

MDL for Pattern Mining

Jilles Vreeken



4 June 2014 (TADA)



UNIVERSITÄT
DES
SAARLANDES



mpi max planck institut
informatik

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 $\mathcal{S} \subseteq \mathcal{P}$ such that

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

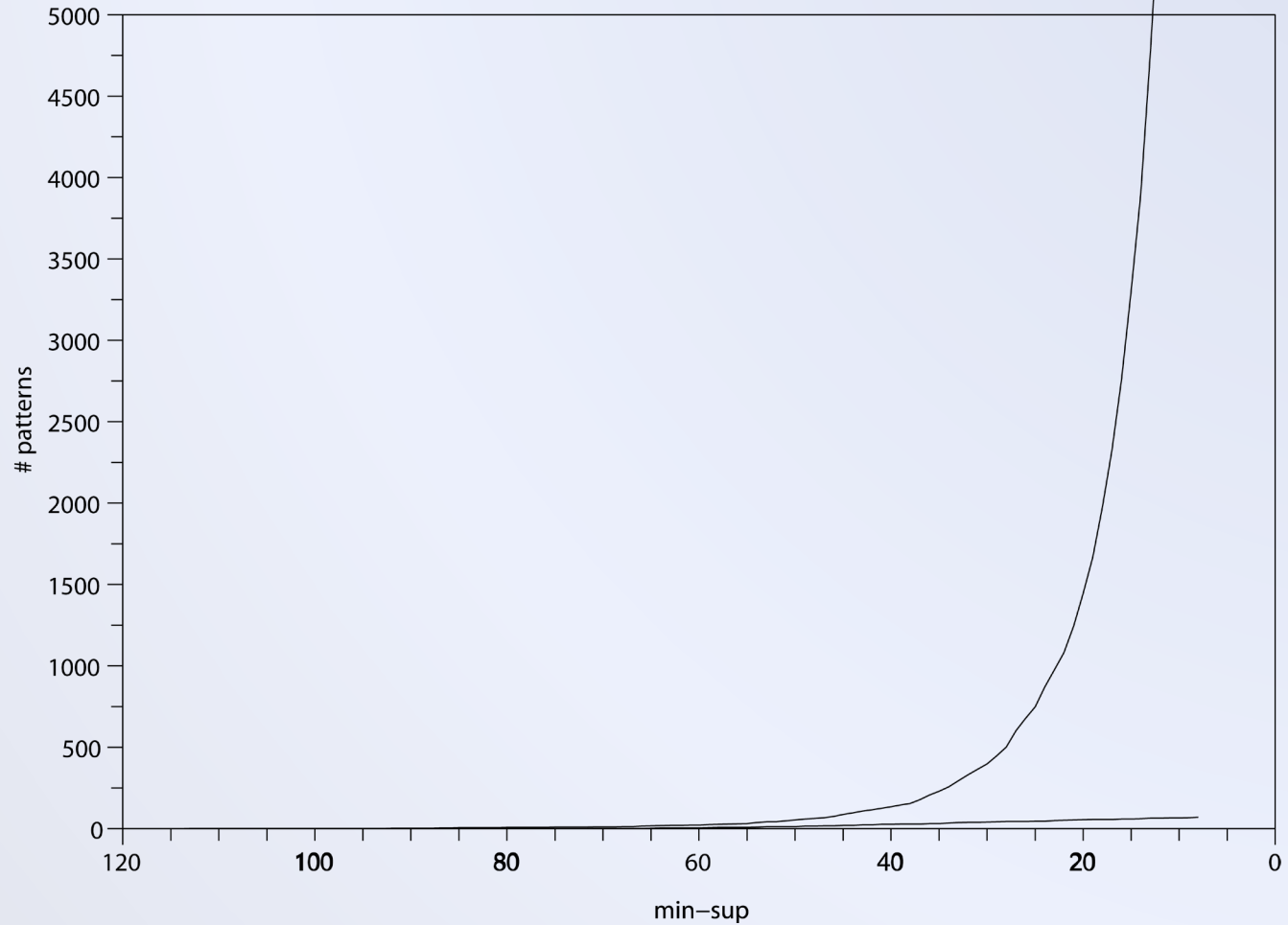
Many patterns are redundant

Unstable

- small data change,
yet different results
- even when distribution
did not really change



The Wine Explosion

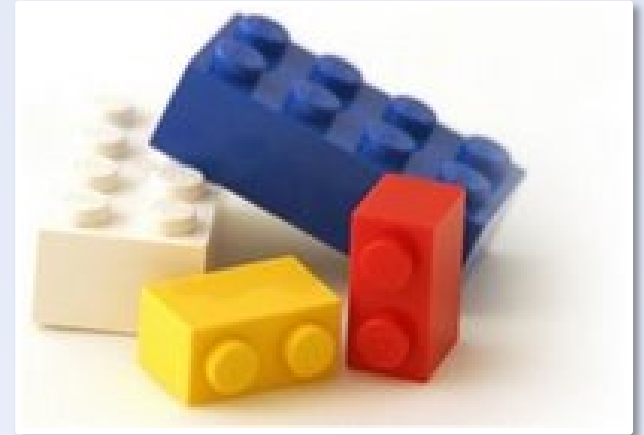


the *Wine* dataset has 178 rows, 14 columns

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

M

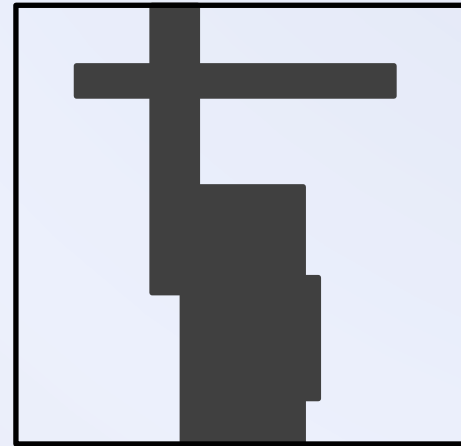


patterns

a pattern identifies
local properties
of the data

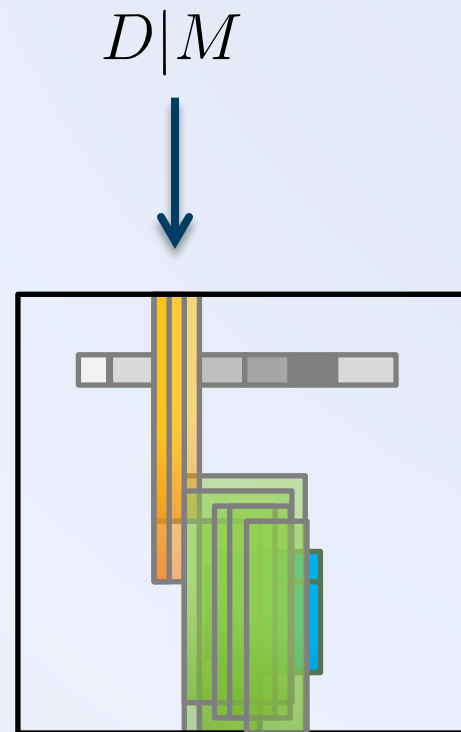
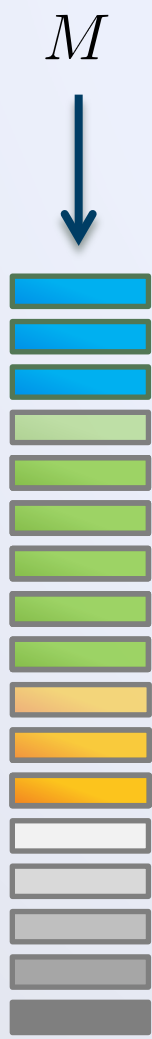
e.g. itemsets

D



a toy 0-1 dataset

Intuition **Bad**

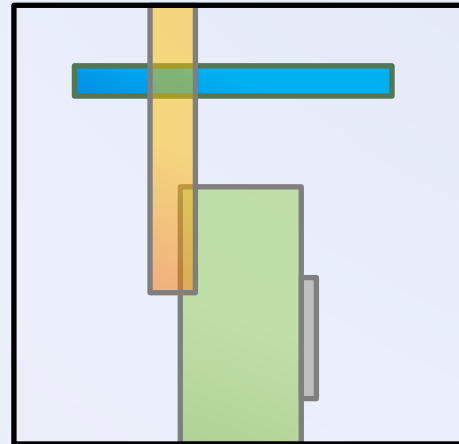


Intuition **Good**

M



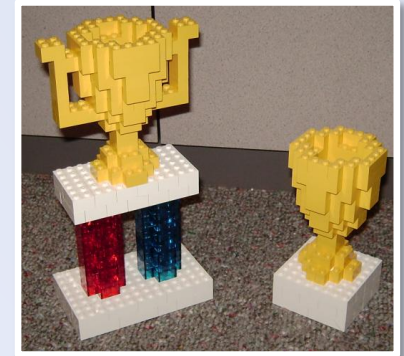
$D|M$



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

MD-what?

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

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

Kolmogorov Complexity

$$K(s)$$

The Kolmogorov complexity of a binary string s is the length of the shortest program s^* for a universal Turing Machine U that generates s and halts.

Kolmogorov Complexity

$$K(s)$$

The Kolmogorov complexity of a binary string s is the length of the shortest program s^* for a universal Turing Machine U that generates s and **halts**.

Conditional Complexity

$$K(s \mid t)$$

The **conditional** Kolmogorov complexity of a string s is the length of the shortest program s^* for a universal Turing Machine U **that given string t as input** generates s and halts.

Two-part Complexity

$$K(s) = K(s') + K(s \mid s')$$

The *two-part* Kolmogorov complexity of a string s decomposes the shortest program s^* into **two** parts

length of the **algorithm'**
length of its **parameters'**

$$s^*(s') = s$$

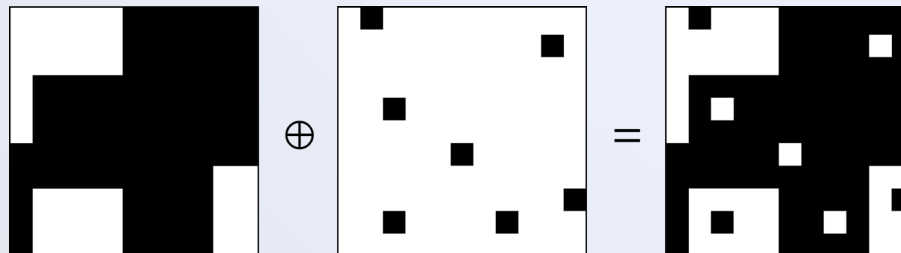
(up to a constant)

Two-part Complexity

$$K(s) = K(s') + K(s \mid s')$$

The *two-part* Kolmogorov complexity of a string s decomposes the shortest program s^* into **two** parts

length of the **model**,
length of **data given model**



MDL

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

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

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

Code Table

<i>Itemset</i>	<i>Code</i>
	
	
	
	
	
	-
	
	





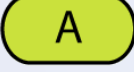
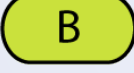



Code Table

<i>Itemset</i>	<i>Usage</i>
A C	0
B D	0
C E	0
A	0
B	0
C	0
D	0
E	0

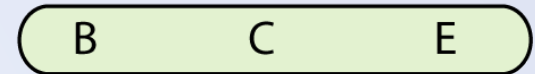
Transaction t

B	C	E
---	---	---


Code Table

<i>Itemset</i>	<i>Usage</i>
 	0
	0
	0
	0
	0
	0
	0
	0

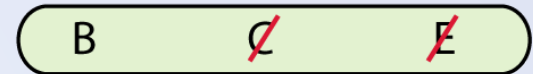
Transaction t



Code Table

Itemset	Usage
<div>A C</div>	0
<div>B D</div>	0
 <div>C E</div>	0 + 1
<div>A</div>	0
<div>B</div>	0
<div>C</div>	0
<div>D</div>	0
<div>E</div>	0

Transaction t



Cover of t

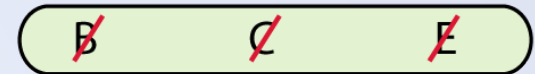


Code Table

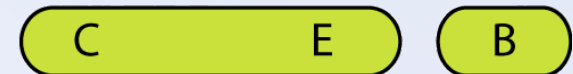
Itemset	Usage
<div>A C</div>	0
<div>B D</div>	0
<div>C E</div>	1
<div>A</div>	0
<div>B</div>	0 + 1
<div>C</div>	0
<div>D</div>	0
<div>E</div>	0



Transaction t

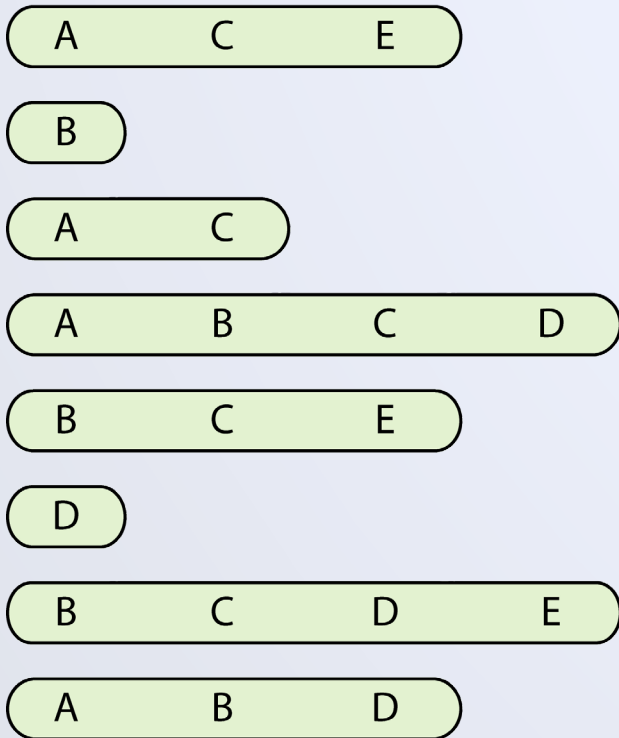


Cover of t

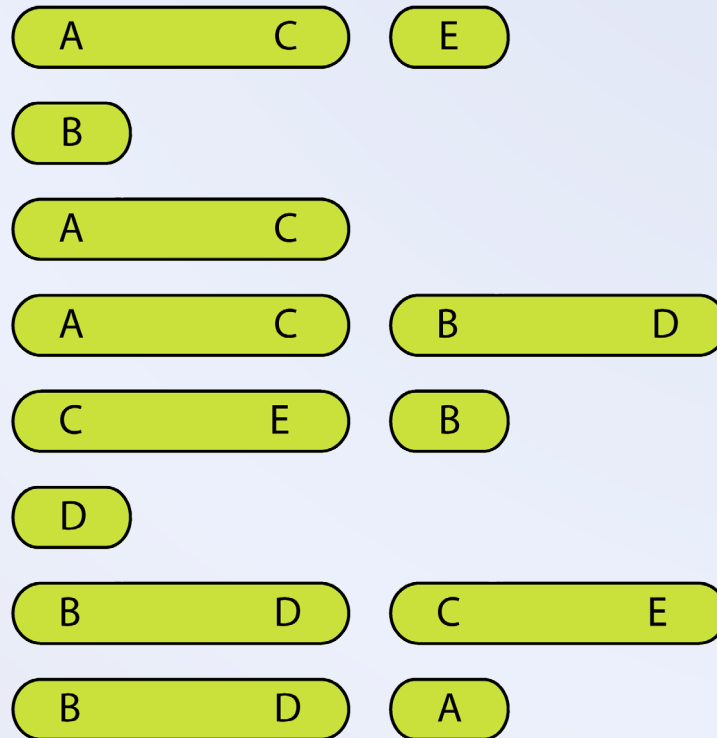


Encoding a database








Database



Database Cover



Code Table

Itemset	Code
A C	
B D	
C E	
A	
B	
C	-
D	
E	

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:

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

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

$$L(CT \mid D) = \sum_{c \in CT: usage(c) \neq 0} L(c \mid ST) + L(c \mid CT)$$

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

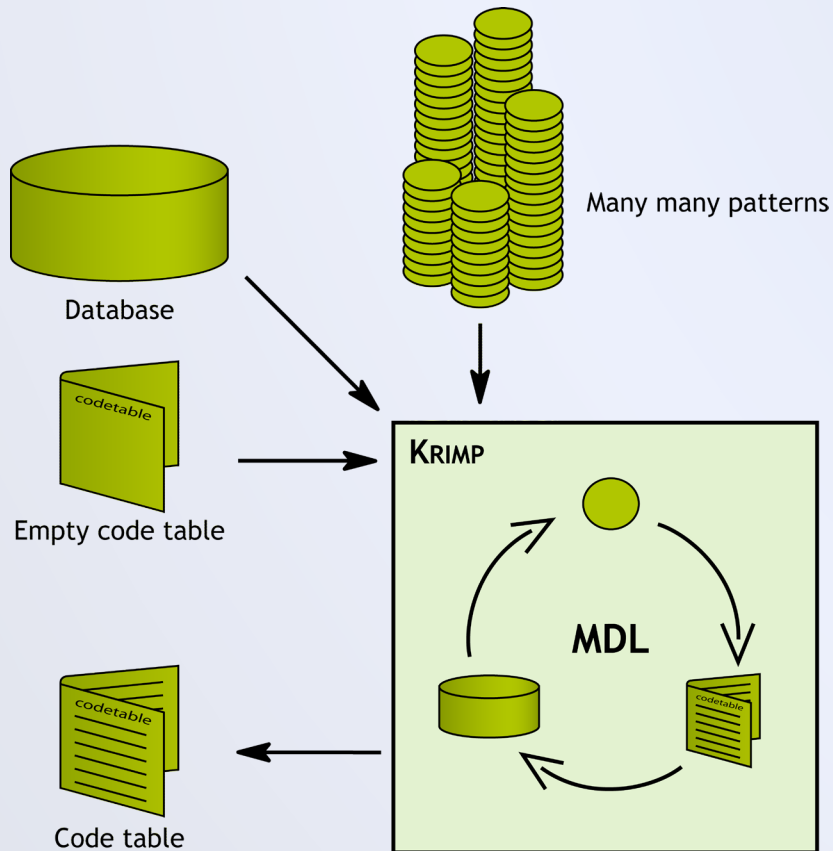
And now, the optimal code table...

Easier said than done

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

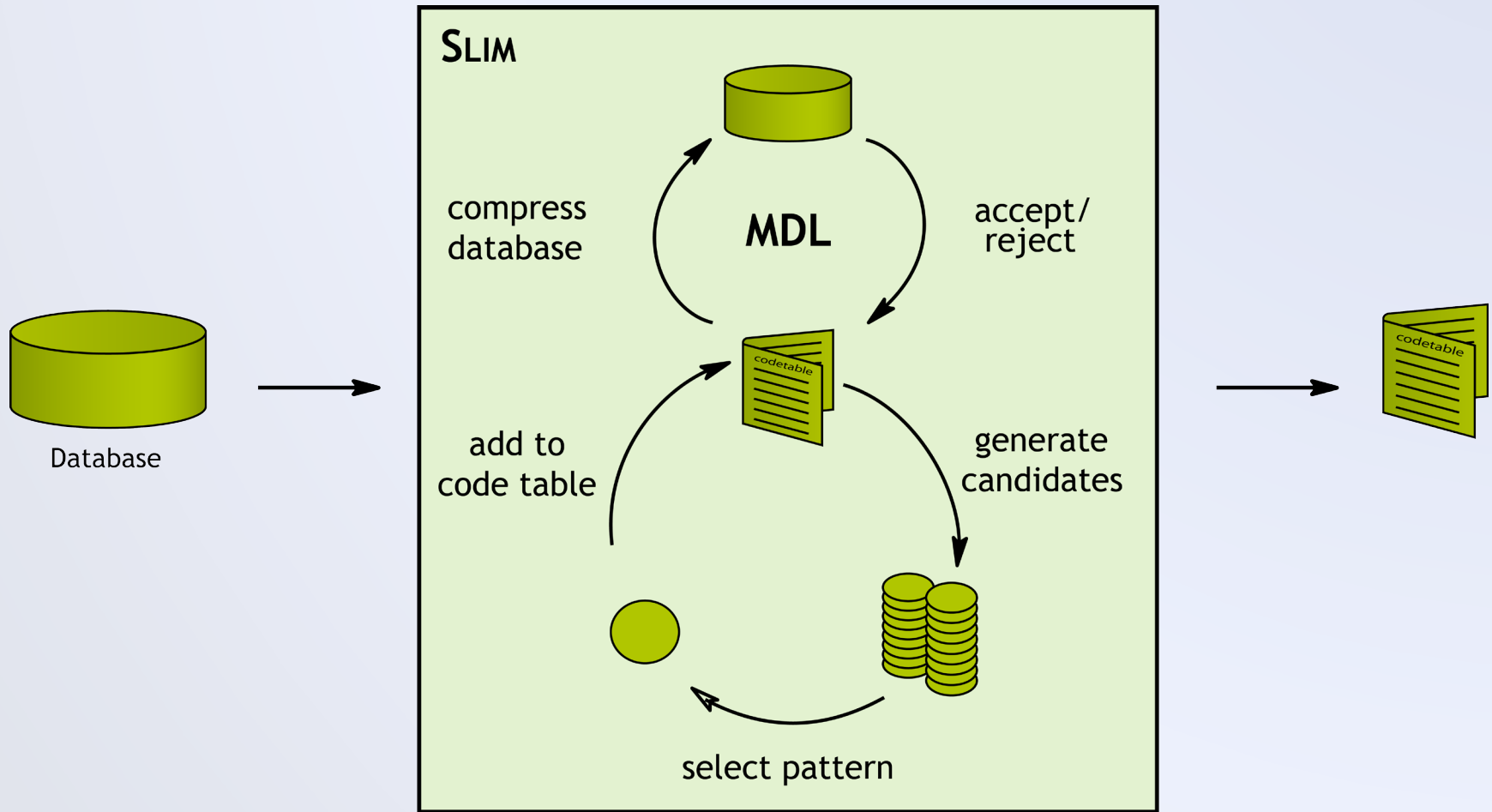
Hence, we resort to heuristics

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

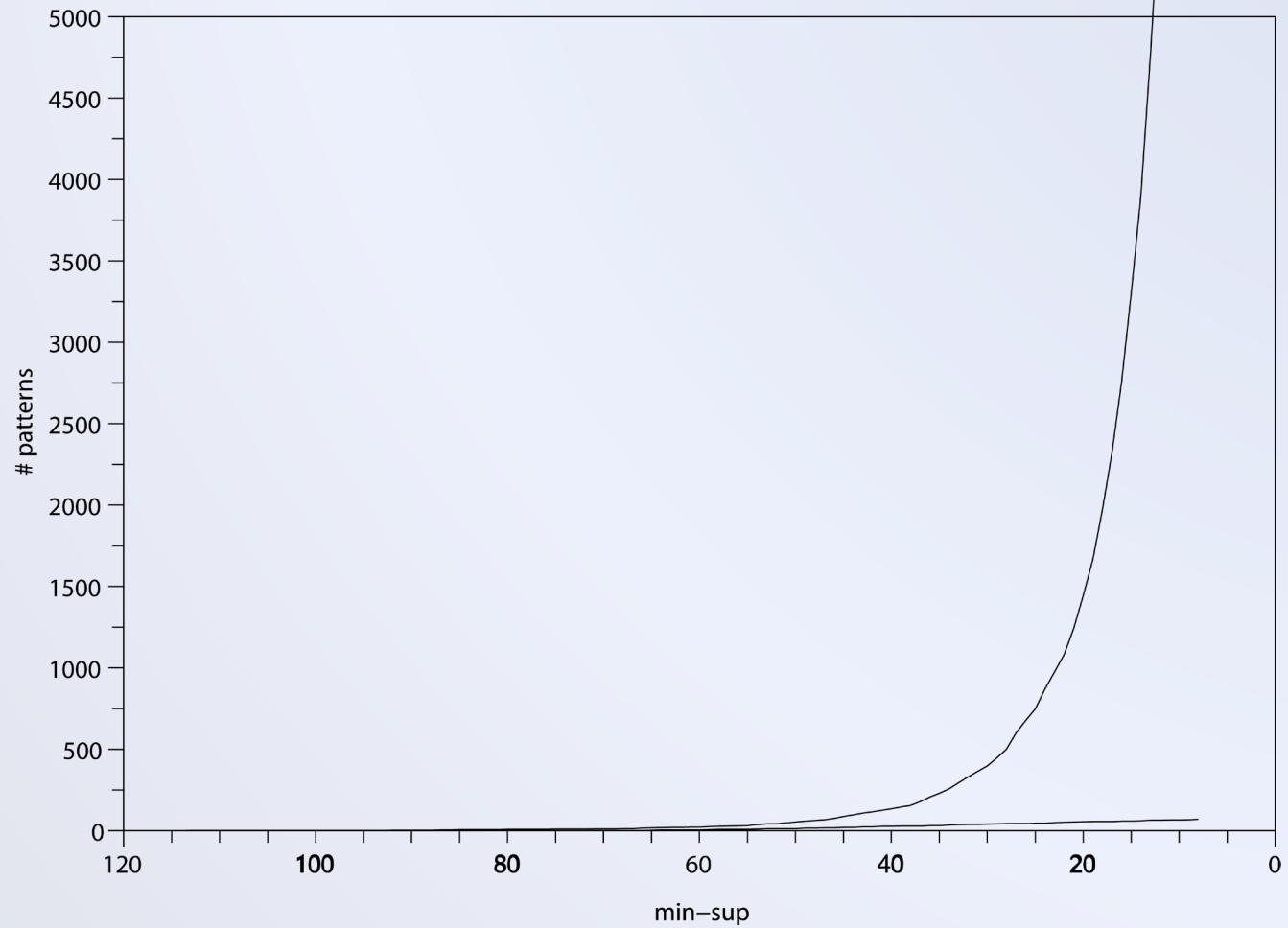
SLIM – smarter KRIMP



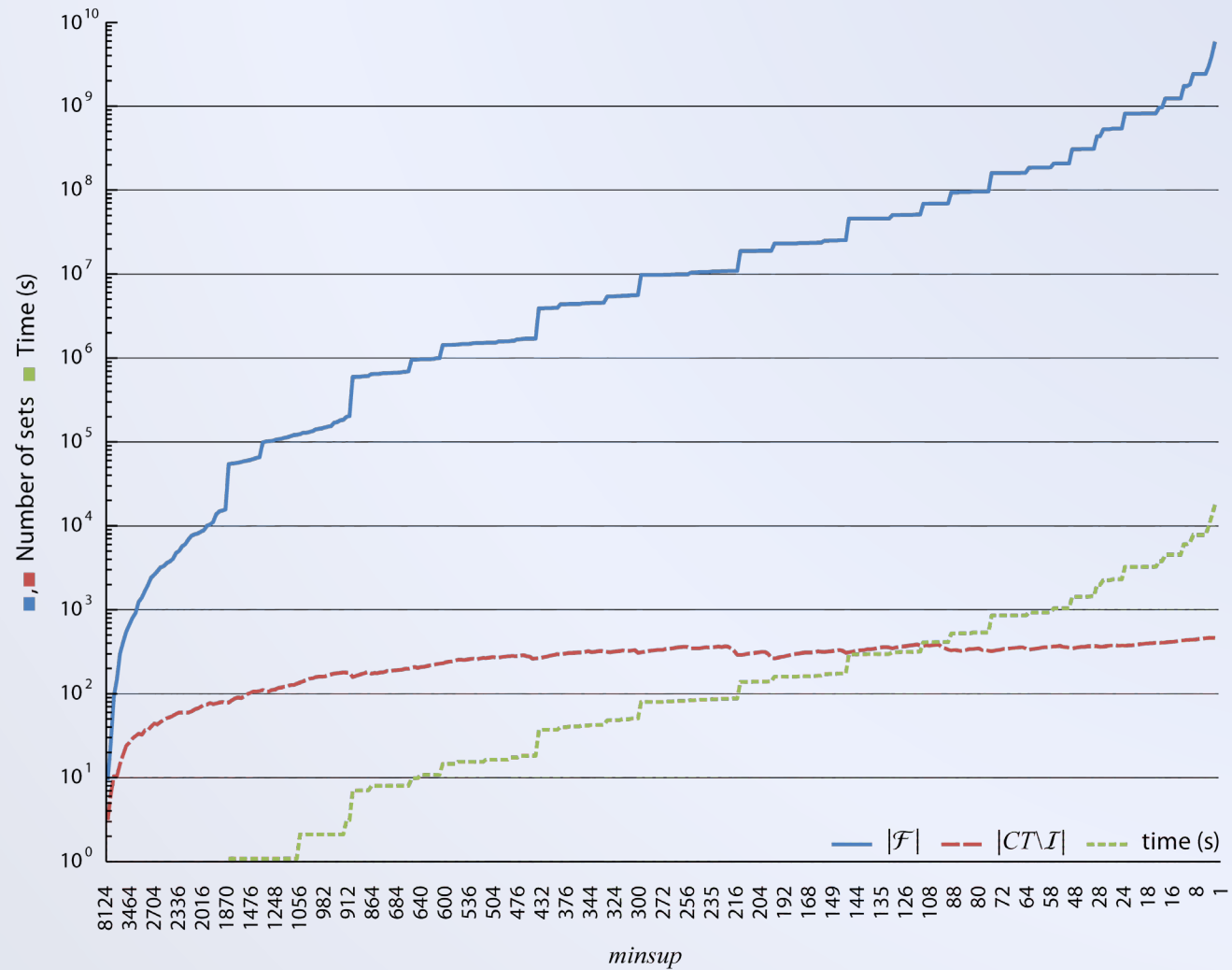
KRIMP in Action

Dataset	$ \mathcal{D} $	$ \mathcal{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



KRIMP in Action



So, are K_{RIMP} 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 over \mathcal{I}
- 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.

KRIMP for Classification

The KRIMP Classifier

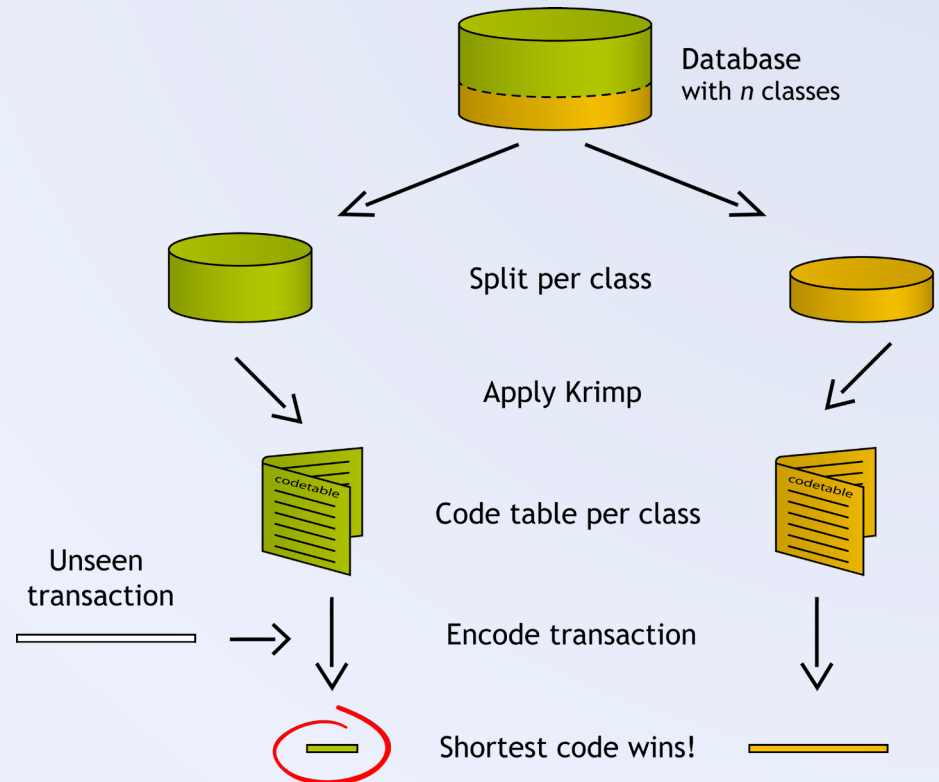
- split database on class
- find code tables
- classify by compression

The Goal

- validation of KRIMP

The Results

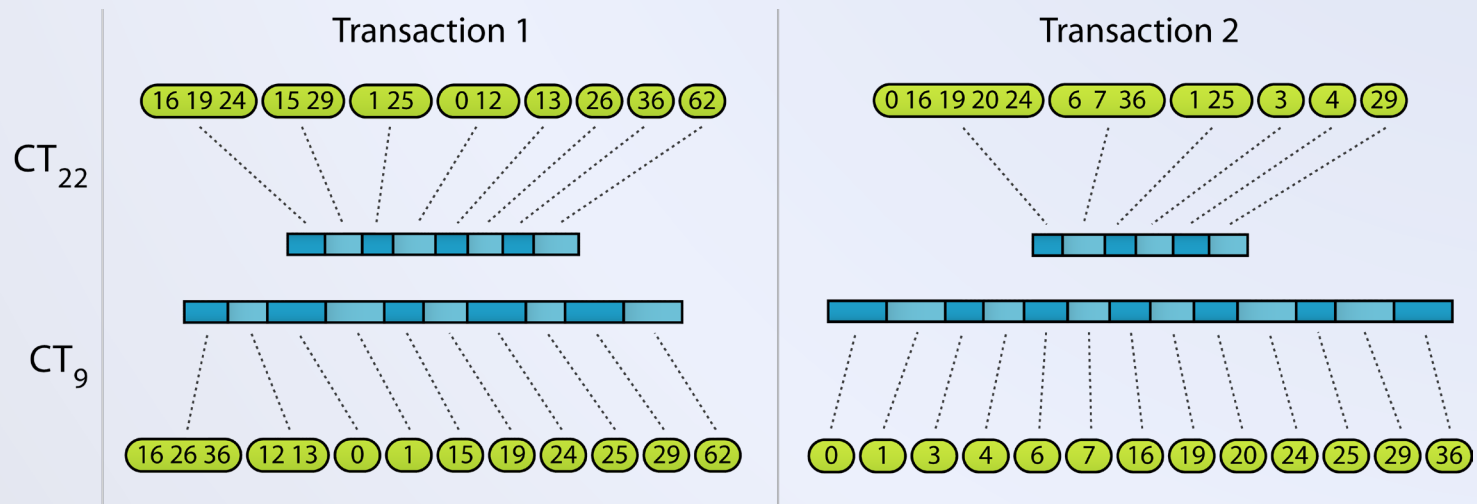
- expected 'ok'
- on par with top classifiers



Classification by Compression

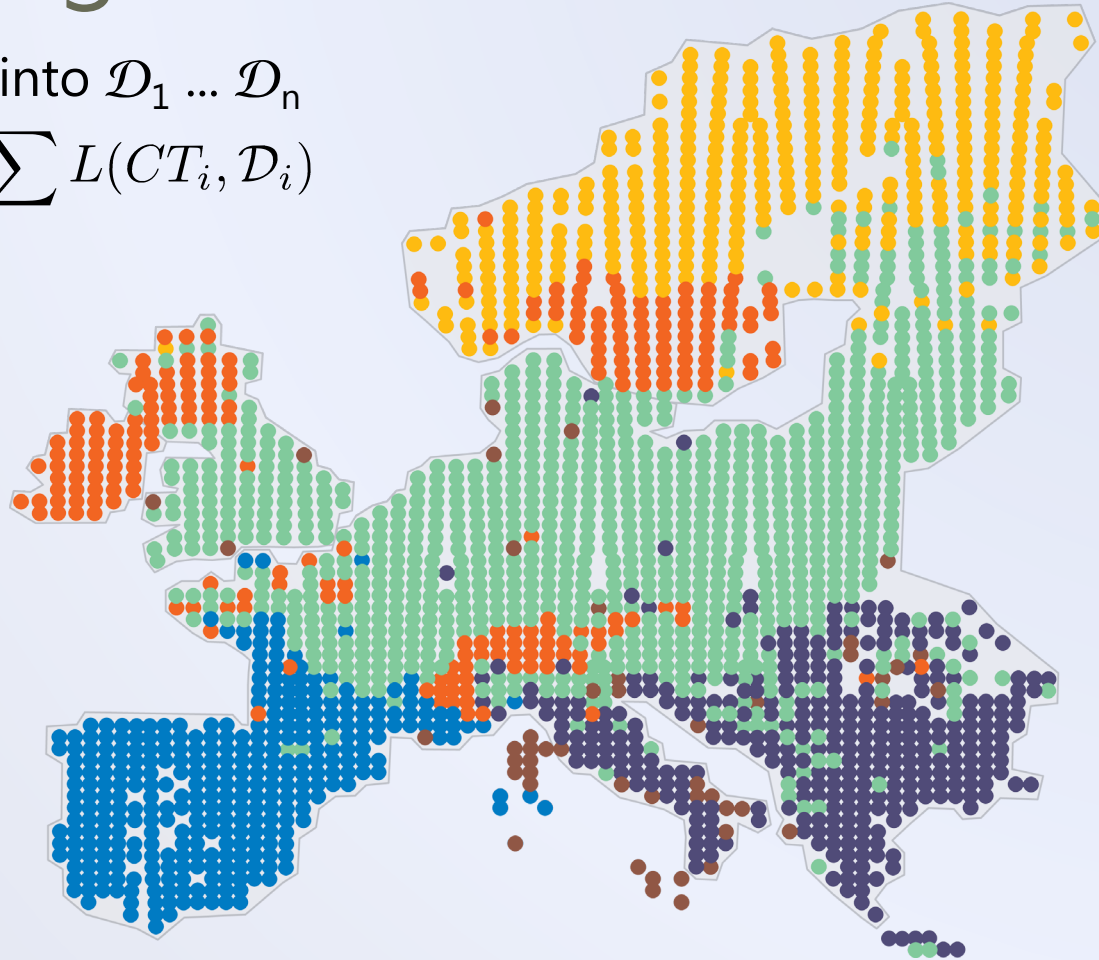
Two transactions encoded by two code tables

- can you spot the true class labels?



Clustering transaction data

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



$k=6$, MDL optimal

The Odd One Out

One-Class Classification (aka anomaly detection)

- lots of data for *normal* situation – insufficient data for *target*

Compression models the *norm*

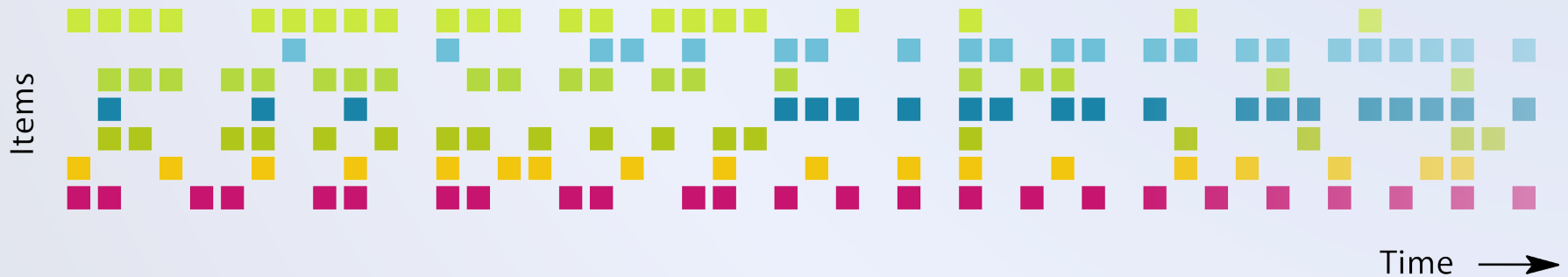
- anomalies will have high description length $L(t \mid CT_{norm})$

Very nice properties

- *performance* high accuracy
- *versatile* no distance measure needed
- *characterisation* this part of t can't be compressed well

STREAMKRIMP

Given a *stream* of itemsets



STREAMKRIMP

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 expenditur

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

Thank you!

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