelief.com/tag/boost/)

cURL (1) (http://www.codebelief.com/tag/curl/)

scikit-learn (1) (http://www.codebelief.com/tag/scikit-learn/)

排序 (1) (http://www.codebelief.com/tag/sort/)

科学计算 (1) (http://www.codebelief.com/tag/scientific-computing/)

二叉树 (1) (http://www.codebelief.com/tag/binary-tree/)

Express (1) (http://www.codebelief.com/tag/express/)

并查集 (1) (http://www.codebelief.com/tag/union-find/)

Eclipse (1) (http://www.codebelief.com/tag/eclipse/)