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：必修

实验题目：实现k-means聚类方法和混合高斯模型

学号：1190200501

姓名：林燕燕

一、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

实验要求

- 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

OS： Windows 11

Python： 3.7.9

三、设计思想

1. k-means算法

给定样本集 $D=\{X_1,X_2,...,X_m\}$ ，针对聚类所得簇划分 $C=\{C_1,C_2,...,C_k\}$ ，最小化平方误差：

$$E=\sum_{i=1}^{k}\sum_{X\in C_i}||X-\mu_i||_2^2=\sum_{i=1}^{k}\sum_{X\in C_i}\sum_{j=1}^{n}(x_j-\mu_{ij})^2$$

其中 $\mu_i=\frac{1}{|C_i|}\sum_{X\in C_i}X$ ，是簇 C_i 的均值向量，$X=\{x_1,x_2,...x_n\}$。

假定聚类簇数 $k=3$ ，随机选取三个样本 X_1,X_2,X_3 作为初始均值向量，即 $\mu_1=X_1,\mu_2=X_2,\mu_3=X_3$ ，

考察样本 X_i ，分别计算与均值向量之间的距离，将其划分进距离最短的聚类中，考察所有样本，可得当前簇划分 C_1,C_2,C_3 ，

对 C_1,C_2,C_3 分别求出新的均值向量 $\mu\grave{}_1,\mu\grave{}_2,\mu\grave{}_3$ ，继续重复上一过程，直到迭代结果不变，得到最终簇划分。

2. 混合高斯模型（GMM）

定义高斯混合分布：

$$p_M(x)=\sum_{i=1}^{k}\alpha_i\cdot p(x|\mu_i,\Sigma_i)$$

其中 $\sum_{i=1}^{k}\alpha_i=1$, α_i 为混合系数

$$p(x|\mu_i,\Sigma_i)=\frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma_i|^{\frac{1}{2}}}e^{-\frac{1}{2}(x-\mu_i)^T(x-\mu_i)\Sigma_i^{-1}}$$

训练集 $D=\{x_1,x_2,\dots x_m\}$ 由高斯混合分布生成，令 $z_j\in\{1,2,\dots,k\}$ 表示生成样本 x_j 的高斯混合成分，

则 z_j 的先验概率 $P(z_j=i)=\alpha_i$ ，由贝叶斯定理，z_j 的后验分布为：

$$p_M(z_j=i|x_j)=\frac{P(z_j=i)\cdot p_M(x_j|z_j=i)}{p_M(x_j)}$$

记 $\gamma_{ji}=p_M(z_j=i|x_j)$ ，为样本 x_j 对第 i 个高斯混合成分的后验概率

把样本集 D 划分为 k 个簇 $C=\{C_1,C_2,\dots,C_k\}$ ，每个样本 x_j 的簇标记为 $\lambda_j=\arg_{i\in\{1,2,\dots,k\}}\max\gamma_{ji}$

最大化对数似然求解：

$$LL(D)=ln(\prod_{i=1}^{m}p_M(x_j))=\sum_{j=1}^{m}ln(\sum_{i=1}^{k}\alpha_i\cdot p(x_j|\mu_i,\Sigma_i))$$

对 μ_i 求导 $\frac{\partial LL(D)}{\partial\mu_i}=0$ ：

$$\sum_{j=1}^{m}\frac{\alpha_i\cdot p(x_j|\mu_i,\Sigma_i)}{\sum_{l=1}^{k}\alpha_l\cdot p(x_j|\mu_l,\Sigma_l)}(x_j-\mu_i)=0$$

可得：

$$\mu_i=\frac{\sum_{j=1}^{m}\gamma_{ji}x_j}{\sum_{j=1}^{m}\gamma_{ji}}$$

同理，对 Σ_i 求导 $\frac{\partial LL(D)}{\partial\Sigma_i}=0$ ，可得：

$$\Sigma_i=\frac{\sum_{j=1}^{m}\gamma_{ji}(x_j-\mu_i)(x_j-\mu_i)^T}{\sum_{j=1}^{m}\gamma_{ji}}$$

需满足$\sum_{i=1}^{k}\alpha_i=1$，$\alpha_i\geq 0$ ，则考虑 $LL(D)$ 拉格朗日形式

$$LL(D)+\lambda(\sum_{i=1}^{k}\alpha_i-1)$$

对 α_i 求导：

$$\sum_{j=1}^{m}\frac{p(x_j|\mu_i,\Sigma_i)}{\sum_{l=1}^{k}\alpha_l\cdot p(x_j|\mu_l,\Sigma_l)}+\lambda=0$$

两边同时乘以 α_i 得：$\sum_{j=1}^{m}p_M(z_j=i|x_j)+\alpha_i\lambda=0$

对 i 求和得：$\sum_{i=1}^{k}\sum_{j=1}^{m}p_M(z_j=i|x_j)+\sum_{i=1}^{k}\alpha_i\lambda=0$ ，可得 $\lambda=-m$ ，代入上一式得：

$$\alpha_i=\frac{1}{m}\sum_{j=1}^{m}\gamma_{ji}$$

迭代过程：假定高斯混合成分个数 $k=3$ ，将高斯混合分布参数初始化，混合成分概率 $\alpha_1=\alpha_2=\alpha_3=\frac{1}{3}$ ，均值向量 $\mu_1=x_1,\mu_2=x_2,\mu_3=x_3$ ，协方差矩阵 $\Sigma_1=\Sigma_2=\Sigma_3=\begin{pmatrix}0.1 & 0.0\\0.0 & 0.1\end{pmatrix}$ ，

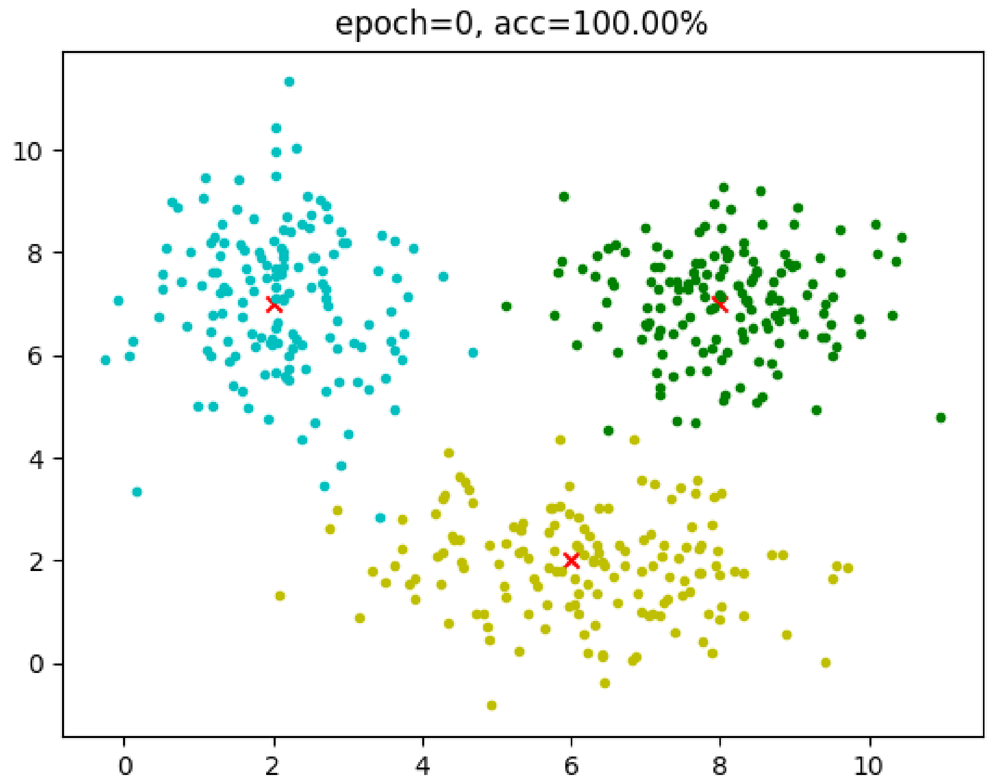
对样本 x_j ，计算后验概率 $\gamma_{j1},\gamma_{j2},\gamma_{j3}$ ，

计算完所有样本后，得到新的模型参数 $\alpha_1^`,\alpha_2^`,\alpha_3^`$，$\mu_1^`,\mu_2^`,\mu_3^`$ ，$\Sigma_1^`,\Sigma_2^`,\Sigma_3^`$ ，重复上一过程。

四、实验结果分析

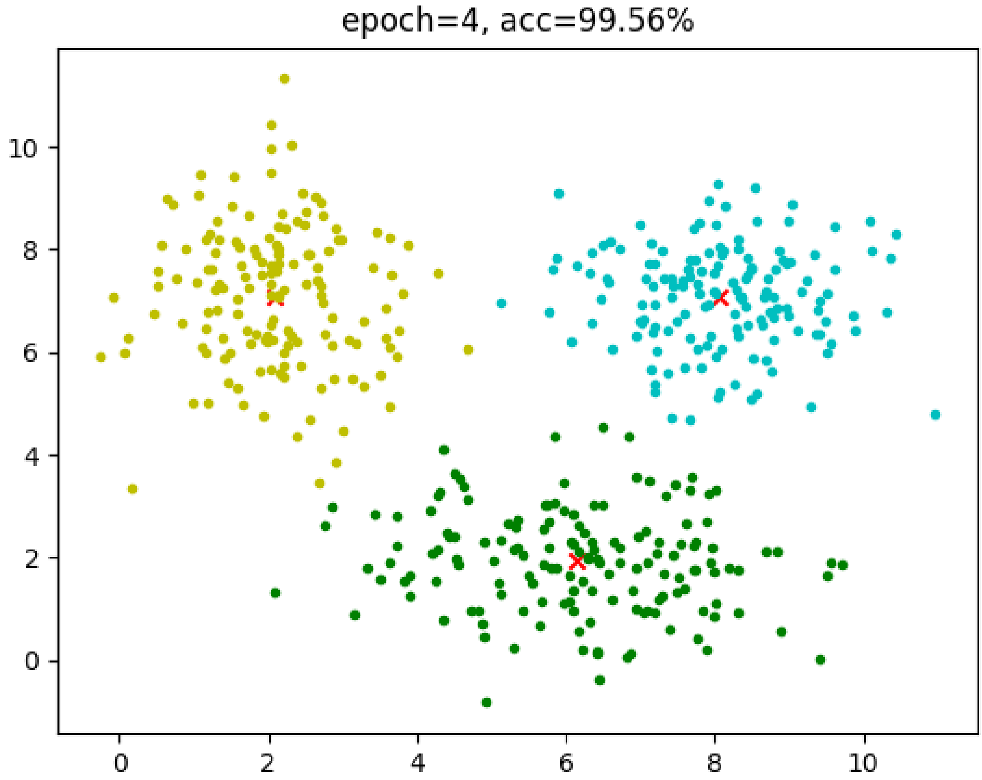
1.生成数据

将类别数 k 设置为 3，均值向量分别为 $(2,7)$、$(6,2)$ 和 $(8,7)$，协方差矩阵分别为 $\begin{pmatrix}1 & 0\\0 & 2\end{pmatrix}$、$\begin{pmatrix}2 & 0\\0 & 1\end{pmatrix}$ 和 $\begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}$，每个类别的样本数设置为$150$，生成数据如下：



2.k-means算法

设置精确度为 10^{-7}，算法最大轮数为50，结果如下：

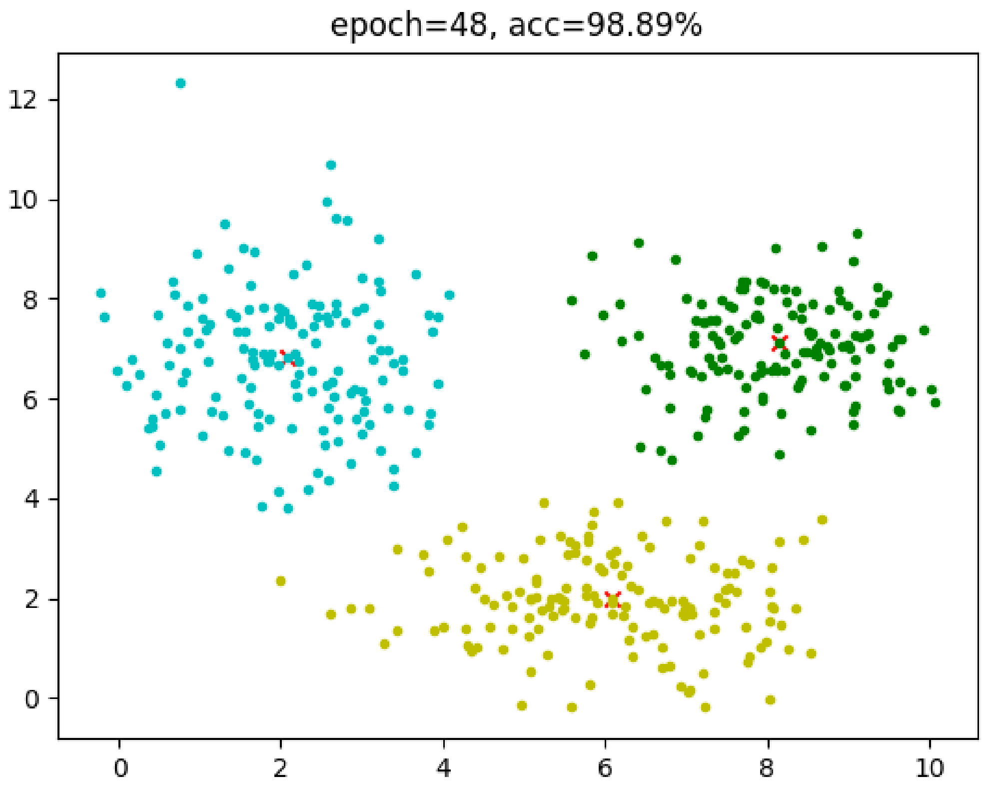


迭代轮数为 4 ，准确率为 99.56 。

可以看出k-means算法收敛速度快，准确率高。

3.GMM

设置精确度为 10^{-7} ，算法最大轮数为100，结果如下：



迭代次数为 48， 准确率为 98.89

在迭代48次后收敛，收敛速度快，准确率高。

4.UCI数据集

数据集中有 150 个数据，类别为 3， 使用k-means算法和GMM算法实现聚类

```
PS E:\Program\Machine Learning\3-GMM\Code> python .\uci.py
k_means: epoch=9, acc=88.67%
GMM: epoch=38, acc=96.67%
```

> k_means: epoch=9, acc=88.67%
>
> GMM: epoch=38, acc=96.67%

在UCI数据集上，可以看出GMM算法比k-means准确率更高，但收敛速度较慢

五、结论

k-means算法较易理解与实现，在简单数据集上收敛较快；

GMM的实现复杂，推导繁琐，在各种数据集上都能取得良好的效果，但收敛速度较k-means缓慢。

六、参考文献

《机器学习》

[人人都懂EM算法]

七、附录:源代码(带注释)

```python
####################data.py####################
import numpy as np
import matplotlib.pyplot as plt
from itertools import permutations

# 获取数据
def data_generator(k, size, loc, cov):
    X = np.zeros((k * size, loc.shape[1]))
    Y = np.zeros((k * size), dtype=np.int64)
    for i in range(k):
        X[i*size: (i+1)*size] = np.random.multivariate_normal(loc[i], cov[i], size)
        Y[i*size: (i+1)*size] = i
    center = np.zeros((k, loc.shape[1]))
    C = np.zeros((k * size), dtype=np.int64)
    shuffle = np.random.permutation(X.shape[0])
    X, Y = X[shuffle], Y[shuffle]
    # draw(k, X, Y, Y, loc, 0)
    return X, Y, C, center

# 计算准确度
def get_accuracy(k, Y, C):
    max_acc = 0
    classes = list(permutations(range(k), k)) # 获取分类的全排列
    for i in range(len(classes)):
        acc = 0
```

```python
        for j in range(len(Y)):
            if(Y[j] == classes[i][C[j]]):
                acc += 1
        acc /= len(Y)
        # print(acc)
        max_acc = max(max_acc, acc)
    return max_acc * 100

# 画图
def draw(k, X, Y, C, center, epoch):
    colors = ['c','y', 'g', 'b']
    for i in range(X.shape[0]):
        plt.plot(X[i, 0], X[i, 1], '.', c = colors[C[i]])
    plt.scatter(center[:,0],center[:,1], marker = 'x', c = 'r')
    plt.title('epoch={}, acc={:.2f}%'.format(epoch, get_accuracy(k, Y, C)))
    plt.show()

####################k_means.py####################
import numpy as np

from data import *

def kmeans(k, X, C, center, epochs=50, epsilon=1e-7):
    """
    K-means算法实现，算得分类结果和中心
    """
    # 初始化均值向量
    for i in range(k):
        center[i] = X[i]
    distance = np.zeros(k)
    epoch = 0
    # 开始迭代
    for t in range(epochs):
        # 计算距离，划分进距离最小得簇
        for j in range(X.shape[0]):
            for i in range(k):
                distance[i] = np.linalg.norm(X[j]-center[i])
            C[j] = np.argmin(distance) # 划分至距离最短处
        # 计算新的均值向量
        count = np.zeros((k),dtype=np.int64)
        new_center = np.zeros((k, X.shape[1]))
        for j in range(X.shape[0]):
            new_center[C[j]] += X[j]
            count[C[j]] += 1  # 簇中元素个数
        for i in range(k):
            new_center[i] = new_center[i] / count[i]
        # 判断是否符合结束迭代条的件
        if(np.linalg.norm(new_center - center) < epsilon):
            center = new_center
            epoch = t
            break
        # 更新均值向量
        center = new_center
    return epoch, center, C # 返回迭代次数 均值向量 簇划分



if __name__ == "__main__":
```

```
    k = 3
    size = 150
    loc = np.array([[2, 7], [6, 2], [8, 7]])
    cov = np.array([[[1, 0], [0, 2]], [[2, 0], [0, 1]], [[1, 0], [0, 1]]])
    X, Y, C, center = data_generator(k, size, loc, cov)
    # print(X.shape[0], X.shape[1], Y.shape[0]) # 450, 2   450
    epoch, center, C = kmeans(k, X, C, center)
    draw(k, X, Y, C, center, epoch)

#####################Gmm.py#####################
import numpy as np
from scipy.stats import multivariate_normal

from data import *

def GMM(k, X, Y, C, center, epochs=100, epsilon=1e-7):
    # 初始化高斯混合成分概率
    _alpha = np.ones(k) * (1.0 / k)
    # 初始化均值向量
    center = X[:k]
    # print(center)
    # 初始化协方差矩阵
    _sigma = np.array([0.1 * np.eye(X.shape[1])] * k)
    # print(_sigma[0])
    likelihood = 0
    for t in range(epochs):
        # 计算后验概率 E步
        _gamma = cal_gamma(k, X, _alpha, _sigma, center)

        # print(_gamma[0])
        # 划分簇
        C = np.argmax(_gamma, axis=1)
        # M步 计算新参数值
        new_alpha = cal_alpha(_gamma, _alpha)
        new_sigma = cal_sigma(X, _gamma, center, _sigma)
        new_center = cal_center(X, _gamma, center)
        # 计算极大对数似然
        new_likelihood = log_likelihood(X, center, _sigma, _alpha)
        # 判断退出条件
        if abs(new_likelihood - likelihood) < epsilon:
            epoch = t
            break
        _alpha = new_alpha
        _sigma = new_sigma
        center = new_center
        likelihood = new_likelihood
    return epoch, C, center

def cal_gamma(k, X, _alpha, _sigma, _mu, epsilon=1e-7):
    _gamma = np.zeros((X.shape[0], k))
    # print(_sigma)
    for i in range(X.shape[0]):
        sum = 0
        p = 0
        for j in range(k):
            p = multivariate_normal.pdf(X[i], mean=_mu[j], cov=_sigma[j])
            if abs(p) < epsilon:
                p += epsilon
```

```python
                _gamma[i][j] = _alpha[j] * p
                sum += _gamma[i][j]
            for j in range(k):
                _gamma[i][j] /= sum
        return _gamma

    def cal_alpha(_gamma, _alpha):
        m = _gamma.shape[0]
        k = _gamma.shape[1]
        for i in range(k):
            for j in range(m):
                _alpha[i] += _gamma[j][i]
            _alpha[i] /= m
        return _alpha

    def cal_center(X, _gamma, center):
        m = _gamma.shape[0]
        k = _gamma.shape[1]
        for i in range(k):
            sum = 0
            for j in range(m):
                sum += _gamma[j][i]
                center[i] += _gamma[j][i] * X[j]
            center[i] /= sum
        return center

    def cal_sigma(X, _gamma, center, _sigma):
        _sigma = np.zeros_like(_sigma)
        m = _gamma.shape[0]
        k = _gamma.shape[1]
        for i in range(k):
            sum = 0
            for j in range(m):
                sum += _gamma[j][i]
                v = (X[j]-center[i]).reshape(-1, 1)
                _sigma[i] += _gamma[j][i] * np.dot(v,v.T)
            _sigma[i] /= sum
        return _sigma

    def log_likelihood(X, _mu, _sigma, _alpha):
        """
        计算极大似然对数
        """
        log = 0
        for i in range(X.shape[0]):
            pi_times_pdf_sum = 0
            for j in range(_mu.shape[0]):
                pi_times_pdf_sum += _alpha[j] * multivariate_normal.pdf(X[j],
    mean=_mu[j], cov=_sigma[j])
            log += np.log(pi_times_pdf_sum)
        return log

    if __name__ == '__main__':
        k = 3
        size = 150
        loc = np.array([[2, 7], [6, 2], [8, 7]])
        cov = np.array([[[1, 0], [0, 2]], [[2, 0], [0, 1]], [[1, 0], [0, 1]]])
        X, Y, center, C = data_generator(k, size, loc, cov)
```

```
    epoch, C, center = GMM(k, X, Y, C, center)
    draw(k, X, Y, C, center, epoch)


###################uci.py###################
import numpy as np

from data import *
from k_means import *
from Gmm import *

def get_data():
    with open('bezdekIris.data', 'r') as f:
        lines = [l[:-1].split(',') for l in f.readlines()]
    np.random.shuffle(lines)
    label_map = {'Iris-setosa':0, 'Iris-versicolor':1, 'Iris-virginica':2}
    X = np.array([[float(l[i]) for i in range(len(l) - 1)] for l in lines])
    Y = np.array([label_map[l[i]] for l in lines for i in range(len(l) - 1,
len(l))])
    return X, Y


if __name__ == '__main__':
    X, Y = get_data()
    center = np.zeros((3, X.shape[1]))
    C = np.zeros((X.shape[0]), dtype=np.int64)
    epoch, center, C = kmeans(3, X, C, center)
    print('k_means: epoch={}, acc={:.2f}%'.format(epoch, get_accuracy(3, Y,
C)))
    epoch, C, center = GMM(3, X, Y, C, center)
    print('GMM: epoch={}, acc={:.2f}%'.format(epoch, get_accuracy(3, Y,
C)))
```