模式识别实验报告

实验二 GMM 分类器

学院：计算机科学与技术

姓名：张文强

学号：18S003044

一、实验内容

1、使用 C 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, K, eps=1e-6):
    self.K = K
    self.eps = eps

def gaussian_pdf(self, x, mean, cov):
    # x :  d x 1
    # mean : d x 1
    # cov: d x d
    centered = x - mean
    cov_inv = np.linalg.inv(cov)
    cov_det = np.linalg.det(cov)
    exponent = np.dot(np.dot(centered.T, cov_inv), centered)
    return np.exp(-0.5 * exponent) / np.sqrt(cov_det *
np.power(2 * np.pi, len(mean)))  # here

def init_params(self, data):
    data_shuffled = data.copy()
    np.random.shuffle(data_shuffled)


    # (n_data//K, dim_data)
    data_splited = np.array_split(data_shuffled, self.K)
```

```
        # estimate means & covs & alphas
        self.means = np.array([np.mean(data_splited[i], axis=0) for
i in range(self.K)])
        self.covs = np.array([np.cov(data_splited[i].T) for i in
range(self.K)])
        self.alphas = np.repeat(1.0 / self.K, self.K)  # (self.K, )

    def EM(self, data):
        n_data = data.shape[0]

        norm_densities = np.empty((n_data, self.K), np.float)
        responsibilities = np.empty((n_data, self.K), np.float)

        old_log_likelihood = 0

        self.init_params(data)

        iteration = 0
        while True:
            for i in range(n_data):
                x = data[i]
                for j in range(self.K):
                    norm_densities[i][j] = self.gaussian_pdf(x,
self.means[j], self.covs[j])

            # log likelihood
            log_vector = np.log(np.array([np.dot(self.alphas,
norm_densities[i]) for i in range(n_data)]))
            log_likelihood = log_vector.sum()



            # print ('loss: %s' % abs(old_log_likelihood -
log_likelihood))
            if abs(old_log_likelihood - log_likelihood) < self.eps:
                break

            # E-step: estimate y
            for i in range(n_data):
                x = data[i]
                normalizer = np.dot(self.alphas.T,
norm_densities[i])
                for j in range(self.K):
```

```
                responsibilities[i][j] = self.alphas[j] *
norm_densities[i][j] / normalizer

          # M-step: re-estimate the params
          for i in range(self.K):
              responsibility = (responsibilities.T)[i]

              normalizer = np.dot(responsibility, np.ones(n_data))

              self.alphas[i] = normalizer / n_data
              self.means[i] = np.dot(responsibility, data) /
normalizer
              diff = data - np.tile(self.means[i], (n_data, 1))
              # pdb.set_trace()
              self.covs[i] =
np.dot((responsibility.reshape(n_data, 1) * diff).T, diff) /
normalizer

          old_log_likelihood = log_likelihood
          iteration += 1
          print ('Iter : %d' % iteration)

    def display_result(self):
        print ('alphas:', self.alphas)
        print ('means:', self.means)
        print ('covs:', self.covs)


class Classifier:
    def __init__(self, gmms, priors):
        self.gmms = gmms
        self.priors = priors
        self.classes = len(priors)


    def classify(self, data, label):
        # data  n x d
        # label n
        n_data = data.shape[0]
        if isinstance(label, int):
            label = np.full((n_data,), label)
        log_vectors = np.empty((self.classes, n_data),
dtype=np.float)
        for idx, gmm in enumerate(self.gmms):
```

```
            norm_densities = np.empty((n_data, gmm.K), np.float)
            for i in range(n_data):
                for j in range(gmm.K):
                    norm_densities[i][j] = gmm.gaussian_pdf(data[i],
gmm.means[j], gmm.covs[j])

            # log_vectors[idx] =
np.log(np.array([np.dot(gmm.alphas, norm_densities[i]) for i in
range(n_data)]) * self.priors[idx])
            log_vectors[idx] = np.array([np.dot(gmm.alphas,
norm_densities[i]) for i in range(n_data)]) * self.priors[idx]

        predict = np.argmax(log_vectors, axis=0)
        n_correct = (predict == label).sum()
        accuracy = n_correct / n_data
        print ('[%d/%d]=%.2f%%' % (n_correct, n_data, accuracy *
100))
```

三、实验结果

1、 仿真数据实验结果：给出估计出的两个 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	0.658906	(-0.0487716, -0.03493012)	[[2.85162771 0.97072874] [0.97072874 0.96895128]
GMM1-Gauss2	0.341094	(9.9702672, 9.9534738)	[[2.01631084 2.35543114] [2.35543114 5.31829474]]
GMM2-Gauss1	0.332000	(14.97097406, 19.99245787)	[[5.28809397 2.18044656] [2.18044656 1.12285598]]
GMM2-Gauss2	0.668000	(2.02206041, 10.16700955)	[[0.96728744 0.91060643] [0.91060643 2.74892245]]

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	9429	9511	9502	9532
正确识别率	93.32%	94.29%	95.11%	95.02%	95.32%