Simply Mining Data

Jilles Vreeken
So, how do you pronounce...

Jilles  Vreeken
Yill-less  Fray-can

Okay, now we can talk.
Exploratory Data Analysis

Jilles Vreeken

17 August 2015
The goal

Theory and methods for getting **insight** from data summarising its main characteristics in **easily understandable terms**
So, what kind of data?

- **Real-valued**
  - GRIM: 0.7, 0.5, 0.3
  - REAPER: 0.9, 0.8, 0.0
  - INDY: 1.0, 0.2, 0.8

- **Binary**
  - A: 0, 1, 0
  - B: 0, 1, 1
  - C: 1, 1, 1

- **Multi-relational**
  - Graphs: (un)directed, (un)weighted, (un)attributed

- **Categorical**
  - Eyes: Blue, Brown
  - Hair: Red, Brown, Blue
  - Course: TADA, WTF

- **Time series**
So, what kinds of analysis?

pattern mining, summarisation, clustering, outlier detection, correlations measures, distance measures, missing value estimation, graph clustering, influence propagation, classification...

all at an *explorative* angle as assumption-free as possible identify interesting local structure describing structures in simple terms such that *you* learn about *your* data
So, what is your signature then?

defining well-founded objective functions for exploratory tasks

using information theory for measuring how many bits of information a result gives

MDL, Kolmogorov Complexity, Kullback-Leibler, Maximum Entropy, (cumulative) entropy
What is your signature?

data analysis ↔ communication

transfer the data to the analyst in as few as possible bits

‘induction by compression’
The Menu

In three lectures, I will give an introduction to modern **exploratory data analysis**.

Lecture 1: **How can we summarise** complex data?

Lecture 2: **How to solve** data mining **tasks by compression**?

Lecture 3: **How to discover causal relations** from data?
Simply Summarising Data

Jilles Vreeken

17 August 2015
How can we mine a small, non-redundant set of interesting patterns that together describe the data well?
Traditional 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 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

patterns

a pattern identifies local properties of the data

e.g. itemsets

a toy 0-1 dataset
Intuition Bad

\[ M \]

\[ D|M \]
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
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 $M$

(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*

and is related to Kolmogorov complexity
Kolmogorov Complexity

$K(s)$

The Kolmogorov complexity of a binary string $s$ is the length of the shortest program $k(s)$ for a universal Turing Machine $U$ that generates $s$ and halts.

(Kolmogorov, 1963)
Kolmogorov Complexity

\[ K(s) \]

The Kolmogorov complexity of a binary string \( s \) is the length of the shortest program \( k(s) \) for a universal Turing Machine \( U \) that generates \( s \) and halts.

(Kolmogorov, 1963)
Conditional Complexity

\[ K(s \mid t) \]

The **conditional** Kolmogorov complexity of a string \( s \) is the length of the shortest program \( k(s) \) for a universal Turing Machine \( U \) that given string \( t \) as input generates \( s \) and halts.
Two-part Complexity

\[ K(s) \triangleq K(s') + K(s \mid s') \]

The \textit{two-part} Kolmogorov complexity of a string \( s \) decomposes the shortest program \( k(s) \) into \textbf{two} parts

length of the `\texttt{algorithm}'
length of its `\texttt{parameters}'

(equality up to a constant)
Two-part Complexity

\[ K(s) \triangleq K(S) + \log |S| \]

The *two-part* Kolmogorov complexity of a string \( s \) decomposes the shortest program \( k(s) \) into two parts

- length of the `model` \( S \ni s \)
- length of `data given model` \( \log |S| \)
The Minimum Description Length (MDL) principle

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

$$L(M) + L(D \mid M)$$

in which

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

$L(D \mid M)$ 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

If we could use Kolmogorov complexity, we’d be done.

To use MDL, however, 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

And, of course, an algorithm to discover the optimal model for this score
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 ↔ defining a prior
- codes and probabilities are tightly linked:
  higher probability ↔ shorter code

So, although we don’t know overall probabilities

- we can exploit knowledge on local probabilities
$\mathcal{I} = \{ A, B, C, D, E \}$

**Code Table**

<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>
<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>
<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</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>
<tr>
<td>AC</td>
<td>0</td>
</tr>
<tr>
<td>BD</td>
<td>0</td>
</tr>
<tr>
<td>CE</td>
<td>0</td>
</tr>
</tbody>
</table>

Transaction $t$: B C E
<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></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>

Transaction $t$: B C E
### 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>0 + 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
<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>1</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0 + 1</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

Cover of t
Encoding a database

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

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

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

$$P(X \mid CT, D) = \frac{usage(X)}{\sum_{Y \in CT} usage(Y)}$$

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

$$L(X \mid CT) = -\log(P(X \mid CT, D))$$

(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

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

For $t \in D$ we have

$$L(t \mid CT) = \sum_{X \in \text{cover}(t, CT)} L(X \mid CT, D)$$

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

Hence we have
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
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)
### SLIM in Action

| Dataset          | $|\mathcal{D}|$ | $|\mathcal{F}|$   | $|\mathcal{C} \setminus \mathcal{I}|$ | $L\%$  |
|------------------|-----------|----------------|-----------------|--------|
| 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   |
SLIM in Action
SLIM in Action
SLIM in Action

Reductions of up to $10^7$, only one in $10,000,000$ is chosen!

Are you impressed?
Permutations

How characteristic are our models for our data?

This we can test through permutation testing.

We generate 1000 random datasets of the same row and column margins as $D$.

We can now determine empirical p-values.
Results on text data

**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

(top-5 from 563)  
(selection from top-25)

(Tatti & Vreeken, KDD’12)
So, are 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?
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)
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