Sequential Pattern Discovery using Equivalence Classes (SPADE)

Introduction

Sequential pattern mining is a topic of data mining concerned with finding statistically relevant patterns between data examples where the values are delivered in a sequence. It is usually presumed that the values are discrete, and thus time series mining is closely related, but usually considered a different activity. Sequential pattern mining is a special case of structured data mining. There are several key traditional computational problems addressed within this field. These include building efficient databases and indexes for sequence information, extracting the frequently occurring patterns, comparing sequences for similarity, and recovering missing sequence members.

Data Cleaning

Since the given dataset was in **.seq** format, there was no missing values in the dataset as according to the data description, the number of hits on a website and the number of visits on the web can be variable. So, the entire numerical data was good for use. However, the introduction to the data that was present was truncated so that no garbage values were included.

Implementation

The algorithm goes as follows:-

SPADE (min_sup, *D*):
    F_1 = { frequent items or 1-sequences };
    F_2 = { frequent 2-sequences };
    ξ = { equivalence classes $[X]_{\theta 1}$ };
    for all [X] ϵ ξ **do** *Enumerate-Frequent-Seq*([X]);

In the horizontal format the database consists of a set of input-sequences. Each input-sequence has a set of events, along with the items contained in the event. In contrast our algorithm uses a vertical database format, where we maintain a disk-based id-list for each item. Each entry of the id-list is a (sid, eid) pair where the item occurs. This enables us to check support via simple id-list joins.

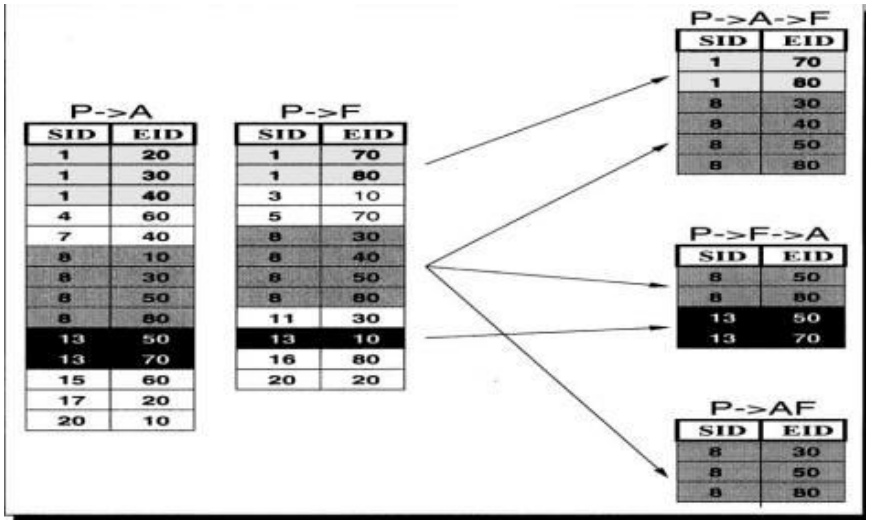
Computing F_1: Given the vertical id-list database, all frequent 1-sequences can be computed in a single database scan. For each database item, we read its id-list from the disk into memory. We then scan the id-list, incrementing the support for each new sid encountered.

Computing F_2: Let N = $|F_1|$ be the number of frequent items, and A the average id-list size in bytes. A naive implementation for computing the frequent 2-sequences requires ${}^{N}C_2$ id-list joins for all pairs of items. The amount of data read is $A \cdot N \cdot (N-1)/2$, which corresponds to around N/2 data scans. This is clearly inefficient. Instead of the naive method we propose two alternate solutions:

Enumerating frequent sequences of a class:-

Enumerate-Frequent-Seq(S):
    for all atoms $A_i \in S$
    $T_i = \{\}$
        for all atoms $A_j \in S$, with $j \geq i$ **do**
            $R = A_i \cup A_j$;
            if (Prune(**R**) == **FALSE**) **then**
            $\boldsymbol{L}(R) = \boldsymbol{L}(A_i) \cap \boldsymbol{L}(A_j)$
            if $\sigma(R) \geq$ *min_sup* **then**
                $T_i = T_i \cup \{R\}$;
                $F_{|R|} = F_{|R|} \cup \{R\}$;
        end
        if (Depth-First-Search) **then** *Enumerate-Frequent-Seq*(T_i);
    end
    if(Breadth-First-Search) **then**
        for all $T_i \neq \{\}$ **do** *Enumerate-Frequent-Seq*(T_i);

Following this steps, the list is joined temporally:-



Finally the sequence is pruned as follows:-

Prune (β):
    for all (k-1)-subsequences, α<β
    if ([α_1] has been processed, and α ∉ F_{k-1}) **then**
        return TRUE;
    return FALSE;

Conclusion

- **Pros**
  - Allows for different search strategies: Breadth-first and depth-first search.
- **Cons**
  - A huge set of candidates could be generated.
  - Mining long sequential patterns.
    - Needs an exponent number of short candidates.
    - A length 100 sequential pattern needs 10^{30} candidate sequences.

GitHub Link

The GitHub Link for the project:- https://github.com/archishman97/SPADE-Implementation