33

Clustering Methods in Scikit-Learn

Scikit-Learn 聚类

K 均聚类、四种高斯混合 GMM 聚类

> 只有想象力无界的人，方能开创不可能的事。
>
> ***Those who can imagine anything, can create the impossible.***
>
> —— 艾伦·图灵 (Alan Turing) | 英国计算机科学家、数学家，人工智能之父 | 1912 ~ 1954

- ◄ matplotlib.patches.Ellipse() 创建并绘制椭圆形状的图形对象
- ◄ matplotlib.pyplot.quiver() 绘制向量箭头
- ◄ numpy.arctan2() 计算反正切，返回弧度值
- ◄ numpy.linalg.svd() 完成奇异值分解
- ◄ numpy.sqrt() 计算平方根
- ◄ sklearn.cluster.KMeans() 执行 K 均值聚类算法，将数据点划分成预定数量的簇
- ◄ sklearn.mixture.GaussianMixture() 用于拟合高斯混合模型，以对数据进行聚类和概率密度估计

33.1 聚类

本书前文介绍过，聚类 (clustering) 是无监督学习 (unsupervised learning) 中的一类问题。

聚类是指将数据集中的样本按照某种相似性指标进行分组的过程。常用的聚类算法包括。

如图 1 所示，删除鸢尾花数据集的标签，即 target，仅仅根据鸢尾花花萼长度 (sepal length)、花萼宽度 (sepal width) 这两个特征上样本数据分布情况，我们可以将数据分成两簇 (clusters)。

在机器学习中，决定将数据分成多少个簇是一个重要而且有挑战性的问题，通常称为聚类数目的选择或者簇数选择。不同的聚类算法可能需要不同的方法来确定合适的聚类数目。本章后文在介绍具体算法时，会介绍如何选择合适的簇数。

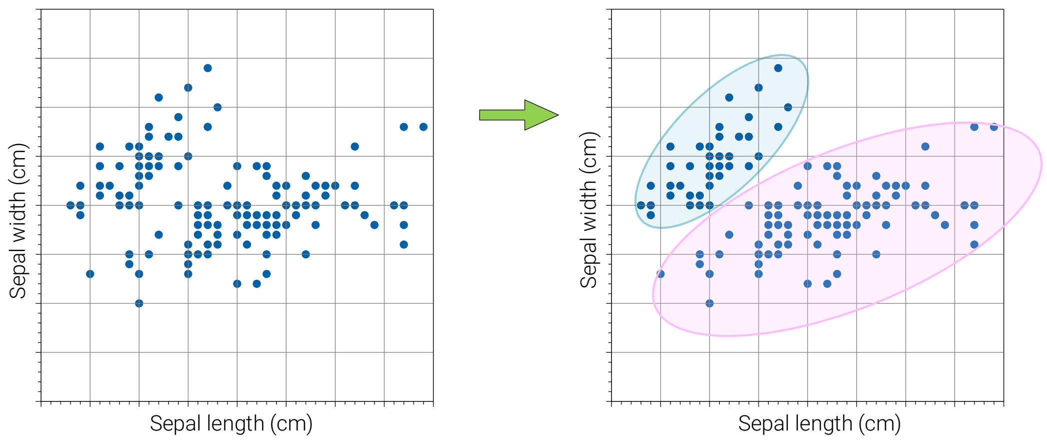

图 1. 用删除标签的鸢尾花数据介绍聚类算法

大家在使用 Scikit-Learn 聚类算法时，会发现有些算法有 predict() 方法。也就是说，如图 2 所示，已经训练好的模型，有可能你将全新的数据点分配到确定的簇中。有这种功能的聚类算法叫做归纳聚类 (inductive clustering)。本章后文要介绍的 k 均值聚类、高斯混合模型都属于归纳聚类。如图 2 所示，归纳聚类算法也有决策边界。这就意味着归纳聚类模型具有一定的泛化能力，可以推广到新的、之前未见过的数据。

不具备这种能力的聚类算法叫做非归纳聚类 (non-inductive clustering)。

非归纳聚类只能对训练数据进行聚类，而不能将新数据点添加到已有的模型中进行预测。这意味着模型在训练时只能学习训练数据的模式，无法用于对新数据点进行簇分配。比如，层次聚类、DBSCAN 聚类都是非归纳聚类。

归纳聚类强调模型的泛化能力，可以适应新数据，而非归纳聚类则更侧重于建模训练数据内部的结构。

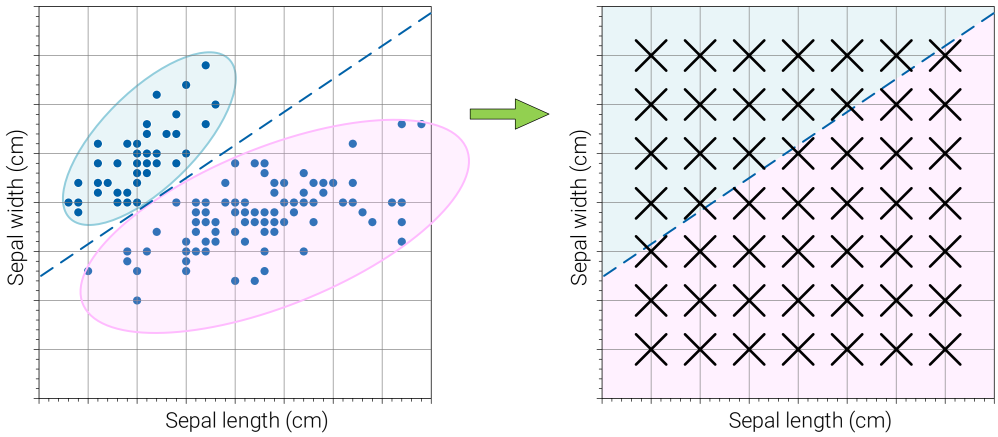

图 2. 归纳聚类算法

下面我们就用最通俗的语言，以几乎没有数学公式的方式，介绍几种常用聚类算法。

33.2 *K* 均值聚类

K 均值算法 (*K*-Means) 将样本分为 *K* 个簇，使得每个数据点与其所属簇的中心 (也叫质心 (centroid)) 之间的距离最小化。一般情况，每个簇的中心点是该簇中所有样本点的平均值。

图 3 以二聚类为例，展示 *K* 均值聚类的操作流程。从样本数据开始，首先从样本中随机选取 2 个数据作为均值向量 $\boldsymbol{\mu}_1$ 和 $\boldsymbol{\mu}_2$ 的初始值，然后进入如下迭代循环。

a) 计算每一个样本点分别到均值向量 $\boldsymbol{\mu}_1$ 和 $\boldsymbol{\mu}_2$ 的距离；

b) 比较每个样本到 $\boldsymbol{\mu}_1$ 和 $\boldsymbol{\mu}_2$ 距离，确定簇的划分；

c) 根据当前簇，重新计算并更新均值向量 $\boldsymbol{\mu}_1$ 和 $\boldsymbol{\mu}_2$。

直到均值向量 $\boldsymbol{\mu}_1$ 和 $\boldsymbol{\mu}_2$ 满足迭代停止条件，得到最终的簇划分。

图 4 所示为利用 *K* 均值算法根据鸢尾花花萼长度、花萼宽度特征划分为 2 和 3 簇两种情况。

根据前文介绍的内容，我们知道 *K* 均值算法为归纳聚类算法；因此，如图 4 所示，*K* 均值算法可以用训练好的模型预测其他新样本数据的聚类，从而获得聚类决策边界。容易发现 *K* 均值聚类算法决策边界为直线段。图 4 中的 × 为 *K* 均值算法的簇质心。

图 5 代码绘制图 4 两幅子图，下面聊聊其中关键语句。

ⓐ 从 sklearn.cluster 模块导入 *K* 均值算法对象 KMeans。请大家注意变量大小写。

ⓑ 加载经典鸢尾花数据集。在聚类算法中，我们仅仅用到鸢尾花的特征数据 (data)，不会用到标签数据 (target)。ⓒ 提取鸢尾花数据中的前两个特征 (花萼长度、花萼宽度) 数据。

ⓓ 利用 matplotlib.colors.ListedColormap 创建离散颜色映射，以在图表中对不同的离散值进行颜色编码。颜色映射在本例中可视化鸢尾花聚类区域。

ⓔ 实例化了一个 KMeans 对象，并指定了要进行的聚类数目。参数 n_clusters 参数就是用来指定 *K* 均值聚类算法 *K* 的值，即希望将数据划分成多少个簇。

ⓕ 执行了 KMeans 聚类算法，拟合模型并预测数据点所属的簇标签。fit_predict(X) 同时拟合 (fit) 数据并预测 (prefict) 数据点所属的簇标签。大家也可以用 fit(X).predict(X) 来分两步执行。其中，X 是一个二维数组，表示输入的数据，每行代表一个数据样本，每列代表一个特征。请大家自行查看返回结果。

ⓖ 利用训练好的 KMeans 模型对全新的数据进行聚类预测。ⓗ 调整数组形状，用于后续可视化。

ⓘ 用填充等高线可视化聚类区域。ⓙ 用等高线可视化聚类决策边界。

ⓚ 获取 KMeans 聚类算法拟合后得到的聚类质心的坐标。ⓛ 用散点可视化聚类质心。

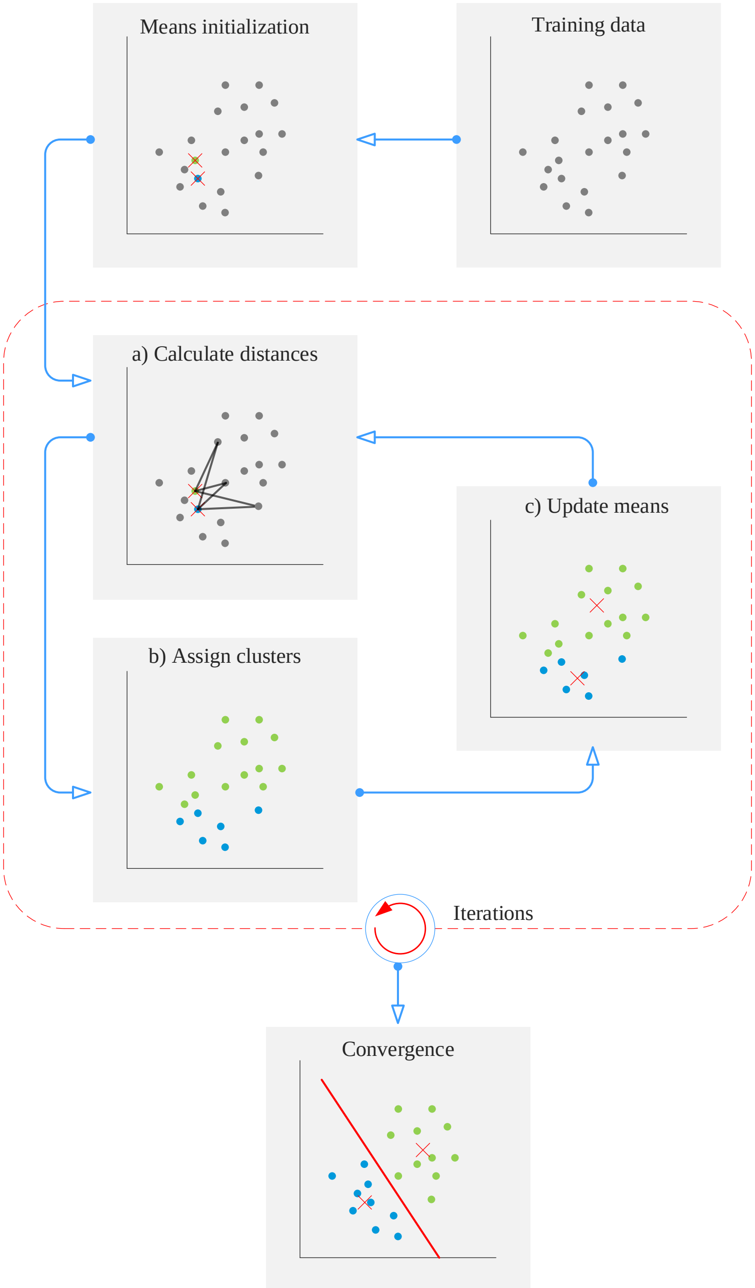

图 3. *K* 均值算法流程图

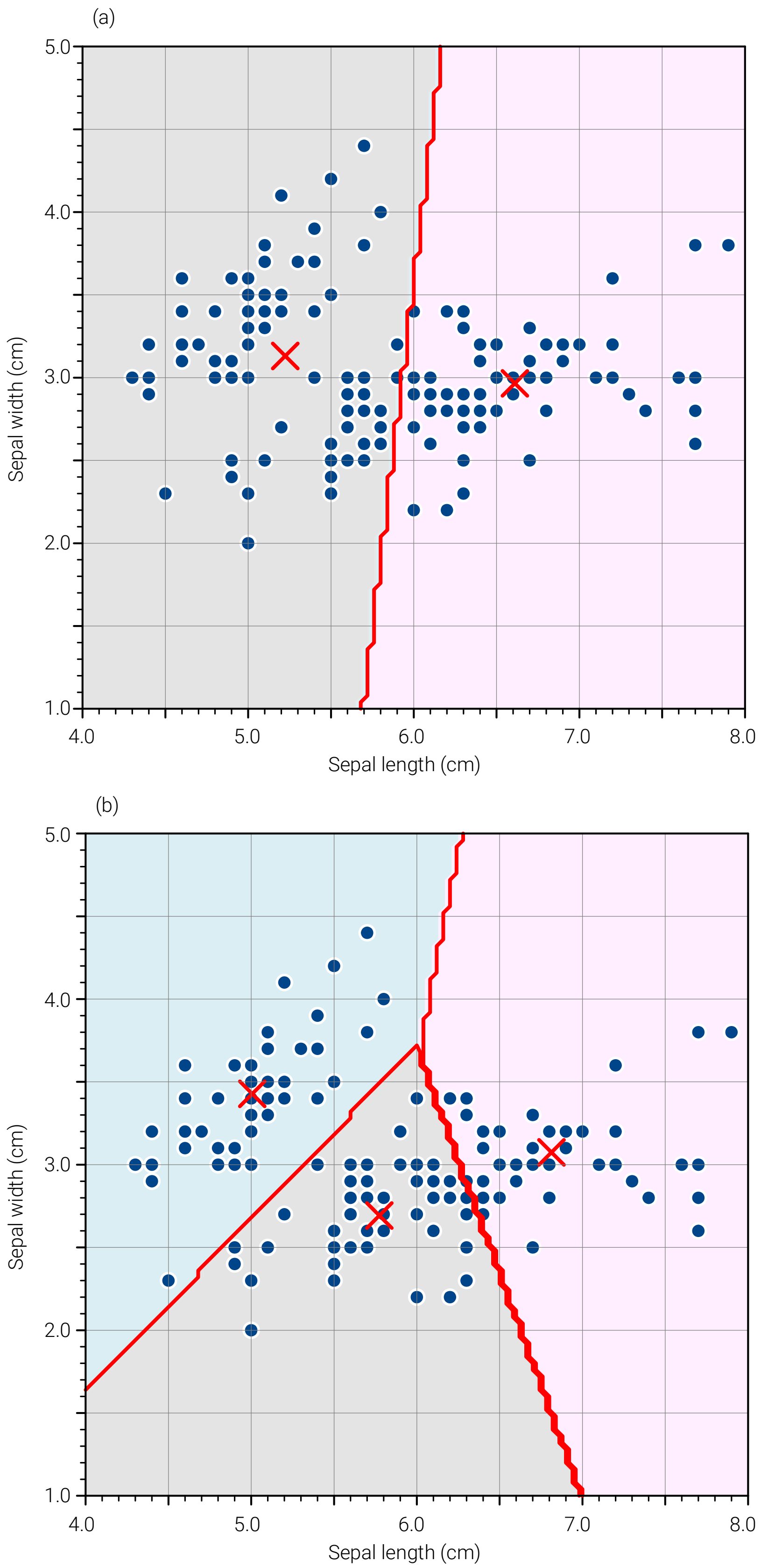

图 4. *K* 均值聚类确定决策边界，簇数分别为 2、3

*K*均值聚类

```python
from sklearn import datasets
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap

# 导入并整理数据
iris = datasets.load_iris()
X = iris.data[:, :2]

# 生成网格化数据
x1_array = np.linspace(4,8,101)
x2_array = np.linspace(1,5,101)
xx1, xx2 = np.meshgrid(x1_array,x2_array)
# 创建色谱
rgb = [[255, 238, 255],
       [219, 238, 244],
       [228, 228, 228]]
rgb = np.array(rgb)/255.
cmap_light = ListedColormap(rgb)

# 采用KMeans聚类
kmeans = KMeans(n_clusters=2)
cluster_labels = kmeans.fit_predict(X)

# 预测聚类
Z = kmeans.predict(np.c_[xx1.ravel(), xx2.ravel()])
Z = Z.reshape(xx1.shape)

fig, ax = plt.subplots()

ax.contourf(xx1, xx2, Z, cmap=cmap_light)
ax.scatter(x=X[:, 0], y=X[:, 1],
           color=np.array([0, 68, 138])/255.,
           alpha=1.0,
           linewidth = 1, edgecolor=[1,1,1])
# 绘制决策边界
levels = np.unique(Z).tolist();
ax.contour(xx1, xx2, Z, levels=levels,colors='r')
centroids = kmeans.cluster_centers_
ax.scatter(centroids[:, 0], centroids[:, 1],
           marker="x", s=100, linewidths=1.5,
           color="r")

ax.set_xlim(4, 8); ax.set_ylim(1, 5)
ax.set_xlabel(iris.feature_names[0])
ax.set_ylabel(iris.feature_names[1])
ax.grid(linestyle='--', linewidth=0.25,
        color=[0.5,0.5,0.5])
ax.set_aspect('equal', adjustable='box')
```

图 5. 根据花萼长度、花萼宽度，用 *K* 均值聚类算法确定聚类决策边界，代码

33.3 高斯混合

高斯混合模型 (Gaussian Mixture Model, GMM) 将样本分为多个高斯分布，每个高斯分布对应一个簇。与 K 均值聚类不同，GMM 不仅能够将数据点分配到不同的簇，还可以为每个簇分配一个概率值，表明数据点属于该簇的可能性。

如图 6 所示，多元高斯分布中，协方差矩阵决定高斯分布的形状。

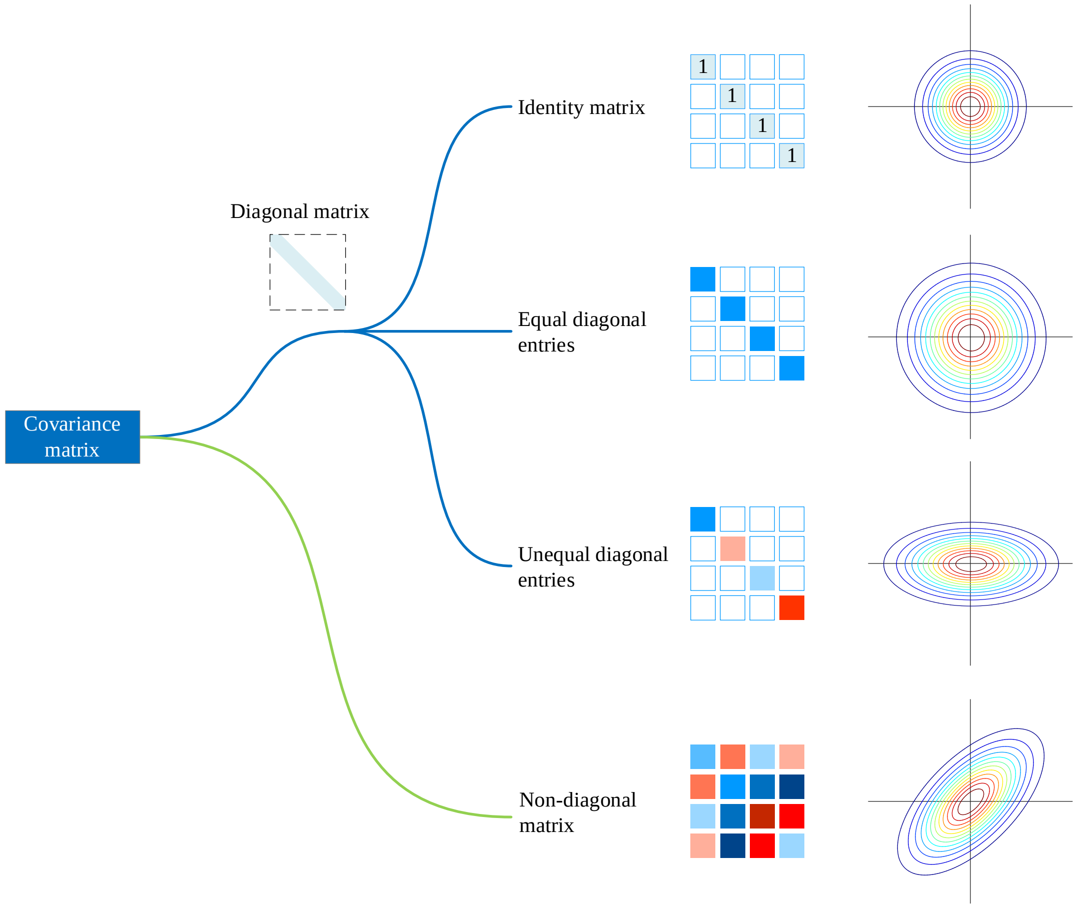

图 6. 协方差矩阵的形态影响高斯密度函数形状

如表 1 总结，scikit-learn 工具包中 `sklearn.mixture` 高斯混合模型支持四种协方差矩阵——`tied` (平移)、`spherical` (球面)、`diag` (对角)和 `full` (完全)。

`tied` 指的是，所有分量共享一个非对角协方差矩阵 Σ。每个簇对应的多元高斯分布等高线为大小相等旋转椭圆。`tied` 对应的决策边界为直线。

`spherical` 指的是，每个分量协方差矩阵 Σ_j ($j = 1,2, \ldots, K$) 不同，但是每个分量 Σ_j 均为对角阵；且 Σ_j 对角元素相同，即特征方差相同。每个簇对应的多元高斯分布等高线为正圆。`spherical` 对应的决策边界为圆形弧线。

`diag` 指每个分量有各自独立的对角协方差矩阵，也就是 Σ_j 为对角阵，特征条件独立；但是对 Σ_j 对角线元素大小不做限制。每个簇对应的多元高斯分布等高线正椭圆，`diag` 对应的决策边界为正圆锥曲线。

full 指每个分量有各自独立协方差矩阵，即对 $\boldsymbol{\Sigma}_j$ 不做任何限制。full 对应的决策边界为任意圆锥曲线。

表 1. 根据方差-协方差矩阵特点将高斯混合模型分为 4 类

参数设置	$\boldsymbol{\Sigma}_i$	$\boldsymbol{\Sigma}_i$ 特点	多元高斯分布 PDF 等高线	决策边界
tied	相同	非对角阵	任意椭圆	直线
spherica		对角阵，对角线元素等值	正圆	正圆
diag	不相同	对角阵	正椭圆	正圆锥曲线
full		非对角阵	任意椭圆	圆锥曲线

和 K 均值聚类算法一样，高斯混合模型 GMM 也需要指定 K 值；高斯混合模型也是利用迭代求解优化问题。不同的是，GMM 利用协方差矩阵，可以估算后验概率/成员值。前文提过，GMM 的协方差矩阵有四种类型，每种类型对应不同假设，获得不同决策边界类型。

K 均值聚类可以看作是高斯混合模型一个特例。如图 7 所示，K 均值聚类对应的 GMM 特点是，各簇协方差矩阵 $\boldsymbol{\Sigma}_j$ 相同，$\boldsymbol{\Sigma}_j$ 为对角阵，并且 $\boldsymbol{\Sigma}_j$ 主对角线元素相等。

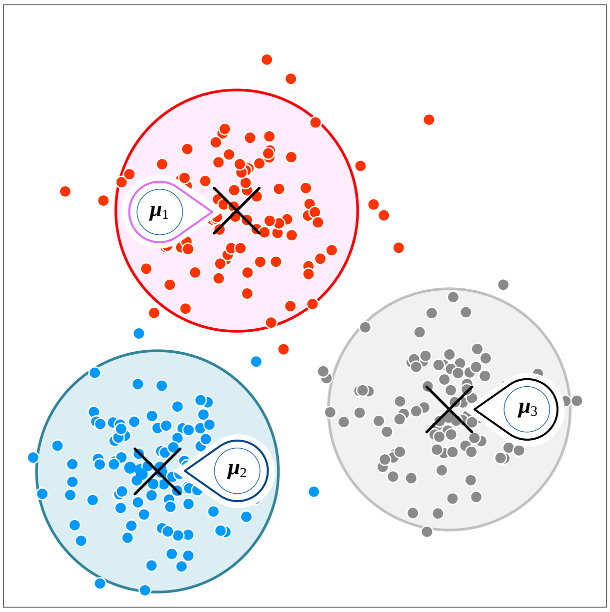

图 7. K 均值聚类可以看作是高斯混合模型一个特例

图 8 ~ 图 11 所示为利用 GMM 聚类鸢尾花数据。这四幅图采用四种不同的协方差矩阵完成 GMM 聚类。大家可以通过比较这四幅图的椭圆形状很容易理解表 1。图 12 定义的可视化函数绘制了这四幅图中的椭圆和向量。图 13 是完成 GMM 的代码，这段代码调用了图 12 的可视化函数。下面让我们聊聊这两段代码。

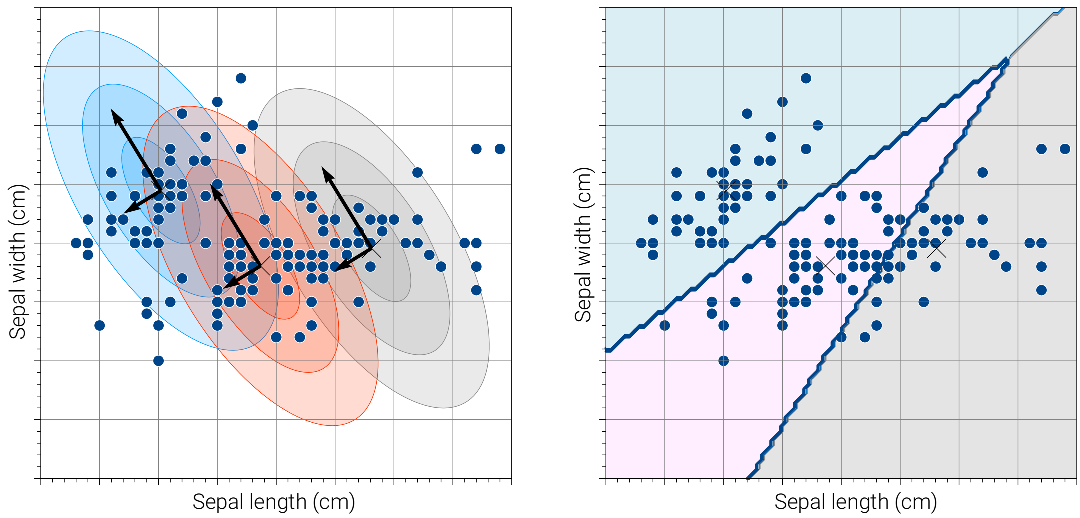

图 8. *K* 均值聚类，协方差矩阵为 'tied'

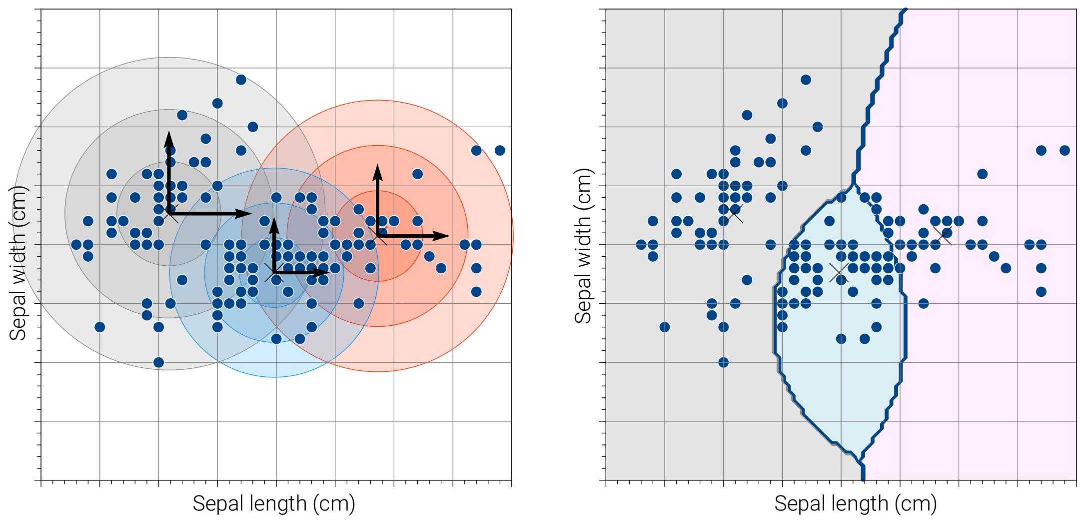

图 9. *K* 均值聚类，协方差矩阵为 'spherical'

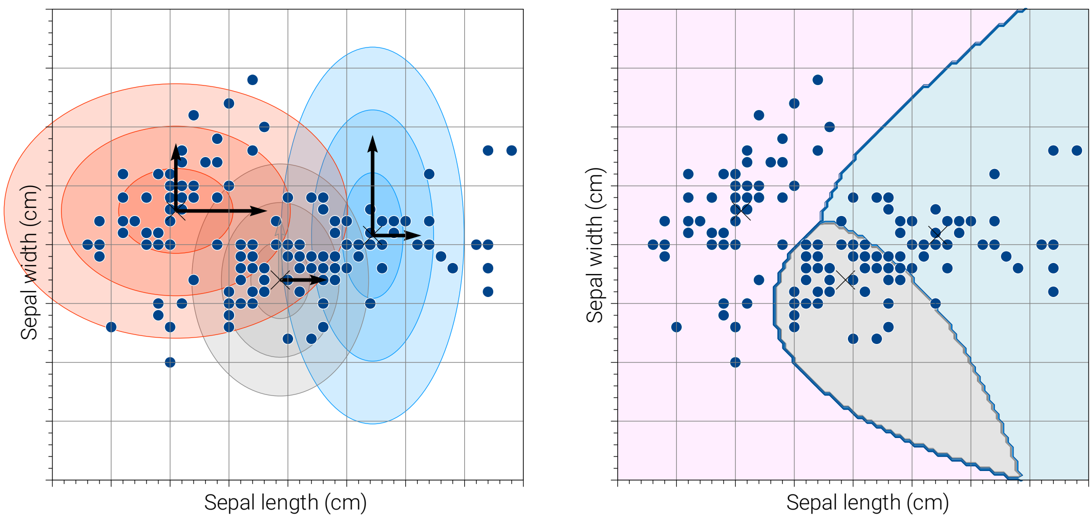

图 10. *K* 均值聚类，协方差矩阵为 'diag'

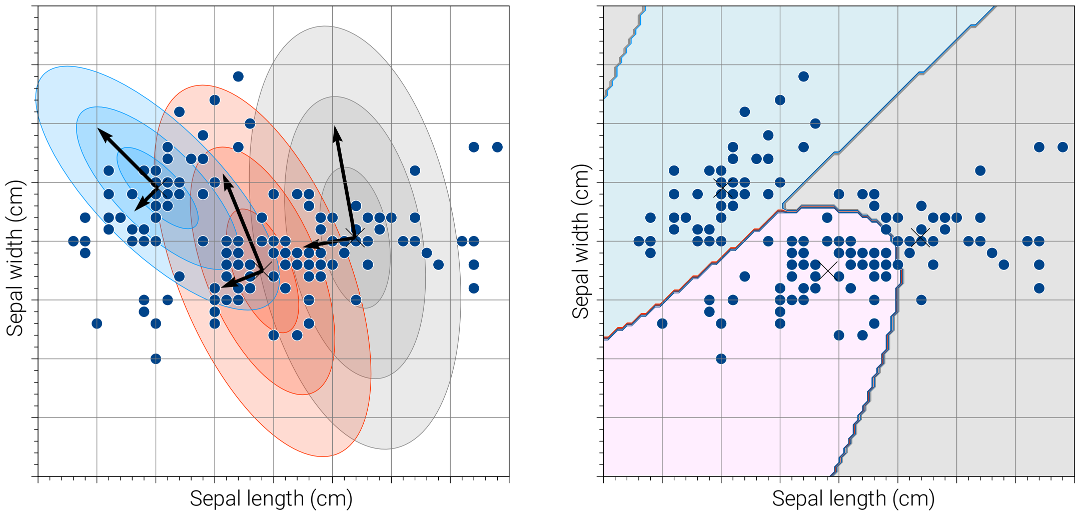

图 11. *K* 均值聚类，协方差矩阵为 'full'

让我们首先看看图 12 可视化代码。

ⓐ从 matplotlib.patches 导入 Ellipse 类，Ellipse 用来绘制椭圆形状。

前文提过，GMM 可以有不同的协方差类型，包括 'full'、'tied'、'diag' 和 'spherical'，它们分别表示完整协方差矩阵、共享协方差矩阵、对角协方差矩阵和球状协方差矩阵。

ⓑ这个条件判断语句检查 GMM 对象的协方差类型是否为 'full'。根据技术文档，这种情况下，协方差矩阵形状为 (n_components, n_features, n_features)，三维 NumPy 数组。其中，axis = 0 对应的是不同簇。也就是说，如图 11 所示，不同簇协方差矩阵不同，gmm.covariances_[j] 提取的是不同簇的协方差矩阵，结果为二维 NumPy 数组。

ⓒ判断 GMM 对象的协方差类型是否为 'tied'。根据技术文档，这种情况下，协方差矩阵形状为 (n_features, n_features)，二维 NumPy 数组。这意味着不同簇的协方差矩阵完全相同，如图 8 所示。

ⓓ判断 GMM 对象的协方差类型是否为 'diag'。根据技术文档，这种情况下，协方差矩阵形状为 (n_components, n_features)，二维 NumPy 数组。其中，axis = 0 对应的是不同簇，axis = 1 对应的是不同特征的方差。也就是说，如图 10 所示，从 GMM 对象的 gmm.covariances_[j] 属性中获取第 j 个分量的协方差矩阵，结果为一维数组；然后，使用 np.diag() 函数将其转换为对角矩阵形式，结果为二维数组。

ⓔ判断 GMM 对象的协方差类型是否为 'spherical'。根据技术文档，这种情况下，协方差矩阵形状为 (n_components,)，一维 NumPy 数组。其中，axis = 0 对应不同簇。也就是说，如图 9 所示，将单位矩阵的每个维度上的方差都乘以相应的协方差值，从而形成一个球状的协方差矩阵。

ⓕ实际上用奇异值函数 numpy.linalg.svd() 完成的是协方差矩阵的特征值分解。这个矩阵分解，可以帮我们了解一个旋转椭圆的半长轴、半短轴的长度，以及椭圆的旋转角度。《矩阵力量》将具体讲解数学工具背后的原理。

ⓖ计算椭圆长轴、短轴的长度。ⓗ计算椭圆旋转角度弧度。

ⓘ绘制 GMM 每个簇的质心。

ⓙ使用 Matplotlib 的 quiver 函数来在二维图中绘制箭头，用来表示椭圆长轴方向 (矩阵 $\boldsymbol{U}$ 的第 1 列)。ⓚ绘制椭圆短轴方向 (矩阵 $\boldsymbol{U}$ 的第 2 列)。

Ⓛ建了一个椭圆对象，指定了椭圆的中心坐标、长轴宽度、短轴宽度、旋转角度、边缘颜色和填充颜色。然后，我们使用 ax.add_patch() 将椭圆添加到图中。

可视化函数

```
from matplotlib.patches import Ellipse
# 定义可视化函数
def make_ellipses(gmm, ax):

    # 可视化不同簇
    for j in range(0,K):
        # 四种不同的协方差矩阵
        if gmm.covariance_type == 'full':
            covariances = gmm.covariances_[j]
        elif gmm.covariance_type == 'tied':
            covariances = gmm.covariances_
        elif gmm.covariance_type == 'diag':
            covariances = np.diag(gmm.covariances_[j])
        elif gmm.covariance_type == 'spherical':
            covariances = np.eye(gmm.means_.shape[1])
            covariances = covariances*gmm.covariances_[j]

        # 用奇异值分解完成特征值分解
        U, S, V_T = np.linalg.svd(covariances)
        # 计算长轴、短轴长度
        major, minor = 2 * np.sqrt(S)

        # 计算椭圆长轴旋转角度
        angle = np.arctan2(U[1,0], U[0,0])
        angle = 180 * angle / np.pi

        # 多元高斯分布中心
        ax.plot(gmm.means_[j, 0],gmm.means_[j, 1],
                 color = 'k',marker = 'x',markersize = 10)

        # 绘制半长轴向量
        ax.quiver(gmm.means_[j,0],gmm.means_[j,1],
                  U[0,0], U[1,0], scale = 5/minor)

        # 绘制半短轴向量
        ax.quiver(gmm.means_[j,0],gmm.means_[j,1],
                  U[0,1], U[1,1], scale = 5/major)

        # 绘制椭圆
        for scale in np.array([3, 2, 1]):

            ell = Ellipse(gmm.means_[j, :2],
                          scale*minor,
                          scale*major,
                          angle,
                          color=rgb[j,:],
                          alpha = 0.18)
            ax.add_artist(ell)
```

图 12. 定义可视化函数

图 13 和图 5 代码比较类似。这部分代码请大家自行学习。

GMM聚类

```
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import numpy as np
from sklearn import datasets
from sklearn.mixture import GaussianMixture

# 创建色谱
rgb = [[255, 51, 0],
       [0, 153, 255],
       [138,138,138]]
rgb = np.array(rgb)/255.
cmap_bold = ListedColormap(rgb)

# 生成网格化数据
x1_array = np.linspace(4,8,101)
x2_array = np.linspace(1,5,101)
xx1, xx2 = np.meshgrid(x1_array,x2_array)

# 鸢尾花数据
iris = datasets.load_iris(); X = iris.data[:, :2]

K = 3 # 簇数
# 协方差类型
covariance_types = ['tied', 'spherical', 'diag', 'full']

for covariance_type in covariance_types:
    # 采用GMM聚类
    gmm = GaussianMixture(n_components=K,
                          covariance_type=covariance_type)
    gmm.fit(X)
    Z = gmm.predict(np.c_[xx1.ravel(), xx2.ravel()])
    Z = Z.reshape(xx1.shape)

    # 可视化
    fig = plt.figure(figsize = (10,5))
    ax = fig.add_subplot(1,2,1)
    ax.scatter(x=X[:, 0], y=X[:, 1],
               color=np.array([0, 68, 138])/255.,
               alpha=1.0,
               linewidth = 1, edgecolor=[1,1,1])
    # 绘制椭圆和向量
    make_ellipses(gmm, ax)
    ax.set_xlim(4, 8); ax.set_ylim(1, 5)
    ax.set_xlabel(iris.feature_names[0])
    ax.set_ylabel(iris.feature_names[1])
    ax.grid(linestyle='--', linewidth=0.25,
            color=[0.5,0.5,0.5])
    ax.set_aspect('equal', adjustable='box')

    ax = fig.add_subplot(1,2,2)
    ax.contourf(xx1, xx2, Z, cmap=cmap_bold, alpha = 0.18)
    ax.contour(xx1, xx2, Z, levels=[0,1,2],
               colors=np.array([0, 68, 138])/255.)
    ax.scatter(x=X[:, 0], y=X[:, 1],
               color=np.array([0, 68, 138])/255.,
               alpha=1.0,
               linewidth = 1, edgecolor=[1,1,1])
    centroids = gmm.means_
    ax.scatter(centroids[:, 0], centroids[:, 1],
               marker="x", s=100, linewidths=1.5,
               color="k")
    ax.set_xlim(4, 8); ax.set_ylim(1, 5)
    ax.set_xlabel(iris.feature_names[0])
    ax.set_ylabel(iris.feature_names[1])
    ax.grid(linestyle='--', linewidth=0.25,
            color=[0.5,0.5,0.5])
    ax.set_aspect('equal', adjustable='box')
```

图 13. GMM 聚类代码