Author Name

Book title goes here

List of Figures

List of Tables

Contents

Symbol Description

α	To solve the generator maintenance scheduling, in the past, several mathematical techniques have been applied.
σ^2	These include integer programming, integer linear programming, dynamic programming, branch and bound etc.
$\sum$	Several heuristic search algorithms have also been developed. In recent years expert systems,
abc	fuzzy approaches, simulated annealing and genetic algorithms have also been tested.
$\theta\sqrt{abc}$	This paper presents a survey of the literature

Part I

This is a Part

Chapter 1

Incremental Maintenance of Emerging Patterns

Mengling Feng

Data Mining Department, Institute for Infocomm Research

Guozhu Dong

Department of Computer Science and Engineering, Wright State University

Due to the advance in data acquisition, storage and transfer technologies, data is nowadays dynamically updated all the time. In addition, in many interactive data mining applications, data is often modified repeatedly. Re-generating the corresponding emerging patterns from scratch every time when the underlying data is updated/modified is obviously inefficient. It is more advantageous to incrementally maintain the emerging patterns.

In this chapter, we first present the motivations and potential applications for incremental maintenance of emerging patterns. We then formally define the maintenance problem, and discuss the challenges associated with solving the problem. After that, we discuss how an emerging pattern space can be concisely represented by its *border* [7]. Finally, we demonstrate how an emerging pattern space, represented by its *border*, can be effectively maintained under various data updates or modifications [13].

1.1 Background & Potential Applications

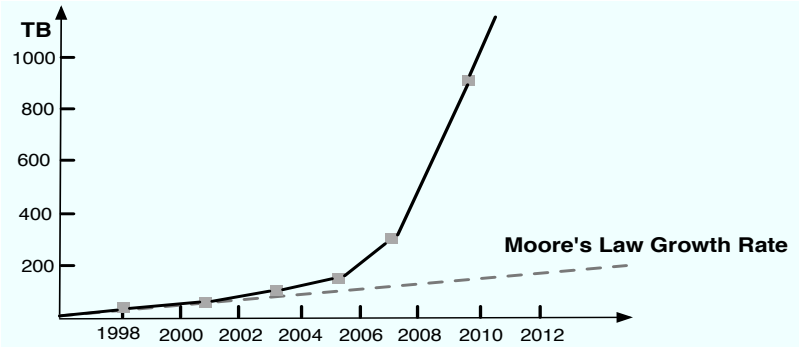


FIGURE 1.1: Projected average size of data warehouses [3].

Due to the advance in data generation and collection technologies, the increased popularity of internet and the invention of cloud computing, the amount of available data has exploded in recent years. Figure 1.1 shows the projected sizes of data warehouses over the coming years. As pointed out by the Vice President of *Google*, Marissa Mayer, "data explosion is bigger than Moore's law." According to various studies, the amount of available data has grown over 15-fold in the past few years, and the annual data growth rate is projected to increase to 45-fold by 2020 [1]. In this era of data explosion, data is no longer static. Data is dynamically updated all the time: new records are inserted, obsolete records are removed and records containing error are corrected. When the underlying data is updated, the corresponding emerging patterns also need to be updated. The most straightforward approach is to re-discover the emerging patterns. However, as graphically illustrated in Figure 1.2, the size of data affected by the updates is usually much smaller than the overall data size, and the original and the updated emerging pattern spaces have very large overlap. Moreover, the re-discovery approach also leads to large computational overheads, and it is likely to be practically infeasible. A more practical solution is to incrementally maintain the emerging patterns to reflect updates in the underlying data. In real-life applications, incremental maintenance of emerging patterns has been employed for real-time monitoring of critical assets [11] and diagnosis of medical conditions [14]. This chapter discusses how emerging patterns can be incrementally maintained when the pattern space is concisely represented by its *border*.

Incremental maintenance of emerging patterns is also a useful tool for interactive mining applications. One potential application is to answer hypothetical queries, including, the "*what if*" queries. Data analyzers are often interested in finding out "*what*" might happen to the discovered emerging

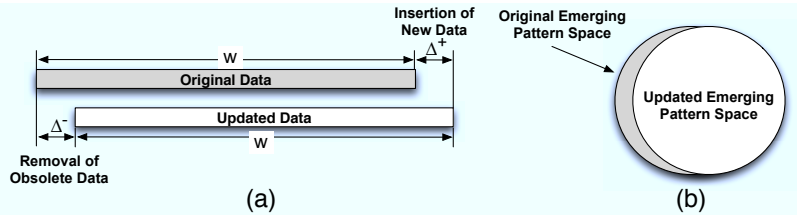


FIGURE 1.2: (a) A graphical illustration of dynamic data updates. (b) Overlaps between the original and the updated emerging pattern spaces.

patterns "*if*": some new transactions were inserted to the dataset, some existing transactions were removed, some existing transactions were replaced, or some sets of items are included or excluded, etc. Useful insights can often be discovered with interactive hypothetical queries.

Figure 1.3 (a) illustrates the naïve approach to answer *what if* hypothetical queries. First, the naïve approach requires the re-discovery of emerging patterns, which involves large amount of repeated computation effort. Moreover, to answer *what if* queries, the naïve approach needs to compare the original and updated pattern spaces. Since the size of emerging pattern spaces is usually very large, the comparison is computationally expensive.

As illustrated by Figure 1.3 (b), incremental maintenance of emerging patterns can be used to efficiently support the query answering process, without using the pattern re-generation and comparison steps of the naïve approach.

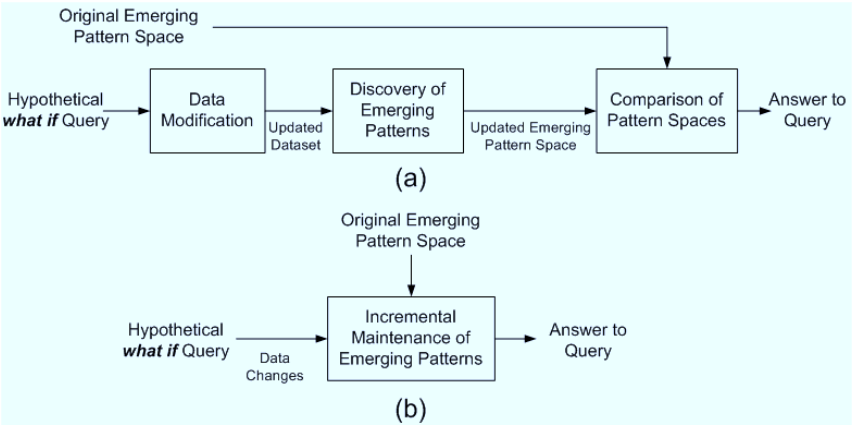


FIGURE 1.3: (a) The naïve approach and (b) the incremental maintenance approach to answer hypothetical *what if* queries.

1.2 Problem Definition & Challenges

Following the definitions in Chapter **??**, given two contrasting datasets D_P and D_N and a *support ratio threshold*, σ_r, a pattern (or an itemset) P is an "emerging pattern" for the *positive* dataset D_P if and only if $SuppRatio(P, D_P, D_N) = \frac{Supp(P,D_P)}{Supp(P,D_N)} \geq \sigma_r$, where $Supp(P, D_x)$ denotes the support of P in dataset D_x. In some applications, the differences between the contrasting datasets are measured based on *support delta*. In this case, a pattern P is an emerging pattern for the *positive* dataset if and only if $SuppDelta(P, D_P, D_N) = Supp(P, D_P) - Supp(P, D_N) \geq \sigma_d$, where σ_d is the *minimum support delta threshold*. Given σ_r or σ_d, the "space of emerging pattern" or "emerging pattern space", denoted as $EP(D_P, D_N, \sigma_r)$ or $EP(D_P, D_N, \sigma_d)$, is the set of all valid emerging patterns. Since the *support ratio* threshold, σ_r, is more commonly used, for ease of discussion, in this chapter we will assume σ_r is used unless otherwise stated.

Types of Updates

Insertion of new instances is an update where new *positive* instances, Δ_P, and/or new *negative* instances, Δ_N, are inserted into the original contrasting datasets. Suppose the original contrasting datasets are D_P and D_N. The updated datasets then are $D'_P = D_P \cup \Delta_P$ and $D'_N = D_N \cup \Delta_N$, and $|D'_P| = |D_P| + |\Delta_P|$ and $|D'_N| = |D_N| + |\Delta_N|$. (We assume that $D_P \cap \Delta_P = \emptyset$ and $D_N \cap \Delta_N = \emptyset$.) Insertion of new instances is the most common data management operation. It allows new data to be included in the data analysis.

Deletion of existing instances is an update where existing *positive* instances, Δ_P, and/or *negative* instances, Δ_N, are removed from the current contrasting datasets. Suppose the original contrasting datasets are D_P and D_N. The updated datasets then are $D'_P = D_P - \Delta_P$ and $D'_N = D_N - \Delta_N$, and $|D'_P| = |D_P| - |\Delta_P|$ and $|D'_N| = |D_N| - |\Delta_N|$. (We assume that $\Delta_P \subset D_P$ and $\Delta_N \subset D_N$.) Deletion of existing instances is a data management operation that allows obsolete and invalid instances to be removed. It ensures the generated emerging patterns are not undesirably influenced by out-of-date or invalid instances that contain error.

> When the original contrasting datasets, D_P and D_N, are updated into D'_P and D'_N by inserting or removing instances, the task of incremental maintenance is to obtain the updated emerging pattern space, $EP(D'_P, D'_N, \sigma_r)$, by updating the original pattern space, $EP(D_P, D_N, \sigma_r)$.

Expansion of query item space is an update where new items are included in the existing "*query item space*".

Definition 1 *Given the positive and negative datasets, D_P and D_N, let $\mathcal{I} = \{i_1, i_2, \cdots\}$ be the "complete item space" that includes all the items in D_P and D_N. The "query item space", denoted as $\mathcal{I}_Q$, defines an item subspace where $\mathcal{I}_Q \subseteq \mathcal{I}$. In the context of $\mathcal{I}_Q$, only items in $\mathcal{I}_Q$ will be considered for the formation of emerging patterns, and all other items will be ignored.*

In the context of *query item space*, $\mathcal{I}_Q$, the emerging pattern space, $EP_{\mathcal{I}_Q}(D_P, D_N, \sigma_r)$, is defined as the set of emerging patterns P that $P \subseteq \mathcal{I}_Q$.

Expansion of *query item space* is an update where the existing *query item space*, $\mathcal{I}_Q$, expands with new items $\mathcal{I}_\Delta$. $\mathcal{I}_\Delta \neq \emptyset$ and $\mathcal{I}_\Delta \subset \mathcal{I}$. The updated *Query Item Space* becomes $\mathcal{I}'_Q = \mathcal{I}_Q \cup \mathcal{I}_\Delta$. Therefore, $\mathcal{I}_Q \subset \mathcal{I}'_Q \subseteq \mathcal{I}$.

Shrinkage of query item space is an update where the existing *query item space*, $\mathcal{I}_Q$, shrinks by removing items $\mathcal{I}_\Delta$. $\mathcal{I}_\Delta \neq \emptyset$ and $\mathcal{I}_\Delta \subset \mathcal{I}_Q$. The updated *query item space* becomes $\mathcal{I}'_Q = \mathcal{I}_Q - \mathcal{I}_\Delta$. This implies that $\mathcal{I}'_Q \subset \mathcal{I}_Q$.

When the original *query item space*, $\mathcal{I}_Q$, expands or shrinks into $\mathcal{I}'_Q$, the task of incremental maintenance is to obtain the updated emerging pattern space, $EP_{\mathcal{I}'_Q}(D_P, D_N, \sigma_r)$, by updating the original pattern space, $EP_{\mathcal{I}_Q}(D_P, D_N, \sigma_r)$.

We note that, for the insertion and removal of instance updates, the *query item space* is assumed to remain unchanged.

1.2.1 Potential Challenges

Data updates may invalidate existing emerging patterns and introduce new emerging patterns. As a result, the incremental maintenance of emerging patterns consists of two main computational tasks: to update existing patterns and to generate new emerging patterns.

To update the existing emerging patterns, the naïve way is to scan through all the existing patterns to find out which patterns are affected by the data updates and then update the patterns accordingly. Suppose the dataset is incrementally updated with m new instances. The computational complexity of the naïve approach is $O(N_{EP} \times m)$, where N_{EP} refers to the number of the existing emerging patterns. Since N_{EP} can often be very large, the naïve approach is often computationally too expensive to be practically feasible. Therefore, how to update the existing patterns effectively is one of the major challenges in incremental maintenance of emerging patterns.

The generation of new emerging patterns is also technically challenging. In theory, the number of possible candidates for the new emerging patterns equals to $(2^n - N_{EP})$, where n is the total number of items (attribute-value pairs) and N_{EP} is the number of existing emerging patterns. In most applications, the number of items can be around $500 \sim 1,000$. Suppose there are $1,000$ items

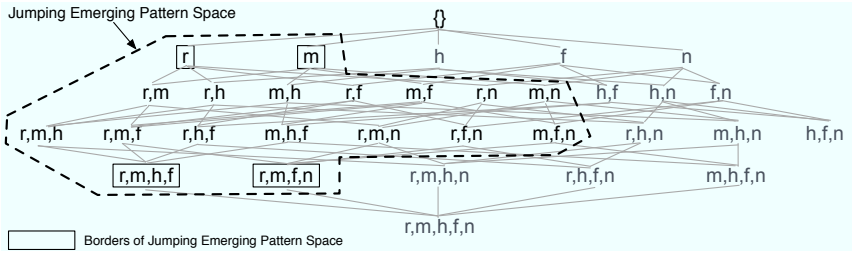
in the contrasting datasets. The potential search space for the new emerging patterns will consist of over 10^{300} candidates. Given such a large search space, effective techniques are needed to extract the new emerging patterns.

In the subsequent sections, we will investigate how the emerging patterns can be effectively maintained by concisely representing the pattern space with its *border*. The concept of *border* was first introduced in [7] as a concise representation of emerging patterns.

1.3 Concise Representation of Pattern Space: The *Border*

ID	Outlook	Temp.	Humidity	Windy	AvoidOutdoor
1	rain	mild	high	false	Yes
2	rain	mild	normal	false	Yes
3	sunny	Hot	high	false	No
4	sunny	Hot	normal	true	No

(a)



(b)

FIGURE 1.4: (a) Sample dataset; "AvoidOutdoor" is the class label; "Yes" indicates the positive class, and "No" indicates the negative class. (b) Emerging pattern space and its border. Here, we represent the attribute values using their first letters. The JEP space for the sample dataset, marked with a dotted line, includes 18 patterns; it can be concisely represented by its *border*, which consists of only 4 patterns highlighted in solid boxes.

This section discusses how the space of emerging patterns can be concisely represented. To illustrate the concept, we focus on a special type of emerging patterns — the "Jumping Emerging Patterns" (JEP) .

Definition 2 *Given the positive and negative contrasting datasets D_P and D_N, a pattern P is a jumping emerging pattern for the positive dataset if and only if $Supp(P, D_P) > 0$ and $Supp(P, D_N) = 0$, or equivalently, $SuppRatio(P, D_P, D_N) = \infty$.*

A jumping emerging pattern for the *positive* dataset is a pattern P that appears only in *positive* instances and does not appear in any *negative* instances. For ease of discussion, when we mention "jumping emerging pattern" in subsequent discussion, it refers to the jumping emerging pattern for the *positive* dataset. The "space of jumping emerging pattern", in short *JEP* space, is then the pattern space that includes all valid jumping emerging patterns. We denote the space of jumping emerging patterns as $JEP(D_P, D_N)$. Jumping emerging patterns are most commonly used in the applications of classification, because they are often the most discriminative emerging patterns.

Figure 1.4 (a) shows a sample dataset on the relationship between the weather conditions and whether one should avoid staying outdoor. ("Yes" and "No" respectively denote the positive and negative classes.) Observe that, for this simple dataset consisting of only 4 instances and 4 attributes, the corresponding *JEP* space consists of 18 patterns. In the theoretical worse case, the size of the *JEP* space can grow exponentially with respect to the number of attributes. Updating and maintaining the *JEP* space can be computationally expensive. Luckily, the *JEP* space has a useful property: it is a convex space.

Definition 3 *A pattern space S is a convex space if, for all X, $Y \in S$, where $X \subseteq Y$, it is the case that $Z \in S$ whenever Z satisfies $X \subseteq Z \subseteq Y$.*

The convexity of the *JEP* space implies that, given any jumping emerging patterns X and Y, where $X \subseteq Y$, any pattern Z such that $X \subseteq Z \subseteq Y$ is also a jumping emerging pattern. This property forms the theoretical foundation for the concise representation of the *JEP* space.

Given a *JEP* space, we define the most general emerging patterns as the *left bound* of the pattern space, denoted as $\mathcal{L}$; and we define the most specific emerging patterns as the *right bound* of the pattern space, denoted as $\mathcal{R}$. The combination of the *left* and *right* bounds forms the *border* of the *JEP* space, denoted as $< \mathcal{L}, \mathcal{R} >$.

Based on the convexity of the *JEP* space, we observe the following four properties of its *border*. (a) Both the *left* and *right* bounds, $\mathcal{L}$ and $\mathcal{R}$, are *antichains*. Here, an *antichain* refers to a collection of patterns in which it is true for any two patterns X and Y that $X \not\subseteq Y$ and $Y \not\subseteq X$. (b) For each pattern X in $\mathcal{L}$, there exists at least one pattern Y in $\mathcal{R}$ such that $X \subseteq Y$. (c) For each pattern Y in $\mathcal{R}$, there exists at least one pattern X in $\mathcal{L}$ such that $X \subseteq Y$. (d) Most importantly, all patterns in the *JEP* space are completely covered by $\mathcal{L}$ and $\mathcal{R}$.

As a result, the *JEP* space can be concisely represented by its *border*, $< \mathcal{L}, \mathcal{R} >$. For the sample dataset of Figure 1.4, the *JEP* space (consisting of 18 patterns) can be concisely summarized by its *border*, which consists of only 4 patterns: $\mathcal{L} = \{\{r\}, \{m\}\}$ and $\mathcal{R} = \{\{r, m, h, f\}, \{r, m, n, f\}\}$.

Moreover, based on the notations of the *left* and *right* bounds, $\mathcal{L}$ and $\mathcal{R}$, we can re-interpret the *JEP* space as a collection of patterns that are supersets of some patterns in $\mathcal{L}$ and subsets of some patterns in $\mathcal{R}$. The *JEP* space can be expressed as $[\mathcal{L}, \mathcal{R}] = \{Z | \exists X \in \mathcal{L}, \exists Y \in \mathcal{R}$ *such that* $X \subseteq Z \subseteq Y\}$. Note

that notations $< \mathcal{L}, \mathcal{R} >$ and $[\mathcal{L}, \mathcal{R}]$ are different from each other. $< \mathcal{L}, \mathcal{R} >$ denotes the *border* of the *JEP* space, which consists of only the *left* bound, $\mathcal{L}$, and the *right* bound, $\mathcal{R}$. On the other hand, $[\mathcal{L}, \mathcal{R}]$ refers to the entire *JEP* space that are bounded by the *border*, $< \mathcal{L}, \mathcal{R} >$.

Recall that a jumping emerging pattern is a pattern that occurs only in the *positive* instances, D_P, but not in the *negative* instances, D_N. Therefore, one can obtain all the jumping emerging patterns by subtracting all patterns, which have *non-zero* support in D_N, from the collection of patterns that have *non-zero* support in D_P. This idea is graphically demonstrated in Figure 1.5. Based on the concept of *border*, the collection of non–zero support patterns in D_P can be expressed as $[\{\emptyset\}, \mathcal{R}_\mathcal{P}]$, where $\mathcal{R}_\mathcal{P}$ is the *right* bound of the pattern space. Similarly, the collection of non-zero support patterns in D_N can be expressed as as $[\{\emptyset\}, \mathcal{R}_\mathcal{N}]$, where $\mathcal{R}_\mathcal{N}$ is the *right* bound of the pattern space. As a result, the *JEP* space for the contrasting data D_P and D_N, denoted as $JEP(D_P, D_N)$, can be re-written as:

$$JEP(D_P, D_N) = [\{\emptyset\}, \mathcal{R}_\mathcal{P}] - [\{\emptyset\}, \mathcal{R}_\mathcal{N}] \tag{1.1}$$

This expression of the *JEP* space will be used in subsequent discussions. By concisely representing the *JEP* space with its *border*, one only needs to incrementally update the *border* patterns instead of the entire pattern space, which effectively reduces the computational complexity.

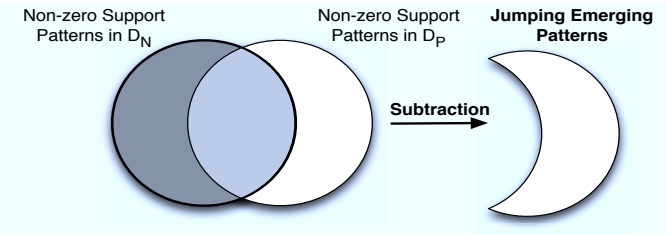


FIGURE 1.5: Pattern space subtraction to obtain the *JEP* space.

1.4 Maintenance of *Border*

This section investigates how the *border* of the *JEP* space can be incrementally maintained. First, Section 1.4.1 introduces some basic *border* operations, including *Subtraction*, *Union* and *Intersection*. The subsequent sections then demonstrate, based on the basic operations, how the *border* can be effectively maintained under various data updates. Section 1.4.2 addresses the insertion

of new instances, Section 1.4.3 the removal of existing instances, and Sections 1.4.4 and 1.4.5 the expansion and shrinkage of the *query item space*.

1.4.1 Basic *Border* Operations

***Border* Subtraction**: Recall that, the *JEP* space can be obtained by subtracting the non-zero support patterns in D_N, represented by $[\{\emptyset\}, \mathcal{R}_\mathcal{N}]$, from the set of non-zero support patterns in D_P, represented by $[\{\emptyset\}, \mathcal{R}_\mathcal{P}]$; see Figure 1.5. That is, $JEP(D_P, D_N) = [\{\emptyset\}, \mathcal{R}_\mathcal{P}] - [\{\emptyset\}, \mathcal{R}_\mathcal{N}]$ (Equation 1.1). Based on this formula, the *border* subtraction operation generates the *JEP* space with respect to the given $\mathcal{R}_\mathcal{P}$ and $\mathcal{R}_\mathcal{N}$. Figure 1.6 describes the detailed algorithm of the *border* subtraction operation, where $\mathcal{R}_\mathcal{P} = \{A_1, ..., A_k\}$. The subroutine of the operation, *BorderDiff*, is described in Figure 1.7.

Input: Two pattern spaces given by $< \{\emptyset\}, \{A_1, ..., A_k\} >$ and $< \{\emptyset\}, \mathcal{R}_\mathcal{N} >$
Output: $< \mathcal{L}, \mathcal{R} >$ such that $[\mathcal{L}, \mathcal{R}] = [\{\emptyset\}, \{A_1, ..., A_k\}] - [\{\emptyset\}, \mathcal{R}_\mathcal{N} >$
Method:
1: $\mathcal{L} \leftarrow \{\ \}$, $\mathcal{R} \leftarrow \{\ \}$;
2: **for** j from 1 to k do
3: *border* = *BorderDiff*$(< \{\emptyset\}, \{A_j\} >, < \{\emptyset\}, \mathcal{R}_\mathcal{N} >)$;
4: $\mathcal{L} = \mathcal{L} \cup$ left bound of *border*;
5: $\mathcal{R} = \mathcal{R} \cup$ right bound of *border*;
6: **end for**
7: **return** $< \mathcal{L}, \mathcal{R} >$;

FIGURE 1.6: Operation *Border* Subtraction.

Input: Two pattern spaces given by $< \{\emptyset\}, \{U\} >$ and $< \{\emptyset\}, \{S_1, ..., S_k\} >$
Output: $< \mathcal{L}, \mathcal{R} >$ such that $[\mathcal{L}, \mathcal{R}] = [\{\emptyset\}, \{U\}] - [\{\emptyset\}, \{S_1, ..., S_k\}]$
Method:
1: $\mathcal{L} = \{\{x\} | x \in U - S_1\}$;
2: **for** i from 2 to k do
3: $\mathcal{L} \leftarrow \{X \cup \{x\} | X \in \mathcal{L}, x \in U - S_i\}$;
4: remove patterns in $\mathcal{L}$ that are not most general;
5: **end for**
6: **return** $< \mathcal{L}, \{U\} >$;

FIGURE 1.7: Procedure *BorderDiff*.

***Border* Union**: Suppose D_N is a set of *negative* instances, D_P is a set of *positive* instances, and D_P is partitioned into two sets, D_{P_1} and D_{P_2}. It is interesting to know how the *JEP* space $JEP(D_P, D_N)$ is related to the *JEP*

spaces $JEP(D_{P_1}, D_N)$ and $JEP(D_{P_2}, D_N)$. We have the following answer to that question:

Fact 4 *Given a negative dataset D_N and a positive dataset D_P and a partition of D_P into D_{P_1} and D_{P_2}, we have $JEP(D_P, D_N) = JEP(D_{P_1}, D_N) \cup JEP(D_{P_2}, D_N)$. This fact is illustrated in Figure 1.8.*

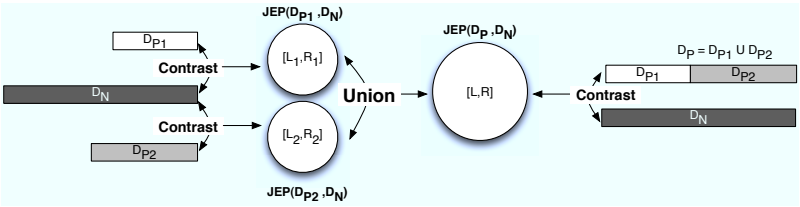


FIGURE 1.8: Union of two *JEP* spaces: given D_N and $D_P = D_{P_1} \cup D_{P_2}$, $JEP(D_P, D_N) = JEP(D_{P_1}, D_N) \cup JEP(D_{P_2}, D_N)$.

In the notion of *border*, suppose $JEP(D_P, D_N) = [\mathcal{L}, \mathcal{R}]$, $JEP(D_{P_1}, D_N) = [\mathcal{L}_1, \mathcal{R}_1]$ and $JEP(D_{P_1}, D_N) = [\mathcal{L}_2, \mathcal{R}_2]$. Then, $[\mathcal{L}, \mathcal{R}] = [\mathcal{L}_1, \mathcal{R}_1] \cup [\mathcal{L}_2, \mathcal{R}_2]$. Based on Fact 4, the *border* union operation in Figure 1.9 obtains the *JEP* space *border* $< \mathcal{L}, \mathcal{R} >$ from the *borders* $< \mathcal{L}_1, \mathcal{R}_1 >$ and $< \mathcal{L}_2, \mathcal{R}_2 >$.

Input: $< \mathcal{L}_1, \mathcal{R}_1 >$ representing $JEP(D_{P_1}, D_N)$ and $< \mathcal{L}_2, \mathcal{R}_2 >$ representing $JEP(D_{P_2}, D_N)$, for some datasets D_{P_1}, D_{P_2}, and D_N
Output: $< \mathcal{L}, \mathcal{R} >$ representing $JEP(D_{P_1} \cup D_{P_2}, D_N)$
% $< \mathcal{L}, \mathcal{R} >$ satisfies $[\mathcal{L}, \mathcal{R}] = [\mathcal{L}_1, \mathcal{R}_1] \cup [\mathcal{L}_2, \mathcal{R}_2]$
Method:
1: $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2$;
2: $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$;
3: remove patterns in $\mathcal{L}$ that are not most general;
4: remove patterns in $\mathcal{R}$ that are not most specific;
return $< \mathcal{L}, \mathcal{R} >$;

FIGURE 1.9: Operation *Border* Union.

***Border* Intersection**: The *border* intersection operation addresses the opposite scenario compared with the *border* union operation. We have the following relationship between the underlying JEP spaces.

Fact 5 *Suppose we have a positive dataset D_P and a negative dataset D_N, and a partition of D_N into D_{N_1} and D_{N_2}. Then we have $JEP(D_P, D_N) = JEP(D_P, D_{N_1}) \cap JEP(D_P, D_{N_2})$. This is illustrated in Figure 1.10.*

In the notion of *border*, suppose $JEP(D_P, D_N) = [\mathcal{L}, \mathcal{R}]$, $JEP(D_P, D_{N_1}) =$

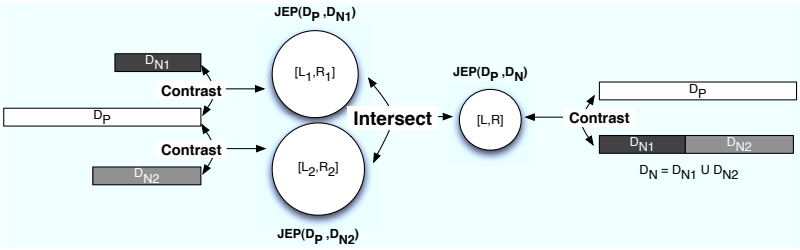


FIGURE 1.10: Intersection of two *JEP* spaces: given D_P and $D_N = D_{N_1} \cup D_{N_2}$, $JEP(D_P, D_N) = JEP(D_P, D_{N_1}) \cap JEP(D_P, D_{N_2})$.

$[\mathcal{L}_1, \mathcal{R}_1]$ and $JEP(D_P, D_{N_2}) = [\mathcal{L}_2, \mathcal{R}_2]$. Then, $[\mathcal{L}, \mathcal{R}] = [\mathcal{L}_1, \mathcal{R}_1] \cap [\mathcal{L}_2, \mathcal{R}_2]$. The *border* intersection operation in Figure 1.11 gives the *JEP* space *border* $< \mathcal{L}, \mathcal{R} >$ based on the *borders* $< \mathcal{L}_1, \mathcal{R}_1 >$ and $< \mathcal{L}_2, \mathcal{R}_2 >$.

Input: $< \mathcal{L}_1, \mathcal{R}_1 >$ representing $JEP(D_P, D_{N_1})$ and $< \mathcal{L}_2, \mathcal{R}_2 >$ representing $JEP(D_P, D_{N_2})$, for some datasets D_P, D_{N_1}, and D_{N_2}
Output: $< \mathcal{L}, \mathcal{R} >$ representing $JEP(D_P, D_{N_1} \cup D_{N_2})$
% $< \mathcal{L}, \mathcal{R} >$ satisfies $[\mathcal{L}, \mathcal{R}] = [\mathcal{L}_1, \mathcal{R}_1] \cup [\mathcal{L}_2, \mathcal{R}_2]$
Method:
1: $\mathcal{R} = \mathcal{R}_1 \cap \mathcal{R}_2$;
2: $\mathcal{L} = \{A \cup B | A \in \mathcal{L}_1, B \in \mathcal{L}_2\}$;
3: remove patterns in $\mathcal{L}$ that are not most general;
4: remove patterns P in $\mathcal{L}$ if $\nexists Q \in \mathcal{R}$ such that $P \subseteq Q$;
return $< \mathcal{L}, \mathcal{R} >$;

FIGURE 1.11: Operation *Border* Intersection.

1.4.2 Insertion of New Instances

Suppose we have D_P and D_N as the original contrasting *positive* and *negative* datasets, and the corresponding non-zero support pattern spaces are $[\{\emptyset\}, \mathcal{R}_\mathcal{P}]$ and $[\{\emptyset\}, \mathcal{R}_\mathcal{N}]$. Thus, the original *JEP* space can be expressed as $JEP(D_P, D_N) = [\{\emptyset\}, \mathcal{R}_\mathcal{P}] - [\{\emptyset\}, \mathcal{R}_\mathcal{N}]$. When new instances are inserted, there can be two scenarios, where new *positive* instances are inserted or new *negative* instances are inserted. We discuss these two scenarios separately.

Insertion of New *Positive* Instances: Suppose a set of new *positive* instances, Δ_P, is inserted, where $\Delta_P \cap D_P = \emptyset$. The updated set of *positive* instances is $(D_P \cup \Delta_P)$. Let $[\{\emptyset\}, \mathcal{R}_P^\Delta]$ denote the non-zero support pattern space of Δ_P. The updated *JEP* space with respect to $(D_P \cup \Delta_P)$ and D_N can be precisely defined by the following formula.

$$
\begin{aligned}
& JEP((D_P \cup \Delta_P), D_N) \\
= \; & ([\{\emptyset\}, \mathcal{R}_{\mathcal{P}}] \cup [\{\emptyset\}, \mathcal{R}_P^{\Delta}]) - [\{\emptyset\}, \mathcal{R}_{\mathcal{N}}] \\
= \; & ([\{\emptyset\}, \mathcal{R}_{\mathcal{P}}] - [\{\emptyset\}, \mathcal{R}_{\mathcal{N}}]) \cup ([\{\emptyset\}, \mathcal{R}_P^{\Delta}] - [\{\emptyset\}, \mathcal{R}_{\mathcal{N}}]) \\
= \; & JEP(D_P, D_N) \cup JEP(\Delta_P, D_N) \qquad (1.2)
\end{aligned}
$$

By Equation 1.2, the updated *JEP* space, $JEP((D_P \cup \Delta_P), D_N)$, is the union of the original *JEP* space, $JEP(D_P, D_N)$, and the *JEP* space contrasting Δ_P and D_N, $JEP(\Delta_P, D_N)$. Let $< \mathcal{L}, \mathcal{R} >$ be the *border* of the original *JEP* space. We can obtain the *border* of the updated *JEP* space in two steps:

Step 1. Discover the border of the *JEP* space contrasting Δ_P and D_N. Let $< \mathcal{L}', \mathcal{R}' >$ denote the resulting *border*.
Step 2. Apply the *border* union operation (Figure 1.9) on $< \mathcal{L}, \mathcal{R} >$ and $< \mathcal{L}', \mathcal{R}' >$.

Insertion of New *Negative* Instances: Suppose a set of new *negative* instances, Δ_N, is inserted, where $\Delta_N \cap D_N = \emptyset$. Thus, the updated *negative* instances are $(D_N \cup \Delta_N)$. Let $[\{\emptyset\}, \mathcal{R}_N^{\Delta}]$ denote the non-zero support pattern space of Δ_N. The updated *JEP* space with respect to D_P and $(D_N \cup \Delta_N)$ can be precisely defined as:

$$
\begin{aligned}
& JEP(D_P, (D_N \cup \Delta_N)) \\
= \; & [\{\emptyset\}, \mathcal{R}_{\mathcal{P}}] - ([\{\emptyset\}, \mathcal{R}_{\mathcal{N}}] \cup [\{\emptyset\}, \mathcal{R}_N^{\Delta}) \\
= \; & ([\{\emptyset\}, \mathcal{R}_{\mathcal{P}}] - [\{\emptyset\}, \mathcal{R}_{\mathcal{N}}]) \cap ([\{\emptyset\}, \mathcal{R}_P] - [\{\emptyset\}, \mathcal{R}_N^{\Delta}]) \\
= \; & JEP(D_P, D_N) \cap JEP(D_P, \Delta_N) \qquad (1.3)
\end{aligned}
$$

We observe that, upon insertion of *negative* instances, Δ_N, the updated *JEP* space, $JEP(D_P, (D_N \cup \Delta_N))$, can be expressed as the intersaction of the original *JEP* space, $JEP(D_P, D_N)$, and the *JEP* space contrasting D_P and Δ_N, $JEP(D_P, \Delta_N)$. Let $< \mathcal{L}, \mathcal{R} >$ be the *border* of the original *JEP* space. We can obtain the *border* of the updated *JEP* space in two steps:

Step 1. Discover the border of the *JEP* space contrasting D_P and Δ_N. Let $< \mathcal{L}', \mathcal{R}' >$ denote the resulting *border*.
Step 2. Apply the intersection operation (Figure 1.11) on $< \mathcal{L}, \mathcal{R} >$ and $< \mathcal{L}', \mathcal{R}' >$.

1.4.3 Removal of Existing Instances

When removing existing instances, there are also two scenarios: removing existing *positive* instances or removing existing *negative* instances. Different

from the insertion case, the maintenance of the *JEP* space under the two removal scenarios is very distinct from each other.

Removal of Existing *Positive* Instances: Suppose a set of *positive* instances, Δ_P, is removed from the original data D_P. The *JEP* space will shrink upon the removal. The updated *JEP* space can be obtained by removing all non-zero support patterns in Δ_P. Let $[\{\emptyset\}, \mathcal{R}_P^{\Delta}]$ denote the non-zero support pattern space of Δ_P. The updated *JEP* space with respect to $(D_P - \Delta_P)$ and D_N is:

$$JEP((D_P - \Delta_P), D_N) = JEP(D_P, D_N) - [\{\emptyset\}, \mathcal{R}_P^{\Delta}] \tag{1.4}$$

Let $< \mathcal{L}, \mathcal{R} >$ be the *border* of the original *JEP* space, $JEP(D_P, D_N)$. The subtraction in Equation 1.4 can be performed in the following steps:

Step 1. Discover the *border* of the non-zero support pattern space of $(D_P - \Delta_P)$. Let $< \{\emptyset\}, \mathcal{R}'_P >$ denote the resulting *border*.
Step 2. Let $\mathcal{L}' = \mathcal{L}$.
Step 3. Remove all patterns P in $\mathcal{L}'$ satisfying $\nexists Q \in \mathcal{R}'_P$ satisfying $P \subseteq Q$.

The resulting $< \mathcal{L}', \mathcal{R}'_P >$ is then the *border* of the the updated *JEP* space.

Removal of Existing *Negative* Instances: Suppose a set of *negative* instance, Δ_N, is removed. Upon the removal of existing *negative* instances, the *JEP* space will expand, as new emerging patterns will appear [13]. Under this case, the maintenance of the *JEP* space is much more complicated. We illustrate the process mathematically with the following formula. For ease of discussion, we denote $\mathcal{A} = [\{\emptyset\}, \mathcal{R}_P]$, the non-zero pattern space of the *positive* instances D_P, $\mathcal{B} = [\{\emptyset\}, \mathcal{R}'_N]$, the non-zero pattern space of the updated *negative* instances $(D_N - \Delta_N)$, and $\mathcal{C} = [\{\emptyset\}, \mathcal{R}_N^{\Delta}]$, the non-zero pattern space of the removed *negative* instances Δ_N.

$$\begin{aligned} & JEP(D_P, (D_N - \Delta_N)) \\ = \; & \mathcal{A} - \mathcal{B} = \mathcal{A} - \mathcal{B} - \mathcal{A} \cap \mathcal{B} \\ = \; & \mathcal{A} - \mathcal{B} - \mathcal{A} \cap \mathcal{B} - \mathcal{C} \cup \mathcal{A} \cap \mathcal{C} \\ = \; & (\mathcal{A} - \mathcal{B} \cup \mathcal{C}) \cup (\mathcal{A} \cap (\mathcal{C} - \mathcal{B})) \\ = \; & JEP(D_P, D_N) \cup (\mathcal{A} \cap (\mathcal{C} - \mathcal{B})) \end{aligned} \tag{1.5}$$

We observe that the first term of the union, $(\mathcal{A} - \mathcal{B} \cup \mathcal{C})$, is equivalent to the original *JEP* space, $JEP(D_P, D_N)$. The term $(\mathcal{C} - \mathcal{B}) = ([\{\emptyset\}, \mathcal{R}_N^{\Delta}] - [\{\emptyset\}, \mathcal{R}'_N])$ is then the *JEP* space contrasting Δ_N and D_N, $JEP(\Delta_N, D_N)$. Let $< \mathcal{L}, \mathcal{R} >$ be the *border* of the original *JEP* space, $JEP(D_P, D_N)$. The *border* of the updated *JEP* space can be obtained as follows.

Step 1. Discover the *border* of $JEP(\Delta_N, D_N)$. Let $< \mathcal{L}', \mathcal{R}' >$ denote the resulting *border*.
Step 2. Remove all patterns P in $\mathcal{L}'$ that does not appear in D_P.
Step 3. Apply the *border* union operation on $< \mathcal{L}, \mathcal{R} >$ and $< \mathcal{L}', \mathcal{R}' >$.

1.4.4 Expansion of Query Item Space

Let D_P and D_N denote the *positive* and *negative* datasets respectively. Let $\mathcal{I}$ be the *complete item space* that includes all the items in D_P and D_N and $\mathcal{I}_Q$ be the original *query item space*. Suppose a new item $e \in \mathcal{I}$ is inserted in the *query item space*, $\mathcal{I}_Q$. The *query item space* expands. The updated *query item space* becomes $\mathcal{I}'_Q = \mathcal{I}_Q \cup \{e\}$.

Let $[\{\emptyset\}, \mathcal{R}_P]$ be the non-zero support patterns in D_P under the original *query item space*, $\mathcal{I}_Q$, and let $[\{\{e\}\}, \mathcal{R}'_P]$ be the non-zero support patterns in D_P containing item e. Under the updated updated *query item space*, $\mathcal{I}'_Q = \mathcal{I}_Q \cup \{e\}$, the non-zero support patterns in D_P can then be expressed as $[\{\emptyset\}, \mathcal{R}_P] \cup [\{\{e\}\}, \mathcal{R}'_P]$.

Similarly, let $[\{\emptyset\}, \mathcal{R}_N]$ be the non-zero support patterns in D_N under $\mathcal{I}_Q$, and let $[\{\{e\}\}, \mathcal{R}'_N]$ be the non-zero support patterns in D_N containing item e. Under the updated updated *query item space*, $\mathcal{I}'_Q = \mathcal{I}_Q \cup \{e\}$, the non-zero support patterns in D_N can then be expressed as $[\{\emptyset\}, \mathcal{R}_N] \cup [\{\{e\}\}, \mathcal{R}'_N]$.

Based on Equation 1.1, the updated *JEP* space for the expanded *query item space*, $\mathcal{I}'_Q$, can be expressed as:

$$
\begin{aligned}
& JEP_{\mathcal{I}'_Q}(D_P, D_N) \\
= \; & ([\{\emptyset\}, \mathcal{R}_P] \cup [\{\{e\}\}, \mathcal{R}'_P]) - ([\{\emptyset\}, \mathcal{R}_N] \cup [\{\{e\}\}, \mathcal{R}'_N]) \\
= \; & ([\{\emptyset\}, \mathcal{R}_P] - [\{\{e\}\}, \mathcal{R}'_N] - [\{\emptyset\}, \mathcal{R}_N]) \cup \\
& ([\{\{e\}\}, \mathcal{R}'_P] - [\{\emptyset\}, \mathcal{R}_N] - [\{\{e\}\}, \mathcal{R}'_N]) \\
= \; & ([\{\emptyset\}, \mathcal{R}_P] - [\{\emptyset\}, \mathcal{R}_N]) \cup ([\{\{e\}\}, \mathcal{R}'_P] - [\{\{e\}\}, \mathcal{R}'_N]) \\
= \; & JEP_{\mathcal{I}_Q}(D_P, D_N) \cup ([\{\{e\}\}, \mathcal{R}'_P] - [\{\{e\}\}, \mathcal{R}'_N]) \qquad (1.6)
\end{aligned}
$$

According to Equation 1.6, the updated *JEP* space can be obtained based on the *border*, $< \mathcal{L}, \mathcal{R} >$, of the original *JEP* space, and it can be achieved in two steps:

Step 1. Apply *border* subtraction operation to obtain the border of $[\{\{e\}\}, \mathcal{R}'_P] - [\{\{e\}\}, \mathcal{R}'_N]$. Let $< \mathcal{L}', \mathcal{R}' >$ be the resulting *border*.
Step 2. Apply the *border* union operation on $< \mathcal{L}, \mathcal{R} >$ and $< \mathcal{L}', \mathcal{R}' >$.

1.4.5 Shrinkage of Query Item Space

Shrinkage of *query item space* is the opposite operation of the expansion of *query item space*. Let $\mathcal{I}_Q$ be the original *query item space*. Suppose an item $e \in \mathcal{I}_Q$ is removed from $\mathcal{I}_Q$. The *query item space* shrinks, and the updated *query item space* becomes $\mathcal{I}'_Q = \mathcal{I}_Q - \{e\}$. Following the notations in Section 1.4.4, the updated *JEP* space for the shrunken *query item space* can then be formulated as:

$$
\begin{aligned}
& JEP_{\mathcal{I}'_Q}(D_P, D_N) \\
= \; & ([\{\emptyset\}, \mathcal{R}_P] - [\{\{e\}\}, \mathcal{R}'_P]) - ([\{\emptyset\}, \mathcal{R}_N] - [\{\{e\}\}, \mathcal{R}'_N]) \\
= \; & ([\{\emptyset\}, \mathcal{R}_P] - [\{\{e\}\}, \mathcal{R}'_P] - [\{\emptyset\}, \mathcal{R}_N]) \cup \\
& (([\{\emptyset\}, \mathcal{R}_P] - [\{\{e\}\}, \mathcal{R}'_P]) \cap [\{\{e\}\}, \mathcal{R}'_N]) \\
= \; & [\{\emptyset\}, \mathcal{R}_P] - [\{\emptyset\}, \mathcal{R}_N] - [\{\{e\}\}, \mathcal{R}'_P] \\
= \; & JEP_{\mathcal{I}_Q}(D_P, D_N) - [\{\{e\}\}, \mathcal{R}'_P] \qquad (1.7)
\end{aligned}
$$

By Equation 1.7, the updated *JEP* space can be obtained by removing all existing emerging patterns that contain item e. Suppose the *border* of the original JEP space is $< \mathcal{L}, \mathcal{R} >$. The *border* of the updated *JEP* space, $< \mathcal{L}', \mathcal{R}' >$, can be effectively obtained as follows:

Step 1. $\mathcal{L}' = \{P | P \in \mathcal{L}$ *and* $e \notin P\}$.
Step 2. $\mathcal{R}' = \{P | P \in \mathcal{R}$ *and* $e \notin P\}$.
Step 3. Remove all patterns P in $\mathcal{R}'$ that are not most specific.
Step 4. Remove all patterns P in $\mathcal{R}'$ that do not contain any pattern in $\mathcal{L}'$.

1.5 Related Work

Incremental maintenance of emerging patterns has not received as much research attention as its discovery task. However, considerable amount of research effort has been committed to the incremental maintenance of *frequent patterns*. "Frequent Patterns" [**?**] are patterns that appear frequently in the data. The spaces of frequent patterns and emerging patterns share many similarities. For instance, both pattern spaces grow exponentially with the number of items, and both pattern spaces are convex [13, 12]. Therefore, ideas in frequent pattern maintenance can often be extended to the incremental maintenance of emerging patterns.

In the literature, the frequent pattern maintenance algorithms can be classified into four main categories: the 1) *Apriori-based* algorithms, 2) *Partition-based* algorithms, 3) *Prefix-tree-based* algorithms and 4) *Concise-representation-based* algorithms [9].

FUP [4] is the first *Apriori*-based maintenance algorithm. Inspired by Apriori [**?**], FUP updates the space of frequent patterns iteratively based on the candidate-generation-verification framework. The key technique of FUP is to make use of support information in previously discovered frequent patterns to reduce the number of candidate patterns. Since the performance of candidate-generation-verification based algorithms heavily depends on the size of the

candidate set, FUP outperforms Apriori. Similarly, the partition-based algorithm SWF [10] also employs the candidate-generation-verification framework. However, SWF applies different techniques to reduce the size of candidate set. SWF slices a dataset into several partitions and employs a filtering threshold in each partition to filter out unnecessary candidate patterns. Even with all the candidate reduction techniques, the candidate-generation-verification framework still leads to the enumeration of large number of unnecessary candidates. This greatly limits the performance of both *Apriori*-based and partition-based algorithms.

To address this shortcoming of the candidate-generation-verification framework, tree-based algorithms are proposed. Examples include FP-growth [**?**], a *prefix-tree* based algoirhtm, and Apriori-TFP [6], a *Total Support Tree* (*T-tree*) based algorithm. In [11] and [14], Apriori-TFP was extended to incrementally maintain emerging patterns. Apriori-TFP enumerates and generates patterns with the *T-tree*. A *T-tree* is basically a *set-enumeration tree*; it ensures a complete and non-redundant enumeration of patterns. *Apriori-TFP* (Total From Partial) efficiently constructs the *Total Support Tree*, *T-tree*, based on the *Partial Support Tree*, *P-tree*. *P-tree* is a summarized set-enumeration tree with partial support information of patterns [11]. Benefiting from the "Total From Partial" idea, Apriori-TFP achieves a good balance between time and memory efficiency. However, tree based algorithms still suffer from the undesirably large size of the pattern space.

To break this bottleneck, concise representations of the frequent pattern space are proposed. The commonly used representations include "maximal patterns" [2], "closed patterns" [**?**] and "equivalence classes" [8]. Algorithms, such as Moment [5], ZIGZAG [15] and PSM (Pattern Space Maintainer) [8] have been proposed to maintain these concise representations.

Concise representations are also employed to summarize the pattern space of emerging patterns. The concept of *border* was first introduced in [7] to summarize the the emerging pattern space with its *left* and *right* bounds — its border. Based the concept of *border*, an effective method for incremental maintenance of emerging patterns is proposed in [13]. This method has been discussed in Section 1.3 and 1.4. Reference [**?**] further extended the method to address the maintenance of emerging patterns for data streams. Since a data stream is a continuous stream of data, emerging pattern discovery in data streams, as defined in [**?**], is to contrast the updated data with the obsolete data. As illustrated in Figure 1.12, the contrasting *positive* dataset, D_P, refers to the relatively newer segment of the data stream, and the contrasting *negative* dataset, D_N, refers to the more obsolete segment. Suppose the data stream is updated in a sliding window manner. We can observe from Figure 1.12 that: newly updated data, Δ_P^+, will be included in the updated *positive* dataset, D'_P; some originally *positive* data, Δ_P^-, will become obsolete and will be converted into the *negative* dataset, D'_N; and, according to the concept of sliding window, the most obsolete data, Δ_N^-, will be removed.

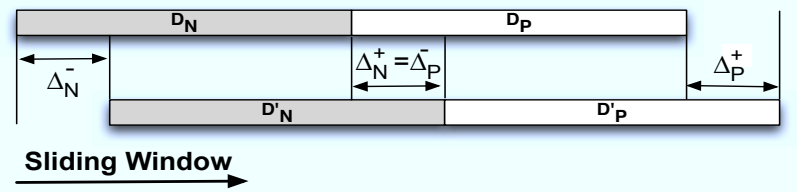


FIGURE 1.12: Maintenance of emerging patterns over data streams. Note: $\Delta_P^+ = \Delta_P^- = \Delta_N^+ = \Delta_N^-$

1.6 Closing Remarks

In the current era of data explosion, data is dynamically updated all the time. In addition, data is also often modified to perform interactive mining. Regenerating emerging patterns every time when the underlying data is updated or modified involves large amount of redundancy and is practically infeasible. The practical solution is to incrementally maintain the discovered emerging patterns.

The pattern space of emerging patterns is usually very large. Updating every single pattern in the pattern space can be computationally expensive. This bottleneck can be addressed by summarizing the emerging pattern space with its concise representation. Compared with the entire pattern space, the concise representation consists of a much smaller number of patterns, thus it can be maintained in a much lower computational cost.

To illustrate the concept, we focused on the jumping emerging patterns — the most discriminative emerging patterns. In Section 1.3, we discussed how the jumping emerging pattern space can be summarized with its concise representation, its *border*. In Section 1.4, we further discussed how the *border* of the jumping emerging pattern space can be effectively maintained under various data updates. The types of data updates that we have addressed include: insertion of new instances, removal of existing instances, expansion and shrinkage of the *query item space*.

Bibliography

[1] AMEinfo. Website: http://www.ameinfo.com/231603.html.

[2] Roberto J. Bayardo. Efficiently mining long patterns from databases. In *ACM SIGMOD International Conference on Management of Data*, pages 85–93, 1998.

[3] BeyeNetwork. Website: http://www.b-eye-network.com/view/7188.

[4] David Wai-Lok Cheung, Jiawei Han, Vincent T. Y. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: an incremental updating technique. In *Twelfth International Conference On Data Engineering (ICDE)*, pages 106–114, 1996.

[5] Yun Chi, Haixun Wang, Philip S. Yu, and Richard R. Muntz. Catch the moment: maintaining closed frequent itemsets over a data stream sliding window. *Knowledge and Information Systems*, 10(3):265–294, 2006.

[6] Frans Coenen, Paul H. Leng, and Shakil Ahmed. Data structure for association rule mining: T-trees and p-trees. *IEEE Transaction on Knowledge and Data Engineering*, 16(6):774–778, 2004.

[7] Guozhu Dong and Jinyan Li. Efficient mining of emerging patterns: Discovering trends and differences. In *ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 43–52, 1999.

[8] Mengling Feng, Guozhu Dong, Jinyan Li, Yap-Peng Tan, and Limsoon Wong. Pattern space maintenance for data updates and interactive mining. *Computational Intelligence*, 26(3):282–317, 2010.

[9] Mengling Feng, Jinyan Li, Guozhu Dong, and Limsoon Wong. *Maintenance of Frequent Patterns: A Survey*. 2009.

[10] Chang-Hung Lee, Cheng-Ru Lin, and Ming-Syan Chen. Sliding window filtering: an efficient method for incremental mining on a time-variant database. *Information Systems*, 30(3):227–244, 2005.

[11] Jong Bum Lee, Minghao Piao, and Keun Ho Ryu. Incremental emerging patterns mining for identifying safe and non-safe power load lines. In *10th IEEE International Conference on Computer and Information Technology*, pages 1424–1429, 2010.

[12] Haiquan Li, Jinyan Li, Limsoon Wong, Mengling Feng, and Yap-Peng Tan. Relative risk and odds ratio: a data mining perspective. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 368–377, 2005.

[13] Jinyan Li, Thomas Manoukian, Guozhu Dong, and Kotagiri Ramamohanarao. Incremental maintenance on the border of the space of emerging patterns. *Data Min. Knowl. Discov.*, 9(1):89–116, 2004.

[14] Jin Hyoung Park, Heon Gyu Lee, and Jong Heung Park. Real-time diagnosis system using incremental emerging pattern mining. In *5th International Conference on Ubiquitous Information Technologies and Applications (CUTE)*, pages 1–5, 2010.

[15] Adriano Veloso, Wagner Meira Jr., Márcio de Carvalho, Bruno Pôssas, Srinivasan Parthasarathy, and Mohammed Javeed Zaki. Mining frequent itemsets in evolving databases. In *Second SIAM International Conference on Data Mining*, 2002.

Index