Discovery of frequent episodes in event sequences
Andres Kauts, Kait Kasak

University of Tartu 2009
MTAT.03.249 Combinatorial Data Mining Algorithms
What is sequential data mining

Sequential data mining is a branch of data mining that deals with datasets in which events have a time of occurrence.
Sequencial data-mining - where to use?

- Log analysis
- Security (intrusion detection)
- Analysing financial events (stock markets)
- Genetics (DNA-sequences)
- Document collection
- Time based shopping basket prediction

...or anything else that looks like this:
Basics

Data consists of **events** in a sequence.

Given a set $E$ of *event types*, an event is a pair $(A, t)$ where $A \in E$ and $t$ is an integer, the (occurrence) *time* of the event.

An event sequence $s$ on $E$ is a triple $(s, T_s, T_e)$ where

$$s = \langle (A_1, t_1), (A_2, t_2), \ldots, (A_n, t_n) \rangle$$
Example

Example Sequence:
\[ s = \langle (E,31), (T,32), (F,33), (A,35), (B,37), (C,38), ..., (D,67) \rangle \]

Example Window of size 5:
\[ s = \langle (A,35), (B,37), (C,38), (E,39) \rangle, 35, 40 \]
Episodes

Serial

Parallel

Complex

E  F
A
B
A  C
B
Episodes

Episode is a collection of events, with predefined order of appearance. Episodes, in concept, are similar to itemsets. The main difference is, that items (events) they consist of, must appear in a certain timeframe (window) and might have a particular order.
At first there was apriori ...
At first there was apriori ... but sequential data makes it a bit more complicated:

- Multiple events at the same time
- Order of appearance
Apriori ...

1. Gather all possible event types from sequence
2. Generate "first level" candidates (episodes with one event)
3. Find if generated candidates are frequent
4. Generate next level super episodes of the frequent episodes found as new candidates
5. Wash, Rinse, & Repeat

...  

6. Output rules

   1. /* Find frequent episodes (Algorithm 2): */
   2. compute \( F(s, \text{win}, \text{min}_fr) \);
   3. /* Generate rules: */
   4. for all \( \alpha \in F(s, \text{win}, \text{min}_fr) \) do
   5.     for all \( \beta < \alpha \) do
   6.         if \( fr(\alpha) / fr(\beta) \geq \text{min}_\text{conf} \) then
   7.             output the rule \( \beta \rightarrow \alpha \) and the confidence \( fr(\alpha) / fr(\beta) \);
Two basic algorithms for finding frequent episodes:

**WINEPI** - "Sliding window" approach

**MINEPI** - Minimal occurrences approach
Winepi

• Candidate episodes are generated
• A window is slid through the event-based data sequence
• Occurrence of episodes is counted in every window
• Higher level episode candidates are generated based on frequent episodes found

• Input: window size and minimal frequency
• Output: frequent episodes in defined windows
**Winepi**

**frequency threshold**: $min_{fr}$ is used

Episode $\alpha$ is **frequent** if $fr(\alpha, s, win) \geq min_{fr}$, i.e., „if the frequency of $\alpha$ exceeds the minimum frequency threshold within the data sequence $s$ and with window width $win“"

$F(s, win, min_{fr})$: a collection of frequent episodes in $s$ with respect to $win$ and $min_{fr}$

**Apriori trick holds**: if an episode $\alpha$ is frequent in an event sequence $s$, then all subepisodes $\beta \in \alpha$ are frequent
Parallel episodes:
For each candidate $\alpha$ maintain a counter $\alpha.\text{event\_count}$: how many events of $\alpha$ are present in the window

When $\alpha.\text{event\_count}$ becomes equal to $|\alpha|$, indicating that $\alpha$ is entirely included in the window, save the starting time of the window in $\alpha.\text{inwindow}$

When $\alpha.\text{event\_count}$ decreases again, increase the field $\alpha.\text{freq\_count}$ by the number of windows where $\alpha$ remained entirely in the window
Winepi

Serial and complex episodes:
use a state automata
Winepi

window width is 40 seconds (last point is excluded).
windows start and end before the sequence.

Event a = s.getEventAt(t);
if (a != null) {
    int aCount = eventCount.get(a);
eventCount.put(a, aCount);
    debug(" Adding element: " + a);
    debugln(" Count: " + eventCount.get(a));
    if (!contains.get(a).isEmpty() && contains.get(a).get(aCount) != null) {
        for (Episode ep : contains.get(a).get(aCount)) {
            aCount = ep.eventCount += aCount;
            if (ep.eventCount == ep.length()) {
                debugln(" Full episode in window. Start counting windows from here");
                ep.inWindow = start;
            }
        }
    }
} else {
    debug(" e != null in this loop");
t = start;
a = s.getEventAt(t);
    if (a != null) {
        int aCount = eventCount.get(a);
        if (!contains.get(a).isEmpty() && contains.get(a).get(aCount) != null) {
            for (Episode ep : contains.get(a).get(aCount)) {
                debugln(" Episode with " + aCount + " events " + a
                        + " no longer in window.");
                if (ep.eventCount == ep.length()) {
                    ep.freqCount = ep.freqCount - ep.inWindow + start;
                    debugln(" Full episode was in "
                            + (start - ep.inWindow) + " windows");
                }
                ep.eventCount -= aCount;
            debugln(" " + ep);
        }
    }
    aCount = eventCount.get(a);
Winepi

**Strengths:**
- Intuitive
- Not too heavy on memory

**Weaknesses:**
- Slow with larger frequent episodes
- Some problems:
Minepi

- Candidate episodes are generated
- **Minimal occurrences** of each candidate episode are counted
- Frequency of found minimal episodes is computed
- Higher level episode candidates are joined from level frequent episodes
- Max window width **may** be used.
Formally, given a episode $\alpha$ and an event sequence $s$, the interval $[t_s, t_e]$ is a minimal occurrence $\alpha$ of $s$,

- If $\alpha$ occurs in the window corresponding to the interval
- If $\alpha$ does not occur in any proper subinterval

The set of minimal occurrences of an episode $\alpha$ in a given event sequence is denoted by $mo(\alpha)$:

$$mo(\alpha) = \{ [t_s, t_e] \mid [t_s, t_e] \text{ is a minimal occurrence of } \alpha \}$$
**Example:** Parallel episode $\beta$ consisting of event types $A$ and $B$ has three minimal occurrences in $s$: $\{[30,40], [40,60], [60,70]\}$, $\alpha$ has one occurrence in $s$: $\{[60,80]\}$

Example 2 (might be removed!): The parallel episode $\beta$ consisting of event types $A$ and $B$ has four minimal occurrences in $s$: $mo(\beta) = D \{[35; 38); [46; 48); [47; 58); [57; 60)\}$. Minimal occurrences of the partially ordered episode $\gamma$ are: $[35; 39); [46; 51); [57; 62)$. 
episode rule: $\beta[\text{win}_1] \Rightarrow \alpha[\text{win}_2]$  $\beta \in \alpha$

$\beta$ and $\alpha$ are episodes such that

If episode $\beta$ has a minimal occurrence at interval $[t_s, t_e]$ with $t_e - t_s \leq \text{win}_1$, then episode $\alpha$ occurs at interval $[t_s, t'_e]$ for some $t'_e$ such that $t'_e - t_s \leq \text{win}_2$

confidence of the rule $\beta[\text{win}_1] \Rightarrow \alpha[\text{win}_2]$ is: $|\text{mo}(\alpha)| / |\text{mo}(\beta)|$

where $|\text{mo}(\beta)|$ is the number of minimal occurrences of $\beta$ such that $t_e - t_s \leq \text{win}_1$

$|\text{mo}(\alpha)|$ is the number of such occurrences where there is also an occurrence of $\alpha$ within the interval $[t_s, t_s + \text{win}_2]$

frequency of the rule $\beta[\text{win}_1] \Rightarrow \alpha[\text{win}_2]$ is: $|\text{mo}(\alpha)|$
**Strengths:**

• Good performance with bigger episodes.
• More „natural“ episode rules as there can be several time limits for one rule e.g. „If A and B occur within 15 seconds, then C follows within 30 seconds“

**Weaknesses:**

• Memory hog
Unbounded Episodes

Both Minepi and Winepi have fixed window width. But what if we're more interested in closeness of elements than fixed window width?

To overcome this problem unbounded episodes are introduced:

*Unbounded episodes* define *maximal time* \( t \) between any two events but no window width.
Unbounded Episodes

Unbounded episodes are good, when one is more interested in the **closeness of elements** than the window width itself.
Win-Miner

• Max window size sets constraints to episode length.

• We might be more interested in variable-width episodes.

• Unbounded episodes can help, but are an incomplete solution. They are often open for too long window sizes (reducing confidence).

To overcome these problems Win-Miner was introduced.
Find frequent unbounded episodes.

Then find *optimal window size* by looking when increasing in window size decreases confidence.

Input: support threshold, confidence threshold, maximum gap between events, decrease threshold.
Win-Miner

Fig. 2. Confidence vs width
Case study

Mining episode rules in STULONG dataset
Nicolas Meger, Claire Leschi, Noël Lucas and Christophe Rigotti
Case study: Stulong

- Dataset is the result of a twenty-year long study of risk factors related to atherosclerosis in a population of 1417 middle-aged men.

- *Win-Miner* algorithm was used
Case study: Stulong

First run:
6 results found.

Each rule that had been discovered expresses knowledge that was well known.
- Confidence, that experiment was working correctly.

Additionally the window of importance for rules was found.
Case study: Stulong

Example:
"If the patient has no hypercholesterolemia and if he sometimes follows his diet, then the patient has no hypercholesterolemia with a probability of 0.8 and this, within 40 months, which is the optimal window size for this rule. This rule is supported by 201 examples in the event sequence".
Case study: Stulong

Second run:
217 results found.

Again, many expected results were found. While some new ones and time of importance for some known rules was found.
Fuzzy Frequent Episodes

Similar to *MINEPI* but the event occurrences are not limited to values 0 or 1. Events have a probability of occurrence and the minimal occurrence of an episode is the product of its events.

In this example the minimal occurrence of episode $\beta$ would be:

$$0.2 \cdot 0.7 \cdot 0.3 = 0.042$$
Fuzzy Frequent Episodes

Fuzzy frequent episodes are beneficial:

• If event-attributes can represent quantitative data.
• If event-attributes cannot be easily classified for instance: „How little hair a subject needs to be considered bald ?“
Fuzzy Frequent Episodes

- Events mined: The number of different destination ports during last 2 seconds.
- Anomaly percentage = $m/n \times 100\%$ where $n = \text{total events}$, $m = \text{events not represented in training data}$

$min\text{confidence} = 0.8$, $min\text{support} = 0.1$, $min\text{occurrence} = 0.3$ and $\text{window} = 15s$
Fuzzy Frequent Episodes vs „Vanilla“ Episodes

- \( PN \) was divided into 3 Fuzzy sets or, in case of traditional episodes, 3 fixed intervals (LOW, MEDIUM, HIGH)
- False positive rates on the same training data:
# Performance

<table>
<thead>
<tr>
<th>Frequency threshold</th>
<th>Candidates</th>
<th>Frequent episodes</th>
<th>Iterations</th>
<th>Total time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
<td>4528</td>
<td>359</td>
<td>45</td>
<td>680</td>
</tr>
<tr>
<td>0.002</td>
<td>2222</td>
<td>151</td>
<td>44</td>
<td>646</td>
</tr>
<tr>
<td>0.005</td>
<td>800</td>
<td>48</td>
<td>10</td>
<td>147</td>
</tr>
<tr>
<td>0.010</td>
<td>463</td>
<td>22</td>
<td>7</td>
<td>110</td>
</tr>
<tr>
<td>0.020</td>
<td>338</td>
<td>10</td>
<td>4</td>
<td>62</td>
</tr>
<tr>
<td>0.050</td>
<td>288</td>
<td>1</td>
<td>2</td>
<td>22</td>
</tr>
<tr>
<td>0.100</td>
<td>287</td>
<td>0</td>
<td>1</td>
<td>16</td>
</tr>
</tbody>
</table>

**Winepi serial**

<table>
<thead>
<tr>
<th>Support threshold</th>
<th>Candidates</th>
<th>Frequent episodes</th>
<th>Iterations</th>
<th>Total time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>12732</td>
<td>2735</td>
<td>83</td>
<td>28</td>
</tr>
<tr>
<td>100</td>
<td>5893</td>
<td>826</td>
<td>71</td>
<td>16</td>
</tr>
<tr>
<td>250</td>
<td>2140</td>
<td>298</td>
<td>54</td>
<td>16</td>
</tr>
<tr>
<td>500</td>
<td>813</td>
<td>138</td>
<td>49</td>
<td>14</td>
</tr>
<tr>
<td>1000</td>
<td>589</td>
<td>92</td>
<td>48</td>
<td>14</td>
</tr>
<tr>
<td>2000</td>
<td>405</td>
<td>64</td>
<td>47</td>
<td>13</td>
</tr>
<tr>
<td>4000</td>
<td>352</td>
<td>53</td>
<td>46</td>
<td>12</td>
</tr>
</tbody>
</table>

**Minepi serial**
Figure 5. Number of frequent serial (solid line) and injective parallel (dotted line) episodes as a function of the window width; WINEPI, alarm database, frequency threshold 0.002.
Figure 6. Processing time for serial (solid line) and parallel (dotted line) episodes with MINEPI; alarm database, maximum time bound 60 s.
Case study

Mining Frequent Episodes for Relating Financial Events and Stock Trends

Anny Ng and Ada Wai-chee Fu
Case study

With a dataset of financial news (775 days) harvested interesting keywords from it („telecommunication stocks raise“, „Star TV-HK Telecom“) and tried to find relations between news events and events in stock market.
Case study - performance

Fig. 3. Performance of synthetic datasets D1 and D2
Case study - performance

(a) varying number of days  
(b) varying number of event types

Fig. 4. Synthetic dataset D2 with window size = 3 and threshold = 10%.
Case study - performance

Fig. 5. Real dataset with (a) window size = 3 days. (b) support threshold = 20%.