적응적 격자기반 다차원 데이터 스트림 클러스터링 방법

박 남 훈† · 이 원 석††

요 약

데이터 스트림이란, 빠른 속도로 지속적으로 생성되는 무한한 크기의 방대한 양의 데이터 집합으로 정의된다. 무한한 데이터 스트림에 비해 주어진 메모리 공간은 유한하게 한정되어 있어, 이러한 제약조건을 충족시키는 범위 내에서 일정 한도내의 정확도 오차를 허용하기도 한다. 또한, 변화하는 데이터 스트림 내의 최신 클러스터를 찾기 위해서는 데이터 객체의 저장없이 오래된 데이터 스트림 내의 정보들을 비중을 감소시킬 수 있어야 한다. 본 연구에서는 데이터 스트림 분석을 위한 데이터 스트림 격자 기반 클러스터링 기법을 제시한다. 주어진 초기 격자셀에 대해, 데이터 객체의 빈도가 높은 범위를 반복적으로 보다 작은 크기의 격자셀로 분할하여 최소 크기의 격자셀, 단위 격자셀을 생성한다. 격자셀에서는 데이터 객체들의 분포에 대한 통계값만을 저장하여, 기존의 클러스터링 기법에 비해 데이터 객체에 대한 탐색없이 효율적으로 클러스터를 찾을 수 있다. 또한, 가용 메모리 공간에 따라 단위 격자셀의 크기를 조절하여 클러스터의 정확도를 최대화할 수 있어, 주어진 메모리 공간에 맞게 적응적으로 성능을 조절할 수 있다.

키워드 : 데이터 스트림, 데이터 마이닝, 클러스터링

An Adaptive Grid-based Clustering Algorithm over Multi-dimensional Data Streams

Nam Hun Park† · Won Suk Lee††

ABSTRACT

A data stream is a massive unbounded sequence of data elements continuously generated at a rapid rate. Due to this reason, memory usage for data stream analysis should be confined finitely although new data elements are continuously generated in a data stream. To satisfy this requirement, data stream processing sacrifices the correctness of its analysis result by allowing some errors. The old distribution statistics are diminished by a predefined decay rate as time goes by, so that the effect of the obsolete information on the current result of clustering can be eliminated without maintaining any data element physically. This paper proposes a grid based clustering algorithm for a data stream. Given a set of initial grid cells, the dense range of a grid cell is recursively partitioned into a smaller cell based on the distribution statistics of data elements by a top down manner until the smallest cell, called a unit cell, is identified. Since only the distribution statistics of data elements are maintained by dynamically partitioned grid cells, the clusters of a data stream can be effectively found without maintaining the data elements physically. Furthermore, the memory usage of the proposed algorithm is adjusted adaptively to the size of confined memory space by flexibly resizing the size of a unit cell. As a result, the confined memory space can be fully utilized to generate the result of clustering as accurately as possible. The proposed algorithm is analyzed by a series of experiments to identify its various characteristics

Key Words : Data Stream, Data Mining, Clustering

1. 서 론

최근, 웹 서핑 로그나 통신 로그, 실시간 동영상 데이터, 실시간 주식 거래 등과 같이 방대한 실시간 데이터를 처리하는 응용 환경이 증가함에 따라 이러한 데이터 스트림 분석에 대한 수요가 증가함에 따라 데이터 스트림 분석을 위한 데이터 마이닝 연구[1, 2, 3, 4, 5, 6, 7, 8]들이 진행되어 왔다. 데이터 스트림이란, 빠른 속도로 지속적으로 생성되는 무한한 크기의 방대한 양의 데이터 집합으로 정의된다. 따라서, 기존의 대부분 분석방법처럼 데이터 스트림을 저장하고 반복 분석하는 것은 불가능하기 때문에, 데이터 스트림 처리 기법들은 다음과 같은 제약조건들을 충족시켜야 한다. 첫째, 각 데이터 객체들은 최대 한번의 검색으로 분석 될 수 있어야 한다. 둘째, 무한한 데이터 스트림에 비해 이를 분석하는 메모리 공간은 한정되어 있다. 셋째, 최근 생성된 데이터 객체는 최대한 빠르게 분석되어, 앞으로의 분석작업에 그 결과를

※ 이 논문은 2007년도 정부(과학기술부)의 재원으로 한국과학재단의 국가지정연구실사업으로 수행된 연구임 (No.R0A-2006-000-10225-0)
† 준 회 원 : 연세대학교 대학원 컴퓨터과학과 박사과정
†† 종신회원 : 연세대학교 컴퓨터과학과 부교수
논문접수 : 2007년 2월 15일, 심사완료 : 2007년 7월 2일

반영할 수 있어야 한다. 이러한 제약조건 내에서 데이터 스트림 분석 작업은 일정 한도내의 오차를 허용하기도 한다.

클러스터링은 데이터 마이닝 분석 방법 중 하나로서, 주어진 유사도 척도에 따라 다수의 데이터 객체들을 유사한 집단으로 구분하는 데이터 마이닝 기법으로서 위성 이미지 분석, 고객 분석을 통한 타겟 마케팅 및 침입 탐지를 위한 대용량 로그 분석 등의 다양한 응용분야에서 효율적인 방법으로 사용되어 왔다. 기존의 대부분 클러스터링 기법[7, 8, 9, 10]들은 한정된 데이터 집합을 대상으로 분석 결과의 정확도를 유지하면서 수행시간과 메모리 사용량을 최소화하였다. 비록 클러스터링 기법들이 많은 응용분야에서 활용되고 있지만, 방대한 양의 데이터 집합을 대상으로 수행할 수 있는 클러스터링 기법은 많지 않다. 점진적으로 증가하는 데이터 집합을 대상으로 점진적 클러스터링 기법(Incremental Clustering Algorithm)[8, 11]들이 제시되었다. 이들 연구들은 이전 분석 결과들을 적용하여 새로 생성된 데이터 집합에 영향을 받는 최근 클러스터를 효율적으로 찾는 방법을 제시하였다. 하지만, 이전 데이터 집합을 모두 저장되어, 일부 데이터들을 반복 분석해야하기 때문에, 데이터 스트림 환경에서는 적합하지 못하다. [12]의 연구에서는 데이터 스트림을 대상으로 분할 클러스터링(Partitioning Clustering) 기반의 분석 방법을 제시하였다. 분할된 데이터 스트림의 각 부분 스트림들을 대상으로 O(1)-approximate K-medoid를 수행하였으며, 기존 K-medoid[7] 기법들이 데이터 집합을 반복해서 분석하는 것에 비해 각 부분 스트림들의 k개의 중심점을 지속적으로 유지한다.

본 연구에서는 데이터 스트림 분석을 위한 격자 기반 클러스터링 기법을 제시한다. 정밀하게 분할된 각 격자셀에 위치한 데이터 객체들의 통계적 분포정보만을 지속적으로 유지하여 데이터 객체들을 저장하지 않으면서 한번의 검색으로 클러스터를 찾을 수 있다. 먼저, 데이터 스트림의 다차원 데이터 공간을 중첩되지않는 동일한 크기의 초기 격자셀로 분할한다. 데이터 객체가 생성됨에 따라 각 초기 격자셀은 해당 범위 내의 데이터 객체들의 통계적 분포정보를 갱신한다. 초기 격자셀의 지지도가 높은 경우, 데이터 객체들의 통계적 분포정보를 기반으로 분할차원을 선택하여 이를 기준으로 두 개의 격자셀, 중간 격자셀을 생성한다. 중간 격자셀 내의 데이터 객체들의 통계적 분포 정보는 기존 격자셀의 분포 정보로부터 추정할 수 있다. 이와 같이, 데이터 객체의 빈도가 높은 범위의 중간 격자셀들은 반복적으로 보다 작은 크기의 중간 격자셀로 분할하게 되며, 최종적으로 최소 크기의 격자셀, 단위 격자셀이 생성된다.

격자셀을 분할하는 방법으로는 *μ-partition*[13], *σ-partition*[14], *hybrid-partition*의 세가지 분할 방법을 제시한다. μ-partition 방법은 해당 격자셀 내의 데이터 객체들의 평균값 μ를 기준으로 데이터 객체의 빈도가 균등한 두 개의 격자셀로 분할을 수행하는 반면, σ-partition 방법은 표준편차 σ를 기준으로 빈도가 높은 격자셀과 그 외의 격자셀로 분할한다. hybrid-partition방법은 앞의 두 분할 방법 중 해당 격자셀의 분포정보로부터 보다 효율적인 방법을 선택해서 수행하는 방법이다. 클러스터는 이러한 분할방법을 거쳐 생성된 연속된 범위의 빈도 높은 단위 격자셀의 집합으로 정의된다. 단위 격자셀의 크기가 작을수록 보다 정밀한 클러스터를 찾을 수 있는 반면에 클러스터를 구성하는 격자셀의 수는 많아진다. 따라서, 메모리 사용량을 최소화하기 위해서 빈도가 낮은 중간 격자셀 또는 단위 격자셀에 대해 전지과정(pruning)을 수행할 수 있다. 또한, 한정된 메모리 공간 내에서 수행할 수 있도록 단위 격자셀의 크기를 동적으로 조절하여 메모리 사용량을 조절할 수 있다.

데이터 스트림에 내재된 지식은 시간에 따라 변화하기 때문에 과거의 오래된 데이터 객체의 정보들은 더 이상 의미가 없다. 따라서, 변화하는 데이터 스트림 내의 최근 클러스터를 찾는 것이 데이터 스트림을 보다 가치있게 분석하는 것이라 할 수 있다. 이를 위해서 각 격자셀 내의 과거의 데이터 객체에 대한 통계정보를 주어진 감쇄율(decay rate)에 의해 감소시켜, 데이터 객체의 저장없이 현재 클러스터 결과에서의 비중을 시간에 따라 점차적으로 줄이도록 한다.

본 논문은 다음과 같이 구성된다. 2장에서는 관련된 기존 연구들에 대해서 설명을 하고, 3장에서는 본 논문에서 제안하는 격자 기반의 데이터 스트림 클러스터링 방법을 설명한다. 4장에서는 실험을 통해 제안한 방법의 정확도 및 성능을 분석하고, 마지막으로 5장에서 결론을 제시한다.

2. 관련 연구

클러스터링 기법은 분할기반, 계층기반, 밀도기반 그리고 격자기반 클러스터링으로 분류된다. 분할기반 클러스터링 기법으로는 K-means[15]과 K-medoid[7] 등이 있으며, 주어진 데이터 공간을 k개의 중첩되지 않는 클러스터로 나누는 것을 목적으로 한다. 대상 데이터에 내재된 클러스터의 수가 미리 정의되어 있어야 하며, 먼저 임의의 k개의 클러스터 중심을 선택한 후에 전체 데이터 집합을 반복해서 수행하여 보다 정확한 k 클러스터 중심으로 이를 대체한다. 하지만, 이러한 분할기반 클러스터링 기법들은 잡음 데이터 객체(Noise Data Element)가 존재할 경우 정확한 클러스터를 찾지 못하는 단점이 있다. 계층기반 클러스터링 기법으로는 BIRCH[8]와 CURE[9]가 있다. BIRCH는 CF(Clustering Feature) 트리를 생성하며 데이터 객체들을 해당 범위의 클러스터로 계층적으로 분류한다. CF 트리가 완료된 후에 보다 정확한 클러스터들을 분류하기 위해 기존의 분할기반 클러스터링 기법을 다시 수행하여 데이터 객체들을 노드간에 재분배하기 때문에 무한한 데이터 객체들을 대상으로 빠른 수행을 필요로하는 데이터 스트림 환경에는 부적합하다. CURE에서는 클러스터를 찾기위해 하나의 중심점이 아닌 클러스터 내에 균등히 분산된 다수의 데이터 객체들을 사용한다. 수축인자(Shrinking Factor)에 따라 클러스터를 나타내는 해당 데이터 객체들은 클러스터의 중심에 인접하도록 위치를 재조정한 후, 사용자에 의해 정의된 수의 클러스터를 찾을 때까지 인접한 데이터 객체들부터 병합한다.

전형적인 밀도기반 클러스터링 기법인 DBSCAN[10]에서는 클러스터를 주어진 반경 내에 밀도가 높은 연속된 데이터 객체들의 집합으로 정의한다. DBSCAN에서는 인접한 밀도 높은 데이터 객체들을 순차적으로 탐색하기 때문에 분할기반 클러스터링에 비해 임의의 형태의 클러스터도 정확히 찾을 수 있을 뿐더러 잡음 데이터 객체를 효율적으로 처리할 수

있다. 격자기반 클러스터링 기법에서는 먼저, 주어진 데이터 공간을 동등한 크기의 육방형의 공간(rectangular space), 즉 격자셀들로 나눈다. 각 격자셀에서는 해당 범위 내의 데이터 객체들의 수를 저장하여 임계값 이상의 데이터 객체가 존재하는 연속된 격자셀들의 집합으로부터 클러스터를 찾을 수 있다. STING[16]은 다레벨 격자셀을 사용한 격자기반 클러스터링 기법으로 레벨별로 각각 다른 격자셀로 데이터 공간을 구성하였다. Opti-Grid[20]에서는 고차원 데이터 클러스터링에 있어 주어진 밀도함수에 따라 클러스터가 가장 잘 표출된 사영면(projection)을 찾아 이를 기준으로 격자셀을 생성한다. 고차원 데이터에 대해서 기존의 기법에 비해 뛰어난 성능으로 클러스터를 찾을 수가 있다. 하지만, 지속적으로 생성되는 데이터 스트림 환경에서는 무한한 수의 데이터 객체에 대해 각 밀도함수에 따른 사영면을 구하는 것이 불가능하며, 새로운 데이터 객체로 인해 최적의 사영면은 매시간 변화할 수가 있어 이를 적용하는 것은 불가능하다.

기존의 연구들은 한정된 데이터 집합 내의 클러스터를 찾을 수 있지만, 한정되지 않은, 즉 데이터 객체의 수가 증가하는 데이터를 대상으로는 수행할 수 없다. 이러한 증가하는 데이터 집합을 대상으로 점진적 클러스터링 기법이 연구되었다. 새로 생성된 데이터 객체들에 대해, 전체 데이터 집합을 탐색하는 대신 생성된 데이터 객체와 이에 영향받는 데이터 객체들을 다시 검색해서 클러스터를 찾는다. 따라서, 기존 방법에 비해 전체 데이터 집합을 재검색하지 않고 효율적으로 클러스터를 찾을 수 있다.

[12]에서는 연속해서 데이터 객체가 생성되는 데이터 스트림을 대상으로 클러스터를 찾는 K-median 방법을 제시하였다. 데이터 스트림을 각각의 부분 스트림들로 구성된 집합으로 간주하고, 부분 스트림이 생성될 때마다 O(1)-approximate K-medoid 알고리즘을 사용하는 LSEARCH를 수행하여 각 부분 스트림에서의 k개의 중심점을 찾는다. 한정된 메모리 공간에서 무한한 데이터 스트림을 처리하므로 i번째 부분 스트림을 수행한 결과 ik개의 중심점이 메모리 허용 공간을 초과하는 경우 ik개의 중심점을 클러스터링하여 k개의 중심점만을 저장한다. 하지만, K-means와 K-medoid와 같이 대부분의 분할기반 클러스터링 기법들이 잡음 객체를 처리하지 못하듯이 K-median도 이러한 경우 정확한 클러스터를 찾을 수 없다. 또한, 클러스터의 수가 정해지지 않은 경우, K-median은 보다 좋은 클러스터를 찾기 위해 데이터 스트림을 반복수행한다.

CluStream[17] 또한 분할기반 클러스터링 기법들 중 하나로서, 데이터 스트림 내의 클러스터의 변화를 분석하기 위해 제안되었다. CluStream은 온라인과 오프라인의 두 과정으로 구성되어 있다. 온라인 과정에서는 K-means 기법을 사용하여 메모리 허용 내의 최대 수의 극소 클러스터(Micro Cluster)들을 생성한다. 각 극소 클러스터의 정보는 BIRCH[8]과 유사하게 CF 벡터(Cluster Feature Vector)를 사용하여 저장하며, 이 때 저장된 CF 벡터들은 해당 시점의 클러스터들을 반영하여 이 후 오프라인 과정에서 분석된다. 새롭게 생성된 데이터 객체에 대해 인접한 클러스터 중심과의 거리를 비교하여 클러스터 내의 데이터 객체들의 root-mean-square(RMS) 이하인 경우, 해당 클러스터의 CF벡터를 새로운 데이터 객체에 대해 갱신한다. 반대로, root-mean-square(RMS) 이상인 경우, 최근 데이터 객체로 구성된 새로운 클러스터를 생성한다. 이 때, 증가된 극소 클러스터의 수가 메모리 허용 공간을 초과할 경우, 기존의 극소 클러스터들 중에서 가장 인접한 부 클러스터를 하나의 클러스터로 병합하게 된다.

3. 격자기반 데이터 스트림 클러스터링

3.1 감쇄율

데이터 스트림 상의 최근 클러스터들을 찾기 위해, 각 데이터 객체들은 생성된 시간에 따라 그 비중을 차별화한다. 즉, 오래된 데이터 객체일수록 그 내재된 정보는 현재의 클러스터링 결과에 반영되는 비율이 낮아진다. (그림 1)의 (a)가 기존의 생성 시간에 무관하게 동등한 비중을 나타내며 (b)는 현재로부터 주어진 윈도우 크기 내의 데이터 객체들의 비중만 저장하는 SWF[19]에 의한 비중을 나타낸다. 하지만 SWF는 데이터 객체의 비중을 생성시간에 따라 구분하지 않기 때문에, 최근 데이터 스트림에 존재하지 않는 데이터 객체들도 클러스터로 간주될 수 있으며, 또한 윈도우 내의 데이터 객체들을 물리적으로 저장하여 새로운 데이터 객체가 생성됨에 따라 저장된 과거 데이터 객체들이 유효한지 검사하여 윈도우 범위 밖에 데이터 객체들을 현재의 분석 결과에서 제외시켜야 한다.

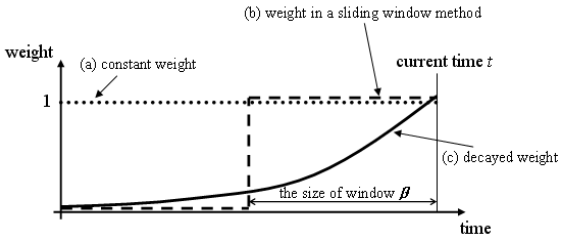
본 논문에서는 최근 데이터 스트림 내에 존재하는 클러스터를 찾기 위해서 데이터 객체의 비중을 점진적으로 감소시켜가는 감쇠율(decay rate) 기법을 사용한다. 감쇠율은 감쇠기저(decay base) b와 감쇠주기(decay period) w에 의해 결정된다. 최근 생성된 데이터 객체의 비중을 1로 가정했을 때, 감쇠주기 w는 데이터 객체의 비중이 $1/b$로 감소될 때까지의 시간을 의미한다. 감쇄율 $\boldsymbol{\tau}$를 수식으로 표현하면 다음과 같다.

$$\tau = b^{-(1/w)} \qquad (b>1,\ w\geq 1,\ b^{-1}\leq \tau < 1)$$

b와 w의 값이 1로 주어진 경우, 데이터 객체의 정보는 감쇄되지 않고 시간에 관계없이 동등하게 적용된다. (그림 1)의 (c)는 시간에 따른 감쇄된 데이터 객체의 비중을 나타낸다. 데이터 객체가 오래될수록 데이터 객체의 비중은 감쇄되어, 최근 데이터 스트림 내의 클러스터를 효율적으로 찾을 수 있다.

3.2 격자셀과 데이터 객체의 분포

d차원 데이터 공간 $\boldsymbol{N}=N_1\times N_2\times\cdots\times N_d$에서 주어진 데이터 스트림 $\boldsymbol{D}$내의 j번째 생성된 데이터 객체는 $\boldsymbol{e}^j=\langle e_1^j, e_2^j, \cdots, e_d^j\rangle$, $e_i^j\in N_i$, $1\leq i\leq$d로 나타낸다. t번째 차례에 $\boldsymbol{e}^t$가 생성된 경우 최근 데이터 스트림 $\boldsymbol{D}^t$는 t번째까지 생성된 데이



(그림 1) 감쇄율에 의한 데이터 객체의 비중

터 객체들의 집합 $\boldsymbol{D}^t = \{\boldsymbol{e}^1, \boldsymbol{e}^2, \cdots, \boldsymbol{e}^t\}$으로 정의하며, 최근 데이터 스트림 $\boldsymbol{D}^t$내의 데이터 객체의 수는 $|\boldsymbol{D}^t|$으로 표기한다. 데이터 스트림 $\boldsymbol{D}^t$의 클러스터링은 데이터 객체의 빈도가 높은 지역을 찾는 방법으로 가정할 수 있다. 따라서, 데이터 객체 간의 유사도를 기준으로 삼는 기존의 클러스터링 기법과 유사하게 격자기반 클러스터링에서는 사용자 정의 거리 λ를 기준으로 모든 차원축 상의 범위가 λ이하인 격자셀을 단위 격자셀로 정의하고, $\boldsymbol{D}^t$내의 데이터 객체의 수에 대한 격자셀 내의 데이터 객체의 수의 비율을 격자셀의 지지도로 정의한다. 데이터 스트림 $\boldsymbol{D}^t$의 클러스터는 지지도가 주어진 최소지지도 S_{min}이상인 인접한 단위 격자셀들의 집합으로 정의된다.

먼저, 각 차원축 N_i를 주어진 p개의 동일한 크기의 범위 $I_i{}^j = [s_i^j, f_i^j)$ $1 \le j \le p$, 로 나눈다. s_i^j와 f_i^j는 i차원상의 j번째 범위의 시작좌표와 끝좌표를 나타낸다. 즉, 각 초기 격자셀은 d개의 범위가 $\{I_1, I_2, \cdots, I_d\}$ $I_i \subseteq N_i$ $1 \le i \le d$로 표시되며, 초기 격자셀 g의 범위 $R(g)$는 각 범위가 교차하는 육방형 공간 $\boldsymbol{rs} = I_1 \times \cdots \times I_d$로 정의된다. 결과적으로 다차원 데이터 공간 $\boldsymbol{N}$은 p^d개의 초기 격자셀로 구성된다. 초기 격자셀의 범위는 이 후 분할과정을 수행하여 분할된 공간 수 q개의 육방형 공간의 집합 $\boldsymbol{RS} = \{\boldsymbol{rs}_1, \boldsymbol{rs}_2, \cdots, \boldsymbol{rs}_q\}$으로 정의된다. 이 때, 격자셀 g의 i번째 차원축에 대한 범위는 $IS_i(g) = \{I_i^1, I_i^2, \cdots, I_i^q\}$로 나타낸다. 즉, i번째 차원에 대한 격자셀 g의 크기 $|IS_i(g)|$는 이들 범위의 합으로 정의하며, 다차원 공간에서의 격자셀 g의 범위는 육방형 공간 $\boldsymbol{rs}_1, \cdots, \boldsymbol{rs}_q$의 합, $R(g) = \bigcup_{i=1}^{q} rs_i$ 로 나타낸다. 각 격자셀은 해당 공간 내의 데이터 객체들의 분포에 대한 통계정보를 저장하며, 정의 1과 같이 정의할 수 있다.

정의 1. 격자셀 $\boldsymbol{g(RS, c, \mu, \sigma)}$

최근 d차원 데이터 스트림 $\boldsymbol{D}^t$에 대해, $g(\boldsymbol{RS}, c^t, \boldsymbol{\mu}^t, \boldsymbol{\sigma}^t)$는 격자셀 g의 공간 $\boldsymbol{RS}$내의 데이터 객체들의 분포에 대한 통계정보를 나타낸다. 격자셀 g내의 데이터 객체들의 집합 $\boldsymbol{D}_g{}^t = \{\ \boldsymbol{e} |\ \boldsymbol{e} \in \boldsymbol{D}^t$ and $\boldsymbol{e} \in R(g)\ \}$에 대해, 격자셀 g의 통계정보는 다음과 같이 정의된다.

i) c^t : $\boldsymbol{D}_g{}^t$내의 감쇄율을 적용한 데이터 객체의 수.

$$c^t = \sum_{j=1}^{c^t} \tau^{(t-j)}$$

ii) $\boldsymbol{\mu}^t = \langle \mu_1^t, \cdots, \mu_d^t \rangle$: $\boldsymbol{D}_g{}^t$내의 데이터 객체들의 i번째 차원에서의 평균값 μ_i^t

$$\mu_i^t = \sum_{j=1}^{c^t} e_i^j \times \tau^{(t-j)} / c^t \ ,\ 1 \le i \le d$$

iii) $\boldsymbol{\sigma}^t = \langle \sigma_1^t, \cdots, \sigma_d^t \rangle$: $\boldsymbol{D}_g{}^t$내의 데이터 객체들의 i번째 차원에서의 표준편차 σ_i^t

$$\sigma_i^t = \sqrt{\sum_{j=i}^{c^t} (e_i^j \times \tau^{(t-j)} - \mu_i^t)^2 / c^t} \ ,\ 1 \le i \le d \qquad \square$$

최근 데이터 스트림 $\boldsymbol{D}^t$에 생성된 데이터 객체 $\boldsymbol{e}^t$에 대해서 p^d초기 격자셀 중 이를 포함하는 초기 격자셀 g를 검색한다. 이전 $v (v \le t)$번째 생성된 데이터 객체에 의해 갱신된 격자셀 $g(\boldsymbol{RS}, c^v, \boldsymbol{\mu}^v, \boldsymbol{\sigma}^v)$는 $\boldsymbol{e}^t$에 의해 다음과 같이 $g(\boldsymbol{RS}, c^t, \boldsymbol{\mu}^t, \boldsymbol{\sigma}^t)$로 갱신된다.

$$c^t = c^v \times \tau^{(t-v)} + 1, \qquad \text{(식 1)}$$

$$\mu_i^t = \frac{\mu_i^v \times c^v \times \tau^{(t-v)} + e_i^t}{c^t}\ ,\ \sigma_i^t = \sqrt{\frac{c^v}{c^t} \times \tau^{(t-v)} \times (\sigma_i^v)^2 + \frac{(\mu_i^v)^2 + (e_i^t)^2}{c^t} - (\mu_i^t)^2}$$

$$\text{for} \forall i,\ 1 \le i \le d \qquad \text{(식 2)}$$

데이터 스트림 $\boldsymbol{D}^t$에 대해서, 격자셀 $g(\boldsymbol{RS}, c^t, \boldsymbol{\mu}^t, \boldsymbol{\sigma}^t)$의 지지도는 $\boldsymbol{D}^t$내의 데이터 객체의 수에 대한 격자셀 내의 데이터 객체의 수의 비율, $c^t / |\boldsymbol{D}^t|$로 정의된다. 격자셀의 지지도가 주어진 분할지지도 $S_{splt} (S_{splt} < S_{min})$ 이상인 경우, 해당 공간 내에 클러스터가 존재하는 것으로 간주하고 보다 정확한 범위를 찾기 위해 해당 격자셀을 두 개의 중간 격자셀 g_1 과 g_2를 해당 초기 격자셀의 하위로 생성되며, 이 때 생성되는 중간 격자셀들의 범위와 통계정보는 사용된 분할방법에 따라 결정된다. 분할방법은 3.2장에서 자세히 기술한다.

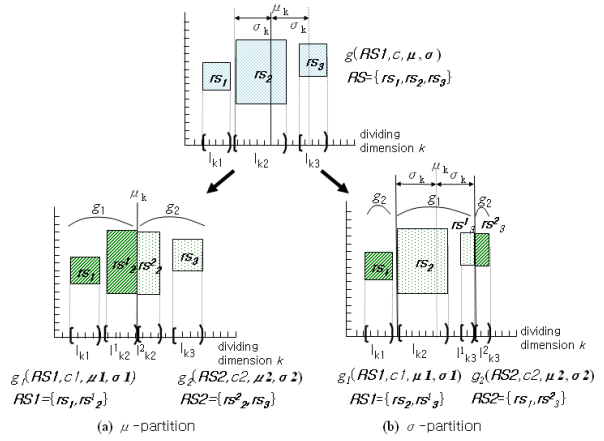
초기 격자셀의 하위로 중간 격자셀이 존재하는 경우, 이들 중 새로 생성된 데이터 객체 e^t를 포함하는 중간 격자셀 g를 검색하여 이의 통계정보를 갱신한다. 해당 중간 격자셀의 지지도가 S_{splt} 이상인 경우 마찬가지로 분할과정을 수행하게 되며, 초기 격자셀과 달리 분할된 중간 격자셀은 보다 작은 중간 격자셀들로 대체되어 해당 초기 격자셀의 하위에 위치한다.

3.3 격자셀 분할방법

본 연구에서는 격자셀의 분할방법으로는 *μ-partition*[13], *σ-partition*[14]과 *hybrid-partition*방법을 제안한다. 먼저, 다차원 데이터 공간 $\boldsymbol{N}$에서의 각 차원들 중 격자셀 내의 데이터 객체들의 분포에 따라 클러스터를 효율적으로 찾을 수 있도록 분할차원(Dividing Dimension)을 결정하고, 선택된 분할차원 상의 범위를 분할한다. 분할방법에 따른 분할차원 선택방법은 각각 다르게 정의된다. μ-partition은 격자셀 내의 데이터 객체들을 가능한 균등하도록 양분하는 분할방법이며, σ-partition은 객체의 빈도가 높은 격자셀과 그 이외의 격자셀을 차등하도록 나누는 방법이다. 그리고, hybrid-partition 방법은 앞의 두 방법 중 데이터 객체들의 분포를 기반으로 보다 효율적인 방법을 동적으로 선택하는 방법이다.

주어진 데이터 스트림 $\boldsymbol{D}^t$에서 데이터 객체의 밀도가 높은 격자셀 g를 μ-partition방법을 통해 g_1과 g_2로 분할을 수행하는 경우, 격자셀 범위의 크기가 λ보다 큰 차원 중 데이터 객체들의 표준편차가 가장 큰 차원 $\sigma_k^t (1 \le k \le d)$을 데이터 객체들이 양분된 차원으로 간주하고 분할차원으로 선택한다. 표준편차 값이 클수록 대다수의 데이터 객체들이 평균에서 멀리 분포하기 때문에 k차원에서의 평균값 μ_k^t를 기준으로 격자셀 g를 보다작은 공간의 격자셀 g_1과 g_2로 분할한다. (그림 2-(a))는 격자셀 $g(\boldsymbol{RS}, c, \boldsymbol{\mu}, \boldsymbol{\sigma})$를 μ-partition방법을 도시하였다.

격자셀 g내의 데이터 객체의 통계값으로부터 이들을 데이터 분포함수에 적용하여 분할된 격자셀 g_1($\boldsymbol{RS1}$, $c1^t$, $\boldsymbol{\mu 1}^t$, $\boldsymbol{\sigma 1}^t$)와 g_2($\boldsymbol{RS2}$, $c2^t$, $\boldsymbol{\mu 2}^t$, $\boldsymbol{\sigma 2}^t$)의 통계값을 추정할 수 있다. 다양한 분포함수를 적용할 수 있지만, 본 연구에서는 이들



(그림 2) 격자셀 분할방법

중 대표적인 정규분포함수를 적용한다. 분할된 격자셀 g_1과 g_2내의 데이터 객체의 수는 $c1^t=c2^t=c^t/2$으로 균등히 분할되며, 정규분포함수 $\varphi(x)=\frac{1}{\sqrt{2\pi\sigma^t}}e^{\frac{(x-\mu^t)^2}{2(\sigma^t)^2}}$ 에 따라 통계값은 다음과 같이 구할 수 있다.

$\mu 1^t = \mu 2^t = \mu^t$ except $\mu 1_k^{\,t}$ and $\mu 2_k^{\,t}$,

$$\mu 1_k^{\,t}=\int_{s_k(g_1)}^{f_k(g_1)} x\,\varphi(x)\,dx \qquad \mu 2_k^{\,t}=\int_{s_k(g_2)}^{f_k(g_2)} x\,\varphi(x)\,dx$$

$\sigma 1^t = \sigma 2^t = \sigma^t$ except $\sigma 1_k^{\,t}$ and $\sigma 2_k^{\,t}$,

$$\sigma 1_k^{\,t}=\sqrt{\int_{s_k(g_1)}^{f_k(g_1)} x^2\,\varphi(x)\,dx-\left(\mu 1_k^{\,t}\right)^2} \qquad \sigma 2_k^{\,t}=\sqrt{\int_{s_k(g_2)}^{f_k(g_2)} x^2\,\varphi(x)\,dx-\left(\mu 2_k^{\,t}\right)^2}$$

$s_k(g_w)$와 $f_k(g_w)$는 분할된 격자셀 g_w, (w=1 or 2)의 분할차원 k상의 범위의 시작좌표와 끝좌표를 의미한다. 초기 격자셀이 분할되는 경우 초기 격자셀의 통계값 c^t=0, $\mu_i^t=\sigma_i^t=0$ ($\forall i$, $1\le i\le$ d)은 초기화되고, 해당 범위내의 데이터 객체들의 정보는 g_1과 g_2를 통해 관리된다.

σ-partition 분할방법에서는 격자셀의 범위가 주어진 λ이상인 차원중 표준편차 $\sigma_l^t(1\le l\le d)$가 가장 작은 차원을 분할차원으로 선택한다. 데이터 객체들의 표준편차가 작은 경우 대다수 데이터 객체들이 평균값에 근접해서 분포하는 것으로 추정할 수 있다. 따라서, 표준편차가 가장 작은 차원을 분할차원으로 선택하여 객체가 빈발한 범위를 보다 효율적으로 분할할 수 있다. 분할차원 l상의 표준편차 σ_l^t에 대해, 정규분포함수에 따라 68%의 데이터 객체가 존재하는 것으로 간주할 수 있는 범위 $[\mu_l-\sigma_l, \mu_l+\sigma_l]$와 이를 제외한 범위로 나눈다. 이를 도시하면 (그림 2-(b))와 같다.

주어진 격자셀 $g(\boldsymbol{RS}, c^t, \boldsymbol{\mu}^t, \boldsymbol{\sigma}^t)$을 σ-partition방법으로 g_1 ($\boldsymbol{RS_1}$, $c1^t$, $\boldsymbol{\mu 1}^t$, $\boldsymbol{\sigma 1}^t$)과 g_2($\boldsymbol{RS_2}$, $c2^t$, $\boldsymbol{\mu 2}^t$, $\boldsymbol{\sigma 2}^t$)로 분할하는 경우, 데이터 객체의 수는 정규분포함수에 따라 다음과 같이 추정된다.

$$c1^t=c^t\times\int_{\mu_k^t-\sigma_k^t}^{\mu_k^t+\sigma_k^t}\varphi(x)\,dx, \quad c2^t=c^t-c1^t \qquad \text{(식 3)}$$

분할된 격자셀들의 분할차원 k상의 통계값 $\mu 1_k^{\,t}$, $\mu 2_k^{\,t}$, $\sigma 1_k^{\,t}$과 $\sigma 2_k^{\,t}$ 는 다음과 같이 추정할 수 있다.

$$\mu 1_k^{\,t}=\int_{s_k(g_1)}^{f_k(g_1)} x\,\varphi(x)\,dx \quad \text{and} \quad \sigma 1_k^{\,t}=\sqrt{\int_{s_k(g_1)}^{f_k(g_1)} x^2\,\varphi(x)\,dx-\left(\mu 1_k^{\,t}\right)^2}$$

if $s_k(g_2) < \mu 1_k^{\,t} < f_k(g_2)$,

$$\mu 2_k^{\,t}=\int_{s_k(g_2)}^{f_k(g_2)} x\,\varphi(x)\,dx \quad \mu 1_k^{\,t}\text{ , and}$$

$$\sigma 2_k^{\,t}=\sqrt{\int_{s_k(g_2)}^{f_k(g_2)} x^2\,\varphi(x)\,dx-\int_{s_k(g_1)}^{f_k(g_1)} x^2\,\varphi(x)\,dx-\left(\mu 2_k^{\,t}\right)^2}$$

else $\mu 2_k^{\,t}=\int_{s_k(g_2)}^{f_k(g_2)} x\varphi(x)dx$ and $\sigma 2_k^{\,t}=\sqrt{\int_{s_k(g_2)}^{f_k(g_2)} x^2\,\varphi(x)\,dx-\left(\mu 2_k^{\,t}\right)^2}$.

분할차원을 제외한 차원에서의 통계값은 μ-partition 분할방법과 동일하게 구할 수 있다.

hybrid-partition 분할방법은 위의 두 분할방법 중 보다 효율적인 분할방법을 선택적으로 수행하는 방법이다. 본 클러스터링 방법에서는 적은 수의 분할과정으로 단위 격자셀을 빠르게 찾는 것이 본 성능을 결정하는 중요한 요소가 된다. 따라서, 두 분할방법 중 보다 단위 격자셀을 빨리 찾을 수 있는 방법을 선택하여 클러스터링에 필요한 분할과정을 줄일 수 있다. 데이터 객체들이 균일하게 분포하는 i차원의 표준편차 σ^e_i에 대해서, 분할임계값 $\beta_k(g)$는 격자셀 g의 표준편차와 σ^e_i와의 차이로 다음과 같이 정의할 수 있다.

$$\beta_k(g) = |\sigma^e_k - \sigma^t_k|, \quad \text{where} \quad \sigma^e_k=\sqrt{\frac{1}{f_k(g)-s_k(g)}\int_{s_k(g)}^{f_k(g)}x^2dx-(\mu_k^t)^2}$$

주어진 격자셀 g에 대해서, μ-partition과 σ-partition에 의해 선택된 분할차원을 각각 $k1$과 $k2$라 간주한다. 즉, 데이터 객체들의 표준편차가 가장 큰 차원을 $k1$, 가장 작은 차원을 $k2$라 한다. 이 때, 각 분할차원에 대한 분할임계값을 비교하여, $\beta_{k1}(g) > \beta_{k2}(g)$인 경우 데이터 객체들이 $k1$차원에서 분산된 정도가 $k2$차원에서 응집된 정도보다 큰 것으로 판정하고 μ partition을 선택해서 수행할 수 있다. 반대로, $\beta_{k1}(g) < \beta_{k2}(g)$인 경우, σ partition 방법을 선택하여 효율적으로 분할과정을 수행할 수 있으며, $\beta_{k1}(g) = \beta_{k2}(g)$의 경우에는 분할임계값이 동일하기 때문에 둘 중 어느 방법으로도 분할을 수행할 수 있다. 데이터 객체의 분포에 따라 효율적인 분할방법을 선택하여 클러스터링 수행 동안의 분할과정의 수를 줄이고, 불필요한 격자셀의 생성을 방지할 수 있다.

3.4 격자셀 전지과정

중간 격자셀이나 단위 격자셀의 지지도가 주어진 전지지지도(Pruning Support) S_{prn}보다 낮은 경우($c^t/|\boldsymbol{D}^t|\le S_{prn}$), 격자셀의 범위 내에 클러스터가 존재하지 않는 것으로 간주할 수 있다. 따라서, 이러한 격자셀은 삭제하고, 삭제된 격자셀 범위의 데이터 객체들의 통계정보는 다시 해당 초기 격자셀로 반영하여 클러스터와 무관한 격자셀의 수를 줄일 수 있다. 이러한 과정을 전지과정이라 한다. v번째 데이터 객체에 의해 최근 갱신된 초기 격자셀 $g_p(\boldsymbol{RS_p}, cp^v, \boldsymbol{\mu p}^v, \boldsymbol{\sigma p}^v)$와 데이터 객체들의 통계정보 $\boldsymbol{\mu p}^v = <\mu p_1^v, \mu p_2^v, \cdots, \mu p_d^v>$, $\boldsymbol{\sigma p}^v = <\sigma p_1^v, \sigma p_2^v, \cdots, \sigma p_d^v>$에 대해, 격자셀 $g(\boldsymbol{RS}, c^t, \boldsymbol{\mu}^t, \boldsymbol{\sigma}^t)$에 대해 전지과정을 수행한 경우 초기 격자셀은 다음과 같이 갱신된다.

$$cp^t = cp^v \times \tau^{(t-v)} + c^t ,$$

$$\mu p_i^t = \frac{\mu p_i^v \times c^v \times \tau^{(t-v)} + \mu_i^t \times c^t}{cp^t} \quad \text{for all dimensions } i\ (1 \le i \le d)$$

$$\sigma p_i^t = \sqrt{\frac{cp^v \times (\sigma p_i^v)^2 \times \tau^{(t-v)} + c^t \times (\sigma_i^t)^2}{cp^t} + \frac{(\mu p_i^v)^2 + (\mu_i^t)^2}{cp^t} - (\mu p_i^t)^2}$$

σ-partition에 의해 격자셀을 분할한 경우, 데이터 객체의 수는 수식 (3)과 같이 68%인 격자셀과 그 나머지 수의 격자셀로 나누어 진다. 따라서, 전지지지도 S_{prn}은 분할지지도 S_{splt}의 32% 이하로 정의되어야 분할된 격자셀이 너무 일찍 삭제되는 것을 방지할 수 있다. 동일한 이유로 μ-partition에서 S_{prn}은 S_{splt}의 1/2 이하로 정의되어야 하며, hybrid-partition에서도 σ-partition와 같이 전지지지도 S_{prn}의 값을 정의해야 한다.

데이터 객체의 빈도가 낮은 중간 격자셀 또는 단위 격자셀은 해당 범위 내에 최근 데이터 객체가 생성되었을 때, 해당 격자셀을 탐색하여 전지과정을 수행할 수 있다. 하지만, 빈도가 낮은 격자셀의 범위에 데이터 객체가 생성될 가능성은 매우 낮으므로 다수의 빈도 낮은 격자셀들은 클러스터가 존재하지 않으면서도 전지과정이 수행되지 않을 수 있다. 따라서, 모든 중간 격자셀과 단위 격자셀들을 탐색하여 지지도에 따라 전지과정을 수행할 수 있다. 이러한 과정을 강제전지과정(Force-pruning Operation)이라 한다. 모든 중간 격자셀과 단위 격자셀을 탐색해야하기 때문에, 강제전지과정은 수행시간의 관점에서 부하가 큰 작업이므로 주기적으로 또는 메모리 사용량에 따라 선택적으로 이를 수행할 수 있다.

무한한 데이터 스트림에 비해 주어진 메모리 공간은 한정되어 있기 때문에, 본 클러스터링 방법에서는 주어진 단위 격자셀의 크기 λ를 적응적으로 변화하여 메모리 사용량을 조절한다. 주어진 λ에 대해 메모리 사용량이 메모리 공간을 초과하는 경우, 모든 단위 격자셀에 대해 전지과정을 수행하고 포함된 데이터 객체들의 통계값들을 해당 초기 격자셀로 갱신한다. 또한, 중간 격자셀들 중에서 각 차원에 대한 범위가 2λ이하인 격자셀에 대해서도 전지과정을 수행한 후, 단위 격자셀의 크기를 2λ로 정의한 후 새로 생성되는 데이터 객체들에 대해 동일하게 클러스터링을 수행한다. λ의 값이 증가함에 따라 클러스터를 찾는데 필요한 격자셀의 수는 줄어드는 반면에 클러스터의 정확도는 감소한다. 반면에, 가용 메모리 공간이 현재 메모리 사용량의 2배 이상인 경우, λ를 1/2로 감소시켜서 주어진 메모리 공간을 모두 활용하여 클러스터의 정확도를 높일 수 있다.

3.5 알고리즘

본 데이터 스트림 클러스터링 기법의 구체적인 알고리즘은 (그림 3)과 같다. 본 알고리즘은 4단계, 즉 갱신단계, 분할단계, 전지단계 그리고 동적 메모리 관리 단계로 구분된다. 주어진 데이터 스트림 $\boldsymbol{D}^t$에서 최근 데이터 객체 $\boldsymbol{e}^t$가 생성되면서 메모리 관리 단계를 제외한 세 단계를 $\boldsymbol{e}^t$를 포함하는 격자셀에 대해 반복적으로 수행한다. 갱신단계(그림3의 6번줄)에서는 (식 1, 2)에 따라 격자셀 g내의 데이터 객체 분포의 통계정보를 갱신한다. 갱신된 격자셀 g의 지지도에 따라 지지도가 S_{splt} 이상이며 각 차원에 대한 범위의 크기가 λ를 초과하는 경우 분할과정 (7~18번줄)을 수행한다. 선택한 분할방법에 따라 분할차원을 먼저 선택한 후 3.3장의 분할과정을 수행한다. hybrid-partition 분할방법을 사용할 경우, μ-partition과 σ-partition의 분할차원 $k1$과 $k2$의 분할임계값 $\beta(g_{k1})$과 $\beta(g_{k2})$를 비교하여 사용할 분할방법을 결정한다. 선택된 분할방법에 따라 격자셀 g는 g_1과 g_2로 분할과정을 수행하여 분포함수에 따라 데이터 객체 분포에 대한 통계값을 갱신한다. g_1과 g_2는 해당 범위의 초기 격자셀의 하위로 위치되며, 격자셀 g가 중간 격자셀인 경우 g는 g_1과 g_2로 대체된다.

반면에, 격자셀 g가 초기 격자셀이 아니면서 지지도가 전지지지도 S_{prn}이하인 경우 전지단계(19~25번줄)가 수행된다. 격자셀 g는 삭제되고 g내의 데이터 객체 분포에 대한 통계값은 해당 초기 격자셀에 반영된다. 동적 메모리 관리 단계(26~36번줄)은 주기적으로 또는 메모리 사용량 조건에 따라 수행된다.

격자셀 분할과정에 있어서 분할된 격자셀 범위 내의 데이터 객체의 수는 분포함수에 따라 추정하게 되지만, 실제 데이터 스트림은 분포함수와 항상 일치하지 않기 때문에 추정에 의한 오차가 발생한다. 즉, 분할과정이 수행될수록 분할된 격자셀 내의 데이터 객체의 수는 오차를 포함하게 되며, 단위 격자셀까지 분할이 반복된 범위는 이러한 오차가 누적되게 된다. 하지만, 단위 격자셀이 생성된 범위는 더 이상의 분할과정을 수행하지 않으므로 이후에는 오차없이 정확한 데이터 객체의 수를 저장하게 된다. 따라서, 무한한 데이터 스트림에 대해 단위 격자셀에 저장된 데이터 객체의 수에 대한 오차는 상수로 정의할 수 있다. 반면에 특성1에 의해 단위 격자셀의 지지도오차는 지속적으로 감소한다. 주어진 데이터 스트림 $\boldsymbol{D}^t$에 대해 $|\boldsymbol{D}_g^t|$를 단위 격자셀 $g(\boldsymbol{RS}, c^t, \boldsymbol{\mu}, \boldsymbol{\sigma})$의 범위에 존재하는 실제 데이터 객체의 수라 가정하자. 이 때, 데이터 객체의 수에 대한 오차 E(g)는 추정 데이터 객체의 수 c^t와 실제 데이터 객체의 수 $|\boldsymbol{D}_g^t|$간의 차이 E(g)= $|\,|\boldsymbol{D}_g^t| - c^t|$로 나타낸다.

특성 1. (지지도오차 감소 특성) 주어진 최근 데이터 스트림 $\boldsymbol{D}^t$에 생성된 단위 격자셀 g에 대해, 현재까지 분할과정을 통해 추정된 데이터 객체의 수에 대한 오차 E(g)는 상수이며 지지도오차는 E(g)/$|\boldsymbol{D}|^t$로 표현된다. 이 후 m개의 데이터 객체들을 처리한 후에 전체 데이터 객체의 수는 $|\boldsymbol{D}^{t+m}|$으로 증가하고, 단위 격자셀 g의 지지도오차는 $\frac{E(g)}{|\boldsymbol{D}^{t+m}|}$이며, 이전의 지지도 오차에 대해 $\frac{E(g)}{|\boldsymbol{D}^{t+m}|} < \frac{E(g)}{|\boldsymbol{D}^{t}|}$를 만족한다. m이 무한히 증가함에 따라 지지도오차 $\frac{E(g)}{|\boldsymbol{D}^{t+m}|}$는 0에 수렴한다. 즉, $\lim_{m\to\infty} \frac{E(g)}{|\boldsymbol{D}^{t+m}|} \approx 0$로서 무시할 수 있다. □

단위 격자셀의 지지도가 주어진 최소지지도 S_{min} 이상인 경우 이를 데이터 객체의 빈도가 높은 의미있는 영역으로 간주한다. 즉, 클러스터는 지지도가 S_{min}이상인 연속된 단위 격자셀들의 집합으로 정의된다. 단위 격자셀의 크기 λ가 작을수록 정밀한 클러스터를 정확히 찾을 수 있다. 반면에, 분할지지도 S_{splt}값이 작을수록 분할과정을 일찍 수행하게 되며, 결과적으로 단위 격자셀을 빠르게 생성할 수 있어, 분할과정에서의 데이터 객체 수에 대한 오차를 보다 줄일 수 있다. 또한, 전지지지도 S_{prn}과 분할지지도 S_{splt}간의 차이가 클

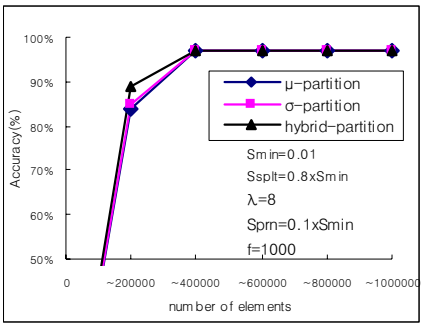
```
1: divide N1,…,Nd into p intervals and create p^d initial cells;
 /*  S(g) : the support of cell g , S(g)=c^t / |D^t|  */

2: for a new data element e^t generated in D^t do
3:    t=t+1;
4:    |D^t|=|D^(t-1)|+1;

    /* Updating Phase */
5:    search the cell g whose range includes e^t, i.e.,e^t∈R(g);
6:    update μ^t, σ^t, c^t  of the cell g;

    /* Partition Phase */
7:    if S(g) >= Sspit {
8:       if ∃ dimension i, 1≤ i≤ d, such that |f_i(g) − s_i(g)| > λ {
9:          find the largest σ_k1^t among σ^t where |f_k1(g) − s_k1(g)| > λ and
            find the smallest σ_k2^t among σ^t where |f_k2(g) − s_k2(g)| > λ;
10:         select the dividing dimension k;
11:         divide g into g1 and g2 with respect to dimension k;
12:         initialize the distribution statistics of g1 and g2;
13:         if g is an initial cell
14:            set c^t=0 and μ_i^t=σ_i^t=0 for ∀ i dimension;
15:         else
16:            eliminate g; continue;
17:      }
18:   }

    /* Pruning Phase */
19:   if g is not an initial cell {
20:      if S(g) <= Sprn {
21:         find the parent initial cell g_p including e^t, i.e., e^t∈R(g_p) ;
22:         update g_p with distribution statistics of g;
23:         eliminate g; continue;
24:      }
25:   }

    /* Memory-Space Adjusting Phase */
26:   if no free memory space {
27:      for all intermediate and unit cells g do {
28:         add the distribution statistics of g to its parent initial cell;
29:         eliminate g;
30:      }
31:      λ =λ × 2;
32:   }
33:   if free memory space stays larger than two-fold of the total size
      of all unit cells {
34:      λ=λ/2;
35:   }
36: end
```

(그림 3) 격자기반 데이터 스트림 클러스터링 알고리즘

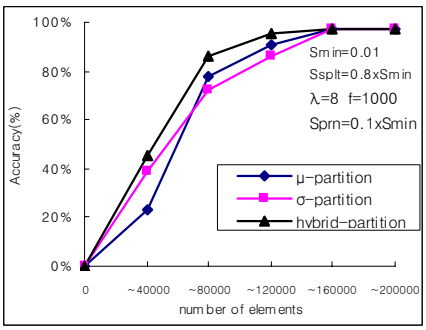
수록 보다 많은 중간 격자셀들이 유지되어 보다 신중히 정확한 단위 격자셀을 찾을 수 있다.

4. 실험 결과

본 데이터 스트림 클러스터링 방법의 성능을 분석하기 위해서, ENCLUS[18]의 데이터 생성기를 사용하여 40차원의 백만개의 데이터 객체들로 구성된 데이터 스트림을 생성하였다. 데이터 객체들은 임의의 10개의 영역에 밀집하여 생성되었으며, 밀집된 영역의 크기는 5~20으로 생성되었다. 본 클러스터링 방법의 정확도는 기존의 대표적인 격자기반 클러스터링 기법인 STING과 비교하였다. 즉, 전체 데이터 객체들의 수에 대해 본 클러스터링 수행결과 STING과 동일한 클러스터로 판별된 데이터 객체들의 수의 비율로 정확도를 표시하였다. 본 클러스터링에 필요한 변수와 임계값들은 $S_{min}=0.01$, $S_{splt}=S_{min}x0.8$, $S_{prn}=S_{min}x0.1$, $\lambda=8$로 주어졌으며 강제전지 주기 $f=1000$으로 실험하였고, 실험 목적에 따라 특정변수에 대한 성능들을 비교 분석하였다. 주어진 다



(그림 4) 클러스터링 정확도

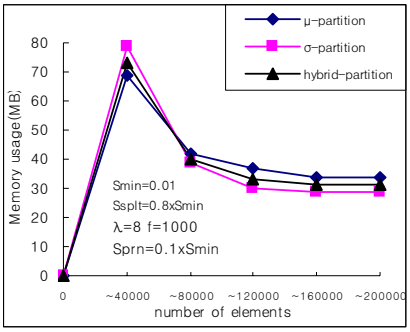


(그림 5) 첫번째 구간의 정확도

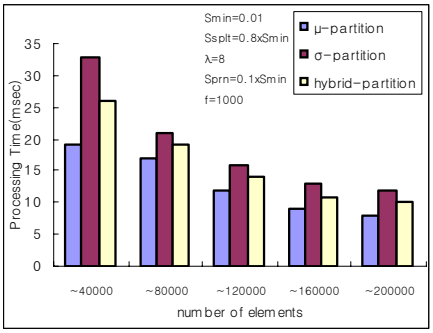
차원 데이터 공간은 4개의 초기 격자셀로 나누었으며, 모든 실험에서 데이터 스트림 환경을 모의실험하기 위해 데이터 객체는 하나씩 순차적으로 처리되었다.

(그림 4)는 각 분할방법에 따른 정확도의 변화를 도시하였다. 전체 데이터 객체의 수에 대한 STING과 동일한 클러스터로 분류된 데이터 객체의 수의 비율을 도시하였으며, 데이터 스트림은 200,000개 구간으로 구분하여 각 구간에서의 평균 정확도를 도시하였다. 강제전지과정은 40,000개의 데이터 객체를 주기로 수행하였다. (그림 5)는 (그림 4)의 실험에서 격자셀 분할이 잦은 첫번째 구간을 세분하여 도시하였다. 클러스터링 시작단계에서는 각 격자셀의 지지도가 불안정하게 변화하므로 이후단계에 비해 정확도가 아주 낮지만, 분할과정 결과 단위 격자셀을 생성하면서 점차적으로 정확도가 높아지고, 성능이 안정화된다. 클러스터링 첫번째 구간에서는 hybrid-partition 방법이 가장 높은 정확도를 보인다. 격자셀 내의 데이터 객체들의 분포를 반영하여 분할방법을 선택하기 때문에 적은 수의 분할과정으로 단위 격자셀을 타방법보다 빠르게 생성하기 때문이다. 따라서, hybrid-partition 분할방법이 다른 분할방법에 비해 정확한 결과를 빠르게 반영할 수 있어, 기존 방법에 비해 데이터 스트림 분석에 적합하다고 할 수 있다.

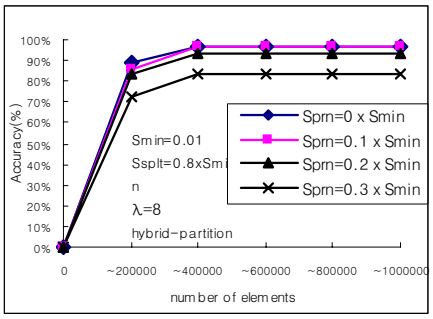
(그림 6)은 첫번째 구간에서의 각 분할방법에 따른 메모리 사용량의 변화를 도시하였다. 최대 메모리 사용량을 비교하기 위해서 메모리 공간은 한정되지 않은 것으로 간주하였으며, 각 분할방법의 메모리 사용량은 40,000개로 세분화된 각 구간에서의 최대 메모리 사용량을 도시하였다. 각 분할방법에 따라 측정된 최대 메모리 사용량은 미세한 차이를 보여준다. (그림 7)은 각 분할방법의 평균수행시간을 도시하였다.



(그림 6) 메모리 사용량



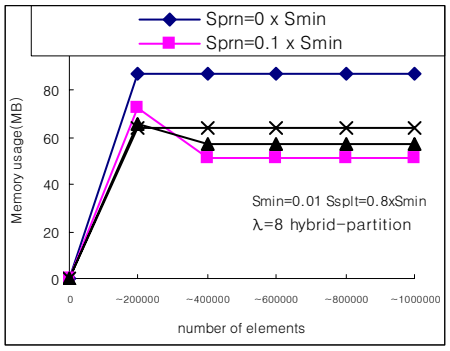
(그림 7) 수행 시간



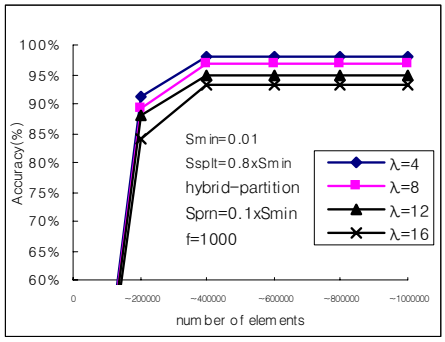
(그림 8) S_{prn}의 정확도 변화

최근 생성된 데이터 객체에 대해 해당 격자셀에서의 분할과정 또는 전지과정까지의 수행시간들의 각 구간에서의 평균수행시간으로 도시하였다. 첫번째 구간에서는 빈발한 분할과정으로 인해 평균수행시간이 크게 도시되었지만, 이후 성능이 안정화되면서 수행시간은 감소되어 동일하게 유지된다.

(그림 8)은 전지지지도 S_{prn}의 변화에 따른 정확도의 변화를 도시하였다. S_{prn}이 증가할수록 단위 격자셀을 생성하기 전에 다수의 중간 격자셀들을 전지하기 때문에 정확도는 감소하게 된다. (그림 9)는 동일한 상황에서의 메모리 사용량의 변화를 도시하였다. 성능이 안정화된 두번째 구간 이후에서도 S_{prn}을 통해 메모리 사용량을 조절할 수 있다. S_{prn}이 $0.3\times S_{min}$인 경우, 분할된 중간 격자셀들이 빠르게 전지되어 다시 초기 격자셀로부터 분할을 반복하기 때문에, 메모리 사용량은 오히려 증가하였다. 반면에, S_{prn}이 $0.1\times S_{min}$인 경우에는 중간 격자셀로부터 효율적으로 단위 격자셀을 생성하고, 불필요한 격자셀만을 전지할 수 있어서 메모리 사용



(그림 9) S_{prn}의 메모리 사용량 변화



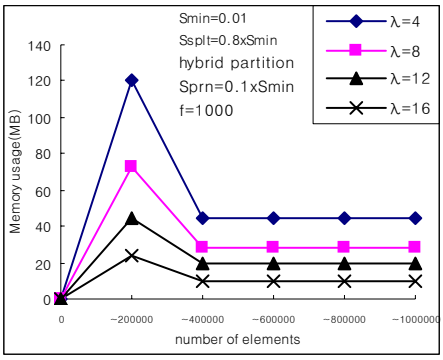
(그림 10) λ의 정확도 변화

량을 감소할 수 있었다. 전지지지도를 조절하여 약간의 정확도의 손실로 메모리 사용량을 최소화시킬 수 있다.

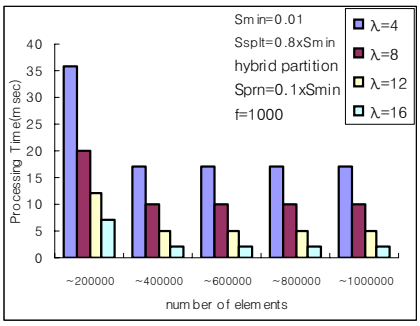
(그림 10)은 단위 격자셀의 크기 λ의 변화에 따른 정확도를 도시하였다. λ에 의해 클러스터의 정밀도가 결정되기 때문에, λ가 작을수록 클러스터의 경계를 정확히 찾을 수 있어 정확도가 향상되는 반면에 많은 수의 단위 격자셀을 생성하기 때문에 메모리 사용량이 증가한다. (그림 11)과 (그림 12)는 동일한 상황에서의 메모리 사용량과 평균수행시간을 도시하였다. λ가 증가함에 따라 필요한 격자셀의 수는 줄어들어 메모리 사용량과 평균수행시간은 크게 감소한다.

(그림 13)은 한정된 메모리 공간 내에서의 본 클러스터링 방법의 동적 메모리 관리 성능을 도시하였다. 성능비교를 위해 λ를 동적으로 변화하는 *adjust* 계열과 λ를 변화하지 않는 *non-adjust* 계열을 비교하였으며, 메모리 공간은 30MB로 제한하였다. 단위 격자셀의 크기 λ는 16으로 초기화되었으며, 첫 구간에서의 메모리 사용량은 동일하다. 하지만, 이 후 전지과정이 수행되면서 *non-adjust*의 메모리 사용량은 급격히 감소되어 성능이 안정화되는 반면, *adjust*에서는 가용 메모리 공간이 증가함에 따라 λ를 8로 변화하면서 보다 많은 단위 격자셀들을 생성하게 된다. 결과적으로 *adjust*에서의 메모리 사용량은 주어진 메모리 공간에 맞게 계속 조절하게 된다. (그림 14)는 본 실험에서의 정확도를 도시하였다. λ를 조절함에 따라 첫구간에서의 정확도는 *adjust*가 낮지만, 이 후 변화된 λ의 단위 격자셀을 생성함에 따라 *non-adjust*에 비해 크게 증가하게 된다.

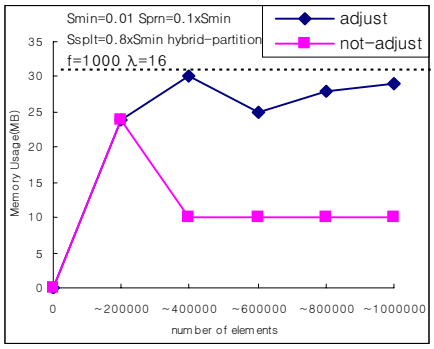
(그림 15)는 데이터 스트림의 변화에 따른 클러스터링 결과의 변화를 보여준다. 감쇄율 실험을 위해서 두 개의 데이



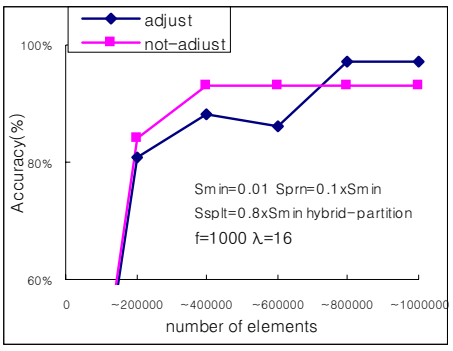
(그림 11) λ의 메모리 사용량



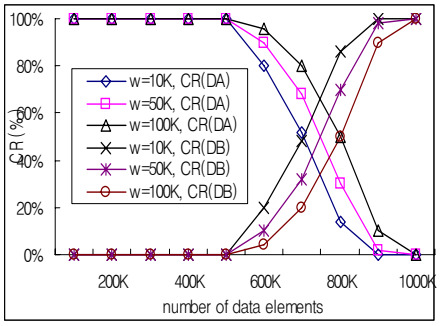
(그림 12) λ의 수행시간 변화



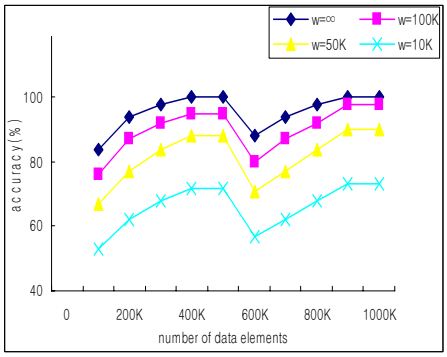
(그림 13) 적응적 메모리 관리



(그림 14) 적응적 메모리 관리에 따른 정확도



(그림 15) 감쇄율에 의한 변화



(그림 16) 감쇄주기에 의한 정확도

터 집합 D_A와 D_B를 연결한 데이터 스트림 D_{AB}를 구성하였다. D_A와 D_B는 각각 500K의 데이터 객체들로 구성되어 있으며, 각 데이터 집합의 결과 클러스터들은 서로 중첩되지 않는 범위에 존재하도록 하였다. S_{min}과 b의 값은 0.1과 2로 설정하였다. 데이터 스트림의 변화에 대해 본 알고리즘이 변화된 클러스터를 찾아가는 성능을 분석하기 위해 클러스터 적응율 CR(coverage rate)을 사용하였다. 데이터 집합 $\boldsymbol{D_i}$ 중에서 클러스터에 포함된 데이터 객체들의 집합을 $P(\boldsymbol{D_i})$라 정의할 때, 두 데이터 집합 D_A와 D_B에 대해서 클러스터 적응율 CR은 다음과 같이 정의된다.

$$CR(\boldsymbol{D_i}) = \frac{|P(D_i)|}{|P(D_A) \cup P(D_B)|} \times 100\ (\%)\ ,\ i=A \text{ or } B$$

감쇄주기 w가 감소할수록, 본 클러스터링 방법은 (그림 15)와 같이 최근 변화된 클러스터들을 빠르게 찾아간다. 따라서, 감쇄주기 w를 조절하여 클러스터의 변화에 대한 본 클러스터링 방법의 적응도를 조절할 수 있으며, 또한 감쇄기저 b를 감쇄주기와 유사하게 성능을 조절할 수 있다. (그림 16)은 감쇄주기에 따른 정확도의 변화를 보여준다. 본 실험에서의 정확도는 감쇄율 방법을 STING에 적용한 *dSTING*과 결과 클러스터를 비교하였다. 500Kth 데이터 객체에서부터 D_A의 클러스터가 D_B로 변화하면서 D_A에서 클러스터에 속했던 격자셀들의 지지도가 낮아지면서 정확도는 감소하게 된다. 하지만, 다시 D_B의 클러스터에 대한 지지도가 높은 격자셀을 찾으면서 600Kth 데이터 객체 이후 점점 정확도는 향상된다.

5. 결 론

본 논문에서는 데이터 스트림 분석을 위한 격자기반 데이터 스트림 클러스터링 방법을 제시하였다. 다차원 데이터 공

간을 그 분포에 따라 동적 범위의 격자셀들로 분할하며, 각 격자셀에서는 데이터 객체들의 분포에 대한 통계값만을 저장하여 기존의 클러스터링 기법에 비해 데이터 객체의 저장없이 효율적으로 수행할 수 있다. 데이터 객체의 밀도가 높은 범위는 단위 격자셀이 생성될 때까지 반복해서 분할하게 되며, 클러스터는 연속한 최소지지도 이상인 단위 격자셀들을 탐색하여 찾을 수 있다. 격자셀의 분할을 위해 μ-partition, σ-partition과 hybrid-partition 분할방법을 제시하였다. μ-partition와 σ-partition 분할방법은 고정된 비율로 격자셀을 분할하는 반면, hybrid partition 방법은 동적으로 보다 효율적인 분할방법을 선택적으로 수행하여 데이터 객체의 분포를 분할과정에 반영할 수 있도록 하였다. 또한, 동적 메모리 관리 방법을 통해 적응적으로 λ를 조절하여 제한된 메모리 공간을 최대한 사용하여 클러스터의 정확도를 주어진 자원에 따라 적응적으로 극대화하였다.

참 고 문 헌

[1] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. *In Proc. Of the 28th Int'l Conference on Very Large Databases*, Hong Kong, China, Aug. 2002.

[2] M. Garofalakis, J. Gehrke and R. Rastogi. Querying and mining data streams: you only get one look. *In the tutorial notes of the 28th Int'l Conference on Very Large Databases*, Hong Kong, China, Aug. 2002.

[3] J. H. Chang & W. S. Lee. Finding Frequent Itemsets over Online Data Streams. *Information and Software Technology*, 48(7), July 2006.

[4] J. H. Chang & W. S. Lee. Finding Recently Frequent Itemsets Adaptively over Online Transactional Data Streams. *Information Systems*, 31(8), December 2006

[5] Hua-Fu Li, Suh-Yin Lee, Man-Kwan Shan: Online Mining Changes of Items over Continuous Append-only and Dynamic Data Streams. J. UCS 11(8), page 1411-1425, 2005

[6] Mohamed Medhat Gaber, Arkady B. Zaslavsky, Shonali Krishnaswamy: Mining data streams: a review. SIGMOD Record 34(2), page 18-26, 2005

[7] L. Kaufman and P.J. Rousseeuw. Finding Groups in Data. An Introduction to Cluster Analysis. Wiley, New York, 1990.

[8] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. *In Proc. SIGMOD*, pages 103 114, 1996

[9] S. Guha, R.Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. *In Proc. SIGMOD,* pages 73 84, 1998

[10] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases, 1996.

[11] M. Ester, H. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental clustering for mining in a data warehousing environment, *In Proc. VLDB 24th*, New York, 1998

[12] Liadan O'Callaghan, Nina Mishra, Adam Meyerson, Sudipto Guha, and Rajeev Motwani. STREAM-data algorithms for high-quality clustering. *In Proc. of IEEE International Conference on Data Engineering*, March 2002.

[13] Nam Hun Park and Won Suk Lee. A statistical μ-partitioning method for clustering data streams. *In Proc. of Eighteenth International Symposium on Computer and Information Sciences*, November 2003.

[14] Nam Hun Park and Won Suk Lee. Statistical σ-partition Clustering over Data Streams. *In Proc. of 7th European Conference on Principles and Practice of Knowledge Discovery in Databases*, September 2003.

[15] R. O. Duda and P. E. Hart. Pattern Classification and Scene Analysis. Wiley, 1972.

[16] W. Wang, J. Yang, and R. Muntz. Sting: A statistical information grid approach to spatial data mining, 1997.

[17] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, Philip S. Yu. A Framework for Clustering Evolving Data Streams. In Proc. VLDB 29th, Berlin, 2003.

[18] Cheng, C., Fu, A., and Zhang, Y. Entropy based subspace clustering for mining numerical data. KDD-99, 84-93, San Diego, August 1999.

[19] C.-H. Lee, C. R. Lin, and M.-S. Chen, Sliding-window filtering: An efficient algorithm for incremental mining, Proceedings of the 10th International Conference on Information and Knowledge Management, Atlanta, GE, November 2001, pp.263-270.

[20] A. Hinneburg and D. A. Keim, "Optimal Grid Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering", In Proc. Int' Conf. on Very Large Data Bases(VLDB), Edinburgh, Scotland, pp.506-517, Sept. 1999.

박 남 훈

e-mail : zyonix@database.yonsei.ac.kr

2000년 연세대학교 컴퓨터과학과(학사)

2002년 연세대학교 대학원 컴퓨터과학과 (석사)

2002년 ~ 현 재 연세대학교 대학원 컴퓨터과학과 박사과정 재학

관심분야 : 데이터 스트림, 데이터 마이닝

이 원 석

e-mail : leewo@database.yonsei.ac.kr

1985년 미국 보스턴대학교 컴퓨터공학과 (공학사)

1987년 미국 퍼듀대학교 컴퓨터공학과 (공학석사)

1990년 미국 퍼듀대학교 컴퓨터공학과 (공학박사)

1990년 ~ 1992년 삼성전자 선임연구원

1993년 ~ 1999년 연세대학교 컴퓨터과학과 조교수

1999년 ~ 2004년 연세대학교 컴퓨터과학과 부교수

2004년 ~ 현 재 연세대학교 컴퓨터과학과 교수

관심분야 : 분산데이터베이스, 미디에이터시스템, 데이터마이닝, 데이터스트림