实验五 聚类算法实现

一、实验目的及要求

1、掌握 k-means 与 GMM，聚类算法的基本原理

2、掌握 k-means 聚类算法的工作原理与实现方法。

3、掌握 GMM 算法的工作原理与实现方法。

4、理解 k-means 与 GMM 算法的区别和联系。

二、预习要求

阅读本实验例程部分，实现基本的 k-means 与 GMM 聚类方法，以便能够充分用

实验时间编程调试。

三、实验设备

硬件：PC 机。

软件：Python 及相关集成开发环境。

四、实验内容

（一）利用利用 Python 编程实现 k-means 算法，并将聚类结果以及聚类中心标准来。

具体要求如下：

（1）随机生成满足高斯分布的待聚类二维数据 200 个，其中 100 个均值值为[2,2],协方

差矩阵为 [0.5,0;0,0.5],另外 100 个均值值为[0,0]，协方差矩阵为 [0.5,0;0,0.5]。

（2）利用 k-mean 算法将技术聚为 2 类，打印聚类结果以及聚类中心。

（3）修改聚类中心的初始化方法，分别采用随机聚类中心和值域均匀采样聚类中心

法。比较聚类结果的区别。

（二）利用利用 Python 编程实现 k-means 算法，并将聚类结果以及聚类中心标准来。

具体要求如下：

（1）将给定的数据利用 GMM 算法聚成 4 类，打印每个 GMM 模型的权重 w，均值

mu 以及协方差矩阵 sigma

（2）利用 matplotlib 显示聚类结果。

（3）利用 K-means 算法对该数据进行聚类，比较 GMM 以及 k-means 算法的区别。

五、实验报告内容要求

1. 列出编写的 python 代码。对主要的语句进行注释

GMM.PY

import numpy as np

import matplotlib.pyplot as plt

from kmeans import run\_kmeans

*# GMM 参数初始化*

*# dataset：[N,D] 训练数据*

*# K ：高斯成分的个数*

def inti\_GMM(dataset, K):

N, D = np.shape(dataset)

val\_max = np.max(dataset, axis=0)

val\_min = np.min(dataset, axis=0)

centers = np.linspace(val\_min, val\_max, num=K + 2)

mus = centers[1:-1, :]

sigmas = np.array([0.5 \* np.eye(D) for i in range(K)])

ws = 1.0 / K \* np.ones(K)

return mus, sigmas, ws

*# 计算一个高斯的pdf*

*# x: 数据 [N,D]*

*# sigma 方差 [D,D]*

*# mu 均值 [1,D]*

def getPdf(x, mu, sigma, eps=1e-12):

N, D = np.shape(x)

if D == 1:

sigma = sigma + eps

A = 1.0 / (sigma)

det = np.fabs(sigma[0])

else:

sigma = sigma + eps \* np.eye(D)

A = np.linalg.inv(sigma)

det = np.fabs(np.linalg.det(sigma))

*# 计算系数*

factor = (2.0 \* np.pi) \*\* (D / 2.0) \* (det) \*\* (0.5)

*# 计算 pdf*

dx = x - mu

pdf = [(np.exp(-0.5 \* np.dot(np.dot(dx[i], A), dx[i])) + eps) / factor for i in range(N)]

return pdf

def train\_GMM\_step(dataset, mus, sigmas, ws):

N, D = np.shape(dataset)

K, D = np.shape(mus)

*# 计算样本在每个成分上的pdf*

pdfs = np.zeros([N, K])

for k in range(K):

pdfs[:, k] = getPdf(dataset, mus[k], sigmas[k])

*# 获取r*

r = pdfs \* np.tile(ws, (N, 1))

r\_sum = np.tile(np.sum(r, axis=1, keepdims=True), (1, K))

r = r / r\_sum

*# 进行参数的更新*

for k in range(K):

r\_k = r[:, k]

N\_k = np.sum(r\_k)

r\_k = r\_k[:, np.newaxis] *# [N,1]*

*# 更新mu*

mu = np.sum(dataset \* r\_k, axis=0) / N\_k *# [D,1]*

*# 更新sigma*

dx = dataset - mu

sigma = np.zeros([D, D])

for i in range(N):

sigma = sigma + r\_k[i, 0] \* np.outer(dx[i], dx[i])

sigma = sigma / N\_k

*# 更新w*

w = N\_k / N

mus[k] = mu

sigmas[k] = sigma

ws[k] = w

return mus, sigmas, ws

def train\_GMM(dataset, K=2, m=10):

mus, sigmas, ws = inti\_GMM(dataset, K)

for i in range(m):

*# print("Step ",i)*

mus, sigms, ws = train\_GMM\_step(dataset, mus, sigmas, ws)

return mus, sigms, ws

def getlogPdfFromeGMM(datas, mus, sigmas, ws):

N, D = np.shape(datas)

K, D = np.shape(mus)

weightedlogPdf = np.zeros([N, K])

for k in range(K):

temp = getPdf(datas, mus[k], sigmas[k], eps=1e-12)

weightedlogPdf[:, k] = np.log(temp) + np.log(ws[k])

return weightedlogPdf, np.sum(weightedlogPdf, axis=1)

def clusterByGMM(datas, mus, sigmas, ws):

weightedlogPdf, \_ = getlogPdfFromeGMM(datas, mus, sigmas, ws)

labs = np.argmax(weightedlogPdf, axis=1)

return labs

def draw\_cluster(dataset, lab, dic\_colors, name="0.jpg"):

plt.cla()

vals\_lab = set(lab.tolist())

for i, val in enumerate(vals\_lab):

index = np.where(lab == val)[0]

sub\_dataset = dataset[index, :]

plt.scatter(sub\_dataset[:, 0], sub\_dataset[:, 1], s=16., color=dic\_colors[i])

plt.savefig(name)

if \_\_name\_\_ == "\_\_main\_\_":

'''

聚类测试 1

'''

dic\_colors = {0: (0., 0.5, 0.), 1: (0.8, 0, 0)}

a = np.random.multivariate\_normal([2, 2], [[.5, 0], [0, .5]], 100)

b = np.random.multivariate\_normal([0, 0], [[0.5, 0], [0, 0.5]], 100)

dataset = np.r\_[a, b]

lab\_ture = np.r\_[np.zeros(100), np.ones(100)].astype(int)

*# 训练GMM*

mus, sigmas, ws = train\_GMM(dataset, K=2, m=10)

print(mus)

print(sigmas)

print(ws)

*# 进行聚类*

labs\_GMM = clusterByGMM(dataset, mus, sigmas, ws)

*# k-menas 比较*

labs\_kmeans = run\_kmeans(dataset, K=2, m=20)

*# 画结果*

draw\_cluster(dataset, lab\_ture, dic\_colors, name="c\_ture1.jpg")

draw\_cluster(dataset, labs\_GMM, dic\_colors, name="c\_GMM1.jpg")

draw\_cluster(dataset, labs\_kmeans, dic\_colors, name="c\_kmeans1.jpg")

'''

聚类测试 2

'''

with open('Clustering\_gmm.csv', 'r') as f:

lines = f.read().splitlines()[1:]

lines = [line.split(",") for line in lines]

dataset = np.array(lines).astype(np.float64)

lab\_ture = np.ones(np.shape(dataset)[0])

dic\_colors = {0: (0., 0.5, 0.), 1: (0.8, 0, 0), 2: (0.5, 0.5, 0), 3: (0, 0.5, 0.5)}

*# 训练GMM*

mus, sigmas, ws = train\_GMM(dataset, K=4, m=100)

*# 进行聚类*

labs\_GMM = clusterByGMM(dataset, mus, sigmas, ws)

*# k-menas 比较*

labs\_kmeans = run\_kmeans(dataset=dataset, dic\_colors=dic\_colors, K=4, m=20)

*# 画结果*

draw\_cluster(dataset, lab\_ture, dic\_colors, name="c\_ture2.jpg")

draw\_cluster(dataset, labs\_GMM, dic\_colors, name="c\_GMM2.jpg")

draw\_cluster(dataset, labs\_kmeans, dic\_colors, name="c\_kmeans2.jpg")

'''

分类测试

'''

file\_data = 'iris.data'

*# 数据读取*

data = np.loadtxt(file\_data, dtype=np.float64, delimiter=',', usecols=(0, 1, 2, 3))

lab = np.loadtxt(file\_data, dtype=str, delimiter=',', usecols=(4))

*# 分为训练集和测试集*

N = 150

N\_train = 100

N\_test = 50

perm = np.random.permutation(N)

index\_train = perm[:N\_train]

index\_test = perm[N\_train:]

data\_train = data[index\_train, :]

lab\_train = lab[index\_train]

data\_test = data[index\_test, :]

lab\_test = lab[index\_test]

*# 获取 训练标签类型*

unique\_labs = np.unique(lab\_train).tolist()

models = {}

*# 进行GMM 训练，为每类数据训练一个GMM*

for lab in unique\_labs:

*# 进行数据筛选*

index = np.where(lab\_train == lab)[0]

dataset = data\_train[index, :]

*# 利用训练的数据训练 GMM*

mus, sigmas, ws = train\_GMM(dataset, K=2, m=20)

models[lab] = {}

models[lab]["ws"] = ws

models[lab]["mus"] = mus

models[lab]["sigmas"] = sigmas

*# 进测试*

pdfs = np.zeros([N\_test, len(unique\_labs)])

index2lab = {}

*# 计算每条测试数据在不同GMM上的logpdf*

for i, lab in enumerate(unique\_labs):

index2lab[i] = lab

ws = models[lab]["ws"]

mus = models[lab]["mus"]

sigmas = models[lab]["sigmas"]

*# 计算每条测试数据在这个GMM上的pdf*

\_, pdf = getlogPdfFromeGMM(data\_test, mus, sigmas, ws)

pdfs[:, i] = pdf

*# 选取最大似然值 实现分类*

det\_labs\_index = np.argmax(pdfs, axis=1).tolist()

*# 将分类结果转为字符串*

det\_labs\_str = [index2lab[i] for i in det\_labs\_index]

*# 进行测试结果输出并统计准确率*

N\_right = 0

for i, lab\_str in enumerate(det\_labs\_str):

print("测试数据 %d 真实标签 %s 检测标签 %s" % (i, lab\_test[i], lab\_str))

if lab\_str == lab\_test[i]:

N\_right = N\_right + 1

print("准确率为 %.2f%%" % (N\_right \* 100 / N\_test))

print(models)

Kmeans.py

import numpy as np

import matplotlib.pyplot as plt

*# 构造聚类中心*

*# dataset [N,D]*

*# K 聚类中心的数目 [K,D]*

def creat\_centers(dataset,K):

val\_max = np.max(dataset,axis=0)

val\_min = np.min(dataset,axis=0)

centers = np.linspace(val\_min,val\_max,num=K+2)

return centers[1:-1,:]

*# def creat\_centers(dataset,K):*

*# N,D = np.shape(dataset)*

*# index = np.random.permutation(N)*

*# centers = dataset[index[:2],:]*

*# return centers*

*# keams 绘图*

*# dataset (N,D)*

*# lab (N,)*

*# dic\_colors K 种颜色*

*# centers (K,D)*

def draw\_kmeans(dataset,lab,centers,dic\_colors=None,name="0.jpg"):

plt.cla()

vals\_lab = set(lab.tolist())

for i,val in enumerate(vals\_lab):

index = np.where(lab==val)[0]

sub\_dataset = dataset[index,:]

plt.scatter(sub\_dataset[:,0],sub\_dataset[:,1],s=16., color=dic\_colors[i])

for i in range(np.shape(centers)[0]):

plt.scatter(centers[i,0],centers[i,1],color="k",marker="+",s = 200.)

plt.savefig(name)

def run\_kmeans(dataset,K, m = 20,dic\_colors=None, b\_draw=False):

N,D = np.shape(dataset)

*# print(N,D)*

*# 确定初始化聚类中心*

centers = creat\_centers(dataset,K)

lab = np.zeros(N)

if b\_draw:

draw\_kmeans(dataset,lab,centers,dic\_colors,name="int.jpg")

*# 进行m轮迭代*

labs = np.zeros(N) *# 初始聚类结果*

for it in range(m):

*# 计算每个点距离中心的距离*

distance = np.zeros([N,K])

for k in range(K):

center = centers[k,:]

*# 计算欧式距离*

diff = np.tile(center, (N, 1)) - dataset

sqrDiff = diff \*\* 2

sqrDiffSum = sqrDiff.sum(axis=1)

distance[:,k] = sqrDiffSum

*# 距离排序，进行聚类*

labs\_new = np.argmin(distance,axis=1)

error = np.sum(np.min(distance,axis=1))/N

print("第 %d 次聚类 距离误差 %.2f"%(it,error))

*# 绘图*

if b\_draw:

draw\_kmeans(dataset,labs\_new,centers,

dic\_colors,name=str(it)+"\_oldcenter.jpg")

*# 计算新的聚类中心*

for k in range(K):

index = np.where(labs\_new==k)[0]

centers[k,:] = np.mean(dataset[index,:],axis=0)

*# 绘图*

if b\_draw:

draw\_kmeans(dataset,labs\_new,centers,

dic\_colors,name=str(it)+"\_newcenter.jpg")

*# 如果聚类结果和上次相同，退出*

if np.sum(labs\_new-labs)==0:

return labs\_new

else:

labs = labs\_new

return labs

if \_\_name\_\_=="\_\_main\_\_":

a = np.random.multivariate\_normal([2,2], [[.5,0],[0,.5]], 100)

b = np.random.multivariate\_normal([0,0], [[0.5,0],[0,0.5]], 100)

dataset = np.r\_[a,b]

lab\_ture = np.r\_[np.zeros(100),np.ones(100)].astype(int)

dic\_colors={0:(0.,0.5,0.),1:(0.8,0,0)}

labs = run\_kmeans(dataset,K=2, m = 20,dic\_colors=dic\_colors,b\_draw=True)

*# print(dataset.shape)*

*# centers = creat\_centers(dataset,K=2)*

*# print(centers)*

*# draw\_keams(dataset,np.zeros(200),dic\_colors,centers)*

1. 对实验结果进行截图，对结果进行必要的解释并说明实验中使用的参数。

GMM结果

[[ 1.06793514e-01 -1.80921955e-03]

[ 1.98307533e+00 1.98049556e+00]]

[[[0.41505989 0.01634632]

[0.01634632 0.49712063]]

[[0.59876212 0.01723403]

[0.01723403 0.55951457]]]

[0.51279107 0.48720893]

第 0 次聚类 距离误差 1.25

第 1 次聚类 距离误差 0.99

第 2 次聚类 距离误差 0.98

第 3 次聚类 距离误差 0.98

第 0 次聚类 距离误差 4.70

第 1 次聚类 距离误差 2.48

第 2 次聚类 距离误差 2.34

第 3 次聚类 距离误差 2.28

第 4 次聚类 距离误差 2.27

第 5 次聚类 距离误差 2.27

第 6 次聚类 距离误差 2.27

测试数据 0 真实标签 Iris-virginica 检测标签 Iris-virginica

测试数据 1 真实标签 Iris-versicolor 检测标签 Iris-versicolor

测试数据 2 真实标签 Iris-virginica 检测标签 Iris-virginica

测试数据 3 真实标签 Iris-setosa 检测标签 Iris-setosa

测试数据 4 真实标签 Iris-virginica 检测标签 Iris-virginica

测试数据 5 真实标签 Iris-versicolor 检测标签 Iris-versicolor

测试数据 6 真实标签 Iris-virginica 检测标签 Iris-virginica

测试数据 7 真实标签 Iris-setosa 检测标签 Iris-setosa

测试数据 8 真实标签 Iris-setosa 检测标签 Iris-setosa

测试数据 9 真实标签 Iris-setosa 检测标签 Iris-setosa

测试数据 10 真实标签 Iris-virginica 检测标签 Iris-virginica

测试数据 11 真实标签 Iris-setosa 检测标签 Iris-setosa

测试数据 12 真实标签 Iris-virginica 检测标签 Iris-virginica

测试数据 13 真实标签 Iris-virginica 检测标签 Iris-virginica

测试数据 14 真实标签 Iris-versicolor 检测标签 Iris-versicolor

测试数据 15 真实标签 Iris-virginica 检测标签 Iris-virginica

测试数据 16 真实标签 Iris-versicolor 检测标签 Iris-versicolor

测试数据 17 真实标签 Iris-versicolor 检测标签 Iris-versicolor

测试数据 18 真实标签 Iris-setosa 检测标签 Iris-setosa

测试数据 19 真实标签 Iris-setosa 检测标签 Iris-setosa

测试数据 20 真实标签 Iris-setosa 检测标签 Iris-setosa

测试数据 21 真实标签 Iris-virginica 检测标签 Iris-virginica

测试数据 22 真实标签 Iris-virginica 检测标签 Iris-virginica

测试数据 23 真实标签 Iris-versicolor 检测标签 Iris-versicolor

测试数据 24 真实标签 Iris-setosa 检测标签 Iris-setosa

测试数据 25 真实标签 Iris-setosa 检测标签 Iris-setosa

测试数据 26 真实标签 Iris-versicolor 检测标签 Iris-versicolor

测试数据 27 真实标签 Iris-virginica 检测标签 Iris-virginica

测试数据 28 真实标签 Iris-versicolor 检测标签 Iris-versicolor

测试数据 29 真实标签 Iris-virginica 检测标签 Iris-virginica

测试数据 30 真实标签 Iris-versicolor 检测标签 Iris-versicolor

测试数据 31 真实标签 Iris-setosa 检测标签 Iris-setosa

测试数据 32 真实标签 Iris-virginica 检测标签 Iris-virginica

测试数据 33 真实标签 Iris-virginica 检测标签 Iris-virginica

测试数据 34 真实标签 Iris-setosa 检测标签 Iris-setosa

测试数据 35 真实标签 Iris-virginica 检测标签 Iris-virginica

测试数据 36 真实标签 Iris-versicolor 检测标签 Iris-versicolor

测试数据 37 真实标签 Iris-versicolor 检测标签 Iris-versicolor

测试数据 38 真实标签 Iris-versicolor 检测标签 Iris-versicolor

测试数据 39 真实标签 Iris-versicolor 检测标签 Iris-versicolor

测试数据 40 真实标签 Iris-setosa 检测标签 Iris-setosa

测试数据 41 真实标签 Iris-virginica 检测标签 Iris-virginica

测试数据 42 真实标签 Iris-setosa 检测标签 Iris-setosa

测试数据 43 真实标签 Iris-virginica 检测标签 Iris-virginica

测试数据 44 真实标签 Iris-virginica 检测标签 Iris-virginica

测试数据 45 真实标签 Iris-setosa 检测标签 Iris-setosa

测试数据 46 真实标签 Iris-setosa 检测标签 Iris-setosa

测试数据 47 真实标签 Iris-versicolor 检测标签 Iris-versicolor

测试数据 48 真实标签 Iris-virginica 检测标签 Iris-virginica

测试数据 49 真实标签 Iris-versicolor 检测标签 Iris-versicolor

准确率为 100.00%

{'Iris-setosa': {'ws': array([0.51305029, 0.48694971]), 'mus': array([[4.82280729, 3.3077116 , 1.4222564 , 0.24281726],

[5.20481027, 3.53951507, 1.47319066, 0.25152774]]), 'sigmas': array([[[ 0.12704652, 0.0871153 , 0.03508822, 0.01096714],

[ 0.0871153 , 0.13807162, 0.01930405, 0.000506 ],

[ 0.03508822, 0.01930405, 0.02493764, 0.0019142 ],

[ 0.01096714, 0.000506 , 0.0019142 , 0.00365451]],

[[ 0.08284839, 0.09313986, -0.0145841 , 0.01004518],

[ 0.09313986, 0.13956562, 0.00543815, 0.02543333],

[-0.0145841 , 0.00543815, 0.02945494, 0.00709981],

[ 0.01004518, 0.02543333, 0.00709981, 0.01089092]]])}, 'Iris-versicolor': {'ws': array([0.3849102, 0.6150898]), 'mus': array([[5.84901062, 2.50040957, 4.01075268, 1.22410425],

[6.17345236, 2.95496179, 4.56461635, 1.4171702 ]]), 'sigmas': array([[[0.26234542, 0.08605389, 0.18758101, 0.06653919],

[0.08605389, 0.09381842, 0.03856077, 0.02707868],

[0.18758101, 0.03856077, 0.15606769, 0.05502422],

[0.06653919, 0.02707868, 0.05502422, 0.03100201]],

[[0.20999961, 0.04099988, 0.05975757, 0.02469731],

[0.04099988, 0.03344339, 0.01162807, 0.01941632],

[0.05975757, 0.01162807, 0.07445739, 0.03323535],

[0.02469731, 0.01941632, 0.03323535, 0.03061985]]])}, 'Iris-virginica': {'ws': array([0.66175309, 0.33824691]), 'mus': array([[6.26540099, 2.83653358, 5.2370882 , 1.88375369],

[6.95897425, 3.1004617 , 5.92853442, 2.22742664]]), 'sigmas': array([[[ 0.23773686, 0.082196 , 0.11827915, 0.02351173],

[ 0.082196 , 0.07083198, 0.05422888, 0.03407846],

[ 0.11827915, 0.05422888, 0.10682318, 0.00558975],

[ 0.02351173, 0.03407846, 0.00558975, 0.07302456]],

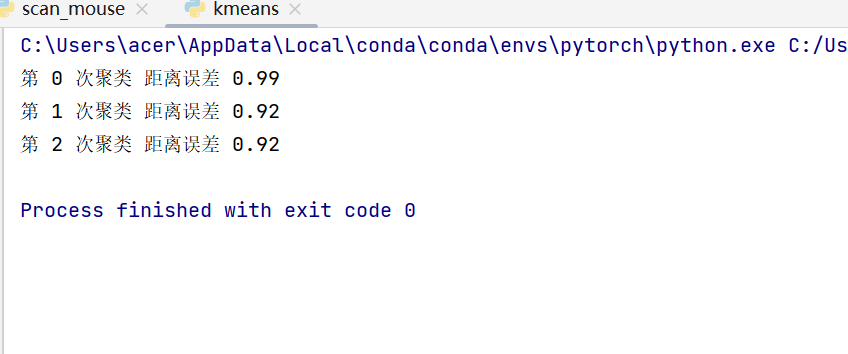
[[ 0.24462581, -0.0873946 , 0.1331243 , -0.07784959],

[-0.0873946 , 0.04033433, -0.05202738, 0.04019863],

[ 0.1331243 , -0.05202738, 0.13303732, -0.0489787 ],

[-0.07784959, 0.04019863, -0.0489787 , 0.05363386]]])}}

Kmeans结果

****

3. 写出调试的过程，说明测试用例及调试中遇到的主要问题和解决方法。

4. 写出实验收获与不足，以及对实验的相关意见。

K-Means采用欧式距离衡量数据点的差异，故它假设数据呈球状分布，每个维度的重要性一样；与之相比GMM使用更加一般的数据表示即高斯分布，它考虑到了协方差。