第5章

类不均衡条件下基于脉内信息的雷达辐射源在线分选

5.1 引言

尽管越来越多的研究者开始试图利用机器学习手段实现雷达辐射源信号分选，但是大部分是将其视为闭集分类问题（Closed-set Classification Problem），即假设所有的待分选的辐射源信号均可提前获取相应的训练样本，然而这在实际中是极其不现实的，因为一些辐射源信号，特别是非合作辐射源的信号其实是很难预先获取的。因此，本书提出了在数据流聚类这一无监督学习框架下实现雷达辐射源信号分选。第3章、第4章也充分论证了这一思路的合理性。

值得注意的是，无论是利用监督还是非监督方法试图解决雷达辐射源信号分选问题，都存在一个重要的假设，即各辐射源雷达脉冲信号样本是均衡的。然而，这个假设在现实中其实很难得到满足。在复杂电磁环境中，各个辐射源信号的数量其实是极其不均衡的，辐射源因作战任务不同、工作次序不同等因素很难实现在一段时间内辐射的信号个数是彼此相当的。例如，对于雷达辐射源而言，不同雷达的脉冲重复频率（PRI）相差甚远，势必造成各个雷达辐射源脉冲信号在数量上的不均衡性。

其实在现实社会中，这种不平衡数据集也是广泛存在的。典型的例子包括网

络中正常邮件和垃圾邮件、销售业中的日常用品和大宗货物、气象数据中的普通气象和极端气象等。可以说，类不平衡是现实世界中的普遍状态，而目前在无监督聚类以及监督分类领域，如何实现对不平衡数据集的聚类以及分类仍然是亟待解决的现实问题。因此本章重点关注如何对非平衡脉冲流进行在线聚类以达到分选目的。

本章各节安排如下：5.2 节主要对雷达非均衡演化数据流的分选问题进行分析与建模，定义辐射源不平衡特性，从理论上分析数据不平衡给聚类算法带来挑战的根本原因，为后续研究铺垫理论基础。5.3 节对 ESC 算法进行简要的总结与分析，重点分析 ESC 算法的局限性，并结合实验对 ESC 算法的局限性进行验证。5.4.1 节首先提出 I-ESC 算法，重点解决 ESC 算法的初始化敏感问题，5.4.2 节提出 DI-ESC 算法，实现对非均衡演化数据流的在线子空间聚类。5.5 节利用实测数据进行大量的仿真实验，对 I-ESC 算法以及 DI-ESC 算法的有效性及优越性进行充分验证与分析，同时对 I-ESC 算法及 DI-ESC 算法的敏感度进行实验分析。5.6 节对本章主要内容进行总结。

5.2 问题分析与建模

5.2.1 雷达辐射源非均衡演化脉冲流在线分选问题建模

本章继续考虑一个典型的雷达辐射源在线分选场景：在 t 时刻，共有 k^t 个雷达辐射源（为了行文简洁，在下文中称其为辐射源）同时工作，每个辐射源可以用 $\mathcal{E}$ 来表示，那么 k^t 个辐射源可以表示为 $\mathbb{E}^t=\{\mathcal{E}_k^t\}_{i=1}^{k^t}$。与此同时，假设有效接收范围内存在一接收机接收这些辐射源持续不断产生的信号脉冲。

假设在 t 时刻，接收机接收到的脉冲信号为 $\boldsymbol{p}^t$，且 $\boldsymbol{p}^t\in\mathbb{R}^{D\times 1}$。假设不存在脉冲之间的交叠，且 1 个时间戳只接收 1 个脉冲。如此，接收到的脉冲流可表示为 $\boldsymbol{P}=\{\boldsymbol{p}^t\}_{t=1}^{N}(N\to\infty)$。为了方便后续讨论，将截至 t 时刻之前（包含 t）接收到的所有脉冲按列依次排序组成矩阵 $\boldsymbol{P}^t$，即 $\boldsymbol{P}^t=[\boldsymbol{p}^1\cdots\boldsymbol{p}^t]_{D\times t}$。

如前所述，辐射源演化特性是客观存在的，本章将继续考虑三种最典型的辐射源演化形式，即辐射源的出现、消失与复现。

- 辐射源出现：是指一个新的辐射源在 t 时刻开始工作。某辐射源 $\mathcal{E}$ 满足 $\mathcal{E}\notin\mathbb{E}^1\cup\mathbb{E}^2\cup\cdots\cup\mathbb{E}^{t-1}$ 且 $\mathcal{E}\in\mathbb{E}^t$，称该辐射源在 t 时刻出现。

- 辐射源消失：是指之前已经存在的辐射源在最近一段时间不再工作，即若存在一个辐射源$\mathcal{E}$满足$\mathcal{E}\in \mathbb{E}^{t_0}\cap \mathbb{E}^{t_0+1}\cap\cdots\cap \mathbb{E}^{t-1}$且$\mathcal{E}\notin \mathbb{E}^{t}$，同时$1\leqslant t_0<t$，则称$\mathcal{E}$消失。
- 辐射源复现：是指一个之前消失的辐射源在t时刻再次出现，即若辐射源$\mathcal{E}$满足$\mathcal{E}\in \mathbb{E}^{t_1}\cap \mathbb{E}^{t_1+1}\cap\cdots\cap \mathbb{E}^{t_2-1}$，$\mathcal{E}\notin \mathbb{E}^{t_2}\cup \mathbb{E}^{t_2+1}\cup\cdots\cup \mathbb{E}^{t-1}$和$\mathcal{E}\in \mathbb{E}^{t}$，其中$1\leqslant t_1<t_2<t$，则称辐射源$\mathcal{E}$在$t$时刻复现。

辐射源除具有演化特性之外，还具有不平衡特性，这种特性导致了各个辐射源信号样本的不平衡，给聚类带来很大的挑战，本章将重点解决辐射源不平衡问题，现对辐射源不平衡特性作如下定义：

辐射源不平衡特性：辐射源之间因具有不同的工作时间以及脉冲重复频率，极易造成接收机接收到的脉冲流中源自各个辐射源的信号样本的数量分布不均衡的现象。

假设在t_1到$t_2(t_2>t_1)$时间段内存在$l_{(t_1,t_2)}$个辐射源工作，其中辐射源$\mathcal{E}_i(i=1,2,\cdots,l_{(t_1,t_2)})$在此阶段共发射$n_i$个脉冲，且辐射源存在不平衡特性。将发射脉冲数量大的辐射源称为过表达辐射源，而将发射脉冲数量小的辐射源称为欠表达辐射源。同时，为了对不平衡度进行量化，将发射脉冲最大的辐射源表示为$\mathcal{E}_{\max}$，将发射脉冲最小的辐射源表示为$\mathcal{E}_{\min}$，则定义不平衡度γ：

$$\gamma=\frac{n_{\max}}{n_{\min}} \tag{5.1}$$

其中，$n_{\max}$与$n_{\min}$分别为辐射源$\mathcal{E}_{\max}$和$\mathcal{E}_{\min}$辐射的脉冲个数。目前，大部分非平衡数据的研究普遍认为一般非平衡数据$\gamma\geqslant 4$[290]。

基于以上分析，本章重点关注雷达非均衡演化脉冲流的分选问题。

雷达辐射源非均衡演化脉冲流的在线分选问题：给定脉冲流$\boldsymbol{P}^t$的条件下，已知$\boldsymbol{P}^t$具有演化性以及非均衡性质，雷达非均衡演化脉冲流的在线分选是实现在任意t时刻，确定辐射源$\mathbb{E}^t=\{\mathcal{E}_i^t\}_{i=1}^{k^t}$以及为每个脉冲$\boldsymbol{p}^t$分配一个对应的辐射源$\mathcal{E}_i(i\in[1,k^t])$。

5.2.2　雷达辐射源非均衡演化脉冲流在线分选问题分析

本质上，脉冲流就是一个具有演化和非均衡性质的特殊的数据流。正如第4章所分析的，来自同一辐射源的脉冲可以被假定位于同一个子空间上，而来自不同辐射源的脉冲位于不同子空间上。从这个角度讲，雷达非均衡演化脉冲流的分选

问题就是对特殊性质数据流的在线子空间聚类问题。

目前，大部分聚类算法都是面向的分布均衡的数据集[220]，即数据集的点在各个类中的分布基本相当。一般地，称这样的数据集为类均衡数据集；反之，将那些在各个类中分布不均衡的数据集称为类不均衡数据集。我们定义类不均衡数据集中点数较少的类为欠表达类，反之为过表达类。

然而，对非均衡演化数据流的子空间聚类不仅在雷达信号处理领域，在数据流处理领域仍然是极具挑战的问题之一。原因在于非均衡数据将破坏数据点的子空间保持特性。

定义 5.1 （子空间保持特性）[291]：由数据点自表示特性（定义 2.1）可知，给定数据集 $\boldsymbol{X}=[\boldsymbol{x}_1\cdots\boldsymbol{x}_N]_{d\times N}\in\mathbb{R}^{d\times N}$，对于 $\forall\,\boldsymbol{x}_i$，存在一个表示向量 $\boldsymbol{c}_i$，满足

$$\begin{cases}\boldsymbol{x}_i=\sum\limits_{i\neq j}\boldsymbol{c}_{ij}\boldsymbol{x}_j\\ \text{且 }\boldsymbol{c}_{ij}\neq 0\text{，仅当 }\boldsymbol{x}_i,\boldsymbol{x}_j\text{ 属于同一个子空间}\end{cases}\tag{5.2}$$

其中，$\boldsymbol{c}_i=[c_{i1}c_{i2}\cdots c_{iN}]^{\mathrm{T}}\in\mathbb{R}^{N\times 1}$ 且 $c_{ii}=0$。这种性质称为子空间保持特性。下面以 SSC 算法为例，具体分析非均衡数据对子空间保持特性的影响。

例证 5.1　非均衡数据对 SSC 算法的影响

现仿真生成两个维度分别为 3，4 的子空间 $\mathcal{S}_1$，$\mathcal{S}_2$ 并将其投射到维度为 30 的仿射空间。不失一般性，假设 $\mathcal{S}_1$ 为过表达子空间，而 $\mathcal{S}_2$ 为欠表达子空间。现从子空间 $\mathcal{S}_1$，$\mathcal{S}_2$ 中分别均匀随机地抽取 n_1 与 n_2 个点。假设 $\mathcal{S}_1$ 与 $\mathcal{S}_2$ 中分别抽取的点集为 $\boldsymbol{X}_1$，$\boldsymbol{X}_2$，将点集组成集合 $\boldsymbol{X}=[\boldsymbol{X}_1\boldsymbol{X}_2]$。不平衡度为 $\gamma=\dfrac{n_1}{n_2}$。利用 SSC 算法[203]对点集 $\boldsymbol{X}$ 进行子空间聚类。当 $\gamma=4,5,\cdots,14$ 时，分别求取聚类准确率（Accuracy），并将结果绘制在图 5.1 中。由图 5.1 观察可知，随着数据集不平衡度增加，聚类准确率逐渐降低。

下面对该结果进行简要的分析。基于式（2.6），SSC 解决的优化问题是

$$\min_{\boldsymbol{c}_i\in\mathbb{R}^N}\|\boldsymbol{c}_i\|_1+\frac{\lambda}{2}\left\|\boldsymbol{x}_i-\sum_{i\neq j}c_{ij}\boldsymbol{x}_j\right\|_2^2\tag{5.3}$$

其中，$\lambda>0$；$\|\cdot\|_1$ 和 $\|\cdot\|_2$ 分别表示 ℓ_1 范数和 ℓ_2 范数；$\boldsymbol{c}_i=[c_{i1},\cdots,c_{iN}]^{\mathrm{T}}$ 且 $c_{ii}=0$ 是为了避免无意义解 $\boldsymbol{x}_i=\boldsymbol{x}_i$。

对于类均衡数据集，点 $\boldsymbol{x}_i$ 的解 $\boldsymbol{c}_i$ 中的非零元素将对应自己的同类，即子空间保持特性。然而，对于类不均衡数据集，若 $\boldsymbol{x}_i$ 为欠表达类的点，则其对应的 $\boldsymbol{c}_i$ 内

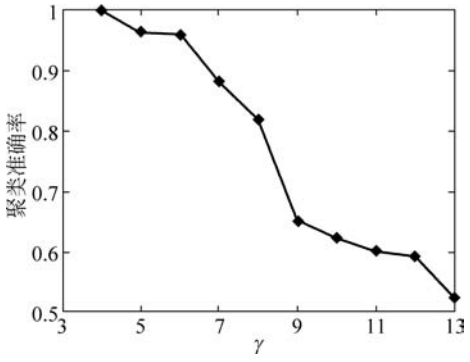

图 5.1　不同不平衡度 γ 下 SSC 算法的聚类准确度

的非零元素更容易对应过表达类，也就是说欠表达类的点容易被过表达类所吞并[220]。尽管这一现象更深层次的原因还未被发现，但对这一现象的研究近期得到广泛关注。文献[220]针对上述现象提出了 ESC 算法解决类不均衡数据集的子空间聚类问题。

5.3　面向非均衡数据的静态聚类算法——ESC 算法

2018 年，You 在计算机科学领域顶级会议 ECCV 上首次提出 Exemplar-based Subspace Clustering(ESC)算法①。ESC 算法主要针对解决数据的非均衡特性影响子空间保持特性这一问题，经仿真对比实验验证，ESC 算法在处理非均衡数据集方面达到先进水平。

假设给定待聚类的点集 $\mathcal{X}=\{\boldsymbol{x}_i\}_{i=1}^{N}$，ESC 算法通过寻找一个子集 $\mathcal{X}_0^*$ 来降低 $\mathcal{X}$ 的不平衡度，$\mathcal{X}_0^* \subseteq \mathcal{X}$。对 $\mathcal{X}_0^*$ 而言，ESC 期待 $\mathcal{X}_0^*$ 能够尽可能压缩其尺寸，同时能最大程度代表原点集 $\mathcal{X}$。因此，ESC 定义了 $\mathcal{X}_0$ 对原点集 $\mathcal{X}$ 的损失函数：

$$F_\lambda(\mathcal{X}_0)=\sup_{\boldsymbol{x}_i\in\mathcal{X}} f_\lambda(\boldsymbol{x}_i,\mathcal{X}_0) \tag{5.4}$$

且

$$f_\lambda(\boldsymbol{x}_i,\mathcal{X}_0)=\min_{\boldsymbol{c}_i\in\mathbb{R}^N}\|\boldsymbol{c}_i\|_1+\frac{\lambda}{2}\left\|\boldsymbol{x}_i-\sum_{j:\,\boldsymbol{x}_j\in\mathcal{X}_0} c_{ij}\boldsymbol{x}_j\right\|_2^2 \tag{5.5}$$

① ESC 算法下载地址：https://github.com/chongyou。

其中，$\lambda\in(1,\infty)$是一个输入调节参数且假定对于所有的$\boldsymbol{x}_i\in\mathcal{X}$，有$f_\lambda(\boldsymbol{x}_i,\varnothing)=\frac{\lambda}{2}$。

由式(5.5)分析可知，$f_\lambda(\boldsymbol{x},\mathcal{X}_0)$衡量的是点$\boldsymbol{x}$被集合$\mathcal{X}_0$的覆盖程度。那么若$\mathcal{X}_0$中含有$\boldsymbol{x}_i$点，式(5.5)的解$\boldsymbol{c}_i$将使$f_\lambda(\boldsymbol{x}_i,\mathcal{X}_0)$接近0($\boldsymbol{c}_i$是稀疏的)。这意味着$\mathcal{X}_0$的选取应该选择那些使得$f_\lambda(\boldsymbol{x}_i,\mathcal{X}_0)$尽量小的点。由于$F_\lambda(\mathcal{X}_0)$被定义为最大的$f_\lambda(\boldsymbol{x}_i,\mathcal{X}_0)$的值，因此$\mathcal{X}_0$的选择应该满足下面约束：

$$\mathcal{X}_0^*=\underset{|\mathcal{X}_0|\leqslant N_0}{\operatorname{argmin}}F_\lambda(\mathcal{X}_0) \tag{5.6}$$

其中，$\mathcal{X}_0^*$中的点称为代表点(Exemplar)，$\mathcal{X}_0^*$称为代表点集(Exemplar Set)；N_0是代表点集中代表点的个数，是一个需要提前由用户设定的参数。通常，一个理想的$\mathcal{X}_0$应该是尽可能多地对数据集$\mathcal{X}$进行覆盖。

式(5.6)这个优化问题其实是一个NP难问题，它需要我们对$\mathcal{X}_0^*$的每一个小于或等于N_0的子集进行评估。因此，ESC通常是由FFS(Farthest First Search)算法进行近似解决的，FFS算法伪代码见算法5.1。

算法5.1 Farthest First Search(FFS)算法

输入：数据集$\mathcal{X}=[\boldsymbol{x}_1,\cdots,\boldsymbol{x}_N]\subseteq\mathbb{R}^{D\times N}$，参数$\lambda>1$，$k\ll N$；

首先在$\mathcal{X}$中随机选择一点$\boldsymbol{x}$，将$\mathcal{X}_0^{(1)}\leftarrow\boldsymbol{x}$

For $i=1,\cdots,k-1$ **do**

$$\mathcal{X}_0^{(i+1)}=\mathcal{X}_0^{(i)}\bigcup\underset{\boldsymbol{x}\in\mathcal{X}}{\operatorname{argmax}}f_\lambda(\boldsymbol{x},\mathcal{X}_0^{(i)})$$

End

输出：$\mathcal{X}_0^{(k)}$

FFS算法的核心思想是随机选择一个点作为代表点集的第一个元素，随后逐个将$\mathcal{X}$中与当前代表点集中最不相似的点吞并到代表点集合中去，如此便使得$\mathcal{X}_0$中尽可能多地囊括了那些与$\mathcal{X}$最不相似的点，从而使得式(5.4)达到最小化。同时，$\boldsymbol{c}_i$可以用作建立相似度图，通过对相似度图进行谱聚类可以得到聚类结果。需要指出的是，ESC仍然具有两个不容忽视的缺点：其一是ESC算法因为FFS的随机初始化而极其不稳定，对初始化很敏感，下面利用例证5.2对ESC算法的性能进行分析。

例证5.2 ESC算法稳定性验证

本例证分别采用两种应用最广泛的手写数字图像集MNIST[288,291]数据集和USPS[286]数据集作为实验数据，两种数据集均为对阿拉伯数字0～9的手写图像。

MNIST 数据集共包含 70000 幅图像，每幅图像尺寸为 28×28；USPS 数据集共包含 9298 幅图像，每幅图像尺寸为 16×16。关于两种数据集更详细的介绍参见 4.4.1 节。

为了简化仿真实验，在本例证中，仅选择数据集中的偶数(0,2,4,6,8)类所对应的图像，并在各自类中随机选取 100 幅分别构成数据集 $\boldsymbol{X}_{\text{mnist}}$ 与 $\boldsymbol{X}_{\text{usps}}$。随后用 ESC 算法分别对 $\boldsymbol{X}_{\text{mnist}}$ 与 $\boldsymbol{X}_{\text{usps}}$ 进行重复实验处理，针对数据集 $\boldsymbol{X}_{\text{mnist}}$ 与 $\boldsymbol{X}_{\text{usps}}$ 依次独立重复 20 次处理，将处理结果绘制在图 5.2 中。由图 5.2 观察可知，ESC 算法对两种数据集的聚类准确率(Accuracy)波动非常大，例如对于 MNIST 数据集，最高准确率与最低准确率之差高达 0.42(最高准确率为 0.78，最低准确率为 0.36)，而 USPS 数据集则接近 0.46(最高准确率为 0.96，最低准确率为 0.50)。

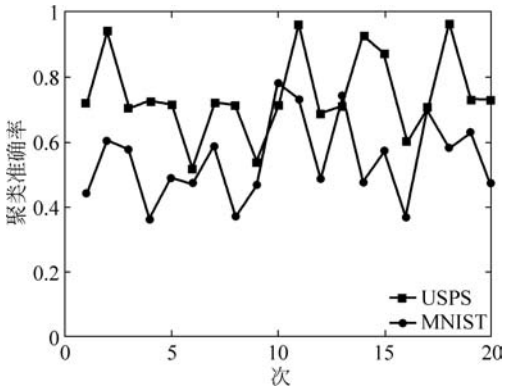

图 5.2　ESC 算法对数据集(MNIST 和 USPS)处理聚类准确率

ESC 算法之所以性能如此波动，是因为 FFS 算法的随机初始化。由算法 5.1 可知，FFS 在确定代表点集合 $\mathcal{X}_0$ 时，随机选择某一点作为 $\mathcal{X}_0^{(1)}$ 的第一个点，并以此为基础，逐步选取那些满足 $\underset{\boldsymbol{x}\in\mathcal{X}}{\operatorname{argmax}} f_\lambda(\boldsymbol{x},\mathcal{X}_0)$ 的点。整个优化达到的仅仅是 $\mathcal{X}_0^{(1)}$ 所引导的局部最优解，因此，ESC 算法的波动性很大。

ESC 算法的第二个缺点是其仅能处理静态数据集，对于存在更广泛的数据流，其没有找到合理的解决方案，因为 ESC 算法是基于数据点的自表示特性与空间保持特性展开的，理论上需要尽可能多的点作为表示字典去寻求稀疏表示向量。而数据流处理则无法存取大量的历史点，这制约了 ESC 算法向数据流处理的应用。

5.4 基于 DI-ESC 的雷达辐射源在线分选算法

脉冲流的演化特性以及非均衡特性对雷达辐射源在线分选均提出了很大的挑战。在本节中，将雷达在线分选问题转化为对具有演化和非均匀的数据流的在线子空间聚类问题，每个辐射源辐射的信号假设分布在同一个子空间上，而不同辐射源辐射的信号分布在不同的子空间上。由于 ESC 算法存在着性能不稳定的缺点，因此本章在 ESC 算法基础上，首先提出了 improved ESC(I-ESC)算法(5.4.1 节)。与原始 ESC 算法相比，I-ESC 算法对均衡及非均衡数据集处理的鲁棒性更强。然后基于 I-ESC 算法，提出了 Dynamic Improved ESC(DI-ESC)算法(5.4.2 节)。DI-ESC 算法具有数据流演化检测的能力。

DI-ESC 不仅能够实现雷达辐射源的在线分选，还能在其他领域处理类的问题。因此，在本章中除特殊说明外，不再强调雷达分选的任务背景，使得 DI-ESC 算法能够作为一个通用模型用于解决其他领域类似问题。脉冲流也被抽象为一个具有特殊性质的数据流，辐射源在本章中被抽象为子空间。

5.4.1 Improved ESC 算法

本质上，ESC 通过式(5.4)为原待聚类集合$\mathcal{X}$寻找代表点子集$\mathcal{X}_0^*$，从而大幅降低了$\mathcal{X}$的不平衡度。根据文献[220]，式(5.4)的近似解可由 FFS 算法获得。5.3 节分析指出，FFS 算法是一个随机初始化算法，即从$\mathcal{X}$中任选一点作为代表点集合$\mathcal{X}_0^*$的第一个代表点$\mathcal{X}_0^{(1)*}$，然后基于$\mathcal{X}_0^{(1)*}$逐步选取其余代表点直到形成$\mathcal{X}_0^*$。然而，这种随机选择第一个代表点的做法导致 ESC 对$\mathcal{X}_0^{(1)*}$极其敏感，造成 ESC 算法不稳定。显然，$\mathcal{X}_0^{(1)*}$的选取对整个 ESC 算法的性能影响很大，因此，$\mathcal{X}_0^{(1)}$不应该任意选择。

由 5.3 节可知，$F_\lambda(\mathcal{X}_0)$的最小化实际上是$\mathcal{X}_0$尽可能包括那些不能被其余点以较少损失进行表示的点[220]。这样，在式(5.4)限制下，那些与其余点不相似的点应该有希望被选择为代表点。因此，下面提出选取第一个代表点的新原则：

$$\mathcal{X}_0^{(1)*}=\min_{\boldsymbol{x}_i\in\mathcal{X}}\Big[\sum_{\substack{\boldsymbol{x}_j\in\mathcal{X}\\ j\neq i}}S(\boldsymbol{x}_i,\boldsymbol{x}_j)\Big] \tag{5.7}$$

其中，$\mathcal{S}(\cdot)$是相似度函数。例如，$\mathcal{S}(\cdot)$可以具体被定义为欧氏空间距离函数、相关函数(Correlation Function)或者任何可以衡量两点相似度的函数。由于在高维空间中，欧氏距离函数对高维数据点的相似度衡量作用有限，因此在本章，将$S(\cdot)$定义为相关函数，即

$$S(\boldsymbol{x}_i,\boldsymbol{x}_j)=\frac{\mathrm{Cov}(\boldsymbol{x}_i,\boldsymbol{x}_j)}{\sqrt{\boldsymbol{x}_i}\sqrt{\boldsymbol{x}_j}}=\frac{E(\boldsymbol{x}_i-E(\boldsymbol{x}_i))E(\boldsymbol{x}_j-E(\boldsymbol{x}_j))}{\sqrt{\boldsymbol{x}_i}\sqrt{\boldsymbol{x}_j}} \tag{5.8}$$

其中，$E(\cdot)$是期望函数。

基于式(5.8)选出来的代表点子集的初始点$\mathcal{X}_0^{(1)*}$，可以利用下式继续逐步选取其余点作为代表点，直至满足终止条件。

$$\mathcal{X}_0^{(i+1)*}=\mathcal{X}_0^{(i)*}\bigcup\underset{\boldsymbol{x}\in\mathcal{X}}{\operatorname{argmax}}f_\lambda(\boldsymbol{x},\mathcal{X}_0^{(i)}) \tag{5.9}$$

由5.3节可知，ESC算法需要用户预先直接指定代表点个数，即N_0。然而不同数据集包括的数据点是不同的，因此所选出来的代表点子集也应该是变化的，因此在处理不同数据集时，需要用户不断变换N_0。与ESC算法不同的是，I-ESC引入一个新的参数η来控制所选的代表点的个数。

$$N_0=\lceil N*\eta\rceil \tag{5.10}$$

其中，$\lceil N*\eta\rceil$表示取整函数，且取不大于$N*\eta$的整数。

由式(5.5)可知，计算$f_\lambda(\boldsymbol{x},\mathcal{X}_0^{(i)})$本身是比较复杂的，因为需要计算稀疏优化问题。下面对式(5.5)的函数$f_\lambda(\boldsymbol{x},)$进行分析。

引理5.1：$f_\lambda(\boldsymbol{x},)$函数对于按集合包含顺序排列的集合是单调的，即对于任何$\varnothing\subset\mathcal{X}'_0\subseteq\mathcal{X}''_0\subseteq\mathcal{X}$，有$f_\lambda(\boldsymbol{x},\mathcal{X}'_0)\geqslant f_\lambda(\boldsymbol{x},\mathcal{X}''_0)$。

证明：已知非空集合$\mathcal{X}$，及其两非空子集$\mathcal{X}'_0$，$\mathcal{X}''_0$，且有如下关系成立：$\varnothing\subset\mathcal{X}'_0\subseteq\mathcal{X}''_0\subseteq\mathcal{X}$。

为了便于表示，对于任一点$\boldsymbol{x}_i\in\mathcal{X}$，现令函数$g(\boldsymbol{c}_i)=\|\boldsymbol{c}_i\|_1+\frac{\lambda}{2}\left\|\boldsymbol{x}_i-\sum\limits_{j:\,\boldsymbol{x}_i\in\mathcal{X}'}c_{ij}\boldsymbol{x}_j\right\|_2^2$，则式(5.5)可变换为

$$f_\lambda(\boldsymbol{x}_i,\mathcal{X}'_0)=\min_{\boldsymbol{c}'_i\in\mathbb{R}^N}g(\boldsymbol{c}'_i,\mathcal{X}'_0)$$

注意为了区分，将上式最优解写为$\boldsymbol{c}'_i$。

现假设对于$\mathcal{X}''_0$，存在一解使得$f_\lambda(\boldsymbol{x},\mathcal{X}'_0)<f_\lambda(\boldsymbol{x},\mathcal{X}''_0)$，说明$\exists\,\boldsymbol{c}''$满足

$$g(\boldsymbol{c}''_i,\mathcal{X}''_0)>g(\boldsymbol{c}'_i,\mathcal{X}'_0) \tag{5.11}$$

且$g(\boldsymbol{c}''_i,\mathcal{X}''_0)$为所有$g(\boldsymbol{c}'_i,\mathcal{X}'_0)$中最小值。

因为$\varnothing\subset\mathcal{X}'_0\subseteq\mathcal{X}''_0\subseteq\mathcal{X}$

所以

$$g(\boldsymbol{c}'_i,\mathcal{X}''_0)=g(\boldsymbol{c}''_i,\mathcal{X}''_0) \tag{5.12}$$

则进一步有

$$g(\boldsymbol{c}''_i,\mathcal{X}''_0)>g(\boldsymbol{c}'_i,\mathcal{X}''_0) \tag{5.13}$$

这与 $g(\boldsymbol{c}''_i,\mathcal{X}''_0)$为所有 $g(,\mathcal{X}''_0)$中最小值相矛盾,因此假设不成立。

故 $f_\lambda(\boldsymbol{x},\mathcal{X}'_0)\geqslant f_\lambda(\boldsymbol{x},\mathcal{X}''_0)$。

基于引理 5.1,可以避免在每次循环计算对所有点都计算 $f_\lambda(\boldsymbol{x},)$。由算法 5.1 可知,代表子集$\mathcal{X}_0$是逐渐增加的,这意味着 $f_\lambda(\boldsymbol{x},\mathcal{X}_0^{(i)})$是非增的,因此可按照如下方式对 FFS 进行加速:

算法 5.2 FFS 的加速算法

输入:数据集$\mathcal{X}=[\boldsymbol{x}_1,\cdots,\boldsymbol{x}_N]\subseteq\mathbb{R}^{D\times N}$,参数 $\lambda>1,\eta$;

1. 按照式(5.7)和式(5.8)选择$\mathcal{X}_0^{(1)}$;
2. 计算 $b_j=f_\lambda(\boldsymbol{x}_j,\mathcal{X}_0^{(1)})$,其中 $j=1,2,\cdots,N$;
3. 计算 $N_0=\lceil N*\eta\rceil$;

For $i=1,\cdots,N_0$ **do**

将 N 个点的 b 值按从大到小进行排序,排序顺序为 $o_1,\cdots,o_N$,满足当 $p\geqslant q$ 有 $b_{o_p}\geqslant b_{o_q}$,

令 max_cost=0,

For $j=1,\cdots,N$ **do**

令 $b_{o_j}=f_\lambda(\boldsymbol{x}_{o_j},\mathcal{X}_0^{(i)})$

If $b_{o_j}>$max_cost **then**

max_cost=b_{o_j} new_index=o_j

End

If $j=N$ 或者 max_cost$\geqslant b_{o_{j+1}}$ **then**

中断

End

End

$\mathcal{X}_0^{(i+1)}=\mathcal{X}_0^{(i)}\cup\{\boldsymbol{x}_{\text{new\_index}}\}$

End

输出:$\mathcal{X}_0^{(N_0)}$

如算法 5.2 所示,在第 2 行,将$\mathcal{X}$中每一个点都计算 $f_\lambda(\boldsymbol{x}_j,\mathcal{X}_0^{(1)})$,由引理 5.1 知这其实是后续 $f_\lambda(\boldsymbol{x}_j,\mathcal{X}_0^{(i)})$的上确界。在每次 i 循环,目标是找到使 $f_\lambda(\boldsymbol{x}_j,\mathcal{X}_0^{(i)})$最大化的点,并将其吸收进$\mathcal{X}_0^{(i)}$。依次按照 $\boldsymbol{x}_{o_1},\cdots,\boldsymbol{x}_{o_N}$ 顺序计算 $f_\lambda(,\mathcal{X}_0^{(i)})$

同时用变量 max_cost 来追踪 $f_\lambda(\cdot,\mathcal{X}_0^{(i)})$的最大值。一旦达到 max_cost$\geqslant b_{o_{j+1}}$，则认为对于任何 $j'>j$，点 $\boldsymbol{x}_{o_{j'}}$ 不会使得 $f_\lambda(\cdot,\mathcal{X}_0^{(i)})$更大。这是因为 $f_\lambda(\boldsymbol{x}_{o_{j'}},\mathcal{X}_0^{(i)})\leqslant b_{o_{j'}}\leqslant b_{o_{j+1}}\leqslant$max_cost，从而避免了计算后续 $\boldsymbol{x}_{o_{j'}}$ 的 f_λ 值。

在$\mathcal{X}_0$确认之后，原集合$\mathcal{X}$的子空间聚类结果可以基于$\mathcal{X}_0$得到。具体地，对于每一个属于$\mathcal{X}$的点 $\boldsymbol{x}_i$，按照下式计算：

$$\min_{\boldsymbol{c}_i\in\mathbb{R}^N}\|\boldsymbol{c}_j\|_1+\frac{\lambda}{2}\Big\|\boldsymbol{x}_i-\sum_{j:\,\boldsymbol{x}_j\in\mathcal{X}_0}c_{ij}\boldsymbol{x}_j\Big\|_2^2 \tag{5.14}$$

其中，$\boldsymbol{c}_i$ 称为 $\boldsymbol{x}_i$ 在$\mathcal{X}_0$下的稀疏表示系数，简称表示系数。由定义 5.1 可知，$\boldsymbol{c}_i$ 将具有子空间保持特性，即对于任意两点$\{\boldsymbol{x}_i,\boldsymbol{x}_j\}\subseteq\mathcal{X}$，$\langle\boldsymbol{x}_i,\boldsymbol{x}_j\rangle\neq0$ 当且仅当 $\boldsymbol{x}_i,\boldsymbol{x}_j$ 来自同一个子空间。在计算$\mathcal{X}$中每一个点的表示系数后，可以由 K-最近邻方法(K-Nearest Neighbor)来对$\mathcal{X}$的分割。具体地，首先将表示系数进行标准化，即

$$\tilde{\boldsymbol{c}}_i=\frac{\boldsymbol{c}_i}{\|\boldsymbol{c}_i\|_2} \tag{5.15}$$

其次，找到 $\tilde{\boldsymbol{r}}_i$ 的 K 个近邻，即计算内积$\langle\boldsymbol{c}_i,\boldsymbol{c}_j\rangle$，其中 $j=1,2,\cdots,N$ 且 $j\neq i$。并选择 K 个绝对值最大的作为 $\tilde{\boldsymbol{r}}_i$ 的近邻。

然后构建亲密度矩阵 $\boldsymbol{W}$，即

$$\boldsymbol{W}=\boldsymbol{A}+\boldsymbol{A}^{\mathrm{T}} \tag{5.16}$$

其中，当 $\tilde{\boldsymbol{r}}_j$ 是 $\tilde{\boldsymbol{r}}_i$ 的 K 近邻时 $\boldsymbol{A}_{ij}=1$，否则 $\boldsymbol{A}_{ij}=0$。

最后运用谱聚类思想对 $\boldsymbol{W}$ 进行分割，从而实现对$\mathcal{X}$集合的聚类。算法 5.3 对 I-ESC 算法进行了总结。

算法 5.3　Improved ESC(I-ESC)算法

输入：数据集$\mathcal{X}=[\boldsymbol{x}_1,\cdots,\boldsymbol{x}_N]\subseteq\mathbb{R}^{D\times N}$，参数 $\lambda>1,\eta,K$；

步骤一：通过算法 5.2 确定代表子集$\mathcal{X}_0$；

步骤二：对于任意点 $\boldsymbol{x}_i\in\mathcal{X}$通过式(5.14)计算其稀疏表示系数 $\boldsymbol{c}_i$；

步骤三：通过式(5.15)对表示系数标准化；

步骤四：通过式(5.16)构建亲密度矩阵；

步骤五：对 $\boldsymbol{W}$ 进行谱分割[292,293]来获取对$\mathcal{X}$的聚类结果。

输出：$\mathcal{X}$的聚类结果。

I-ESC 算法在 ESC 算法基础上，重点解决了 ESC 算法随机初始化带来的性能极其不稳定的问题。与 ESC 算法类似，I-ESC 算法通过确立一个代表点子集，该代表点子集有效降低了原数据集的不平衡度，从而避免了不平衡度对子空间自保持

特性造成的影响。在该代表点子集上，自保持特性得以更好的利用，并且利用该特性可以有效实现对原集合进行的子空间聚类。

尽管 I-ESC 算法对 ESC 算法的性能进行了改进与提升，但值得注意的是，I-ESC 主要是面对静态数据集的，而无法有效处理数据流。为了扩大 I-ESC 算法的应用范围，在下一节将以 I-ESC 算法为基础，提出 Dynamic I-ESC（DI-ESC）算法，使得 I-ESC 的思想可以应用到处理非平衡数据流中。

5.4.2 Dynamic Improved ESC 算法

与 ESC 算法相比，I-ESC 算法以更强的鲁棒性处理非均衡数据集。然而，I-ESC 算法仍然局限于仅可以处理静态数据集，而不可以用于数据流处理。因此本节将 I-ESC 算法扩展为一个在线算法，称为 DI-ESC（Dynamic Improved Exemplar-based Subspace Clustering）算法。DI-ESC 算法可以对非均衡演化数据流进行在线子空间聚类。

将 I-ESC 扩展成 DI-ESC 算法有三个挑战：①在线聚类算法对计算速度和存储有严格的限制，导致 DI-ESC 算法不能存储所有的历史点。然而，子空间聚类需要存储尽可能多的点利用子空间自保持特性。因此，需要平衡这种矛盾。②I-ESC 算法是批量处理算法，但是要处理数据流需要具有增量处理方式，因此 I-ESC 算法需要改变处理方法。③子空间的演化特性需要被有效地检测与适应。综上，在第 1 小节提出了 DI-ESC 概要，来平衡数据流聚类与子空间聚类关于舍弃点与保存点的矛盾。然后，在第 2 小节解释了 DI-ESC 算法基于 DI-ESC 概要实现在线子空间聚类。最后在 3 小节提出了子空间演化检测策略来确保 DI-ESC 算法检测和适应子空间演化特性。

1. DI-ESC 概要

由于数据流是源源不断流入的，具有潜在无限性，为了节约存储资源，既不可能等到收集所有数据点再统一处理，也不能将所有的点都存储并反复读取。因此需要设计一个概要结构实时存储聚类的结果，数据流概要反映了数据流当前的模式。本节主要介绍 DI-ESC 概要。

DI-ESC 概要压缩存储了数据流的必要历史信息以及可能在后续聚类中用到的关键信息。DI-ESC 概要使得 DI-ESC 算法能够更加简洁灵活地处理数据流。将 DI-ESC 用$\mathbb{S}^t$ 表示，上角标 t 是时间戳，表示 DI-ESC 概要是随着时间变化的。DI-ESC 算法实际上是持续地对数据流挖掘子空间的算法。在每一个时刻，

DI-ESC 算法对已经找到的子空间按是否被最近流入的点访问而分为活跃态与非活跃态。活跃态表示该子空间可以反映数据流当前的部分模式；而非活跃态恰恰相反，表示已经在当前一段时间内没有流入的点访问该子空间。子空间的活跃态与非活跃态随着时间可以相互转化。只有活跃态的子空间的概要信息才可以保存在 DI-ESC 概要中。将活跃态子空间的概要信息表示为$\mathcal{S}_i$，将非活跃态子空间的概要信息表示为$\mathcal{D}_i$。假设在 t 时刻，共有 k^t 个活跃子空间与 h^t 个非活跃子空间，则$\mathbb{S}^t=\{\mathcal{S}_i\}_{i=1}^{k^t}$。DI-ESC 算法为非活跃子空间设置了一个存储池，称为非活跃子空间存储池，表示为$\mathbb{D}^t$，则$\mathbb{D}^t=\{\mathcal{D}_i\}_{i=1}^{h^t}$。但值得注意的是，对于活跃以及非活跃子空间来说不是所有的点都值得保存在子空间概要中。相反，子空间概要作为一个对该子空间信息的压缩，应该尽量从宏观对该子空间的历史信息进行压缩表示。

具体地，DI-ESC 对活跃以及非活跃子空间的概要做如下设计：$\mathcal{S}_i^t=\{n_i^t,\boldsymbol{R}_i^t,\mathcal{T}_i,\Omega_i^t\}$，而$\mathcal{D}_i^t=[\tilde{n}_i^t,\widetilde{\boldsymbol{R}}_i^t,\widetilde{\mathcal{T}}_i,\widetilde{\Omega}_i^t]$，其中：

- n_i^t（$\tilde{n}_i^t$）是截至 t 时刻，分入到第 i 个活跃子空间（非活跃子空间）的点的总个数；
- $\boldsymbol{R}_i^t$（$\widetilde{\boldsymbol{R}}_i^t$）称为活跃子空间（非活跃子空间）的保留矩阵，是截至当前 t 时刻，由第 i 个活跃子空间（非活跃子空间）的一些被选择保留下来的点组成的矩阵；
- $\mathcal{T}_i$（$\widetilde{\mathcal{T}}_i$）记录了截至 t 时刻所有分入第 i 个活跃子空间（非活跃子空间）的点的时间戳；
- Ω_i^t（$\widetilde{\Omega}_i^t$）记录了截至 t 时刻所有分入第 i 个活跃子空间（非活跃子空间）的点的 ASCI 指标。

2. DI-ESC 的静态学习与动态聚类

DI-ESC 是两阶段算法，分为静态学习和动态聚类阶段。静态学习阶段的点称为支撑点，而动态聚类阶段的点称为流入点。

1）静态学习阶段

DI-ESC 算法需要一个必要的静态学习阶段来初步对数据流进行分析，同时完成 DI-ESC 模型的初始化，数据流最初到达的点作为支撑点供 DI-ESC 算法进行静态学习，假设最初到达的 T_0 个点是支撑点。

这些支撑点组成了支撑矩阵 $\boldsymbol{X}_{\text{sup}}=[\boldsymbol{x}_1,\cdots,\boldsymbol{x}_{T_0}]$。由于数据流具有非均衡性

质，$\boldsymbol{X}_{\text{sup}}$ 极可能是非均衡的，因此，选用 I-ESC 算法对支撑矩阵进行子空间聚类。由算法 5.2 可知，I-ESC 算法为 $\boldsymbol{X}_{\text{sup}}$ 确定代表点子集 $\mathcal{X}_0$。将 $\mathcal{X}_0$ 中的代表点存入到各个子空间概要中的保留矩阵中，这些点是该空间中最具有代表性的点，将用于帮助后续流入点寻找合适的子空间。

I-ESC 算法通过 $\mathcal{X}_0$ 对支撑矩阵进行子空间聚类，假设共挖掘出 k^{T_0} 个子空间，则可依据聚类结果对各个子空间概要中的变量 n_i 进行初始化。因此，在 T_0 时刻，DI-ESC 通过静态学习阶段完成了对 DI-ESC 概要的初始化，各个子空间概要初始化为

$$\mathcal{S}_i^{T_0}=\{n_i^{T_0},\boldsymbol{R}_i^{T_0},\varnothing,\varnothing\}$$

对于非活跃子空间而言，其子空间概要为 $\mathbb{D}^{T_0}=\varnothing$。

2）动态聚类阶段

在静态学习阶段，DI-ESC 算法对数据流进行了初步学习，并将初步学习结果对 DI-ESC 概要进行初始化。之后，对数据流的处理进入动态聚类阶段。动态处理阶段是对每一个流出点 $\boldsymbol{x}^t(t\geqslant T_0)$ 逐个处理，即在线处理。

流入点也许来自已经发现的子空间，也许来自未发现的子空间，甚至也有可能是噪声点。对于来自已经发现的子空间的点，我们称其为在群点(Inlier)；对于来自未发现的子空间或者噪声点，称其为离群点(Outlier)。对于在群点来说，其可能来自活跃子空间也可能来自非活跃子空间。因此，DI-ESC 首先判断流入点是否为在群点。在这里定义表示矩阵 $\boldsymbol{Z}^t=[\boldsymbol{R}_1^t\cdots\boldsymbol{R}_{k^t}^t\widetilde{\boldsymbol{R}}_1^t\cdots\widetilde{\boldsymbol{R}}_{h^t}^t]$，表示矩阵 $\boldsymbol{Z}^t$ 由各个活跃与非活跃子空间的保留矩阵组成，每一个保留矩阵称为 $\boldsymbol{Z}^t$ 的一个子块。对于流入点 $\boldsymbol{x}^t(t>T_0)$ 而言，求取其在矩阵 $\boldsymbol{Z}^t$ 下的稀疏表示系数，即

$$\min\|\boldsymbol{c}^t\|_0\quad\text{s.t. }\boldsymbol{x}^t=\boldsymbol{Z}^t\boldsymbol{c}^t \tag{5.17}$$

实际中，可以用 $\|\cdot\|_1$ 范数对式(5.17)进行松弛，即

$$\min\|\boldsymbol{c}^t\|_1\quad\text{s.t. }\boldsymbol{x}^t=\boldsymbol{Z}^t\boldsymbol{c}^t \tag{5.18}$$

一般，上式可以转化为下面优化问题：

$$\min_{\boldsymbol{c}_i\in\mathbb{R}^N}\|\boldsymbol{c}_i\|_1+\frac{\lambda}{2}\left\|\boldsymbol{y}_i-\sum_{i\neq j}c_{ij}\boldsymbol{y}_j\right\|_2^2 \tag{5.19}$$

其中，$\lambda>0$ 是一个输入参数；$\|\cdot\|_1$ 和 $\|\cdot\|_2$ 分别表示 ℓ_1 与 ℓ_2 范数。通过式(5.19)求得最优解 $(\boldsymbol{c}^t)^*$，称其为点 $\boldsymbol{x}^t$ 的稀疏表示向量，简称表示向量，为了简化表示，在后文中，将其表示为 $\boldsymbol{c}^t$。

对于在群点来说，由于数据点的子空间自保持特性，其表示向量将具有块稀疏的特性，即表示向量的非零系数集中在 $\boldsymbol{c}^t$ 的某一个子块，该子块对应于该点所对应子空间的保留矩阵 $\boldsymbol{R}$ 在表示矩阵中的 $\boldsymbol{Z}$ 的位置。而对于离群点，由于该点不属于已发现子空间的点，因此其表示向量不具备块稀疏的性质，其非零向量分散在各个子块。为了衡量表示向量非零系数的集中程度，计算 $\boldsymbol{c}^t$ 的 ASCI 指标。ASCI 指标（定义 4.1）是在第 4 章中提出的重要概念，是目前广泛使用的 SCI 指标的推广。

现将 $\boldsymbol{c}^t$ 代入式(5.15)可得

$$\mathrm{ASCI}(\boldsymbol{c}^t)=\frac{(k^t+h^t)\cdot\max\limits_{j^*}\left(\dfrac{\|\delta_j(\boldsymbol{c}^t)\|_1/\zeta_j}{\sum\limits_{i=1}^{k^t+h^t}\|\delta_i(\boldsymbol{c}^t)\|_1/\zeta_i}\right)-1}{(k^t+h^t)-1} \tag{5.20}$$

根据定义 4.1，$\mathrm{ASCI}(\boldsymbol{c}^t)\in[0,1]$ 且 ASCI 值越高表示 $\boldsymbol{c}$ 的非零系数越集中在某一子块，那么说明流入点越有可能来自该子块对应的子空间。为了简化表示，下文将 $\mathrm{ASCI}(\boldsymbol{c}^t)$ 表示为 $\omega(\boldsymbol{c}^t)$。DI-ESC 引入门限 τ 对在群点与离群点进行判断。DI-ESC 认为流入点 $\boldsymbol{x}^t$ 为在群点，若其表示向量 $\boldsymbol{c}^t$ 满足

$$\omega(\boldsymbol{c}^t)\geqslant\tau \tag{5.21}$$

否则，DI-ESC 认为 $\boldsymbol{x}^t$ 为离群点。

对于在群点，其可能是来自活跃子空间或非活跃子空间，因此进一步计算将 $\boldsymbol{x}^t$ 分配到各个子空间所带来的残差，选择最小的残差空间作为 $\boldsymbol{x}^t$ 的子空间，即解决下列优化问题：

$$\min_{j^*} r_j(\boldsymbol{x}^t)\triangleq\|\boldsymbol{x}^t-\boldsymbol{Z}^t\delta_j(\boldsymbol{c}^t)\|_2 \tag{5.22}$$

其中，$r_j(\cdot)$是将 $\boldsymbol{x}^t$ 分到第 j 个子空间所带来的残差；$\delta_j(\cdot)$：$\mathbb{R}^n\to\mathbb{R}^n$ 是将 $\boldsymbol{c}^{*t}$ 的第 j 个子块（$j\in[1,l^t+h^t]$）对应的系数保留并将其余子块系数置 0 的函数。该优化问题意味着将 $\boldsymbol{x}^t$ 分配到残差最小的子空间内。式(5.22)的最优解 j^* 对应着矩阵 $\boldsymbol{Z}^t$ 的第 j^* 个子部分。

若 $j^*\leqslant k^t$，则说明 $\boldsymbol{x}^t$ 属于活跃子空间，相应的子空间概要 $\mathcal{S}_{j^*}$ 需要进行更新，更新规则如下：

$$\begin{cases}n_{j^*}^{t+1}=n_{j^*}^t+1\\ \mathcal{T}_{j^*}^{t+1}=\mathcal{T}_{j^*}^t\cup t\\ \Omega_{j^*}^{t+1}=\Omega_{j^*}^t\cup\omega(\boldsymbol{x}^t)\end{cases} \tag{5.23}$$

类似地，若 $j^* > k^t$，则说明 $\boldsymbol{x}^t$ 属于非活跃子空间，相应的第 $j^* - k^t$ 个非活跃子空间的子空间概要需要更新 $\mathcal{D}_{j^*-k^t}$，其更新规则为

$$\begin{cases} \tilde{n}^{t+1}_{j^*-k^t} = \tilde{n}^{t}_{j^*-k^t} + 1 \\ \widetilde{\mathcal{T}}^{t+1}_{j^*-k^t} = \widetilde{\mathcal{T}}^{t}_{j^*-k^t} \cup t \\ \Omega^{t+1}_{j^*-k^t} = \widetilde{\Omega}^{t}_{j^*-k^t} \cup \omega(\boldsymbol{x}^t) \end{cases} \tag{5.24}$$

离群点并不意味着是毫无价值的点，因为未被发现的子空间的点会被当作离群点。因此，为了能够不断挖掘新的子空间，DI-ESC 不直接将离群点删去，而是将离群点暂时存储在离群点存储池 $\mathcal{O}^t$ 中，具体地：$\mathbb{O}^t = \{\mathcal{O}^i\}_{i=1}^{n_o^t}$ 且 $\mathcal{O}^t = \{\boldsymbol{x}^i, \omega^i, t_i\}$，其中 n_o^t 是 t 时刻离群点的个数。

3. 子空间演化的在线检测

演化是数据流最基本的特征，DI-ESC 算法重点关注急速演化（Abrupt Evolution），具体考虑三种演化形式，即子空间出现、子空间消失与子空间复现。在第 4 章中，已经充分论证了基于 PH 检测与衰减函数的子空间演化检测的有效性，类似地，DI-ESC 算法依然采取类似结构。

1）基于 PH 检测的子空间出现与子空间复现检测

基于第 4 章分析可知，PH 检测可以有效检测子空间出现与复现，但需要对 PH 检测中的参数 p 进行合理设置。子空间出现即在短时间内有大量的离群点被放置在离群点存储池中，因此，定义变量 p^t：

$$p^t = \sqrt{\frac{1}{n_o^t}\sum_{i=1}^{n_o^t}(1+\log(t_i - t_{i-1}))\left(\omega_{t_i} - \frac{1}{n_o^t}\sum_{k=1}^{n_o^t}\omega_{t_k}\right)^2} \tag{5.25}$$

其中，n_o^t 是离群点存储池中的离群点个数。当短时间内存在大量离群点涌入离群点存储池时，p^t 会出现一个下降的趋势，并且这种变化能够被 PH 检测所捕捉。

当检测到子空间出现时，DI-ESC 算法对离群点存储器进行挖掘处理。将离群点按列组成矩阵 $\boldsymbol{O} = [\boldsymbol{x}_1 \boldsymbol{x}_2 \cdots \boldsymbol{x}_{n_o^t}]$，利用 I-ESC 算法（算法 5.3）对 $\boldsymbol{O}$ 进行处理得到聚类结果，将结果表示为 $\mathbb{S}_*$，则需要将 $\mathbb{S}_*$ 更新到 DI-ESC 概要中，即

$$\mathbb{S}^t \leftarrow \mathbb{S}^t \cup \mathbb{S}_* \tag{5.26}$$

子空间复现是指非活跃子空间再次转为活跃状态，这种现象在演化数据流中是经常发生的，因为随着时间的不断流逝，始终保持活跃的模式是极少数的。对于

每一个非活跃子空间，定义变量 $\dot{p}_m^t$，即

$$\dot{p}_m^t = \sqrt{\frac{1}{\tilde{n}_m^t}\sum_{i=1}^{\tilde{n}_m^t}(1+\log(t_i - t_{i-1}))} \tag{5.27}$$

当检测到有子空间复现时，则需将该子空间概要中的 $\widetilde{\mathcal{T}}_m^t$ 以及 $\widetilde{\Omega}_m^t$ 清空，然后将该非活跃子空间更新到 DI-ESC 概要中，即

$$\begin{cases} \mathbb{S}^t \leftarrow \mathbb{S}^t \cup \{\mathcal{D}_m^t\} \\ \mathbb{D}^t \leftarrow \mathbb{D}^t \setminus \{\mathcal{D}_m^t\} \end{cases} \tag{5.28}$$

2）基于衰减函数的子空间消失

子空间消失意味着子空间已经在相当一段时间内没有被流入点访问，即由活跃态转化为非活跃态。DI-ESC 对所有活跃子空间的被访问时间进行监测，并计算子空间最近一次被访问的时间与当前时间的时间间隔，将这段时间间隔称为静默间隔，DI-ESC 为每个活跃子空间定义了变量 $\ddot{p}_l^t$，即

$$\ddot{p}_l^t = 1 - \frac{1}{1+\mathrm{e}^{-(t-\max\{\mathcal{T}_l^t\}-\beta)}} \tag{5.29}$$

其中，$\max\{\mathcal{T}_l^t\}$ 是 l 子空间最后一次被流入点访问的时间。参数 β 被引入用来控制 DI-ESC 算法对活跃子空间的静默间隔的容忍度。β 参数越大，表示 DI-ESC 对活跃子空间的静默间隔容忍度越大。

DI-ESC 对所有活跃子空间的 $\ddot{p}_l^t$ 进行监测，并以 0.5 为门限，当 $\ddot{p}_l^t \geqslant 0.5$ 时，则认定子空间依然保持活跃状态；反之，则判定该子空间已由活跃态转为非活跃态。当判定 l 子空间为非活跃态时，清空其相应的 $\mathcal{T}_m^t$ 以及 Ω_m^t，将其从 DI-ESC 中移出，暂存在非活跃子空间存储池中，即

$$\begin{cases} \mathbb{D}^t \leftarrow \mathbb{D}^t \cup \{\mathcal{D}_l^t\} \\ \mathbb{S}^t \leftarrow \mathbb{S}^t \setminus \{\mathcal{S}_l^t\} \end{cases} \tag{5.30}$$

3）DI-ESC 算法流程

本节对 DI-ESC 算法做总结。算法 5.4 为算法的伪代码，同时，将算法的框架展示在图 5.3 中。

DI-ESC 算法主要分为三个主要步骤，首先对支撑点进行静态学习，并利用 I-ESC 算法对 DI-ESC 算法进行初始化。

然后，DI-ESC 进入动态聚类阶段，开始逐一处理流入点。对于每一个流入点，

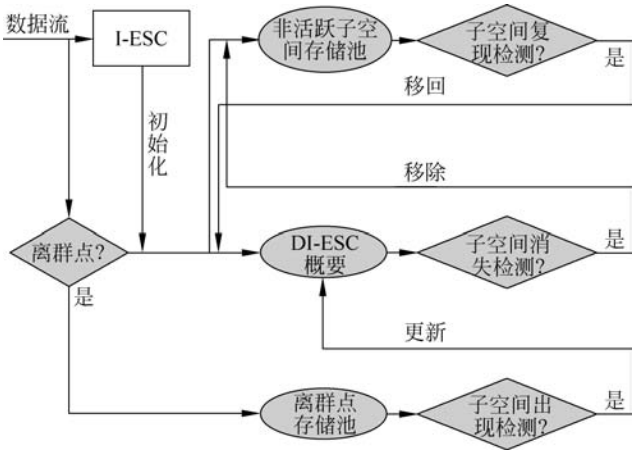

图 5.3 DI-ESC 算法框架

通过式(5.19)和式(5.20)计算其 ASCI 值。通过式(5.21)判定流入点是否为在群点。若为在群点,则计算其在各个子空间表示下的残差,通过式(5.22)将其分入相应子空间,同时更新相应子空间概要$\mathbb{S}$或者$\mathbb{D}$。若判定为离群点,则将该点更新到离群点存储池$\mathbb{O}$中。

接着,DI-ESC 对数据流进行演化检测,分别采用基于 PH 检测的子空间出现[式(5.25)]和子空间复现[式(5.27)]以及基于衰减函数的子空间消失检测[式(5.29)],并对$\mathbb{S}$、$\mathbb{D}$以及$\mathbb{O}$作相应的更新。

算法 5.4 DI-ESC 算法

输入:数据流 $\boldsymbol{x}^1,\cdots,\boldsymbol{x}^t,\cdots$;支撑点数目 T_0;门限 β,τ,f;$\mathbb{S}\leftarrow\varnothing,\mathbb{D}\leftarrow\varnothing,\mathbb{O}\leftarrow\varnothing$;

步骤一:通过算法 5.3 对支撑点进行处理,使得 DI-ESC 概要初始化,即得到$\mathbb{S}^{T_0}$。

步骤二:对每一个流入点 $\boldsymbol{x}^t(t>T_0)$通过式(5.19)和式(5.20)计算其 ASCI 值,即得到 $\omega(\boldsymbol{c}^{*t})$。

步骤三:通过式(5.21)对流入点 $\boldsymbol{x}^t$ 判定是否为离群点。若判定为在群点,则通过计算式(5.22)将 $\boldsymbol{x}^t$ 分入相应子空间,同时更新相应子空间概要$\mathbb{S}$或者$\mathbb{D}$。若判定为离群点,则更新$\mathbb{O}$。

步骤四:进行子空间演化检测:计算 $\dot{p}_m^t,p^t$ 并通过 PH 检测分别对子空间出现、子空间复现进行检测。若被触发,则相应地更新$\mathbb{S}$、$\mathbb{D}$和$\mathbb{O}$。计算 $\ddot{p}_l^t$,并对每一个活跃子空间进行类消失检测,若 $\ddot{p}_l^t\geqslant 0.5$ 则判定该子空间转为非活跃子空间,则相应更新$\mathbb{S}$和$\mathbb{D}$;

输出:数据流在线聚类结果。

5.5　仿真实验与分析

本节利用实测数据对 I-ESC 算法以及 DI-ESC 算法分别在处理非均衡静态数据以及非均衡演化数据流的性能进行验证。同时，依然选择在第 4 章中采用的数据流聚类领域内最先进的算法作为 DI-ESC 算法的对比算法，具体包括 SSSC[205]、SLRR[205]、SLSR[205]以及数据流聚类领域极具代表性的算法 CEDAS[239]和 STRAP[193]。本节分为四部分，5.5.1 节对实验采用的数据集以及各个算法的主要参数设置进行简要介绍与说明。在 5.5.2 节中选用 ESC 算法作为对比算法，对 I-ESC 算法性能进行验证与分析。5.5.3 节主要对 DI-ESC 算法的性能进行仿真验证与分析。5.5.4 节对 DI-ESC 算法进行参数敏感度的分析与讨论。

5.5.1　数据集及实验设置

1. 数据集简介

本节继续采用在第 4 章提到的雷达辐射源数据作为测试数据。本实验用到的静态数据和数据流均基于该实测数据生成得到。在验证 I-ESC 算法性能时，由于 I-ESC 算法只能处理静态数据集，因此将前 4 个辐射源(R1～R4)视为过表达辐射源，而将其余的辐射源(R5～R8)视为欠表达辐射源。在不同的不平衡度，即 γ，生成了具有不同不均衡度的静态数据集，用于验证和比较 I-ESC 算法对不同非均衡数据流的处理性能。

除了上述的静态数据集，本实验还基于实测数据生成了 4 个不同的非均衡演化数据流，分别表示为 DS4～DS7，用来验证 DI-ESC 算法以及对比算法对不同的非均衡演化数据流的处理性能。DS4～DS7 数据流的主要信息总结在表 5.1 中。如表所示，DS4～DS7 主要的演化性质为出现，其中 DS4 与 DS5 数据流在开始阶段包括 4 个子空间，随后又有 4 个子空间出现。而 DS6 数据流在开始阶段包括 2 个子空间，随后共有 6 个子空间出现。DS7 数据流是最复杂的数据流，具有子空间出现、消失与复现三种演化形式。DS4～DS7 数据流更具体的演化性质介绍将在具体实验中阐述。

表 5.1 非均衡演化数据流(DS4～DS7)的主要信息

数 据 流	演 化 性 质	初始子空间个数	子空间总数
DS4	子空间出现	4	8
DS5	子空间出现	4	8
DS6	子空间出现	2	8
DS7	子空间出现/消失/复现	8	8

2. 对比算法与评价指标

在 I-ESC 算法性能验证实验中,采用 ESC 算法[220]作为对比算法,目的是验证 I-ESC 算法处理非均衡数据集的有效性。而在 DI-ESC 的性能验证中,采用五种最先进的数据流聚类算法 CEDAS[239],STRAP[193],SSSC[205],SLSR[205]和 SLRR[205]。除此之外,在 ESC 算法基础上提出了 D-ESC 算法作为 DI-ESC 算法的对比算法。D-ESC 与 D-IESC 有基本相同的框架,唯一不同之处是在静态学习阶段的学习算法不同,分别采用 ESC 算法和 I-ESC 算法。通过 DI-ESC 与 D-ESC 算法对比,可以进一步验证 I-ESC 算法的优越性。CEDAS 和 STRAP 分别是典型的基于密度和基于距离的数据流聚类算法。SSSC、SLSR 和 SLRR 算法是三种最先进的基于表示的子空间聚类算法,在处理高维数据流具有良好的效果。

本实验采用聚类质量(Cluserting Quality)来对各个算法的性能进行评估。聚类质量包括正确率(Accuracy)和 NMI,是通过实际聚类结果与真实标签计算得到的,具体计算公式在 4.4.1 节中进行了详细介绍。NMI 和 Accuracy 取值均在 0～1,数值越大表示聚类结果越贴近真实标签,从而说明聚类效果越好。本实验的 NMI 和 Accuracy 值是对每一个实验独立重复 50 次,然后取 NMI 和 Accuracy 的平均值得到的。整个实验是在装配 2.3GHz 的 CPU 与 4G 主存的个人计算机的 MATLABR2016b 软件上执行的。

5.5.2 I-ESC 算法性能验证与分析

本节主要对 I-ESC 在处理非均衡数据集的性能进行验证与分析。将 I-ESC 与 ESC 算法同时作用不同非均衡度的数据流($\gamma=4,5,\cdots,10$),结果如图 5.4 所示。

由图 5.4 分析可得,I-ESC 算法可以有效处理非均衡数据集,且其性能要超过 ESC 算法。同时,I-ESC 算法的性能十分稳定,50 次重复实验结果的方法几乎为 0。而 ESC 算法的性能波动相对比较大。这主要是由于 ESC 算法随机初始化的原因,导致其性能对随机初始化很敏感,使得 ESC 算法很难具有较好的聚类效果。

而 I-ESC 算法对 ESC 算法进行了改进，通过式(5.7)避免了随机初始化带来的影响。

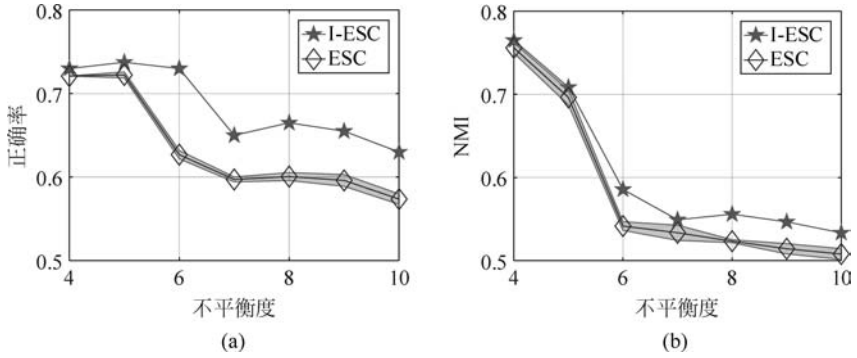

图 5.4　I-ESC 与 ESC 算法对不同非均衡度($\gamma=4,5,\cdots,10$)的数据集处理结果的聚类质量

数据集的非均衡度对 I-ESC 以及 ESC 算法具有较大的影响。如图 5.4 所示，随着不平衡度的持续增加，I-ESC 算法和 ESC 算法的性能都会随之降低。例如，当 $\gamma=4$ 时，I-ESC 算法结果的 Accuracy 值为 0.7309，NMI 值为 0.7684；ESC 算法的 Accuracy 值为 0.7210，NMI 值为 0.7550。然而，当 $\gamma=10$，I-ESC 算法结果的 Accuracy 的值降低为 0.6300，NMI 值为 0.5336；而对于 ESC 算法而言，其结果的 Accuracy 为 0.5736，NMI 值为 0.5081。

除此之外，为了研究由参数 η 控制的代表点子集大小对 I-ESC 算法的影响，下面进一步研究不同 η 下，I-ESC 与 ESC 算法对数据集的处理性能，其中数据集的非均衡度为 4，两种算法处理结果的聚类质量如图 5.5 所示。由图 5.5 可知，I-ESC

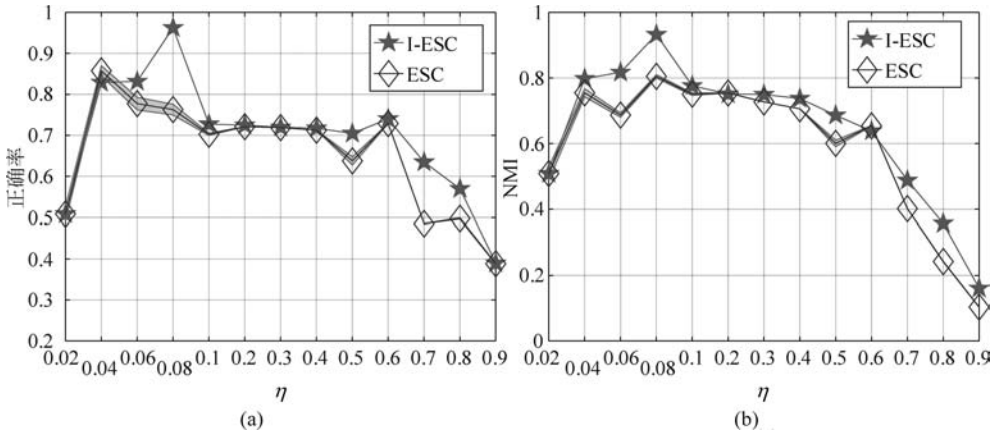

图 5.5　不同参数 η 下，I-ESC 与 ESC 算法对非均衡数据集处理结果的聚类质量($\gamma=4$)

和 ESC 算法的性能随着 η 变化趋势与理论分析一致，随着 η 增加，两种算法的聚类质量整体呈现下降趋势。这主要是由于随着 η 增加，代表点子集中的代表点的个数不断增加，导致代表点子集本身的不平衡度持续增加，特别当 η 足够大时，代表点子集的规模将与原集合相当，具有很高的非平衡度，因此性能会下降。然而，这并不意味着 η 应当设置得足够小，因为 η 过小，导致被选择的代表点数目极少，这样数量不充足的代表点对整个子空间的刻画是不足够的，反而会导致性能的下降。

5.5.3 DI-ESC 算法性能验证与分析

本节将对 DI-ESC 算法的性能进行验证与分析，同时通过与对比算法的对比，验证 DI-ESC 的优越性。在第 1 小节，重点验证 DI-ESC 对基本的演化数据流，即只具有子空间出现演化形式的处理性能；而在第 2 小节中，将验证 DI-ESC 算法对更复杂的演化数据流的处理性能。

1. 对基本的演化数据流的子空间聚类

本节将 DI-ESC 与其余对比算法作用于 DS4～DS6 数据流，DS4～DS6 数据流的演化特性如图 5.6 所示。在图 5.6 中，X 轴是数据流的时间戳，Y 轴代表不同的子空间，按编号 1～8 进行区分。可以看到，按照演化性质，DS4 分为 2 个阶段，分别用 P1、P2 表示。P1 阶段为 DS4 初始阶段，共有 4 个子空间存在，在 P2 阶段，另有 4 个子空间出现。DS2 数据流按照演化性质，可明显分为 3 个阶段（分别对应 P1～P3），P1 阶段有 4 个子空间存在，在 P2 与 P3 阶段分别有 2 个新的子空间出现。DS6 数据流按照演化性质可分为 4 个阶段（分别对应 P1～P4），P1 阶段仅有 2 个子空间存在，随后在 P2 至 P4 阶段，每个阶段分别有 2 个新的子空间出现。

现将 DI-ESC、D-ESC 以及其余的对比算法对 DS4～DS6 数据流进行处理，经重复实验后，将聚类结果质量的平均值统一记录在表 5.2 中。

由表 5.2 分析可知，DI-ESC 和 D-ESC 算法作为两个可以处理非均衡演化数据流的算法，两种算法的聚类质量明显优于其余对比算法。比如，对于 DS4 数据流，其余对比算法中，SLSR 算法的性能达到最优效果，其 Accuracy 值达到 0.5903，NMI 值达到 0.4386。而 DI-ESC 和 D-ESC 算法的 Accuracy 值分别达到 0.7714 和 0.6760，而 NMI 值分别达到 0.8362 和 0.7593。SSSC、SLRR、SLSR 算法尽管可以处理数据流，但是这三种算法只能处理子空间结构不变的数据流，即非演化数据流。而 CEDAS 与 STRAP 算法，二者本质上是在传统欧氏空间的基于距离度量下的聚类，这种聚类方式很难有效刻画高维子空间的分布情况。

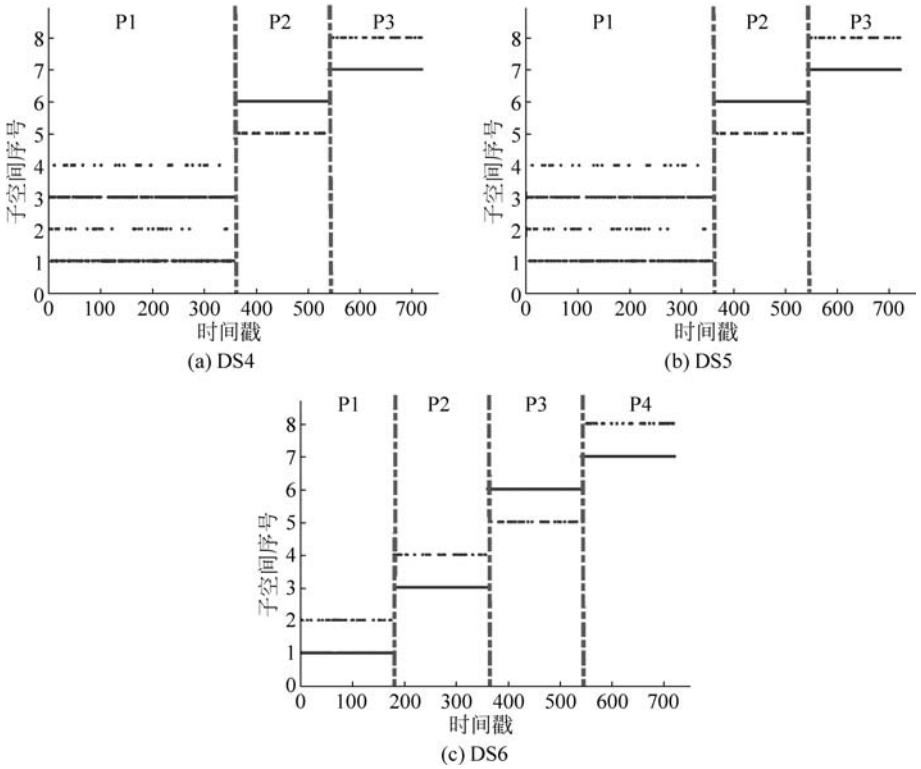

图 5.6　DS4～DS6 数据流的演化性质

表 5.2　不同算法对三种基本演化数据流(DS4～DS6)的处理结果的性能对比

数据流	DS4			DS5			DS6		
算 法	Acc. /%	NMI /%	时间 /s	Acc. /%	NMI /%	时间 /s	Acc. /%	NMI /%	时间 /s
DI-ESC	**77.14**	**83.62**	1.36	**80.49**	**81.46**	1.48	**76.88**	**78.98**	1.72
DIESC	67.60	75.93	1.24	74.57	80.30	1.18	75.73	77.85	1.54
SSSC	58.47	43.48	11.03	59.02	43.48	10.01	25.00	16.48	7.24
SLRR	59.02	43.48	13.07	59.02	43.48	10.78	25.00	16.48	3.07
SLSR	59.03	43.86	**0.52**	59.03	43.86	**0.62**	25.00	16.48	**0.53**
CEDAS	26.81	41.08	1.98	26.39	41.49	2.01	27.08	42.82	2.07
STRAP	26.25	36.17	1.11	26.94	36.43	1.16	28.61	34.59	0.78

随着数据流演化程度的加剧，子空间出现的次数增加，算法对数据流处理结果的聚类质量整体是降低的，由此可见数据流的演化实际上影响了算法的处理性能，

但 DI-ESC 算法受波动程度比较小，因为其本身具有演化检测的机制。而 SSSC、SLRR、SLSR 影响波动比较大，因为它们本身不具备演化检测机制，很难适应高强度的演化。CEDAS 和 STRAP 算法理论上也可以处理演化数据流，但由于传统距离度量的局限性，导致其聚类质量不高。

除了 SLSR 算法，其余对比算法与 DI-ESC 和 D-ESC 相比，需要接近（如 CEDAS 和 STRAP 算法）以及更长的处理时间（如 SSSC、SLRR）。一般地，SSSC 需要最长的处理时间，因为 SSSC 的初始化是基于 SSC 算法[203]完成的，该算法的计算复杂度很高，导致整体算法的效率很慢。而 DI-ESC 与 D-ESC 的初始化主要分别依托 I-ESC 和 ESC 完成，这两种算法通过在代表点子集中进行优化，而不是在原始集合中，从而节约了计算时间。

需要指出的是，DI-ESC 算法的性能明显优于 D-ESC 算法，这是由于 I-ESC 算法对支撑点的处理结果要比 ESC 算法对支撑点的处理结果更准确且稳定。D-IESC 算法比 D-ESC 算法消耗时间略微多一些的主要原因是 I-ESC 算法对第一个代表点的选择不是随机化，而是需要通过式(5.7)来确定，从而引入了额外的计算时间。

2. 对复杂的演化数据流的子空间聚类

本节进一步探讨分析 DI-ESC 算法对更复杂的非均衡演化数据流的处理性能。实验采用 DS7 数据流，DS7 数据流同时具有子空间出现、子空间消失以及子空间复现三种演化形式，按照演化特性，DS7 数据流主要分为三个阶段（分别用 P1、P2 以及 P3 对应表示），如图 5.7 所示。在 P1 阶段，共有 8 个子空间出现，可以观察 8 个子空间的点的分布是不均衡的。从 $t=500$ 开始，有 4 个子空间陆续地开始消失，随后在 P3 阶段，之前消失的子空间陆续复现，直到最后有 8 个子空间存在。

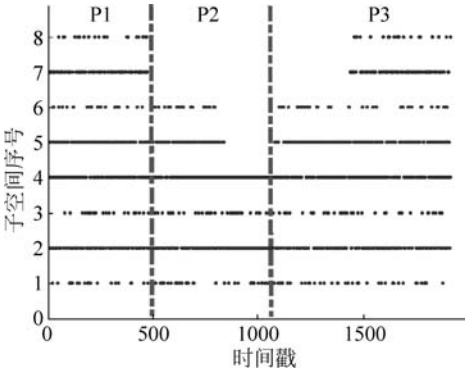

图 5.7 DS7 数据流的演化性质

现将 DI-ESC 算法及对比算法作用于 DS7 数据流，来验证 DI-ESC 算法检测处理演化数据流的有效性和优越性。

在每一个算法独立重复实验之后，将各个算法检测到的子空间数目变化绘制在图 5.8 中。如图 5.8(a)所示，DI-ESC 成功地检测到 DS7 数据流潜在的子空间结构，最开始有 8 个子空间被成功检测出，随后 DI-ESC 成功检测出子空间的消失，直到在 $t=1000$ 附近检测出子空间仅剩 4 个，随后检测出子空间的复现，在数据流末端，再次检测到 8 个子空间。图 5.8(b)是 SSSC、SLRR 与 SLSR 三种算法对数据流变化的检测图，可以观察到，在最开始三者成功检测到了 8 个子空间，但随后，三种算法检测下的子空间结构不再发生变化，这是因为三种算法由于缺少对数据流变化的检测适应机制，从而导致仅能处理非演化数据流。而图 5.8(c)是 CEDAS

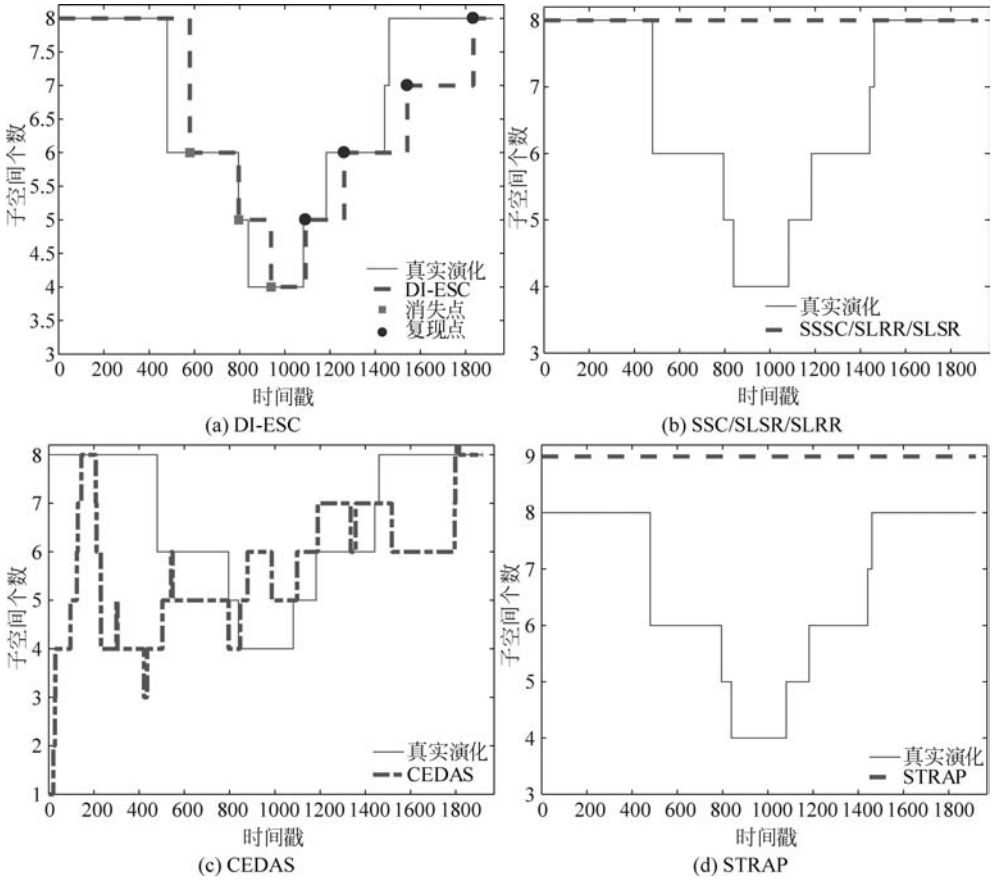

图 5.8　不同算法对复杂演化数据流(DS7)的子空间数目实时结果

算法对数据流变化的检测图，尽管理论上 CEDAS 算法能够检测类的出现与消失，然而，由于其基于传统距离度量，难以刻画子空间的结构，导致其对子空间的聚类效果不理想。类似地，STRAP 算法[图 5.8(d)]也是基于传统距离度量的聚类算法，其在初始阶段的聚类效果就不理想，所发现的类远远高于实际子空间的数量，随后，由于 STRAP 缺乏有效的子空间演化检测与适应机制，导致其挖掘的类数始终保持不变。

5.5.4 DI-ESC 算法参数敏感度分析

对于 DI-ESC 算法而言，共有三个主要参数，包括 η、τ 和 f，对 DI-ESC 性能产生较大的影响。因此，本节重点通过仿真实验来分析讨论 DI-ESC 算法对主要参数的敏感度。

参数 η 是代表点个数占原点集的比例，即参数 η 控制着 I-ESC 算法选出的代表点子集的大小。由 5.5.2 节分析已知，当参数 η 越来越大，实际上会造成代表点子集与原子集在数据点的个数上逐渐接近，但由于原子集是非均衡数据集，会导致代表点子集的非均衡度不断增加，而 I-ESC 对数据集的聚类是基于代表点子集完成的，因此会使得聚类结果质量下降。而当 η 越来越小时，表示选择的代表点越来越少，但过于少的代表点会导致不足以刻画子空间的特征，反而会造成聚类结果质量的下降。

参数 η 主要是通过影响 I-ESC 算法而间接对 DI-ESC 算法产生影响，在 5.5.2 节已经给出了更完整的实验与分析。本节的实验将重点分析 τ 与 f 参数对 DI-ESC 算法的影响。

首先，研究 τ 参数对 DI-ESC 算法的影响。将不同 τ 参数设置下的 DI-ESC 算法作用于 DS7 数据流。结果如图 5.9 所示，可以得出以下结论。

如图 5.9 所示，参数 τ 从 0.5 变化到 0.95。当 τ 相对较小时，DI-ESC 算法处理结果的聚类质量比较差，这是因为 τ 参数实际控制了离群点与在群点判断的门限，当 τ 很小时，有大量的离群点会被误判为在群点而被误分进 DI-ESC 概要中，这不仅会影响聚类质量，同时由于大量的离群点被当作在群点，造成 DI-ESC 对数据流的演化，特别是对子空间出现极其不敏感。然而这并不意味着 τ 应该被设置得很大，因为随着 τ 不断增大，可以观察到聚类质量有所下降，例如，图 5.9 中，当 $\tau>0.8$ 之后，聚类结果的 Accuracy 与 NMI 值均有所下降。这是因为较大的 τ 值导致很多在群点被判断为离群点，这势必会导致聚类质量的直接下降，同时在群点被误

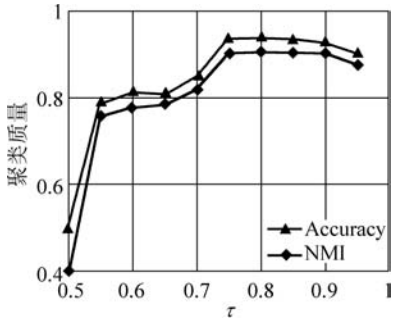

图 5.9　DI-ESC 算法对τ 的敏感度

认为离群点会影响后续聚类。

参数 f 控制着 DI-ESC 算法对数据流演化的敏感度。图 5.10 为不同 f 参数下 DI-ESC 算法对 DS4 处理结果的聚类质量。如图 5.10 所示，当 f 参数很小时，PH 的检测极容易被触发导致 DI-ESC 算法对演化极其敏感，从而使得聚类质量不是很高，同时降低 DI-ESC 算法的稳定性。而随着 f 参数不断增大，DI-ESC 对数据流的演化变得极其不敏感，造成新的子空间无法被挖掘，大量的离群点堆积在离群点存储池中，造成了聚类质量的下降。

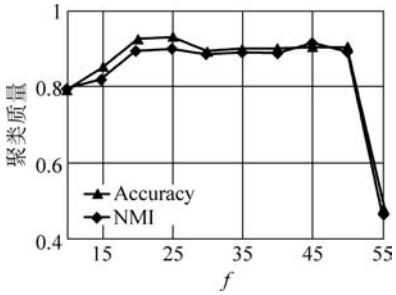

图 5.10　DI-ESC 算法对 f 的敏感度

本章小结

在现实世界的雷达辐射源信号分选任务中，往往极难预先获得非合作源的信号样本用来训练，同时由于各个辐射源的工作任务不同，往往接收到的信号样本在各个辐射源的数量分布是极其不均衡的，这些实际问题给雷达辐射源信号分选带来很大的挑战。

为了解决上述实际问题，本章将雷达辐射源在线分选问题转化为对具有类均衡且演化性质的数据流的在线子空间聚类问题。本章提出两种算法，即 I-ESC 算法与 DI-ESC 算法，分别处理类非均衡静态数据集和非均衡演化数据流。I-ESC 算法有效解决了 ESC 算法对初始化敏感的问题，提高了 ESC 算法的鲁棒性。DI-ESC 算法能够对非均衡演化数据流进行处理，成功突破了目前数据流算法普遍无法处理非均衡演化数据流的瓶颈。

本章主要内容包括：

① 本章首先对类不均衡下的雷达辐射源在线分选问题进行分析与建模，从理论上定义辐射源的演化特性与不平衡特性，提出不平衡度的概念，为后续问题建立理论模型基础。

② 本章首先从理论上分析 ESC 算法性能不稳定的原因，提出 I-ESC 算法，I-ESC 算法通过式(5.7)解决 ESC 算法的随机初始化问题，同时引入参数 η 实现选取代表点数目的自适应化。相比 ESC 算法，I-ESC 算法的性能更加稳定。

③ 本章提出 DI-ESC 算法，实现在类不均衡条件下的雷达辐射源在线分选。DI-ESC 算法通过在线更新维护 DI-ESC 概要实现对数据流模式的实时表达，同时对于子空间出现、子空间消失以及子空间复现三种最典型的子空间演化形式可实现精准检测与适应。

④ 本章利用实测雷达辐射源数据进行大量的仿真实验，验证对比 I-ESC 算法相比于 ESC 算法的优越性。同时也充分验证 DI-ESC 算法在类不均衡条件下对雷达辐射源在线分选的合理性。