Chapter 4: Generalizations and a case study

Algorithmic Methods of Data Mining, Fall 2007, Chapter 4: Generalizations and a case study 1

Contents

- Simple generalization: conjunctions of arbitrary primitive conjuncts
- More complex generalization: pattern classes with generalization and specialization
- An example: episodes in sequences

Pattern classes

- What are the patterns we searched for in finding frequent sets?
- Sets ABC etc.; corresponds to the predicate $A == 1 \wedge B == 1 \wedge C == 1$
- A conjunctive predicate: something **and** something **and** something
- The conjuncts (the somethings) do not necessarily have to be variables (attributes)
- Can be something else
- Example: numeric attributes D and E, string attribute F
- $D \leq 5 \wedge E > 17.1 \wedge substr(F, 1, 3) == "abc"$

The levelwise algorithm

- ... is an algorithm for finding conjuctions from a set of primitive conjucts
- such that the conjunction is true for at least a threshold of the rows in the data set
- Given a data set, think of a transformed dataset with the conjuncts as columns
- And then apply the levelwise method
- First find conjunctions with 1 conjunct
- Build candidates of size 2
- Evaluate; build candidates; evaluate; ...

What do we have to be able to do?

- Find all primitive conjuncts and compute their frequencies
- Form candidates: but this is the same as in the frequent set case

More complex generalization

- Suppose we have a set of patterns
- Such that some patterns are specializations of others
- If P is true for a row, then Q is true for a row
- For frequent sets, specialization is equivalent to "superset"
- If ABC is true for a row, then AB is true for a row
  AB is a generalization of ABC, and ABC is a specialization of AB
- Then a levelwise approach can be used

General levelwise algorithm

- Find all patterns that have no generalizations
- Evaluate whether they are true for sufficiently many rows
- Given a set $\mathcal{F}$ of patterns, find all patterns α such that $\alpha \notin \mathcal{F}$ and all generalizations of α are in $\mathcal{F}$
- Such patterns are the candidate patterns $\mathcal{C}$
- Evaluate these against the data
- Iterate

Example: alarm correlation

- An application example of frequent pattern finding
- Basically the same fundamental ideas as in frequent sets
- Finding frequently occurring conjunctions
- Sequential data

Networks and alarms

- network elements: switches, base stations, transmission equipment, etc.
- 10–1000 elements in a network
- an alarm: a message generated by a network element

  `1234 EL1 BTS 940926 082623 A1 Channel missing`

- hundreds of different alarm types
- up to 100,000 alarms a day
- each contains only local information

Characteristics of the alarm flow

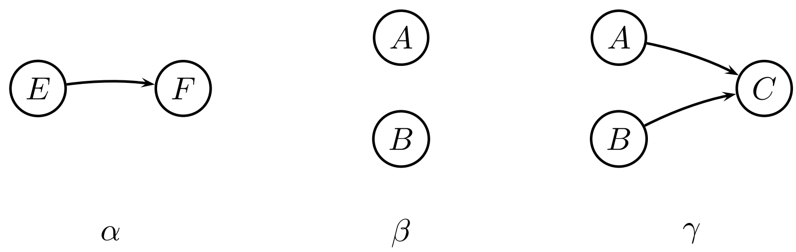
- a variety of situations
- bursts of alarms
- hardware and software change fast
- Difficult to build expertise in the properties of the stream

How to analyze a flow of alarms?

- lots of possibilities: hazard models, neural networks, rule-based representations
- comprehensibility of the discovered knowledge
- simple rule-based representations
- "if certain alarms occur within a time window, then a certain alarm will also occur"

Episodes

Mannila, Toivonen, Verkamo:

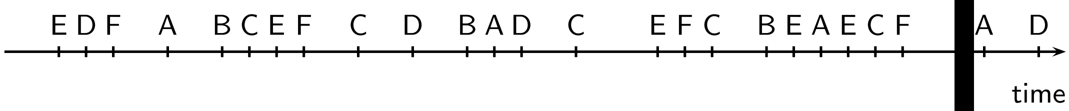


Episodes

Basic solution

- look for repeated occurrences of episodes in the alarm flow sequences
- occurrence: alarms of the specified type occur in the specified order
- why this form?
  - comprehensible
  - represent simple relationships
  - insensitive to inaccurate clocks
  - allows analysis of merged, unrelated sequences

Example sequence



A sequence of alarms

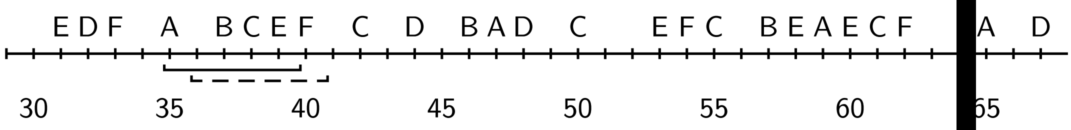
Observations:

- whenever E occurs, F occurs soon
- whenever A and B occur (in either order), C occurs soon

Data

- a set R of *event types*
- an *event* is a pair (A, t)
- $A \in R$ is an event type
- t is an integer, the *(occurrence) time* of the event
- *event sequence* s on R: a triple (s, T_s, T_e)
- $T_s < T_e$ are integer (starting and ending time)
- $s = \langle (A_1, t_1), (A_2, t_2), \ldots, (A_n, t_n) \rangle$
- $A_i \in R$ and $T_s \leq t_i < T_e$ for all $i = 1, \ldots, n$
- $t_i \leq t_{i+1}$ for all $i = 1, \ldots, n-1$

Example



An example event sequence and two windows of width 5.

Windows

- Event sequence $\mathbf{s} = (s, T_s, T_e)$
- A window on it: $\mathbf{w} = (w, t_s, t_e)$
- $t_s < T_e$ and $t_e > T_s$
- w consists of those pairs (A, t) from s where $t_s \leq t < t_e$
- $width(\mathbf{w}) = t_e - t_s$: the *width* of the window $\mathbf{w}$
- $\mathcal{W}(\mathbf{s}, \mathit{win})$: all windows $\mathbf{w}$ on $\mathbf{s}$ such that $width(\mathbf{w}) = \mathit{win}$
- First and last windows need special handling

Episodes

- An *episode* α is a triple $(V, \leq, g)$
- V is a set of nodes
- $\leq$ is a partial order on V
- $g : V \rightarrow R$ is a mapping associating each node with an event type
- Intuition: the events in $g(V)$ have to occur in the order described by $\leq$
- *Size* of α, denoted $|\alpha|$, is $|V|$
- *Parallel episode*: the partial order $\leq$ is trivial
- *Serial episode*: $\leq$ is a total order
- *Injective*: no event type occurs twice in the episode

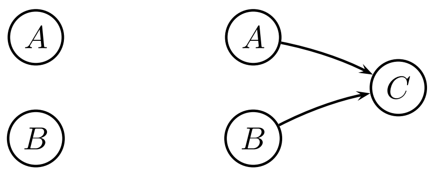
Example

α

An episode

What are the set V and the mapping g?

Example, subepisode



β γ

A subepisode and episode

Subepisodes

$\beta = (V', \leq', g')$ is a *subepisode* of $\alpha = (V, \leq, g)$, $\beta \preceq \alpha$, if:

there exists an injective mapping $f : V' \to V$ such that

- $g'(v) = g(f(v))$ for all $v \in V'$
- for all $v, w \in V'$ with $v \leq' w$ also $f(v) \leq f(w)$

An episode α is a *superepisode* of β if and only if $\beta \preceq \alpha$

$\beta \prec \alpha$ if $\beta \preceq \alpha$ and $\alpha \not\preceq \beta$

In the example: $\beta \preceq \gamma$

Occurrences of episodes

$\alpha = (V, \leq, g)$ *occurs* in an event sequence
$\mathrm{s} = (\langle (A_1, t_1), (A_2, t_2), \ldots, (A_n, t_n) \rangle, T_s, T_e)$, if there exists an injective mapping $h : V \to \{1, \ldots, n\}$ from nodes to events, such that

- $g(x) = A_{h(x)}$ for all $x \in V$
- for all $x, y \in V$ with $x \neq y$ and $x \leq y$ we have $t_{h(x)} < t_{h(y)}$ (or $h(x) < h(y)$)

$(w, 35, 40)$ on the example sequence: events of types A, B, C, and E

both β and γ occur

Frequency of occurrence

- the *frequency* of an episode α in $\mathbf{s}$ is

$$fr(\alpha, \mathbf{s}, win) = \frac{|\{\mathbf{w} \in \mathcal{W}(\mathbf{s}, win) \mid \alpha \text{ occurs in } \mathbf{w}\}|}{|\mathcal{W}(\mathbf{s}, win)|},$$

- i.e., the fraction of windows on $\mathbf{s}$ in which α occurs
- The probability that the episode occurs in a randomly selected window

Pattern discovery task

- A *frequency threshold min_fr*
- α is *frequent* if $fr(\alpha, \mathbf{s}, win) \geq min\_fr$
- $\mathcal{F}(\mathbf{s}, win, min\_fr)$: collection of frequent episodes in $\mathbf{s}$ with respect to *win* and *min_fr*
- size $= l$: $\mathcal{F}_l(\mathbf{s}, win, min\_fr)$.
- Given an event sequence $\mathbf{s}$, a set $\mathcal{E}$ of episodes, a window width *win*, and a frequency threshold *min_fr*, find $\mathcal{F}(\mathbf{s}, win, min\_fr)$

A very simple algorithm for parallel episodes

- Transform the sequence into a 0-1 dataset
- Columns: the set R of event types
- One row for each window $\mathbf{w}$
- In the row corresponding to window $\mathbf{w}$: a 1 for the those event types that occur in the window
- A parallel episode = a set of event types
- Frequent parallel episodes = frequent sets of attributes
- Size of the representation is large

Algorithms

- The same general idea as for frequent sets
- Candidate generation and database pass
- Do not create the contents of windows explicitly

Algorithms

Algorithm

Input: A set R of event types, an event sequence $\mathbf{s}$ over R, a set $\mathcal{E}$ of episodes, a window width *win*, and a frequency threshold *min_fr*.

Output: The collection $\mathcal{F}(\mathbf{s}, \mathit{win}, \mathit{min\_fr})$ of frequent episodes.

Method:

1. compute $\mathcal{C}_1 := \{\alpha \in \mathcal{E} \mid |\alpha| = 1;\}$
2. $l := 1;$
3. **while** $\mathcal{C}_l \neq \emptyset$ **do**
4. // Database pass
5. compute $\mathcal{F}_l(\mathbf{s}, \mathit{win}, \mathit{min\_fr}) := \{\alpha \in \mathcal{C}_l \mid \mathit{fr}(\alpha, \mathbf{s}, \mathit{win}) \geq \mathit{min\_fr}\};$
6. $l := l + 1;$
7. // Candidate generation :
8. compute $\mathcal{C}_l := \{\alpha \in \mathcal{E} \mid |\alpha| = l,$ and $\beta \in \mathcal{F}_{|\beta|}(\mathbf{s}, \mathit{win}, \mathit{min\_fr})$ for all $\beta \in \mathcal{E}$ such that $\beta \prec \alpha$ and $|\beta| < l\};$
9. **for** all l **do** output $\mathcal{F}_l(\mathbf{s}, \mathit{win}, \mathit{min\_fr});$

Basic lemma, once again

Lemma 4.12 If an episode α is frequent in an event sequence $\mathbf{s}$, then all subepisodes $\beta \preceq \alpha$ are frequent. $\square$

Parallel, serial, injective episodes

- *Parallel episode*: the partial order $\leq$ is trivial (= frequent sets)
- *Serial episode*: $\leq$ is a total order (= frequent subsequence)
- *Injective*: no event type occurs twice in the episode (= proper sets, not multisets)
- Useful cases: (serial or parallel) [injective] episodes
  - reduce redundancy in generated episodes
  - keep episodes comprehensible
  - simpler to implement

Generation of candidate episodes

- Parallel episodes, serial episodes (injective or non-injective)
- Same idea as for association rules
- A candidate episode has to be a combination of two episodes of smaller size
- Very small variations to the candidate generation procedure

Recognizing episodes in sequences

- First problem: given a sequence and an episode, find out whether the episode occurs in the sequence
- Finding the number of windows containing an occurrence of the episode can be reduced to this
- Successive windows have a lot in common
- How to use this?
- An incremental algorithm

Parallel episodes

- For each candidate α maintain a counter α.*event_count*: how many events of α are present in the window
- When α.*event_count* becomes equal to $|\alpha|$, indicating that α is entirely included in the window
  - Save the starting time of the window in α.*inwindow*
- When α.*event_count* decreases again, increase the field α.*freq_count* by the number of windows where α remained entirely in the window

Algorithm

Input: A collection $\mathcal{C}$ of parallel episodes, an event sequence $\mathbf{s} = (s, T_s, T_e)$, a window width *win*, and a frequency threshold *min_fr*.

Output: The episodes of $\mathcal{C}$ that are frequent in s with respect to *win* and *min_fr*.

Method:

// Initialization:
for each α in $\mathcal{C}$ **do**
    for each A in α **do**
        $A.\mathit{count} := 0$;
        for $i := 1$ **to** $|\alpha|$ **do** $contains(A, i) := \emptyset$;
for each α in $\mathcal{C}$ **do**
    for each A in α **do**
        $a :=$ number of events of type A in α ;
        $contains(A, a) := contains(A, a) \cup \{\alpha\}$;
    $\alpha.\mathit{event\_count} := 0$;
    $\alpha.\mathit{freq\_count} := 0$;

Algorithm Method:

// Recognition:
for $start := T_s - \mathit{win} + 1$ **to** T_e **do**
    // Bring in new events to the window:
    for all events (A, t) in s such that $t = start + \mathit{win} - 1$ **do**
        $A.\mathit{count} := A.\mathit{count} + 1$;
        for each $\alpha \in$ contains($A, A.\mathit{count}$) **do**
            $\alpha.\mathit{event\_count} := \alpha.\mathit{event\_count} + A.\mathit{count}$;
            if $\alpha.\mathit{event\_count} = |\alpha|$ **then** $\alpha.\mathit{inwindow} := start$;
    // Drop out old events from the window:
    for all events (A, t) in s such that $t = start - 1$ **do**
        for each $\alpha \in$ contains($A, A.\mathit{count}$) **do**
            if $\alpha.\mathit{event\_count} = |\alpha|$ **then**
                $\alpha.\mathit{freq\_count} := \alpha.\mathit{freq\_count} - \alpha.\mathit{inwindow} + start$;
            $\alpha.\mathit{event\_count} := \alpha.\mathit{event\_count} - A.\mathit{count}$;
        $A.\mathit{count} := A.\mathit{count} - 1$;
// Output:
for all episodes α in $\mathcal{C}$ **do**
    if $\alpha.\mathit{freq\_count}/(T_e - T_s + \mathit{win} - 1) \geq \mathit{min\_fr}$ **then** output α;

Theorem 1 *The above algorithm works correctly.*

Proof We consider the following two invariants. (1) For each event type A that occurs in any episode, the variable $A.count$ correctly contains the number of events of type A in the current window.
(2) For each episode α, the counter $\alpha.event\_count$ equals $|\alpha|$ exactly when α occurs in the current window. □

Complexity

Assume that exactly one event takes place every time unit.

Assume candidate episodes are all of size l, and let n be the length of the sequence.

Theorem 2 *The time complexity of Algorithm 102 is* $\mathcal{O}((n + l^2)|\mathcal{C}|)$.

Proof Initialization takes time $\mathcal{O}(|\mathcal{C}|l^2)$.

How many accesses to $\alpha.event\_count$ on lines 7 and 14.

In the recognition phase there are $\mathcal{O}(n)$ shifts of the window. In each shift, one new event comes into the window, and one old event leaves the window. Thus, for any episode α, $\alpha.event\_count$ is accessed at most twice during one shift.

The cost of the recognition phase is thus $\mathcal{O}(n|\mathcal{C}|)$. □

Serial episodes

- Use state automata that accept the candidate episodes
- example: episode A B A B

General episodes

Different alternatives

Experiences in alarm correlation

Useful in

- finding long-term, rather frequently occurring dependencies,
- creating an overview of a short-term alarm sequence, and
- evaluating the consistency and correctness of alarm databases
- discovered rules have been applied in alarm correlation
- lots of rules are trivial