Questions of the day

How can we find useful patterns?

&

How can we use patterns?
Standard pattern mining

For a database $db$

- a pattern language $\mathcal{P}$ and a set of constraints $\mathcal{C}$

the goal is to find the set of patterns $S \subseteq \mathcal{P}$ such that

- each $p \in \mathcal{P}$ satisfies each $c \in \mathcal{C}$ on $db$, and $S$ is maximal

That is, find all patterns that satisfy the constraints
Problems in pattern paradise

The pattern explosion
- high thresholds
  few, but well-known patterns
- low thresholds
  a gazillion patterns

Many patterns are redundant

Unstable
- small data change,
  yet different results
- even when distribution
  did not really change
The Wine Explosion

the Wine dataset has 178 rows, 14 columns
Be careful what you wish for

The root of all evil is,

- we ask for all patterns that satisfy some constraints,
- while we want a small set that shows the structure of the data

In other words, we should ask for a set of patterns such that

- all members of the set satisfy the constraints
- the set is optimal with regard to some criterion
Intuitively

A pattern identifies local properties of the data, e.g., itemsets.

\( M \) patterns

\( D \) a toy 0-1 dataset
Intuition Bad
Intuition Good
Optimality and Induction

What is the **optimal** set?
- the set that generalises the data best
- generalisation = induction
  we should employ an inductive principle

So, which principle should we choose?
- observe: patterns are descriptive for local parts of the data
- MDL is *the* induction principle for descriptions

Hence, MDL is a **natural** choice
The Minimum Description Length (MDL) principle

given a set of models $\mathcal{M}$, the best model $M \in \mathcal{M}$ is that $M$ that minimises

$$L(M) + L(D|M)$$

in which

$L(M)$ is the length, in bits, of the description of $M$

$L(D|M)$ is the length, in bits, of the description of the data when encoded using $M$

(see, e.g., Rissanen 1978, 1983, Grünwald, 2007)
Does this make sense?

Models describe the data
- that is, they capture regularities
- hence, in an abstract way, they compress it

MDL makes this observation concrete:

*the best model gives the best lossless compression*
Does this make sense?

MDL is related to Kolmogorov Complexity

the complexity of a string is the length of the smallest program that generates the string, and then halts

Kolmogorov Complexity is the ultimate compression

- recognizes and exploits any structure
- uncomputable, however
The Minimum Description Length (MDL) principle

given a set of models $\mathcal{M}$, the best model $M \in \mathcal{M}$ is that $M$ that minimises

$$L(M) + L(D|M)$$

in which

$L(M)$ is the length, in bits, of the description of $M$

$L(D|M)$ is the length, in bits, of the description of the data when encoded using $M$

(see, e.g., Rissanen 1978, 1983, Grünwald, 2007)
How to use MDL

To use MDL, we need to define

- how many bits it takes to encode a model
- how many bits it takes to encode the data given this model

... what’s a bit?
How to use MDL

To use MDL, we need to define
- how many bits it takes to encode a model
- how many bits it takes to encode the data given this model

Essentially...
- defining an encoding $\leftrightarrow$ defining a prior
- codes and probabilities are tightly linked: higher probability $\leftrightarrow$ shorter code

So, although we don’t know overall probabilities
- we can exploit knowledge on local probabilities
Model

$I = \{ A, B, C, D, E \}$

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Code</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>D</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
</tr>
</tbody>
</table>

(Vreeken et al 2011 / Siebes et al 2006)
<table>
<thead>
<tr>
<th>Itemset</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>A C</td>
<td>0</td>
</tr>
<tr>
<td>B D</td>
<td>0</td>
</tr>
<tr>
<td>C E</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
</tr>
</tbody>
</table>

Transaction t: B C E
### Code Table

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>E</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
</tr>
</tbody>
</table>

### Transaction $t$

<p>| B | C | E |</p>
<table>
<thead>
<tr>
<th>Code Table</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Itemset</strong></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>E</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transaction t</th>
<th>Cover of t</th>
</tr>
</thead>
<tbody>
<tr>
<td>B C E</td>
<td>C E</td>
</tr>
</tbody>
</table>
### Code Table

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
</tr>
</tbody>
</table>

### Transaction t

- B
- C
- E

### Cover of t

- C
- E
- B
Encoding a database

Database
- A C E
- B
- A C
- A B C D
- B C E
- D
- B C D E
- A B D

Database Cover
- A C E
- B
- A C
- A C B D
- C E B
- D
- D
- B D C E
- B D A

Code Table

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

-
Optimal codes

For $c \in CT$ define:

$$usage(c) = | \{ t \in D \mid c \in Cover(CT, t) \} |$$

$$P(c \mid CT) = \frac{usage(c)}{\sum_{d \in CT} usage(d)}$$

The optimal code for the coding distribution $P$ assigns a code to $c \in CT$ with length:

$$L(c \mid CT) = - \log(P(c \mid CT))$$

(Shannon, 1948; Thomas & Cover, 1991)
Encoding a code table

The size of a code table $CT$ depends on

the left column
- length of itemsets as encoded with independence model

the right column
- the optimal code length

Thus, the size of a code table, is

$$L(CT | D) = \sum_{c \in CT: usage(c) \neq 0} L(c | ST) + L(c | CT)$$
Encoding a database

For $t \in D$ we have

$$L(t \mid CT) = \sum_{c \in \text{Cover}(CT,t)} L(c \mid CT)$$

Hence we have

$$L(D \mid CT) = \sum_{t \in D} L(t \mid CT)$$
The Total Size

The total size of data $D$ and code table $CT$ is

$$L(CT, D) = L(CT \mid D) + L(D \mid CT)$$

Note, we disregard **Cover** as it is identical for all $CT$ and $D$, and hence is only a constant
And now, the optimal code table...

Easier said than done

- the number of possible code tables is huge
- no useful structure to exploit

Hence, we resort to heuristics
**Krimp**

- mine candidates from $D$
- iterate over candidates
  - Standard Candidate Order
  - covers data greedily
  - no overlap
  - Standard Code Table Order
- select by MDL
  - better compression?
  - candidates may stay, reconsider old elements
SLIM – smarter KRIMP

(Smets & Vreeken, SDM’12)
## KRIMP in Action

| Dataset           | $|D|$ | $|F|$ | $|CT\setminus I|$ | $L\%$ |
|-------------------|-----|-----|----------------|-------|
| Accidents         | 340183 | 2881487 | 467 | 55.1 |
| Adult             | 48842  | 58461763 | 1303 | 24.4 |
| Letter Recog.     | 20000  | 580968767 | 1780 | 35.7 |
| Mushroom          | 8124   | 5574930437 | 442  | 24.4 |
| Wine              | 178    | 2276446  | 63   | 77.4 |
KRIMP in Action

# patterns

min-sup
KRIMP in Action
So, are \texttt{KRIMP} code tables good?

At first glance, yes

- the code tables are characteristic in the MDL-sense
  - they compress well

- the code tables are small
  - consist of few patterns

- the code tables are specific
  - contain relatively long itemsets

But, are these patterns useful?
The proof of the pudding

We tested the quality of the KRIMP code tables by

- classification (ECML PKDD’06)
- measuring dissimilarity (KDD’07)
- generating data (ICDM’07)
- concept-drift detection (ECML PKDD’08)
- estimating missing values (ICDM’08)
- clustering (ECML PKDD’09)
- sub-space clustering (CIKM’09)
- one-class classification/anomaly detection (SDM’11, CIKM’12)
- characterising uncertain 0-1 data (SDM’11)
- tag-recommendation (IDA’12)
Compression and Classification

Let’s assume
- two databases, \( db_1 \) and \( db_2 \)
- two corresponding code tables, \( CT_1 \) and \( CT_2 \)

Then, for an arbitrary transaction \( t \)

\[
L(t \mid CT_1) < L(t \mid CT_2) \rightarrow P(t \mid db_1) > P(t \mid db_2)
\]

Hence, the Bayes-optimal choice is to assign \( t \) to that database that gives the best compression.

(Vreeken et al 2011 / Van Leeuwen et al 2006)
**KRIMP for Classification**

**The KRIMP Classifier**
- split database on class
- find code tables
- classify by compression

**The Goal**
- validation of KRIMP

**The Results**
- expected ‘ok’
- on par with top classifiers
Classification by Compression

Two transactions encoded by two code tables

- can you spot the true class labels?
Clustering transaction data

Partition $D$ into $\mathcal{D}_1 \ldots \mathcal{D}_n$ such that $\sum L(CT_i, \mathcal{D}_i)$ is minimal

$k=6$, MDL optimal

(Van Leeuwen, Vreeken & Siebes 2009)
The Odd One Out

One-Class Classification (aka anomaly detection)
- lots of data for normal situation – insufficient data for target

Compression models the norm
- anomalies will have high description length $L(t \mid CT_{norm})$

Very nice properties
- performance high accuracy
- versatile no distance measure needed
- characterisation this part of $t$ can’t be compressed well

(Smets & Vreeken, 2011)
STREAMKRIMP

Given a stream of itemsets

(Van Leeuwen & Siebes, 2008)
Find the point where the distribution changed
Useful?

Yup! with Krimp we can do:

- Classification
- Dissimilarity Measurement and Characterisation
- Clustering
- Missing Value Estimation
- Anonymizing Data
- Detect concept drift
- Find similar tags (subspace clusters)
- and lots more...

And, *better* than the competition

- thanks to patterns! (and compression!) (yay!)
**Sqs - Selected Results**

**JMLR**
- support vector machine
- machine learning
- state [of the] art
- data set
- Bayesian network

**Pres. Addresses**
- unit[ed] state[s]
- take oath
- army navy
- under circumst.
- econ. public expenditure

(Tatti & Vreeken, KDD’12)
Beyond MDL...

Information Theory offers more than MDL

Modelling by Maximum Entropy (Jaynes 1957)
- principle for choosing probability distributions

Subjective Significance Testing
- is result $X$ surprising with regard to what we know?
- binary matrices (De Bie 2010, 2011) real-valued matrices (ICDM’11)

Subjective Interestingness
- the most informative itemset: the one that helps most to predict the data better (MTV) (KDD’11)
Conclusions

MDL is great for picking *important* and *useful* patterns

**KRIMP** approximates the MDL ideal *very well*
- *vast* reduction of the number of itemsets
- works for other pattern types equally well: itemsets, sequences, trees, streams, low-entropy sets

Local patterns and information theory
- naturally induce good classifiers, clusterers, distance measures
- with *instant* *characterisation* and *explanation*,
- and, *without* (explicit) parameters
MDL is great for picking important and useful patterns

KRIMP approximates the MDL ideal very well
- vast reduction of the number of itemsets
- works for other pattern types equally well: itemsets, sequences, trees, streams, low-entropy sets

Local patterns and information theory
- naturally induce good classifiers, clusterers, distance measures
- with instant characterisation and explanation,
- and, without (explicit) parameters