Mining Rare Association Rules by Discovering Quasi-Functional Dependencies: An Incremental Approach

(G. Bruno, P. Garza, E. Quintarelli)

Karel Vaculík

Outline

- Introduction
- Background
- Outlier Detection
- Incremental Outlier Detection
- Experimental Results

Introduction

- Rare events ≈ anomalies ≈ outliers
- Identification of rare rules by using quasi-functional dependencies
- Database is dynamic
- Application:
  - Erroneous data correction
  - Meaning of (correct) exceptions

Introduction

- Rare rule detection process:
  1. Inference of the normal behavior of objects by extracting frequent rules (rules are in the form of quasi-functional dependencies)
  2. Analysis of rare violations

Background

- Functional dependency
  a) Relational databases:

     $X \rightarrow Y$ (X, Y sets of attributes) $\Leftrightarrow$

     $\forall$ tuple t_1, t_2: $(t_1[X] = t_2[X]) \Rightarrow (t_1[Y] = t_2[Y])$
  b) XML:
     - Dependency between elements
     - Using tree tuples which describe the paths in XML

Background

- Example: Functional dependency RegN → Brand

<u>RegN</u>	Brand	Category	...	Wheels	City	Province
A001	Kymco	Scooter	...	2	Verona	Verona
A002	Fiat	Car	...	4	Verona	Verona
A003	Piaggio	Scooter	...	2	Piobesi	Torino
A004	Aprilia	Scooter	...	2	Piobesi	Cuneo
A005	Toyota	Car	...	4	Recetto	Milano
A006	Possl	Auto caravan	...	4	Recetto	Novara
...	...	...	...	...	...	...

Background

- Example: Functional dependency RegN → Brand

RegN	Brand	Category	...	Wheels	City	Province
A001	Kymco	Scooter	...	2	Verona	Verona
A002	Fiat	Car	...	4	Verona	Verona
A003	Piaggio	Scooter	...	2	Piobesi	Torino
A004	Aprilia	Scooter	...	2	Piobesi	Cuneo
A005	Toyota	Car	...	4	Recetto	Milano
A006	Possl	Auto caravan	...	4	Recetto	Novara
...	...	...	...	...	...	...

$\forall$ tuple t_1, t_2: $(t_1[\text{RegN}] = t_2[\text{RegN}]) \Rightarrow (t_1[\text{Brand}] = t_2[\text{Brand}])$

Background

- Given an association rule A ⇒ B:
  - Support: $s(A \Rightarrow B) = s(A \cup B) = \dfrac{count(A \cup B)}{|D|}$
  - Confidence: $c(A \Rightarrow B) = \dfrac{s(A \cup B)}{s(A)}$
  - Dependency degree: $p = \sum_{i \in AR} s_i \cdot c_i$

AR ... set of all association rules relating attributes X and Y

Background

- p = 1:
  - Functional dependency

  ⇔ All rules have a confidence equal to 100%
- *threshold* ≤ p ≤ 1:
  - Quasi-functional dependency

Background

- Example: Functional dependency RegN → Brand

data:

RegN	Brand	...
A001	Kymco	...
A002	Fiat	...
A003	Piaggio	...
A004	Aprilia	...
...	...	...

$$p = \sum_{i \in AR} s_i \cdot c_i = 1$$

association rules:

Body	Head	Sup	Conf
RegN=A001	Brand=Kymco	1	100%
RegN=A002	Brand=Fiat	1	100%
RegN=A003	Brand=Piaggio	1	100%
RegN=A004	Brand=Aprilia	1	100%
...	...	...	...

Background

- Example: Quasi-functional dep. City → Province

data:

...	City	Province
...	Verona	Verona
...	Verona	Verona
...	Piobesi	Torino
...	Piobesi	Cuneo
...	Recetto	Milano
...	Recetto	Novara
...	...	...

$$p = \sum_{i \in AR} s_i \cdot c_i < 1$$

association rules:

Body	Head	Sup	Conf
City=Verona	Province= Verona	19.5%	100%
City=Piobesi	Province= Torino	45.1%	75.2%
City=Piobesi	Province= Cuneo	14.9%	24.8%
City=Recetto	Province= Novara	28.7%	99.7%
City=Recetto	Province= Milano	0.1%	0.3%
...	...	...	...

Outlier Detection

- Association rules and the related quasi-functional dependencies are stored in a relational database, items are stored separately

Outlier Detection

- Procedure:
  1. Retrieve dependencies with degree > *degree_threshold*
  2. Select rules related to these dependencies, where confidence < *confidence_threshold*
  3. If the confidence of some rule is very low (in comparison with confidence of other rules), it is likely to be an error, otherwise, a correct exception

     (error can also be confirmed / disproved by using another database)

Outlier Detection

- Example rules:
  1. City=Piobesi ⇒ Province=Cuneo [s=14.9%, c=24.8%]
  2. City=Recetto ⇒ Province=Milano [s=0.1%, c=0.3%]

Outlier Detection

- Example rules:
  1. City=Piobesi ⇒ Province=Cuneo [s=14.9%, c=24.8%]
  2. ~~City=Recetto ⇒ Province=Milano [s=0.1%, c=0.3%]~~

     error

Incremental Outlier Detection

- Tuples are inserted and deleted over time
  $\Rightarrow$ Set of anomalies has to be updated
- Repetitive extraction is unfeasible

Incremental Outlier Detection

- Procedure:
  1. Add new attributes start time (T_s) and end time (T_e) to define the time interval, in which the tuple is current in the database

Incremental Outlier Detection

- Procedure:
  1. Add new attributes start time (T_s) and end time (T_e) to define the time interval, in which the tuple is current in the database

  Example:

RegN	Brand	Category	…	Wheels	City	Province	T_s	T_e
A001	Kymco	Scooter	…	2	Verona	Verona	0	Now
A002	Fiat	Car	…	4	Verona	Verona	1	Now
A003	Piaggio	Scooter	…	2	Piobesi	Torino	1	5
A004	Aprilia	Scooter	…	2	Piobesi	Cuneo	2	Now
A005	Toyota	Car	…	4	Recetto	Milano	6	Now
…	…	…	…	…	…	…	…	…

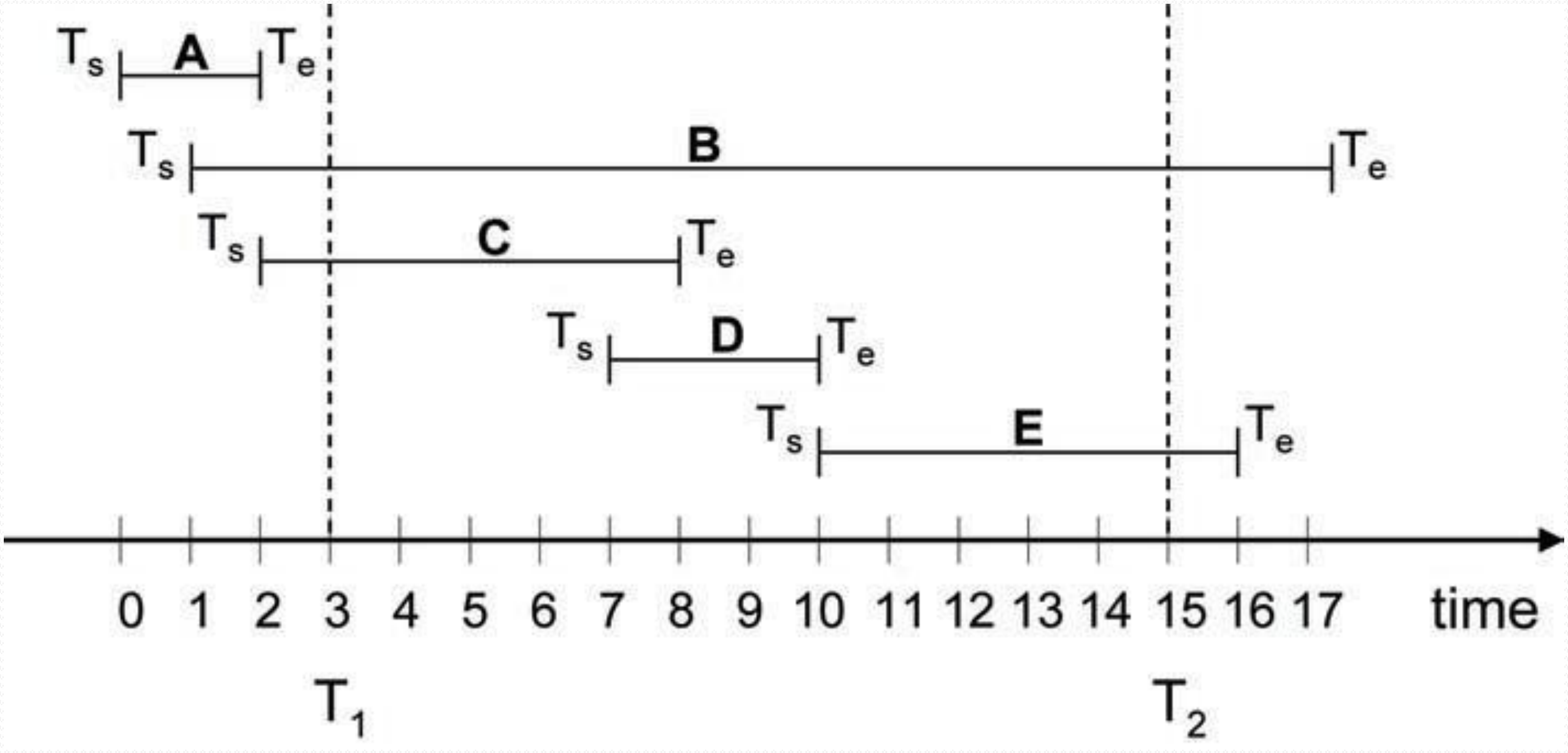
Incremental Outlier Detection

- Procedure:
  1. Add new attributes start time (T_s) and end time (T_e) to define the time interval, in which the tuple is current in the database
  2. Given a time interval $[T_1, T_2]$ and rules with

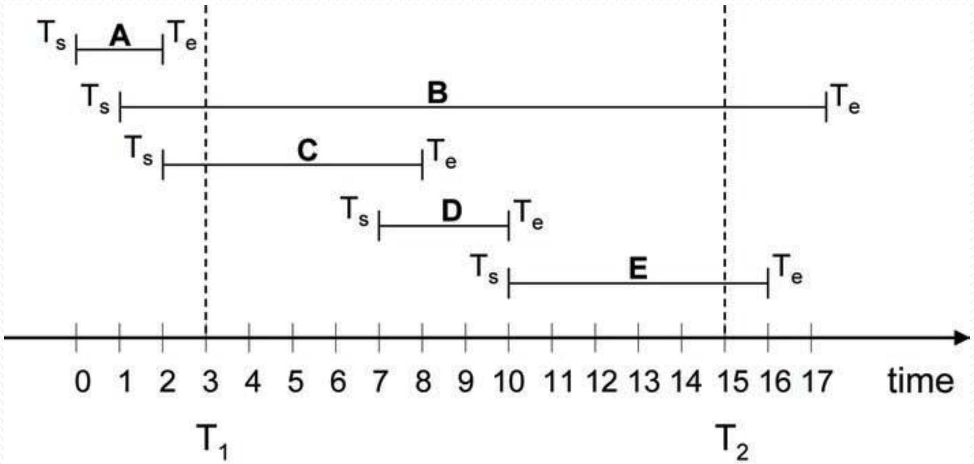
     $T_1 < T_s < T_2$ (insertion) *xor* $T_1 < T_e < T_2$ (deletion), recompute support and confidence of relevant rules
     - update ≈ deletion + insertion
  3. Recompute degrees of relevant dependencies

Incremental Outlier Detection

- Example:



Incremental Outlier Detection



	RegN	Brand	Category	Wheels	City	Province	T_s	T_e
A	A001	Kymco	Scooter	2	Verona	Verona	0	2
B	A002	Fiat	Car	4	Verona	Verona	1	Now
C	A003	Toyota	Car	4	Recetto	Milano	2	8
D	A004	Aprilia	Scooter	2	Piobesi	Cuneo	7	10
E	A005	Piaggio	Scooter	2	Piobesi	Cuneo	10	Now

Experimental Results

- Update time per tuple:
  - After each insertion / deletion
  - Incremental approach is better
- Update time per set of tuples
  - Periodically / on demand from last computation
  - Incremental approach (one tuple at a time) is better on small number of association rules
  - Non-incremental approach is better on large number of rules

Thank you