模式识别实验报告

161440110 张涛,1614402 姚韫玉，1614401 高文钊

2017 年 11 月 21 日

摘要

模式识别是在某些一定量度或观测基础上把待识模式划分到各自的模式类中去。计算机模式识别就是指利用计算机等装置对物体，图像等信息进行自动的识别。实验使用 mnist 数据集 [6] 和 iris[4] 数据集，分类算法使用传统的卷积神经网络模型模型 cnn[2] 和最近 hinton 提出的 capsule 架构 [5]。比较两者在 mnist 数据集上的表现。聚类算法使用 k-means[?] 和混合高斯分布 GMM[?] 在 iris 数据集上做聚类，比较两个算法在此数据集上的表现

1 数据集介绍

实验采用的数据集是机器学习领域最基础的两个数据集，也是很多实验中常用的数据集，本次使用也使用这两个数据集作为基准，比较分类和聚类算法的性能。

1.1 mnist[6]

MNIST 数据集可在http://yann.lecun.com/exdb/mnist/获取, 它包含了四个部分:

- Training set images: train-images-idx3-ubyte.gz (9.9 MB, 解压后 47 MB, 包含 60,000 个样本)
- Training set labels: train-labels-idx1-ubyte.gz (29 KB, 解压后 60 KB, 包含 60,000 个标签)
- Test set images: t10k-images-idx3-ubyte.gz (1.6 MB, 解压后 7.8 MB, 包含 10,000 个样本)
- Test set labels: t10k-labels-idx1-ubyte.gz (5KB, 解压后 10 KB, 包含 10,000 个标签)

数据是 28*28 的图片，图片中是手写体的数字。每张图片都有对应的类标，表示图片上的数字是几，如图所示 任务就是训练模型，识别图片中的数据。

1.2 iris

Iris 也称鸢尾花卉数据集，是一类多重变量分析的数据集。通过花萼长度，花萼宽度，花瓣长度，花瓣宽度 4 个属性预测鸢尾花卉属于（Setosa，Versicolour，Virginica）三个种类中的哪一类。

2 算法介绍

2.1 分类算法

分类算法是处理有标签数据的问题，在测试集上训练模型，然后在测试集上结合真实类标评估分类算法的性能。下面的实验中是使用 mnist 数据集。

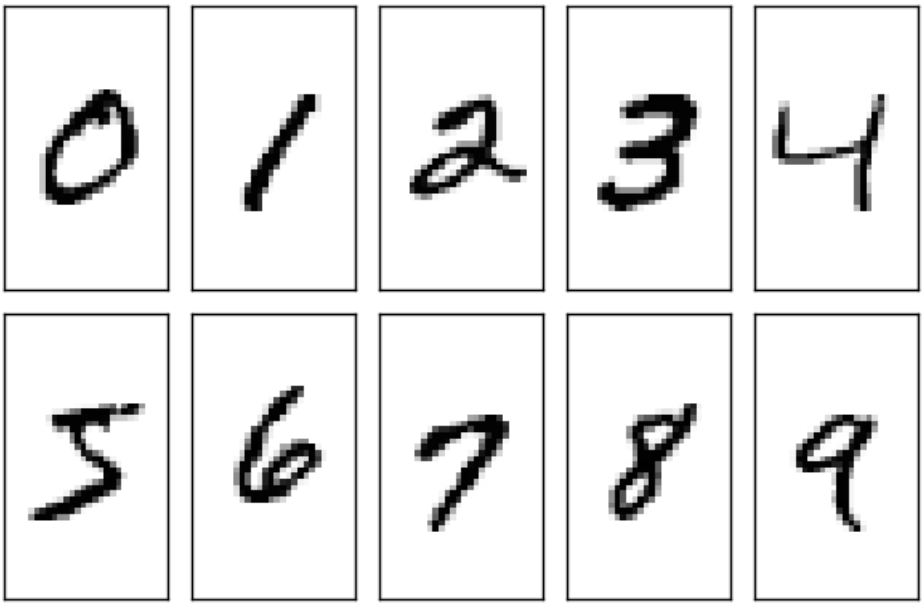

图 1.1: mnist 图片示意

数据集特征:	多变量	**记录数:**	150
属性特征:	实数	**属性数目:**	4
相关应用:	分类	**缺失值?**	无

表 1: iris 数据属性介绍

2.1.1 convolution neural network

卷积神经网络（Convolutional Neural Network,CNN）[1] 是一种前馈神经网络，它的人工神经元可以响应一部分覆盖范围内的周围单元，对于大型图像处理有出色表现. 它包括卷积层 (convolutional layer) 和池化层 (pooling layer). 卷积神经网络是近年发展起来，并引起广泛重视的一种高效识别方法.

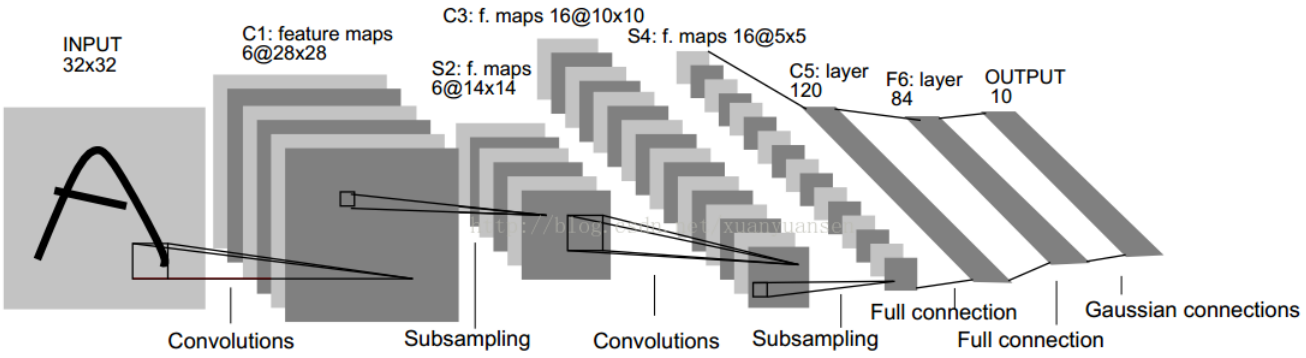

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

图 2.2: LeNet5

2.1.2 capsule

Capsule 是一组神经元，其输入输出向量表示特定实体类型的实例化参数（即特定物体、概念实体等出现的概率与某些属性）。我们使用输入输出向量的长度表征实体存在的概率，向量的方向表示实例化参数（即实体的某些图形属性）。同一层级的 capsule 通过变换矩阵对更高级别的 capsule 的实例化参数进行预测。当多个预测一致时（本论文使用动态路由使预测一致），更高级别的 capsule 将变得活跃。

Capsule 中的神经元的激活情况表示了图像中存在的特定实体的各种性质。这些性质可以包含很多种不同的参数，例如姿势（位置，大小，方向）、变形、速度、反射率，色彩、纹理等等。而输入输出向量的长度表示了某个实体出现的概率，所以它的值必须在 0 到 1 之间

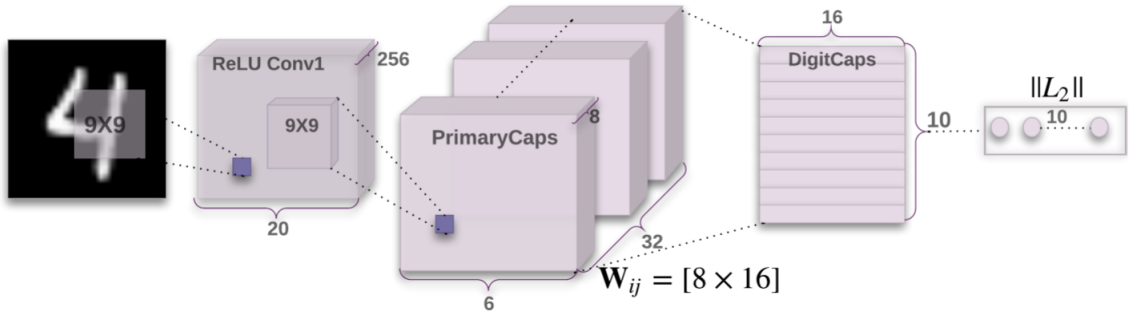

图 2.3: capsule 结构

2.2 聚类算法

聚类算法是一类无标签数据处理的方式，将所有的无标签数据根据不同的属性分成不同的类，本实验使用 iris 数据集，使用聚类算法进行分类，然后通过真实标签，评估不同的聚类算法的表现性能

2.2.1 k-means

K-means 算法是很典型的基于距离的聚类算法，采用距离作为相似性的评价指标，即认为两个对象的距离越近，其相似度就越大。该算法认为簇是由距离靠近的对象组成的，因此把得到紧凑且独立的簇作为最终目标。

k 个初始类聚类中心点的选取对聚类结果具有较大的影响，因为在该算法第一步中是随机的选取任意 k 个对象作为初始聚类的中心，初始地代表一个簇。该算法在每次迭代中对数据集中剩余的每个对象，根据其与各个簇中心的距离将每个对象重新赋给最近的簇。当考察完所有数据对象后，一次迭代运算完成，新的聚类中心被计算出来。如果在一次迭代前后，J 的值没有发生变化，说明算法已经收敛。

2.2.2 GMM

高斯模型就是用高斯概率密度函数（正态分布曲线）精确地量化事物，将一个事物分解为若干的基于高斯概率密度函数（正态分布曲线）形成的模型。

3 实验过程

实验环境:python2.7,pytorch0.4.0[1],scikit-learn0.19[3]

实验数据:mnist,iris

实验方法:cnn,capsule,k-means,GMM

3.1 分类算法实验

3.1.1 convolution neural network

网络架构:

```
Net(
  (conv1): Conv2d (1, 6, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (bn1): BatchNorm2d(6, eps=1e-05, momentum=0.1, affine=True)
  (pool): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1))
  (conv2): Conv2d (6, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True)
  (pool): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1))
  (fc1): Linear(in_features=784, out_features=120)
  (fc2): Linear(in_features=120, out_features=84)
  (fc3): Linear(in_features=84, out_features=10)
  (softmax):Softmax(out_features=10)
)
```

在模型中加入了 batch normalization, 加快了训练的过程. batch size 取 1024, 一共运行 300 个 epoch, 下图是 train loss 的变化情况

然后在测试集上做测试：accuracy of test:96.82%

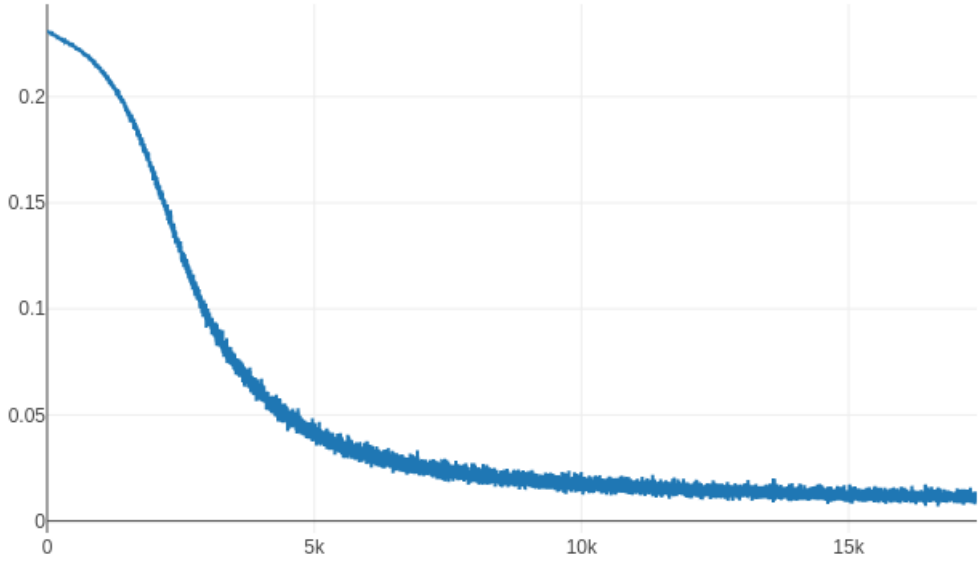

图 3.4: cnn 在 mnist 数据集上的 train loss

3.1.2 capsule

capsule 是 Hinton 提出的另一种神经网络架构，其思想就是将特征从传统的一个点转化为一个向量来表示更多的信息. Sabour 2017[5]，这篇文章中的主要亮点就是提出了 capsNet 的结构和 dynamic routing 的优化算法. 下面介绍一下 capsnet 和优化算法以及 pytorch 实现过程.

conv: 根据图 2 .3，第一层为传统的卷积层，9*9 的 kernel, 激活函数为 relu.

capsconv: 第二层是 capsconv, 也是使用传统的卷积操作进行特征提取，与之前不同的是，会在每一个 feature map 上做 8 次不同的卷积操作，然后合并成一个特征 map.

```
import torch
import torch.nn as nn
import utils
class ConvUnit(nn.Module):
  def __init__(self, in_channels):
        super(ConvUnit, self).__init__()
        self.conv0 = nn.Conv2d(in_channels=in_channels,
                               out_channels=32,  # fixme constant
                               kernel_size=9,  # fixme constant
                               stride=2, # fixme constant
                               bias=True)
    def forward(self, x):
        return self.conv0(x)
class CapsConv(nn.Module):
    def __init__(self,in_channels=256,out_dim=8):
        super(CapsConv,self).__init__()
        self.in_channels = in_channels
        self.out_dim = out_dim
        def create_conv_unit(unit_idx):
            unit = ConvUnit(in_channels=in_channels)
```

```
            self.add_module("unit_" + str(unit_idx), unit)
            return unit
        self.conv = [create_conv_unit(i) for i in range(self.out_dim)]
    def forward(self,x):
        #input x with shape ->[batch_size,in_features,height,width]
        #output with shape->[batch_size,32,6,6]
        x = [self.conv[i](x) for i in range(self.out_dim)]
        #output with shape->[batch_size,8,32,6,6]
        x = torch.stack(x,dim=1)
        #return shape->[batch_size,1152,8]
        x = utils.squash(x,dim=2)
        return x.view(x.size(0),self.out_dim,-1).transpose(1,2)
```

capsnet: 第三层进行特征转换从 [1152,8]->[10,16], 使用全连接，然后使用 dynamic routing 优化 c, 直接求 mean.

```
class CapsNet(nn.Module):
    """
    input :a group of capsule -> shape:[batch_size*1152(feature_num)*8(in_dim)]
    output:a group of new capsule -> shape[batch_size*10(feature_num)*16(out_dim)]
    """

    def __init__(self,in_features,out_features,in_dim,out_dim):
        """
        """
        super(CapsNet,self).__init__()
        #number of output features,10
        self.out_features = out_features
        #number of input features,1152
        self.in_features = in_features
        #dimension of input capsule
        self.in_dim = in_dim
        #dimension of output capsule
        self.out_dim = out_dim

        #full connect parameter W with shape [1(batch共享),1152,10,8,16]
        self.W = nn.Parameter(torch.randn(1,self.in_features,self.out_features,in_dim,out_dim))

    def forward(self,x):
        #input x,shape=[batch_size,in_features,in_dim]
        #[batch_size,1152,8]
        # (batch, input_features, in_dim) -> (batch, in_features, out_features,1,in_dim)
        x = torch.stack([x] * self.out_features, dim=2).unsqueeze(3)
        W = torch.cat([self.W] * conf.batch_size,dim=0)
        # u_hat shape->(batch_size,in_features,out_features,out_dim)=(batch,1152,10,1,16)
        u_hat = torch.matmul(x,W)
        #b for generate weight c,with shape->[1,1152,10,1]
        b = torch.zeros([1,self.in_features,self.out_features,1]).double()
        if self.cuda:
            b = b.cuda()
        b = Variable(b)
        for i in range(3):
            c = F.softmax(b,dim=2)
            #c shape->[batch_size,1152,10,1,1]
            c = torch.cat([c] * conf.batch_size, dim=0).unsqueeze(dim=4)
            #s shape->[batch_size,1,10,1,16]
            s = (u_hat * c).sum(dim=1,keepdim=True)
```

```
            #output shape->[batch_size,1,10,1,16]
            v = utils.squash(s,dim=-1)
            v_1 = torch.cat([v] * self.in_features, dim=1)
            #(batch,1152,10,1,16)matmul(batch,1152,10,16,1)->(batch,1152,10,1,1)
            #squeeze
            #mean->(1,1152,10,1)
            #print u_hat.shape,v_1.shape
            update_b = torch.matmul(u_hat,v_1.transpose(3,4)).squeeze(dim=4).mean(dim=0,keepdim=True)
            b = b+update_b
        return v.squeeze(1).transpose(2,3)
```

最后: 使用 separate margin loss，使用 bp 更新网络中的其他参数. 求向量范数作为概率 $p(y = y_i)$

实验结果: 图是训练过程 loss 的变化趋势，最终在测试集上的结果为 test accuracy:

3.2 聚类算法实验

3.2.1 k-means

实验描述: 实验使用 k-means 对 iris 数据集进行聚类实验，其中利用训练集类标的先验知识，对样本中心进行初始化加速 expectation maximization 算法的收敛

实验步骤:

1. 初始化样本中心， 3 维
2. 根据平方损失函数，计算距离，将数据分成 3 - 簇
3. 根据上一步的结果，更新样本中心
4. 回到第二步，直到簇不在发生变化为止

实验结果 (可视化 23 特征):

3.2.2 GMM

实验描述: 实验使用高斯混合模型对数据进行聚类分析，同样使用训练集的先验知识，对模型参数进行初始化，将数据分成 3/4 训练集,1/4 测试集，选择不同的初始化协方差形式.

实验步骤:

1. 初始化高斯模型均值 μ_k(根据样本标签先验计算均值初始化),协方差 Σ_k(spherical,diagonal,tied,full) 以及比例 $\pi_k(1/3)$
2. 根据优化目标 (极大化似然函数)，求导后可求出 $\gamma(Z_{mk})$, 表示样本 m 属于 k 类的概率
3. 根据 $\gamma(Z_{mk})$ 的值更新 μ_k, Σ_k, π_k
4. 重复 23 步骤，直到小于更新阈值为止

实验结果:

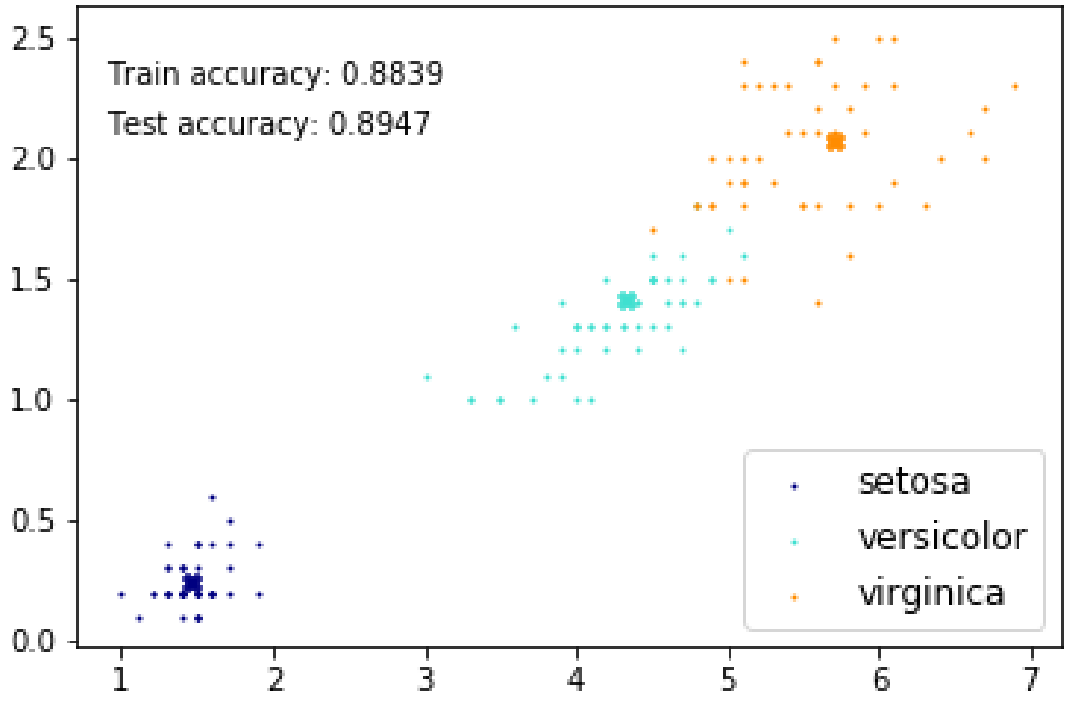

图 3.5: knn 在 iris 数据集上的结果

参考文献

[1] Colesbury Apaszke, Soumith. ”pytorch:deep learning in Python”. 2017. https://github.com/pytorch.

[2] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[4] Michael Marshall R.A. Fisher. ”iris dataset”. 1936.

[5] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. *arXiv preprint arXiv:1710.09829*, 2017.

[6] Christopher J.C. Burges Yann LeCun, Corinna Cortes. ”gradient-based learning applied to document recognition.”. November 1998.

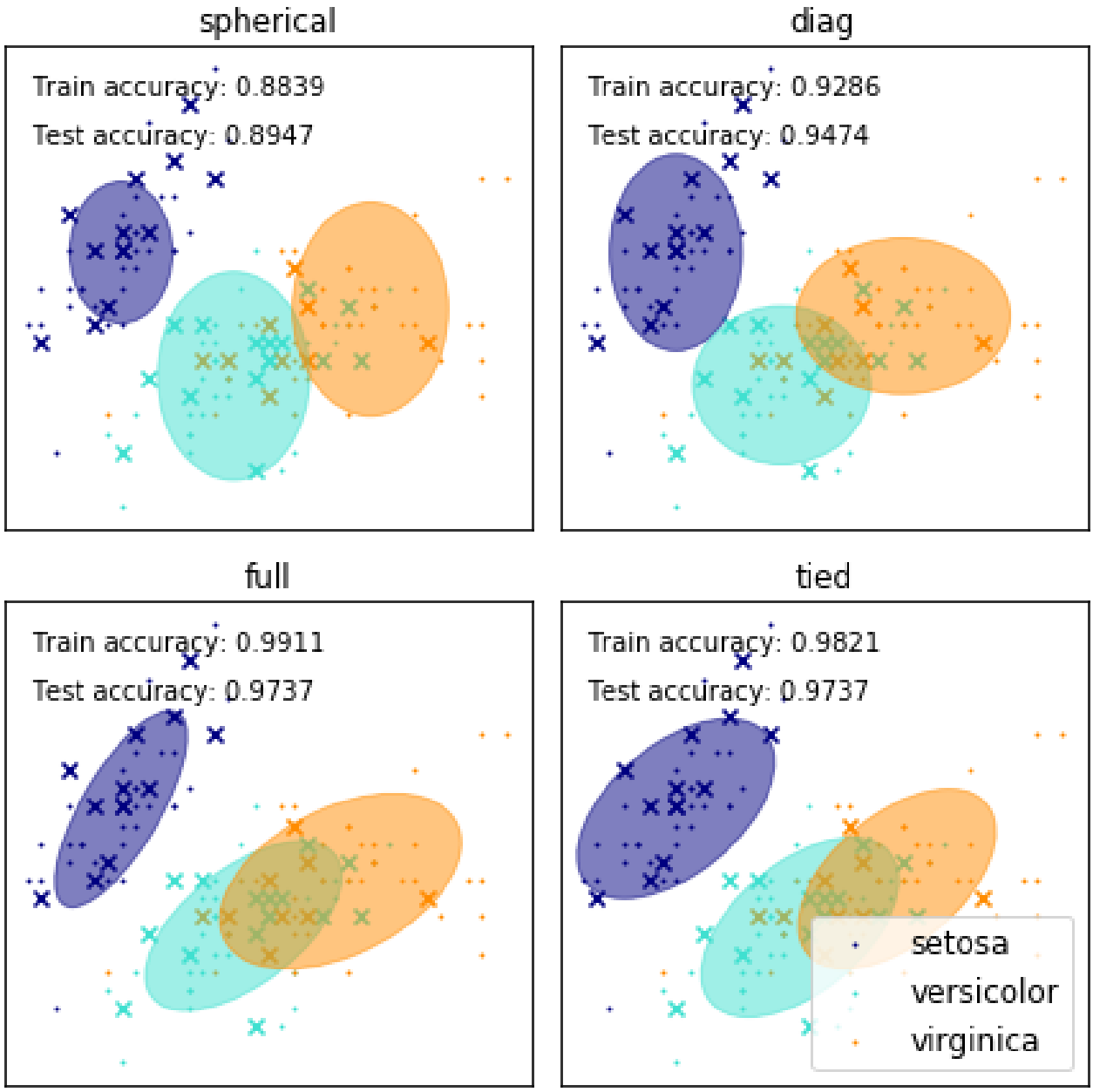

图 3.6: gmm 在 iris 数据集上的结果