Advanced Human–Machine Interaction
Interaction Data Analysis

TDM03: Mining Interaction Sequences with PrefixSpan

INSA Rouen Normandie – Normandie University

Learning objectives

At the end of the session, you should be able to:

- transform a raw interaction log into a database of sequences;
- define and compute trace support for an ordered pattern;
- explain the projected-database principle used by PrefixSpan;
- extract frequent sequential patterns and interpret their limits;
- distinguish session support from user support and avoid data leakage.

The practical session lasts 90 minutes. Parts 1–5 should be completed during the session. Parts 6–8 represent approximately one additional hour of work.

Context and files

The file `interaction_log.csv` contains synthetic traces collected from a learning platform. Each row is one observable event:

event_id	user_id	session_id	timestamp_s	action	source
1	u01	u01_s01	7	open_course	pointer
2	u01	u01_s01	16	open_question	pointer

The dataset also contains an `outcome` column. Do not use it for pattern extraction: it is provided only to make later interpretations possible.

For this practical, every event is represented as a singleton itemset. A session such as

```
open_course, open_question, view_hint, submit
```

therefore represents the sequence

$$\langle \{open_course\}, \{open_question\}, \{view_hint\}, \{submit\} \rangle.$$

We write it more compactly as $\langle open_course, open_question, view_hint, submit \rangle$. This is the single-item special case of the sequences of itemsets presented in the lecture.

The student archive contains:

- `interaction_log.csv`: the data;
- `prefixspan_starter.py`: functions to complete;
- this subject in PDF and LaTeX formats.

No external Python package is required.

1. Formalise the problem – 10 minutes

Complete the following description before programming:

a) What interaction phenomenon are we trying to observe?

b) What is one input trace? What defines its beginning and its end?

c) Give a formal definition of the input database D.

d) What should the algorithm return?

e) What is the support unit: occurrence, session or user?

f) Which data split would be required to claim that a pattern generalises to new users?

Use the following formalisation as a target:

- Σ: alphabet of observable actions;
- $D = \{S_1, \ldots, S_n\}$: database of interaction sequences over Σ;
- $P \sqsubseteq S$: pattern P is a subsequence of trace S;
- $support(P) = |\{S \in D \mid P \sqsubseteq S\}|/|D|$;
- output: all patterns whose support is at least `min_support`, up to a chosen maximum length.

2. Reconstruct interaction sequences – 15 minutes

Complete `build_session_sequences`. The function must:

1. group rows by `session_id`;
2. order events using `timestamp_s`, then `event_id` to resolve ties;
3. return one list of actions per session;
4. retain the user associated with each session.

Checkpoint. You should obtain 789 events, 96 sessions and 36 users. Display five complete sequences and inspect their lengths. Why would grouping rows directly by user create questionable patterns?

3. Subsequence and trace support – 15 minutes

Complete `contains_subsequence` and `trace_support`.

a) A pattern must preserve order, but its elements need not be consecutive.

b) Count a pattern at most once per session, even if it occurs several times.

c) Compute session support for:

- $\langle view_hint, submit \rangle$;
- $\langle submit, read_feedback, retry \rangle$;
- $\langle view_example, bookmark \rangle$.

d) Explain why occurrence count and trace support answer different questions.

Test your function on small examples before using the complete dataset:

```
assert contains_subsequence(["a", "x", "b", "c"], ["a", "b"])
assert not contains_subsequence(["b", "a", "c"], ["a", "b"])
assert contains_subsequence(["a", "a", "b"], ["a", "b"])
```

4. Project a sequence database – 20 minutes

For a prefix P, PrefixSpan constructs a projected database containing only the suffix that follows the earliest embedding of P in each trace.

Example:

$$S = \langle a, x, b, c, b, d \rangle, \quad P = \langle a, b \rangle$$

The earliest embedding of P ends at the first b, so the projected suffix is $\langle c, b, d \rangle$.

Complete:

- `project_sequence`;
- `project_database`;
- `frequent_extensions`.

When counting extensions, count each action at most once in a projected suffix. For example, `[a,b,a]` contributes one trace to the support of `a`, not two.

Checkpoint. Projection with the prefix $\langle open_question, view_hint \rangle$ should retain 68 suffixes. With an absolute minimum count of 10, list the possible extensions and their counts.

5. Complete and use PrefixSpan – 30 minutes

Complete the recursive function `prefixspan`. At each recursive call:

1. find the frequent one-action extensions in the current projected database;
2. append each extension to the current prefix;
3. record its support;
4. project the current database after the extension;
5. continue until `maximum_length` is reached.

Use `math.ceil(min_support * len(database))` to convert a relative threshold into an absolute count.

Run the algorithm with `maximum_length=4` and compare:

```
min_support = 0.10, 0.20, 0.35
```

For each threshold:

a) count all patterns and patterns of length at least two;

b) display the twelve most frequent non-singleton patterns;

c) identify patterns that are frequent but uninformative;

d) identify at least two patterns that describe help-seeking or difficulty;

e) explain why decreasing support produces many redundant outputs.

Checkpoint. With `min_support=0.20`, the implementation should return 302 patterns, including 290 patterns of length at least two.

6. Session support and user support – additional work

The pattern $\langle view_example, bookmark \rangle$ is deliberately repeated by a small group of very active users.

Implement `user_support`: a user supports a pattern when at least one of their sessions contains it. Do not concatenate sessions, since that could create a pattern whose first action occurs in one session and whose last action occurs in another.

Compare session support and user support for the three patterns from Part 3. What population-level conclusion would be misleading if only session support were reported?

7. Generalisation to unseen users – additional work

Create a reproducible 70/30 split of the 36 users. Keep every session of a user in the same subset.

a) Mine patterns only from the training users with `min_support=0.20`.

b) Recompute the support of these patterns in the test users' sessions.

c) Report five patterns that generalise well.

d) Report three patterns whose support decreases the most.

e) Explain why a random split of events would constitute data leakage.

8. Temporal constraints – additional work

The current definition allows an arbitrary number of events between two pattern elements. Implement:

```
contains_subsequence_max_gap(sequence, pattern, maximum_gap)
```

Compare unrestricted support, `maximum_gap=0` and `maximum_gap=2` for:

$$\langle open_question, view_hint, submit \rangle.$$

Conclude in a short paragraph: how do the support unit, gap constraint and session definition change the meaning of the discovered patterns?

Optional extension: sequences of non-singleton itemsets

Group events into five-second windows and represent every window as an itemset. Describe the changes required so that PrefixSpan can extend a prefix either:

- by adding an item to the last itemset;
- or by appending a new itemset.

Implementation is optional. The formal description is sufficient.