目录

第 1 章　介绍

将物理或抽象对象的集合分成由类似的对象组成的多个类的过程被称为聚类。由聚类所生成的簇是一组数据对象的集合，这些对象与同一个簇中的对象彼此相似，与其他簇中的对象相异。“物以类聚，人以群分”，在自然科学和社会科学中，存在着大量的分类问题。聚类分析又称群分析，它是研究（样品或指标）分类问题的一种统计分析方法。聚类分析起源于分类学，但是聚类不等于分类。聚类与分类的不同在于，聚类所要求划分的类是未知的。聚类分析内容非常丰富，有系统聚类法、有序样品聚类法、动态聚类法、模糊聚类法、图论聚类法、聚类预报法等。在数据挖掘中，聚类也是很重要的一个概念。

聚类被用作一个预处理步骤，作为一个知识抽取的工具，将数据分为可管理的部分。聚类或聚类分析是一种探索性数据分析的形式，在这种分析中，数据被分成若干组或子集，从而使每组中的对象具有某种相似性。

传统的聚类分析计算方法主要有如下几种：1. 划分方法；2. 层次方法；3. 基于密度的方法；4. 基于网格的方法；5. 基于模型的方法。

大规模数据聚类主要有两种方法：基于不同增量样式的分布式聚类和通过渐进或随机抽样找到的样本聚类。两种方法都已应用于大规模数据的 FCM 聚类算法中，这些方法也被应用在 GMM（Gaussian-mixture-model）算法和 EM（expectation-maximization）算法中。这两种算法都为实现“可加载数据的加速和不可加载数据的近似”提供了有用的方法。

考虑一个 n 个实体的集合：$O = \{o_1, o_2, \cdots, o_n\}$，每个实体被一个数值的特征向量所表示：$X = \{x_1, x_2, \cdots, x_n\} \subset \mathbb{R}^d$。一个实体的划分被定义为 $cn = \{u_{ik}\}$，u_{ik} 代表实体 o_i 属于第 k 类的概率。`c-partition` 通常被表示为一个 $c \times n$ 的矩阵 $U = [u_{ik}]$。

第 2 章　FUZZY c-MEANS 算法

FCM 算法是一种基于划分的聚类算法，它的思想就是使得被划分到同一簇的对象之间相似度最大，而不同簇之间的相似度最小。模糊 C 均值算法是普通 C 均值算法的改进，普通 C 均值算法对于数据的划分是硬性的，而 FCM 则是一种柔性的模糊划分。

2.1　隶属度函数

隶属度函数 是表示一个对象 x 隶属于集合 A 的程度的函数，通常记作 $\mu_A(x)$。其自变量范围是所有可能属于集合 A 的对象，取值范围是 $[0,1]$，即 $0 \leq \mu_A(x) \leq 1$。$\mu_A(x)=1$ 表示 x 完全隶属于集合 A，相当于传统集合概念上的 $x \in A$。

2.2　K 均值聚类算法

K 均值聚类（K-Means），即众所周知的 C 均值聚类。它的核心思想为：算法把 n 个向量 $x_j(i=1,2,\cdots,n)$ 分为 c 个分类 $v_i(i=1,2,\cdots,c)$，并求每组的聚类中心，使得非相似性（或距离）指标的价值函数（或目标函数）达到最小。目标函数可定义为：

$$J_m(U,V)=\sum_{i=1}^{c}\sum_{j=1}^{n}\|X_j-V_i\|_A^2 \tag{2.1}$$

算法实现主要步骤：

步骤 1：随机确定 k 个初始点作为质心。

步骤 2：对数据集中的每个数据点找到距离最近的簇。

步骤 3：对于每一个簇，计算簇中所有点的均值并将均值作为质心。

步骤 4：重复步骤 2，直到任意一个点的簇分配结果不变。

2.3　模糊 C 均值聚类

模糊 C 均值聚类（FCM），即众所周知的模糊 ISODATA，是用隶属度确定每个数据点属于某个聚类的程度的一种聚类算法。1973 年，Bezdek 提出了该算法，作为早期硬 C 均值聚类（HCM）方法的一种改进。

2.3.1　FCM 思想

1. FCM 算法的目标函数为：

$$J_m(U,V)=\sum_{i=1}^{c}\sum_{j=1}^{n}u_{ij}^m\|X_j-V_i\|_A^2 \tag{2.2}$$

其中 $U = [u_{ij}]$ 为隶属度矩阵，u_{ij} 是第 j 个样本对于第 i 类的隶属度，m 是模糊常数。

2. FCM 算法的约束条件：对于任意一个样本，它对各个聚类的隶属度之和为 1。

$$\sum_{i=1}^{c} u_{ij} = 1 \tag{2.3}$$

3. 求解 U，V：为了求有约束条件下目标函数的极值，我们利用拉格朗日乘子法构造新的函数。

$$F = \sum_{i=1}^{c} u_{ij}^{m} \|X_j - V_i\|^2 + \lambda(\sum_{i=1}^{c} u_{ij} - 1) \tag{2.4}$$

其中 λ 成为 *Lagrange* 乘子，对 F 函数求极值得最优化条件如下：

$$\frac{\partial F}{\partial \lambda} = (\sum_{i=1}^{c} u_{ij} - 1) = 0 \tag{2.5}$$

$$\frac{\partial F}{\partial u_{ij}} = [m(u_{ij})^{m-1} \|X_j - V_i\|^2 - \lambda] = 0 \tag{2.6}$$

$$\frac{\partial F}{\partial v_i} = \sum_{j=1}^{n} (u_{ij})^m x_j - v_i \sum_{i=1}^{c} u_{ij}^m = 0 \tag{2.7}$$

由极值条件解得：

$$u_{ij} = [\sum_{k=1}^{c} (\frac{\|X_j - V_i\|^2}{\|X_j - V_k\|^2})^{\frac{2}{m-1}}]^{-1} \tag{2.8}$$

$$v_i = \frac{\sum_{j=1}^{n} u_{ij}^m X_j}{\sum_{j=1}^{n} (u_{ij}^m)} \tag{2.9}$$

2.3.2 FCM 算法

Algorithm 1 所示为 FCM 算法。

Algorithm 1: LFCM/AO

Input: X, c, m

Output: U, V

while $\max_{1 \le k \le c}\{\|v_{k,new} - v_{k,old}\|^2\} > \epsilon$ **do**

  $u_{ij} = [\sum_{k=1}^{c} (\frac{\|X_j - V_i\|^2}{\|X_j - V_k\|^2})^{\frac{2}{m-1}}]^{-1}$

  $v_i = \frac{\sum_{j=1}^{n} u_{ij}^m X_j}{\sum_{j=1}^{n} (u_{ij}^m)}$

第 3 章　FUZZY c-MEANS 算法扩展

下面将介绍论文中对模糊 C 均值算法的扩展优化方法 [1]。

3.1　采样与非迭代扩展

3.1.1　resFCM

最基本的，也是可能的处理大规模数据的方法是对数据集进行采样，然后使用 FCM 计算采样数据的集群中心。从而得出原数据集的中心。如果对数据进行了有效的采样，则通过对整个数据集进行聚类而产生的聚类中心位置与通过对采样数据进行聚类而产生的聚类中心之间的误差应该是很小的。

resFCM 算法 (random sample and extend FCM) 通过对原数据集进行随机采样而产生新的更小的数据集，然后在新的数据集上使用 FCM 算法进行聚类，得到聚类中心，最后根据产生的聚类中心在整个数据集上求得隶属矩阵。从而实现算法效率上的改进。Algorithm 2 所示为 rseFCM 算法。

Algorithm 2: resFCM to approximately minimize $J_m(U,V)$

Input: X, c, m

Output: U, V

1 Sample n_s objects from X without replacement, denoted X_s

2 $U_s, V = \mathrm{LFCM}(X_s, c, m)$

3 Extend the partition (U_s, V) to X, $\forall x_i \notin X_s$, using $u_{ij} = [\sum_{k=1}^{c} (\frac{\|X_j - V_i\|^2}{\|X_j - V_k\|^2})^{\frac{2}{m-1}}]^{-1}$

4 giving (U, V)

3.2　基于权重的模糊 C 均值算法

3.2.1　wFCM

在 LFCM 中，在集群解决方案中，每个对象都被认为同等重要。*加权 FCM*（(weighted FCM) 模型引入了权重，定义了聚类解决方案中每个对象的相对重要性。wFCM 目标函数：

$$J_{m\mathrm{w}}(U,V) = \sum_{i=1}^{c}\sum_{j=1}^{n} w_j u_{ij}^m \|X_j - V_i\|_A^2 \tag{3.1}$$

其中 $\mathrm{w} \in \mathbb{R}^n$，$w_j \geq 0$，是一组预先确定的权重，用于定义每个特征向量的权重影响。权重较高的对象在确聚类中心位置时更具影响力。Algorithm 3 所示为 wFCM 算法。

Algorithm 3: wFCM/AO to minimize $J_{m\mathrm{w}}(U,V)$

Input: X, c, m, w, (initial V)

Output: U, V

If V is not initialized, initialize V

while $\max_{1\le k\le c}\{\|v_{i,new}-v_{i,old}\|^2\} > \epsilon$ **do**

$u_{ij} = [\sum_{k=1}^{c}(\frac{\|X_j-V_i\|^2}{\|X_j-V_k\|^2})^{\frac{2}{m-1}}]^{-1}$

$v_i = \frac{\sum_{j=1}^{n} w_j u_{ij}^m X_j}{\sum_{j=1}^{n} w_j u_{ij}^m}$

3.2.2 spFCM

spFCM(Single-Pass Fuzzy c-Means) 首先在原数据集上进行随机抽样，随机构造出 s 个大小为 n_s 的数据子集。算法最初将权重向量 w 初始化长度为 n_s，每一位权重都为 $1s$ 的向量。然后计算第一组数据样本的 wFCM 聚类划分和聚类中心。spFCM 然后迭代剩余的 X 的数据子集，每次迭代运用 wFCM 在一组更大的数据集上进行聚类划分，这个数据集由上一次迭代的聚类中心和本次的样本子集 X_l 组成，也就是 $(V \cup X_l)$。

所以，每次迭代有 $c+n_s$ 个实体进行聚类。构造长度为 $c+n_s$ 的权重向量，重复迭代 s 次 wFCM，得到最终的聚类中心。每次迭代，wFCM 聚类中心初始化为上一次迭代所产生的聚类中心会加速 wFCM 聚类中心的收敛。Algorithm 4 所示为 spFCM 算法。

Algorithm 4: spFCM/AO to approximately minimize $J_{m\mathrm{w}}(U,V)$

Input: X, c, m, n_s

Output: V

Load X as n_s-sized randomly chosen subsets, $X = \{X_1, X_2, \cdots, X_s\}$

$\mathrm{w} = 1_{n_s}$

$U, V = \mathrm{wFCM}(X_1, c, m, \mathrm{w})$

for $l = 2$ *to* s **do**

$w_i' = \sum_{j=1}^{n_s}(u_{ij})w_j, i = 1, \cdots, c$

$\mathrm{w} = \{\mathrm{w}' \cup 1_{n_s}\}$

$U, V = \mathrm{wFCM}(\{V \cup X_l\}, c, m, \mathrm{w}, V)$

3.2.3 oFCM

oFCM(Online Fuzzy c-Means) 首先将原数据集划分为 s 个大小为 n_s 的数据子集，划分规则为：$X = \{X_1, X_2, \cdots, X_s\}$, where $X_i = \{x_{(i-1)n_s+1}, \cdots, x_{in_s}\}$

与 spFCM 不同，oFCM 分别对所有 s 个数据子集进行 wFCM 聚类划分，然后对 s 次聚类划分所产生的 $c \times s$ 个聚类中心 ($\{V_1 \cup \cdots \cup V_s\}$) 上进行 wFCM 聚类划分，最终求得整个数据集的聚类中心。同样的，计算每个样本子集的聚类中心时，wFCM 聚类中心初始化为上一个数据子集计算所产生的聚类中心会加速 wFCM 聚类中心的收敛。Algorithm 5 所示为 oFCM 算法。

Algorithm 5: oFCM/AO to approximately minimize $J_{m\mathrm{w}}(U, V)$

Input: X, c, m, n_s

Output: V

Load X as n_s-sized subsets, $X = \{X_1, X_2, \cdots, X_s\}$, where $X_i = \{x_{(i-1)n_s+1}, \cdots, x_{in_s}\}$

$U_1, V_1 = \mathrm{wFCM}(X_1, c, m, 1_{n_s})$

for $l = 2$ *to* s **do**

  $U_l, V_l = \mathrm{wFCM}(X_l, c, m, 1_{n_s}, V_{l-1})$

$\mathrm{w}_1 = \sum_{j=1}^{n_s} (U_l)_j$, $l = 1, \cdots, s$

$V = \mathrm{wFCM}(\{V_1 \cup \cdots \cup V_s\}, c, m, \{\mathrm{w}_1 \cup \cdots \cup \mathrm{w}_s\})$

3.2.4 brFCM

brFCM(Bit-Reduced Fuzzy c-Means) 旨在解决大规模图像中的聚类问题。brFCM 首先将输入数据 X 离散化规约为 X'，然后对 X' 运用 wFCM 进行聚类划分。新的聚类划分中各个实体的权重为其桶内原实体的数目（对于图像，可以是桶内图像像素的总和）。Algorithm 6 所示为 brFCM 算法。

对输入数据进行离散化的方法有很多种，例如首先对图像数据通过移除不重要的位按位规约，然后将相同的实体归为一类。

Algorithm 6: brFCM/AO to approximately minimize $J_{m\mathrm{w}}(U, V)$

Input: X, c, m

Output: V

Bin X into s quantization bins, where $X' = \{\mathrm{x}'_k\},\ k = 1, \cdots, s$,is the set of bin prototypes (typically the means of the binned feature vectors or the bit-reduced binary values)

Set w_k to the number of feature vectors aggregated into x'_k, $\forall k$

$V = \mathrm{wFCM}(X', c, m, \mathrm{x})$

3.3 基于核的改进的模糊 C 均值算法

kFCM 是基于核的改进的模糊 C 均值聚类算法。它是通过核函数将原始空间中的点映射到特征空间中，考虑到原始空间中的点无法用一个线性函数进行划分，于是将其变换到一个更高维度的空间中，可以在这个高维空间中找到一个线性函数，容易对原始数据进行划分。这个高维空间就叫特征空间。从低维到高维空间的映射函数的内积就叫核函数。将核函数引入机器学习的一个重要原因是：当特征空间维数很高而核函数计算量较之特征空间内的内积运算计算量相对很小时，这样做可以提高计算效率。

基于目标函数的 FCM 聚类算法存在两大缺陷:

1. 隶属度和为 1 的约束条件易造成它对孤立点和噪声敏感。

2. 本身是一种迭代下降的算法，使得它初始聚类中心敏感且不易收敛于全局最优。

kFCM 算法提高了聚类性能，使算法对噪声和孤立点具有较好的鲁棒性。

3.3.1 FCM

考虑非线性映射函数 $\phi : \mathrm{x} \to \phi(\mathrm{x}) \in \mathbb{R}^{D_k}$，其中 D_k 是特征向量 x 的维度。在这里，我们不需要明确地知道 x 的转换，我们只需代替地知道它的点积 $\phi(\mathrm{x}) \cdot \phi(\mathrm{x}) = \kappa(\mathrm{x}, \mathrm{x})$。核函数 κ 有很多形式，多项式 $\kappa(\mathrm{x}_i, \mathrm{x}_j) = (\mathrm{x}_i^T \mathrm{x}_j + 1)^p$ 和径向基函数 (RBF)$\kappa(\mathrm{x}_i, \mathrm{x}_j) = \exp(\sigma \|\mathrm{x}_i - \mathrm{x}_j\|^2)$ 是最常用的两种。

给定 n 个实体 X,我们能够构造一个大小为 $n \times n$ 的核矩阵 $K = [K_{ij} = \kappa(\mathrm{x}_i, \mathrm{x}_j)]_{n \times n}$。核矩阵代表所有成对的特征向量的点积。

给定一个核函数 κ，*kernel FCM*(kFCM) 通常被定义为一个约束最小化问题。通过将 V 代入消除可得目标函数为：

$$J_m(U; \kappa) = \sum_{j=1}^{c} (\sum_{i=1}^{n} \sum_{k=1}^{n} (u_{ij}^m u_{kj}^m d_\kappa(\mathrm{x}_i, \mathrm{x}_k))/2 \sum_{l=1}^{n} U_{lj}^m) \tag{3.2}$$

其中 $d_\kappa(\mathrm{x}_i, \mathrm{x}_k) = \kappa(\mathrm{x}_i, \mathrm{x}_i) + \kappa(\mathrm{x}_k, \mathrm{x}_k) - 2\kappa(\mathrm{x}_i, \mathrm{x}_k)$，是第 i 个特征向量核第 j 个特征向量之间基于核的距离。

kFCM 求解最优化问题 $\min(J_m(U; \kappa))$，通过迭代计算：

$$u_{ij} = [\sum_{k=1}^{c} (\frac{d_\kappa(\mathrm{x}_i, \mathrm{v}_j)}{d_\kappa(\mathrm{x}_i, \mathrm{v}_k)})^{\frac{1}{m-1}}]^{-1} \ \forall i, j \tag{3.3}$$

实体 x_i 和聚类中心 v_j 之间基于核的距离为：

$$d_\kappa(\mathrm{x}_i, \mathrm{v}_j) = \|\phi(\mathrm{x}_i) - \phi(\mathrm{v}_j)\|^2 \tag{3.4}$$

其可以通过核矩阵进行计算。定义 $\widetilde{\mathrm{u}}_j = \mathrm{u}_j^m / \sum_i |u_{ij}^m|$,其中 $u_j^m = (u_{1j}^m, u_{2j}^m, \cdots, u_{nj}^m)^T$，化简后我们能得到：

$$d_\kappa(\mathrm{x}_i, \mathrm{v}_j) = \widetilde{\mathrm{u}}_j^T K \widetilde{\mathrm{u}}_j + K_{ii} - 2(\widetilde{\mathrm{u}}_j^T K)_i \tag{3.5}$$

类似于模糊 C 均值算法，聚类中心可以被特征向量线性组合：

$$\phi(\mathrm{v}_j) = \frac{\sum_{l=1}^{n} u_{lj}^m \phi(\mathrm{x}_l)}{\sum_{l=1}^{n} u_{lj}^m} \tag{3.6}$$

3.3.2 wkFCM

我们定义 kFCM 模型 $J_m(U;\kappa)$ 的基于权重扩展的 kFCM 模型（wkFCM）$J_{m\mathrm{w}}(U;\kappa)$，聚类中心 $\phi(\mathrm{v}_j)$ 特征向量的加权和：

$$\phi(\mathrm{v}_j) = \frac{\sum_{l=1}^{n} w_l u_{lj}^m \phi(\mathrm{x}_l)}{\sum_{l=1}^{n} w_l u_{lj}^m} \tag{3.7}$$

将式 3.7 带入式 3.5 中可得：

$$d_\kappa^{\mathrm{w}}(\mathrm{x}_i, \mathrm{x}_j) = \frac{1}{\|\mathrm{w} \circ \mathrm{u}_j\|^2}(\mathrm{w} \circ \mathrm{u}_j)^T K(\mathrm{w} \circ \mathrm{u}_j) + K_{ii} - \frac{2}{\|\mathrm{w} \circ \mathrm{u}_j\|}((\|\mathrm{w} \circ \mathrm{u}_j\|)^T K)_i \tag{3.8}$$

式 3.8 中 $\circ$ 表示哈达玛积。

> 哈达玛积 (Hadamard product) 是矩阵的一类运算，若 $A = (a_{ij})$ 和 $B = (b_{ij})$ 是两个同阶矩阵，若 $c_{ij} = a_{ij} \times b_{ij}$, 则称矩阵 $C = (c_{ij})$ 为 A 和 B 的哈达玛积。

算法 7 所示为 wkFCM 算法。

Algorithm 7: wkFCM/AO to minimize $J_{m\mathrm{w}}(U;\kappa)$

Input: K,c,m,w

Output: U,p $= \{p_1, \cdots, p_c\}$

Initialize $U \in M_{fcn_s}$

while $\max_{1\le k\le c}\{\|\mathrm{u}_{k,new} - \mathrm{u}_{k,old}\|^2\} > \epsilon$ **do**

  Compute $d_\kappa^{\mathrm{w}}(\mathrm{x}_i, \mathrm{v}_j)$

  $u_{ij} = [\sum_{k=1}^{c}(\frac{d_\kappa^{\mathrm{w}}(\mathrm{x}_i,\mathrm{v}_j)}{d_\kappa^{\mathrm{w}}(\mathrm{x}_i,\mathrm{v}_k)})^{\frac{1}{m-1}}]^{-1} \quad \forall i,j$

p_j arg $\min_i\{d_\kappa(\mathrm{x}_i, \mathrm{v}_j)\}$,$\forall j$

3.3.3 rsekFCM

rsekFCM 是基于随机抽样和核扩展的 FCM 算法。算法首先从元数据集 X 中随机抽取一个小的样本集 X_s，在 X_s 上我们使用 wFCM 进行聚类划分，产生聚类中心，然后再扩展到整个数据集求解 U。算法 8 所示为 rsekFCM 算法。

Algorithm 8: rsekFCM/AO to approximately minimize $J_{m\mathrm{w}}(U;\kappa)$

Input: Kernel function κ, X, c, m, n_s

Output: U

Sample n_s vectors from X, denoted X_s

$K = [\kappa(\mathrm{x}_i, \mathrm{x}_j)]$, $\forall x_i, x_j \in X_s$

$U, \mathrm{p} = \mathrm{wkFCM}(K, c, m, 1_{n_s})$

Extend partition to X:

$d_\kappa(\mathrm{x}_i, \mathrm{v}_j) = \kappa(\mathrm{x}_i, \mathrm{x}_i) + \kappa(\mathrm{x}_{p_j}, \mathrm{x}_{p_j}) - 2\kappa(\mathrm{x}_i, \mathrm{x}_{p_j}) \quad \forall i, j$

$u_{ij} = [\sum_{k=1}^{c} (\frac{d_\kappa(\mathrm{x}_i, \mathrm{v}_j)}{d_\kappa(\mathrm{x}_i, \mathrm{v}_k)})^{\frac{1}{m-1}}]^{-1} \quad \forall i, j$

3.3.4 spkFCM

spkFCM 是基于核扩展的 spFCM 算法。二者的区别在于 spkFCM 通过核函数将原始空间中的点映射到特征空间中。通过将原始空间中点的距离计算转换为核函数计算，从而提升算法的聚类性能。而二者在算法逻辑上的实现相同。算法 9 所示为 spkFCM 算法。

Algorithm 9: spkFCM to approximately minimize $J_{m\mathrm{w}}(U;\kappa)$

Input: Kernel function κ, X, c, m, s

Output: p

Randomly, without replacement, draw s (approximately) equal-sized subsets of the intergers $\{1, \cdots, n\}$, denoted $E = \{\xi_1, \cdots \xi_s\}$. n_l is the cardinality of ξ_l.

$\xi' = \xi_1$

$K = [\kappa(\mathrm{x}_i, \mathrm{x}_j)] \quad i, j = \xi'$

$U, \mathrm{p} = \mathrm{wkFCM}(K, c, m, 1_{n_1})$

$w'_j = \sum_{i=1}^{n_1} u_{ij}, \quad \forall j$

for $l = 2$ *to* s **do**

  $\mathrm{w} = \{\mathrm{w}', 1_{n_1}\}$

  $\xi' = \{\xi'(\mathrm{p}), 1_{n_1}\}$

  $K = [\kappa(\mathrm{x}_i, \mathrm{x}_j)], \quad i, j = \xi'$

  $U, \mathrm{p} = \mathrm{wkFCM}(K, c, m, \mathrm{w})$

  $w'_j = \sum_{i=1}^{n_l+c} u_{ij}, \quad \forall j$

$\mathrm{p} = \xi'(\mathrm{p})$

3.3.5 okFCM

okFCM 是基于核扩展的 oFCM 算法。二者的不同在于 okFCM 运用核函数进行了算法的优化。算法 10 所示为 okFCM 算法。

Algorithm 10: okFCM to approximately minimize $J_{m\mathrm{w}}(U;\kappa)$

Input: Kernel function κ, X, c, m, s

Output: U, p

Randomly draw s (approximately) equal-sized subsets of the intergers $\{1,\cdots,n\}$, denoted $E=\{\xi_1,\cdots\xi_s\}$. n_l is the cardinality of ξ_l.

$K=[\kappa(\mathrm{x}_i,\mathrm{x}_j)] \quad i,j=\xi_1$

$U_1,\mathrm{p}_1=\mathrm{wkFCM}(K,c,m,1_{n_1})$

for $l=2$ *to* s **do**

  $K=[\kappa(\mathrm{x}_i,\mathrm{x}_j)], \quad i,j=x'_l$

  $U_l,\mathrm{p}'=\mathrm{wkFCM}(K,c,m,1_{n_1})$

  $\mathrm{p}_l=\xi_l(\mathrm{p}')$

$\mathrm{p}_{all}=\{\mathrm{p}_1,\cdots,\mathrm{p}_s\}$

$K=[\kappa(\mathrm{x}_i,\mathrm{x}_j)], \quad i,j=\mathrm{p}_{all}$

$\mathrm{w}_l=\sum_{j=1}^{n_s}(U_l)_j, \quad \forall l$

$U,\mathrm{p}'=\mathrm{wkFCM}(K,c,m,\mathrm{w})$

$\mathrm{p}=\mathrm{p}_{all}(\mathrm{p}')$

第 4 章　复杂度对比

4.1　理论复杂度对比

不同模糊 C 均值算法的扩展版本的时间复杂性和空间复杂性对比如表 4.1 所示：

表 4.1: TIME AND SPACE COMPLEXITY OF FCM/AO VL ALGORITHMS

Algorithm	**Time**	**Space**
wFCM,LFCM	$O(tc^2dn)$	$O((d+c)n)$
resFCM	$O(tc^2dn/s)$	$O((d+c)(n/s))$
spFCM	$O(tc^2dn)$	$O((d+c)(n/s))$
oFCM	$O(tc^2dn)$	$O((d+c)(n/s)+cs)$
brFCM	$O(tc^2sd)+\text{bin}$	$O((d+c)s)$

不同基于核扩展的模糊 C 均值算法的扩展版本的时间复杂性和空间复杂性对比如表 4.2 所示：

表 4.2: TIME AND SPACE COMPLEXITY OF kFCM VL ALGORITHMS

Algorithm	**Time**	**Space**
wkFCM,kFCM	$O(tc^2n^2)$	$O(n^2)$
akFCM	$O(n^3/s^2+tcn^2/s)$	$O(n^2/s)$
rsekFCM	$O(tcn^2/s^2)$	$O(n^2/s^2)$
spkFCM	$O(tcn^2/s)$	$O(n^2/s^2)$
okFCM	$O(tcn^2/s+tc^3s^2)$	$O(n^2/s^2+s^2)$

4.2　复杂度实验测试

我们组对实现算法进行了实验测试，运用鸢尾花数据集（参见附录一）对算法时间复杂度进行了测量。实验结果如表 4.3 和表 4.4 所示。