13

Active Data Mining

Objectives:

- Introduce an active data mining paradigm that combines recent work in data mining with the rich literature on active database system.
- In this paradigm, data is continuously mined at a desired frequency.
- To be able to specify shape queries, we describe the constructs for defining shapes.
- Discuss how the shape predicates are used in a query construct to retrieve rules whose histories exhibit the desired trends.
- Describe how this query capability is integrated into a trigger system to realize an active mining system.
- The query language described provides the capability to discover interesting information by analyzing rules and their histories in novel ways.
- As the data mining technology is applied in the production mode, the need for active mining arises.

Abstract. We introduce an active data mining paradigm that combines the recent work in data mining with the rich literature on active database system. In this paradigm, data is continuously mined at a desired frequency. As rules are discovered, they are added to a rule base, and if they already exist, the history of the statistical parameter associated with the rules is updated. When the history starts exhibiting certain trends, specified as shape queries in the user-specified triggers, the triggers are fired and appropriate actions are initiated.

To be able to specify shape queries, we describe the constructs for defining shapes and discuss how the shape predicates are used in a query construct to retrieve rules whose histories exhibit the desired trends. We describe how this query capability is integrated into a trigger system to realize an active mining system. This case study is taken from R. Agrawal and G. Psaila, IBM Almaden Research Center, CA.

Data mining (also called *knowledge discovery*) in databases is the efficient discovery of previously unknown patterns in large databases and is emerging as a major application area for databases. In three classes of data mining problems involving associations, sequences and classification were introduced and it was argued that

S. Sumathi and S.N. Sivanandam: *Active Data Mining*, Studies in Computational Intelligence (SCI) **29**, 293–302 (2006)
www.springerlink.com

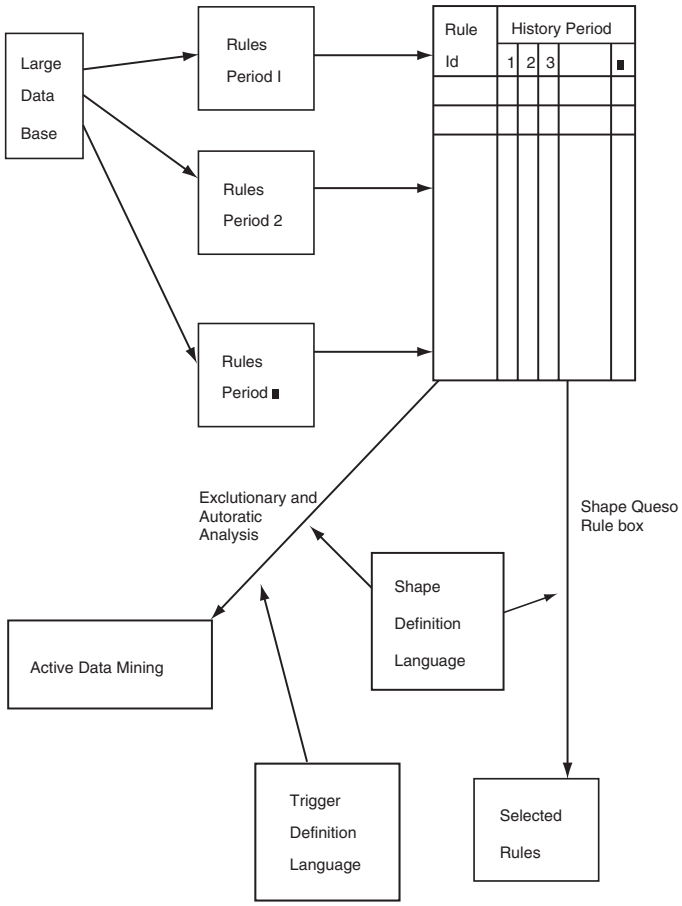

Fig. 13.1. Active Data Mining Process

these problems can be uniformly viewed as requiring discovery of rules embedded in massive data. Attached to every discovered rule are some statistical parameters, such as confidence or support of the rule.

As the data mining technology is applied in the production mode, the need for *active mining* arises. Figure 13.1 shows a schematic of the active data mining process. The basic idea is as follows. Rather than applying a mining algorithm to the whole data, the data is first partitioned according to time periods. The granularity of the time period is application dependent. The amount of data available is large (generally in gigabytes and more), so that this partitioning does not lose significance of the rules discovered. The mining algorithm is now applied to each of the partitioned data set and rules are obtained for each time period. These rules are collected into a rule base. In this rule base, each statistical parameter of a rule will have a sequence of values, called the *history of the parameter* for that rule. We can

now query the rule base using predicates that select rules based on the shape of the history of some or all parameters.

The user can specify triggers over the rule base in which the triggering condition is a query on the shape of the history. As the fresh data comes in for the current time period, the mining algorithm is run over this data, and the rule base is updated with the generated rules. This update causes the histories of the rules to be extended. (A history of a new rule is initialized with zero values for the past time periods.) This, in turn, may cause the triggering condition to be satisfied for some rules and the corresponding actions to be executed.

Such active systems can be used, for instance, to build early warning systems for spotting trends in the retail industry. For example, if we were mining association rules, we will have histories for the support and confidence of each rule. [An association rule is an expression of the form A $\Rightarrow$ C, where A and C are sets of literals. In a database of transactions, where each transaction is a set of literals, the rule A $\Rightarrow$ C signifies that very often when A appears in a transaction, so does C. How often this happens is captured by the "confidence" parameter and is indicative of the strength of the rule. The "support" parameter gives the fraction of transactions in the database in which the given rule is present, and is indicative of the prevalence of the rule]. Following the promotion for an item X, the user may specify a notification trigger on the rule X $\Rightarrow$ Y; the triggering condition being that the support history remains stable, but the confidence history takes the shape of a downward ramp. Firing of this trigger will signify that if the goal of promoting X was to drag the sale of Y, it was not fulfilled. The loyalists continued to buy X and Y together, but the new buyers cherry picked X.

13.1 Shape Definitions

A shape found in a history can be described by considering the transition of values assumed by the shape at the beginning and end of each unit time period. We let the user define classes of transitions and assign symbols to them. These symbols are called *elementary shapes.* We do not have a precanned set of elementary shapes; the user can add or delete shapes or change the definition of any of them. However, to keep the discussion concrete, assume a user-defined set of elementary shapes, consisting of *up, Up, down, Down, appears, disappears, stable and zero.* The shape *up* could be a slightly increasing transitions with a minimum and maximum variation of 0.05 and 0.19 respectively; *up* could be a highly increasing transition with a minimum and maximum variation of 0.20 and 1.00, respectively; *appears* could be a transition from a zero value to a nonzero value; *stable* could be a transition in which the absolute difference between the initial and final value is not more than 0.04; etc.

Complex shapes can be derived by recursively combining elementary shapes and previously defined derived shapes, using the *shape operators.* These operators are summarized in Table 13.1. We give syntax for each operator and how the corresponding derived shape is matched in a history H.

Using these operators, one can describe a wide variety of shapes found in a history, including "blurry," shapes where the user cares about the overall shape but does not care about specific details. The syntax for defining a shape is:

(Shape name (parameters) descriptor).

For example, here is a definition of a double peak:

```
(Shape spike (upcnt dncnt)
        (Concat (at least upcnt (any up Up))
        (atleast dncnt
        (any down Down))))
(Shape doublespeak (width ht1 ht2)
        (in width (in order spike (ht1 ht1)
        spike (ht2 ht2))))
```

We first define *spike* to be a shape that has at least upon number of either *up or Up* transitions followed by at least *dncnt* number of either *down or Down* transitions. Then *doublespeak* is a shape width wide that has two nonoverlapping spikes. Note that width may be wider than the sum of the width of the two spikes and there may be noise on either side of them. As another example, the shape bullish:

```
(Shape bullish (width upcnt dncnt)
        (in width
                (and
                (noless upcnt (any up Up))
                (nomore dncnt
                        (any down Down)))))
```

is defined to have at least upcnt ups (either up or Up) and at most dncnt downs (either down or Down) in width time periods. Finally, the shape drift:

```
(shape drift (width)
        (in width (precisely 0
                        (any up Up down Down))))
```

has no ups or downs in width time periods.

Earlier languages based on regular expressions for finding patterns in sequences were not targeted at defining shapes [Seshadri et al. *Proc. IEEE*

Table 13.1. Shape Operators

Multiple Choice	(any P_1, $P_2, \ldots P_n$)
Match all subsequences of H that match at least one of the P_1 shapes.	
Concatenation	(concat P_1, $P_2, \ldots P_n$)
First, match the shape P_1 if a matching subsequence s is found, match P_2 in the subsequence of H immediately following the last element of s. Accept the match if it is strictly contiguous to s, etc.	
Multiple Occurrences	(exact n P) (atleast n P) (atmost n P)
Match all subsequences of H that contain exactly (at least/at most) no contiguous occurrences of the shape P. In addition, the resulting subsequences must neither be preceded nor followed by a subsequence that matches P.	
Bounded Occurrences	(in length Shape occurrences)
Do "blurry" matching. Here length specifies the length of the shape in number of transitions. The shape-occurrences has two forms given below:	
Shape-Occurrences: Logical combination Using and and or.	(precisely n P) (noless n Q) (nomore n R)
Match all length long subsequences of H that contain precisely (no less than/no more than) n occurrences of the shape P (Q/R). The n occurrences of P (Q/R) need not be contiguous in the matched subsequence; there may be overlap or arbitrary gap between any two.	
Shape-occurrences: Ordered shapes.	(in order $P_1P_2 \ldots P_n$)
Match all length long subsequences of H containing the shapes P_1 through P_n in that order. P_1 and P_{i+1} may not overlap, but may have an arbitrary gap.	

Int.Conf. on Data Engg. 1995]. The difference in design focus influences which expressions are easy to write, understand, optimize, and evaluate.

13.2 Queries

With the machinery for defining shapes in hand, we are ready to specify how we can retrieve rules whose one or more histories contain the desired shapes. The syntax for defining a query is:

(query (shape history-spec))

Here, shape is the descriptor for the shape to be matched. The history-spec is of the form:

history-name start-time end-time

Here history-name specifies the name of the history in which the shape should be matched. The portion in which the matching occurs is constrained

by the interval specified by start time and end time. Matching over the complete history can be specified by using the keywords start and end for start time and end time, respectively.

The result of the execution of a query is the set of all rules that contain the desired shape in the specified history. In addition, the result also contains the list of subsequences of the history that matched the shape. If no subsequence matches the specified shape, the result is an empty set.

Here is an example of a query:

```
(shape ramp( ) (concat Up Up))
        (query ((ramp) (confidence start end)))
```

We have defined a simple shape *ramp,* consisting of two consecutive Ups, and we want to retrieve all the rules whose confidence history contains a ramp. Instead of the shape name, we could have alternatively written its definition in the above query. We also could have limited the range of confidence history in which the shape should be matched. Here is a modified query:

```
(query ((Concat Up Up)
  (confidence start 10)))
```

The user can also retrieve combinations of several shapes in different histories by using the logical operators and and or. Here is an example of a query that is looking for different shapes in the two histories of a rule – an *upramp* in support but a *downramp* in confidence:

```
(shape upramp (len cnt)
  (in len (noless cnt (any up Up))))
(shape dnramp (len cnt)
(in len
  (noless cnt (any down Down))))
(query
  (and
  (upramp 5 3) (support start 10))
  (dnramp 5 3) (confidence start 10))
))
```

13.3 Triggers

The query language we just described provides the capability of discovering interesting information by analyzing rules and their histories in novel ways. Consider a user who is periodically collecting rules in the rule base and wants to discover rules that are assuming critical (or interesting in some other way) behavior. For instance, the user may be interested in rules that have started exhibiting increasing trend. Rather than running queries every time the data for a new period comes and rules are added to the rule base, it will be preferable to post these queries as triggers and let the system initiate appropriate actions (e.g., notification) when the trigger conditions are satisfied.

We use the ECA (Event Condition Action) model by Chakravarthy et al. 1989 as the basis for the trigger system. The interesting aspects of our trigger system are what can be specified as trigger conditions, the semantics of the trigger execution, and how it is used in the active mining process.

The syntax for specifying a trigger is:

```
(trigger trigger-name
        (events events-spec)
        (condition (shape history-spec))
        (actions actions-spec)
)
```

A trigger definition has three sections: *events, condition, and actions.* Let us examine each of them. The trigger system reacts to predefined and user-defined events. The predefined events describe an external update of the rule base. These events are: *createrule and updatehistory.* They occur when a new rule is added to the rule base and the history of rule is updated, respectively. A user-defined event is introduced to the system as:

(event event-name)

where event-name is the name of the event.

The *events-spec* in the events section specifies the events to which the trigger being defined reacts. Predefined and user-defined events and their logical combinations using the logical operators or and can appear in the *events-spec.* A trigger is considered fired if the event specification is true for at least one rule in the rule base. That is, the specified event combination has occurred for some rule.

The condition section is syntactically and semantically similar to the query construct discussed in Section Queries. The difference is that the condition is evaluated only on rules present in the *affected set* produced by the events section, instead of the whole rule base. A condition selects the affected set of rules from the rule base and performs the specified shape query on the relevant histories of those rules. The condition is true if the output set resulting from

the query is not empty. In that case, the action section is executed on the query output.

The *action-specs* in the actions section is a list of actions that are executed for all (and only those) rules that belong to the output set produced by the condition evaluation. An action can be an execution of a function, such as *notify or show,* which can be defined by the user or system supplied. An action can also be a user-defined event name, in which case an occurrence of the specified event is generated. An action does not change the state of the rule base; the goal of model is only to notify that the properties expressed by the condition of a trigger holds for some rules and is accomplished by generating predefined and user-defined events that alert any possibly interested trigger.

13.3.1 Wave Execution Semantics

Several semantics have been proposed for trigger systems in active databases. The execution semantics for trigger system follows what we call the *wave execution model.* This semantics is close to what is known as the deterministic semantics for Datalog like rules in S. Ceri et al. 1990. The attractiveness of this semantics is its simplicity and a good match for our application.

A *wave* is a set of event occurrences that come together to the active system. The trigger execution process starts when a new wave is ready. First, the event specification of every trigger is checked to determine if it will fire. Triggers for which this evaluation is true are selected for firing and their affected set (rules affected by events-spec) is produced. When the event specification has been checked for all the triggers, the current wave has been used up and any event generated as a consequence of the triggers fired will belong to a new wave.

The selected triggers are fired now. For each fired trigger, its condition is evaluated only on rules in the affected set. If the condition is true, the output set is passed to the actions section. The actions section is immediately executed for each rule in the output set. If any event is generated as a consequence, it is added to the new wave.

After the conclusion of the condition evaluation and the eventual actions execution for all the fired triggers, the process is repeated considering the events generated as belonging to the new wave. The process terminates if the evaluation of all the event specifications determines that no trigger needs to be fired.

Example: We now give a simple example to illustrate the trigger facility. Suppose that a user wants to be notified if the support for a rule is increasing but its confidence is decreasing at the same time. The following definitions show how the user can accomplish this goal:

```
(shape uptrend (width upcnt)
        (in width
                (noless upcnt (any up Up))))
        (shape dntrend (width dncnt)
                (in width (noless
                        dncnt (any down Down))))
        (event upward)
        (trigger detect_up
    (events updatehistory)
    (condition
    (uptrend (5 4)
        (support (- end 5) end)))
    (actions upward)
          )
          (trigger detech_dn
    (events upward)
    (condition
        (dntrend (5 4)
                (confidence (- end 5) end)))
        (actions notify)
          )
```

We first specify what is meant by support is increasing and confidence is decreasing by defining two shapes: uptrend and dntrend. We introduce an event named upward to the system using the event construct. We then define the trigger detect_up. This trigger can be fired by the predefined event update history. If this trigger is fired, the condition section of this event checks if the rules that were updated (affected set of the event update history) contain up trend in the last five periods of their support history. If this condition is evaluated as true for some rules, the user-defined event upward is generated for each of these rules.

The second trigger detect_dn reacts to the generation of the occurrences of the upward event and it checks for dntrend in the last five periods of the confidence history of only those rules for which the trigger has been fired

(affected set of the upward event). Thus, the user is notified of only those rules that simultaneously had an uptrend in support and dntrend in confidence in the last five time periods.

13.4 Summary

In this chapter an active data mining paradigm that combines the recent work in data mining with the rich literature on active database system is described. Here data is continuously mined at a desired frequency. As rules are discovered, they are added to a rule base, and if they already exist, the history of the statistical parameter associated with the rules is updated. Thus this section gives details on active data mining.

13.5 Review Questions

1. Explain active data mining process.
2. Write note on shape definitions with its operators in the active mining.
3. Write short note on queries, triggers, and wave execution semantics employed in mining.