模式识别实验报告

实验二 GMM 分类器

学院：计算机学院

姓名：张景润

学号：1172510217

一、实验内容

1、 使用 Python 或 Matlab 编程实现 GMM 算法：要求独立完成算法编程，禁止调用已有函数库或工具箱中的函数；

2、 使用仿真数据测试算法的正确性：两类 2 维各 1000 个训练样本 Train1 和 Train2 分别采样自如下两个 GMM，使用训练样本分别估计包含 2 个分量高斯的 GMM 参数。

GMM1： $\alpha_1 = \frac{2}{3}$， $\boldsymbol{\mu}_1 = (0,0)^t$， $\boldsymbol{\Sigma}_1 = \begin{pmatrix} 3 & 1 \\ 1 & 1 \end{pmatrix}$

$\alpha_2 = \frac{1}{3}$， $\boldsymbol{\mu}_2 = (10,10)^t$， $\boldsymbol{\Sigma}_2 = \begin{pmatrix} 2 & 2 \\ 2 & 5 \end{pmatrix}$

GMM2： $\alpha_1 = \frac{2}{3}$， $\boldsymbol{\mu}_1 = (2,10)^t$， $\boldsymbol{\Sigma}_1 = \begin{pmatrix} 1 & 1 \\ 1 & 3 \end{pmatrix}$

$\alpha_2 = \frac{1}{3}$， $\boldsymbol{\mu}_2 = (15,20)^t$， $\boldsymbol{\Sigma}_2 = \begin{pmatrix} 5 & 2 \\ 2 & 1 \end{pmatrix}$

构造区分两类的 GMM 分类器，测试采样自同样 GMM 的测试样本 Test1 和 Test2。

3、 MNIST 数据集测试：使用 TrainSamples 中的 30000 个 17 维特征手写数字样本训练 GMM 分类器区分 10 个类别，TrainLabels 中包含训练样本的标签；测试设置不同高斯数量 GMM 分类器对 TestSamples 中 10000 个样本的识别正确率。

二、程序代码

（GMM 参数估计部分和 GMM 分类器部分代码）

```
class GMM:
    def __init__(self, gauss_num):
        self.gauss_num = gauss_num  # 高斯混合分布的数目
        self.alphas_np = None  # 每一个高斯分布的权重
        self.means_np = None  # 各维变量的均值
        self.covs_np = None  # 协方差矩阵

    def init_params(self, data_np):  # 初始化高斯分布的三部分参数
        data_shuffled = data_np.copy()
        np.random.shuffle(data_shuffled)
        data_split = np.array_split(data_shuffled, self.gauss_num)
        self.alphas_np = np.repeat(1.0 / self.gauss_num, self.gauss_num)
        self.means_np = np.array([np.mean(data_split[i], axis=0) for i in range(self.gauss_num)])
        self.covs_np = np.array([np.cov(data_split[i].T) for i in range(self.gauss_num)])

    def pdf(self, x_np, idx):  # 概率密度计算
        mean_np = self.means_np[idx]
        cov_np = self.covs_np[idx]
        left = 1 / np.sqrt(np.power(2 * np.pi, x_np.shape[0]) * np.linalg.det(cov_np))
```

```
        right = np.exp(-0.5 * np.dot(np.dot((x_np - mean_np).T, np.linalg.inv(cov_np)), x_np - mean_np))
        return left * right

    def fit(self, data_np, fit_num, eps):  # EM 过程计算参数
        self.init_params(data_np)
        data_num = data_np.shape[0]  # 数据数目
        norm_densities = np.empty((data_num, self.gauss_num), np.float)
        responsibilities = np.empty((data_num, self.gauss_num), np.float)
        pre_log_likelihood = 0
        for idx in range(fit_num):
            for i, one_data in enumerate(data_np):
                for j in range(self.gauss_num):
                    norm_densities[i][j] = self.pdf(one_data, j)
            log_vec = np.log(np.array([np.dot(self.alphas_np, norm_density) for norm_density in norm_densities]))
            log_likelihood = log_vec.sum()
            if abs(log_likelihood - pre_log_likelihood) < eps:
                break
            for i in range(data_num):
                normalizer = np.dot(self.alphas_np.T, norm_densities[i])
                for j in range(self.gauss_num):
                    responsibilities[i][j] = self.alphas_np[j] * norm_densities[i][j] / normalizer
            for i in range(self.gauss_num):
                responsibility = responsibilities.T[i]
                normalizer = np.dot(responsibility, np.ones(data_num))
                self.alphas_np[i] = normalizer / data_num
                self.means_np[i] = np.dot(responsibility, data_np) / normalizer
                diff = data_np - np.tile(self.means_np[i], (data_num, 1))
                self.covs_np[i] = np.dot((responsibility.reshape(data_num, 1) * diff).T, diff) / normalizer
            pre_log_likelihood = log_likelihood
            print('第%d 次迭代' % (idx + 1))
        print(self.alphas_np)
        print(self.means_np)
        print(self.covs_np)
class Classifier:
    def __init__(self, gmm_lst, weight_lst):
        self.gmm_lst = gmm_lst
        self.priors = weight_lst
        self.classes = len(weight_lst)
```

```python

    def classify(self, data_np, label):
        data_num = data_np.shape[0]
        if isinstance(label, int):
            label = np.full((data_num,), label)
        log_vec = np.empty((self.classes, data_num), dtype=np.float)
        for idx, gmm in enumerate(self.gmm_lst):
            norm_densities = np.empty((data_num, gmm.gauss_num), np.float)
            for i in range(data_num):
                for j in range(gmm.gauss_num):
                    norm_densities[i][j] = gmm.pdf(data_np[i], j)
            log_vec[idx] = np.array([np.dot(gmm.alphas_np, norm_density) for norm_density in norm_densities]) * \
                               self.priors[idx]
        predict_np = np.argmax(log_vec, axis=0)
        right_num = (predict_np == label.reshape(predict_np.shape)).sum()
        accuracy = right_num / data_num
        print('[%d/%d]=%.2f%%' % (right_num, data_num, accuracy * 100))
```

三、实验结果

1、 仿真数据实验结果：给出估计出的两 GMM 模型参数，以及测试样本的识别结果。

GMM 估计模型参数

	α	μ	Σ
GMM1-Gauss1	0.65890605	[-0.04877182, 0.03493031]	[2.85162688,0.97072797] [0.97072797,0.96895055]
GMM1-Gauss2	0.34109395	[9.97026666, 9.95347321]	[2.01631378,2.35543427] [2.35543427,5.31829801]
GMM2-Gauss1	0.66799982	[2.02206041 10.16700955]	[0.96728744,0.91060643] [0.91060643,2.74892244]
GMM2-Gauss2	0.33200018	[14.97097406 19.99245787]	[5.288094,2.18044657] [2.18044657,1.12285599]

GMM 分类器识别结果

	正确识别数	正确识别率
Test1	1000	100%
Test2	1000	100%

2、 MNIST 数据集实验结果：

GMM 分类器识别正确率

高斯数	1	2	3	4	5
正确识别数	9332	9421	9547	9491	9526
正确识别率	93.32%	94.21%	95.47%	94.91%	95.26%