SLIM: Directly Mining Descriptive Patterns

Koen Smets    Jilles Vreeken

SDM12
Pattern mining

- Discovering all patterns
  - Fulfilling constraints
  - Potentially interesting
Pattern mining has drawbacks

- Discovering all patterns
  - Fulfilling constraints, but too many
  - Potentially interesting, but redundant
Pattern mining has drawbacks
Pattern set mining

- Discovering all patterns
  - Fulfilling constraints, but too many
  - Potentially interesting, but redundant

- Discovering high-quality set of patterns
  - Small
  - Useful
Pattern mining has drawbacks

Pattern set mining

- Discovering all patterns
  - Fulfilling constraints, but too many
  - Potentially interesting, but redundant

- Discovering high-quality set of patterns
  - Small
  - Useful

- Identify the best set of patterns
  - Together describe the data best
MDL approach to pattern set mining

(Siebes et al 2006 / Vreeken et al 2011)

Minimum Description Length principle

Given a set of models $M$, the best model is

$$\text{argmin}_{M \in M} L(M) + L(D | M)$$

MDL for pattern set mining

▶ Model = set of patterns
▶ Lossless compression of the data

Properties

▶ Non-redundant
▶ Not overly simple
▶ Not overly complex
Minimum Description Length principle

Given a set of models $\mathcal{M}$, the best model is

$$\arg\min_{M \in \mathcal{M}} L(M) + L(\mathcal{D} | M)$$
MDL approach to pattern set mining
(Siebes et al. 2006 / Vreeken et al. 2011)

Minimum Description Length principle
Given a set of models $\mathcal{M}$, the best model is

$$\arg\min_{M \in \mathcal{M}} L(M) + L(\mathcal{D} | M)$$

MDL for pattern set mining

- Model = set of patterns
- Lossless compression of the data
MDL approach to pattern set mining

(Siebes et al 2006 / Vreeken et al 2011)

Minimum Description Length principle
Given a set of models $\mathcal{M}$, the best model is

$$\arg\min_{M \in \mathcal{M}} L(M) + L(D | M)$$

MDL for pattern set mining

- Model = set of patterns
- Lossless compression of the data
- Properties
  - Non-redundant
  - Not overly simple
  - Not overly complex
MDL approach to pattern set mining
(Siebes et al 2006 / Vreeken et al 2011)

Transaction database

A B C
A B C
A B C
A B C
A B C
A B C
A B
A
B
MDL approach to pattern set mining
(Siebes et al 2006 / Vreeken et al 2011)

### Code table

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Code</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>0.85bits</td>
<td>5</td>
</tr>
<tr>
<td>A</td>
<td>2.17bits</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>2.17bits</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>--</td>
<td>0</td>
</tr>
</tbody>
</table>

### Transaction database

- A B C
- A B C
- A B C
- A B C
- A B C
- A B
- A
- B
MDL approach to pattern set mining  
*(Siebes et al 2006 / Vreeken et al 2011)*

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Code</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>0.85bits</td>
<td>5</td>
</tr>
<tr>
<td>A</td>
<td>2.17bits</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>2.17bits</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>—</td>
<td>0</td>
</tr>
</tbody>
</table>

**Transaction database**

- A B C
- A B C
- A B C
- A B C
- A B C
- A B C
- A B
- A
- B

**Covered database**

- A B C
- A B C
- A B C
- A B C
- A B C
- A B C
- A B
- A
- B

**Encoded database**

- 
- 
- 
- 
- 
- 
- 
- 
-
How to mine the optimal code table?

Easier said than done

The number of possible code tables is huge

Exponential in the number of candidate itemsets

No useful structure to exploit

Hence, we resort to heuristics
How to mine the optimal code table?

▶ Easier said than done
How to mine the optimal code table?

▶ Easier said than done

▶ The number of possible code tables is huge
  ▶ Exponential in the number of candidate itemsets
How to mine the optimal code table?

- Easier said than done
  - The number of possible code tables is huge
    - Exponential in the number of candidate itemsets
  - No useful structure to exploit
How to mine the optimal code table?

- Easier said than done

- The number of possible code tables is huge
  - Exponential in the number of candidate itemsets
  - No useful structure to exploit

- Hence, we resort to heuristics
**Krimp**

*(Siebes et al 2006 / Vreeken et al 2011)*

---

**Standard two phase approach**

1. Mining candidate patterns

2. Considering candidates once in a fixed order


**KRIMP**

*(Siebes et al 2006 / Vreeken et al 2011)*

---

**Standard two phase approach has drawbacks**

1. Mining candidate patterns

2. Considering candidates once in a fixed order
Standard two phase approach has drawbacks

1. Mining candidate patterns is expensive

2. Considering candidates once in a fixed order
**KRIMP**  
*(Siebes et al 2006 / Vreeken et al 2011)*

Many many patterns

K\textsc{rimp} select pattern

K\textsc{rimp} accept / reject

K\textsc{rimp} add to code table

K\textsc{rimp} compress database

Many many patterns

**Standard two phase approach has drawbacks**

1. Mining candidate patterns is expensive
   - Lower support threshold correspond to better results

2. Considering candidates once in a fixed order
**KRIMP**

*(Siebes et al 2006 / Vreeken et al 2011)*

---

**Standard two phase approach has drawbacks**

1. Mining candidate patterns is **expensive**
   - Lower support threshold correspond to better results
   - Pattern explosion prohibits detailed analysis

2. Considering candidates once in a fixed order
**KRIMP**

*(Siebes et al 2006 / Vreeken et al 2011)*

Many many patterns

Database

![Diagram of the KRIMP process]

**Standard two phase approach has drawbacks**

1. **Mining candidate patterns is expensive**
   - Lower support threshold correspond to better results
   - Pattern explosion prohibits detailed analysis
   - Most candidates will be rejected

2. Considering candidates once in a fixed order
Standard two phase approach has drawbacks

1. Mining candidate patterns is expensive
   - Lower support threshold correspond to better results
   - Pattern explosion prohibits detailed analysis
   - Most candidates will be rejected

2. Considering candidates once in a fixed order is suboptimal
**KRIMP**

*(Siebes et al 2006 / Vreeken et al 2011)*

---

**Standard two phase approach has drawbacks**

1. **Mining candidate patterns is expensive**
   - Lower support threshold correspond to better results
   - Pattern explosion prohibits detailed analysis
   - Most candidates will be rejected

2. **Considering candidates once in a fixed order is suboptimal**
   - Rejecting candidates that could be useful later on
Many many patterns

\[ \text{Database} \rightarrow \text{SLIM} \]

- Select pattern
- MDL
- Add to code table
- Compress database
- Accept / reject

\[ \text{Code table} \]
Maximise compression locally

- Select the best addition out of all candidates
Maximise compression locally

- Select the best addition out of all candidates
- Converging quickly to better compression
Maximise compression locally

► Select the best addition out of all candidates

► Converging quickly to better compression

![Graph showing relative compression (L%) vs. number of iterations for Adult dataset. The graph compares Kramp and Krimp algorithms.]
Maximise compression locally

- Select the best addition out of all candidates is infeasible
  - Reordering all candidates means recompressing database
  - Converging quickly to better compression takes 2 months
Greedily construct code table bottom-up

- Start with code table containing only singletons

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Code table</th>
<th>Covered database</th>
<th>Encoded database</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>
Greedyly construct code table bottom-up

- Start with code table containing only singletons
- Optimise current code table locally

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Code table</th>
<th>Covered database</th>
<th>Encoded database</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td>A B C</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>A B C</td>
</tr>
</tbody>
</table>
Greedily construct code table bottom-up

- Start with code table containing only singletons
- Optimise current code table locally
  - Consider all pairwise combinations of itemsets in code table
Greedily construct code table bottom-up

- Start with code table containing only singletons
- Optimise current code table locally
  - Consider all pairwise combinations of itemsets in code table
  - Add candidate with highest gain in total compression
Greedily construct code table bottom-up

- Start with code table containing only singletons
- Optimise current code table locally
  - Consider all pairwise combinations of itemsets in code table
  - Add candidate with highest gain in total compression

Covered database

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Code table</th>
<th>Covered database</th>
<th>Encoded database</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Encoded database
Greedily construct code table bottom-up

- Start with code table containing only singletons
- Optimise current code table locally
  - Consider all pairwise combinations of itemsets in code table
  - Add candidate with highest gain in total compression

- Continue to refine current code table
Greedily construct code table bottom-up

- Start with code table containing only singletons
- Optimise current code table locally
  - Consider all pairwise combinations of itemsets in code table
  - Add candidate with highest gain in total compression

- Continue to refine current code table
Greedy construct code table bottom-up

- Start with code table containing only singletons
- Optimise current code table locally
  - Consider all pairwise combinations of itemsets in code table
  - Add candidate with highest gain in total compression
  - Remove code table elements that no longer contribute
- Continue to refine current code table

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Code table</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>A B</td>
<td></td>
</tr>
<tr>
<td>A C</td>
<td></td>
</tr>
<tr>
<td>B C</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Code table</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>A B C</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Covered database</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B</td>
<td>A B C</td>
</tr>
<tr>
<td>A B C</td>
<td>A B C</td>
</tr>
<tr>
<td>A</td>
<td>A B C</td>
</tr>
<tr>
<td>B</td>
<td>A B C</td>
</tr>
<tr>
<td>C</td>
<td>A B C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Encoded database</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>A C</td>
<td></td>
</tr>
<tr>
<td>B C</td>
<td></td>
</tr>
<tr>
<td>A B C</td>
<td></td>
</tr>
</tbody>
</table>
Greedily construct code table bottom-up

- Start with code table containing only singletons
- Optimise current code table locally
  - Consider all pairwise combinations of itemsets in code table
  - Add candidate with highest gain in total compression
  - Remove code table elements that no longer contribute
- Continue to refine current code table

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Code table</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Covered database</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
</tr>
<tr>
<td>A B C</td>
</tr>
<tr>
<td>A B C</td>
</tr>
<tr>
<td>A B C</td>
</tr>
<tr>
<td>A B C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Encoded database</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Greedily construct code table bottom-up

- Start with code table containing only singletons
- Optimise current code table locally
  - Consider all pairwise combinations of itemsets in code table
  - Add candidate with highest gain in total compression
  - Remove code table elements that no longer contribute
- Continue to refine current code table, or stop

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Code table</th>
<th>Covered database</th>
<th>Encoded database</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>A B</td>
<td></td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>A C</td>
<td></td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>B C</td>
<td></td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>A B C</td>
<td></td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>A B</td>
<td></td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>A C</td>
<td></td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>B C</td>
<td></td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>A B C</td>
<td></td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>A B</td>
<td></td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>A C</td>
<td></td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>B C</td>
<td></td>
<td>A B C</td>
<td></td>
</tr>
<tr>
<td>A B C</td>
<td></td>
<td>A B C</td>
<td></td>
</tr>
</tbody>
</table>

Covered database and encoded database visualised in the diagram.
Greedily construct code table bottom-up

- Selecting the candidate pair with highest gain
- Converging quickly to better compression
Greedily construct code table bottom-up

- Selecting the candidate pair with highest gain is expensive
  - Need to compress database for each candidate
  - Converging quickly to better compression takes 1 week
Accurate and efficient heuristic to estimate gain

- Calculate gain through usage counts of code pairs
  - Disregard effects on usage of other codes
Accurate and efficient heuristic to estimate gain

- Calculate gain through usage counts of code pairs
  - Disregard effects on usage of other codes
Accurate and efficient heuristic to estimate gain

- Calculate gain through usage counts of code pairs
  - Disregard effects on usage of other codes
Accurate and efficient heuristic to estimate gain

- Calculate gain through usage counts of code pairs
  - Disregard effects on usage of other codes
- Use branch-and-bound to find highest estimated gain
Accurate and efficient heuristic to estimate gain

- Calculate gain through usage counts of code pairs
  - Disregard effects on usage of other codes
- Use branch-and-bound to find highest estimated gain
**SLIM vs. KRIMP: setup**

- Use wide range of benchmark and real datasets
- Limit processing time to 24 hours
- For KRIMP, mine itemsets with lowest feasible minsup
Better compression
comparing results after at most computing 1 day

![Bar chart showing difference in relative compression (ΔL%) for various datasets. The x-axis represents different datasets including Abstracts, Accidents, Adult, BMS-pos, BMS-wv1, Chess (k-k), Chess (kr-k), Connect-4, DNA amp., Ionosphere, Letter recog., Mammals, MCADD, Mushroom, Pen digits, Plants, Pumsb, Pumsbstar, and Waveform. The y-axis represents the difference in relative compression (ΔL%) ranging from -10 to 80. The chart compares Slim and Krimp.]
Better compression comparing results after at most computing 1 day

- High difference
Better compression comparing results after at most computing 1 day

- High difference → mine at lower minsup threshold
Better compression comparing results after at most computing 1 day

- High difference → mine at lower minsup threshold
  - Impossible to mine all of those

![Graph showing difference in relative compression (\(\Delta L\%\)](image-url)
Better compression
comparing results after at most computing 1 day

- High difference → mine at lower minsup threshold
  - Impossible to mine all of those, need only a few good ones
Code tables at least as characteristic validated using classification experiment

<table>
<thead>
<tr>
<th>Code Table</th>
<th>Adult Chess</th>
<th>Chess (kr-k)</th>
<th>Connect-4</th>
<th>Ionosphere</th>
<th>Letter recog.</th>
<th>Mushroom</th>
<th>Pen digits</th>
<th>Waveform</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(k-k)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(kr-k)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Accuracy (%)

- Slim
- Krimp
Code tables at least as characteristic validated using classification experiment

- Split data and build code table per class
Code tables at least as characteristic validated using classification experiment

- Split data and build code table per class
- Assign class label based on encoded length
Code tables at least as characteristic validated using classification experiment

- Split data and build code table per class
- Assign class label based on encoded length
SLIM vs. KRIMP: highlights

- Describing data more succinct
SLIM vs. KRIMP: highlights

- Describing data more succinct
- Providing high-quality descriptions
SLIM vs. KRIMP: highlights

- Describing data more succinct
- Providing high-quality descriptions
- Instantiating fewer candidate patterns
**SLIM vs. KRIMP: highlights**

- Describing data more succinct
- Providing high-quality descriptions
- Instantiating fewer candidate patterns
- Converging faster
SLIM vs. KRIMP: highlights

- Describing data more succinct
- Providing high-quality descriptions
- Instantiating fewer candidate patterns
- Converging faster
  - most time is invested in tail of convergence
SLIM: Directly Mining Descriptive Patterns

- Iteratively refining current description of the data
- Reconsidering candidates providing highest estimated gain
SLIM: Directly Mining Descriptive Patterns

- Iteratively refining current description of the data
- Reconsidering candidates providing highest estimated gain
- Any-time & parameter-free
SLIM: Directly Mining Descriptive Patterns

- Iteratively refining current description of the data
- Reconsidering candidates providing highest estimated gain
- Any-time & parameter-free
  - Providing detail only when necessary
  - Feasible to analyse large and dense data in more detail
For implementation and further reading

K. Smets & J. Vreeken.
SLIM: Directly Mining Descriptive Patterns.
Proceedings of the SIAM International Conference on Data Mining (SDM), SIAM, 2012.

J. Vreeken, M. van Leeuwen & A. Siebes.
KRIMP: Mining Itemsets that Compress.