Ensemble of Code Tables (Master Thesis Defense)

Jaspreet Singh

Utrecht University

July 10th 2018

Outline

Introduction: Machine Learning Models

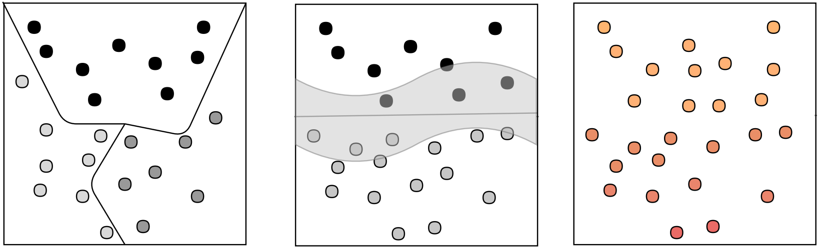
- **Goal**: Give a description of the database by using some model
- ... by approximating the underlying data distribution.
- *Example*: Find interesting patterns in the data.
- *Example*: Find groups of transactions that are similar.
- *Example*: Detect anomalies in the data.

Introduction

- **Goal**: *Cluster* the data such that *overlap* is allowed between the clusters.
- **How**: Use a *series of code tables*, such that each code table captures *a certain aspect* of the data.

Introduction: Clustering

- A group of data points which are similar in some sense.
- Is an unsupervised machine learning method.
- Performed in exploratory data analysis.
- Different types of clustering methods:



(A) Disjoint (B) Kernel (C) Non-disjoint

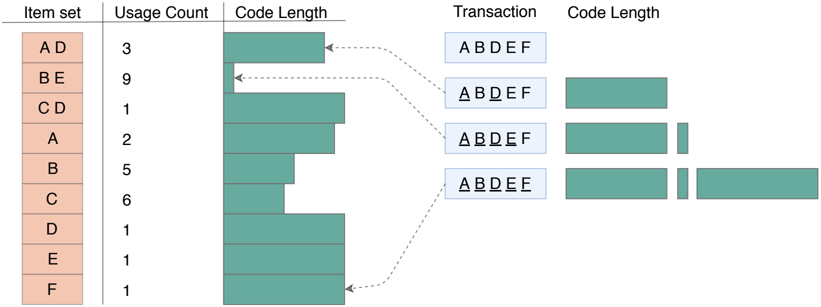
Introduction: Code Tables

- A code table is used to encode (compress) all the transactions in a dataset.
- A code table alone is not very useful, an algorithm which uses a code table is required.
- This algorithm is called the `Cover` algorithm

Introduction: Code Tables

Item set	Usage Count	Code Length
A B	3	
B E	9	
C D	1	
A	2	
B	5	
C	6	
D	1	
E	1	

Introduction: Code Tables



Preliminaries

- Basic Definitions
- Minimum Description Length Principle (MDL)
- Compression and Machine Learning
- Clustering

Preliminaries: Basic Definitions

- A dataset $\mathcal{D}$ is represented as a $N \times M$ binary matrix.
- N denotes the number of rows (transactions).
- M denotes the number of columns (items).
- $(t, i) = 1$ denotes that item i is used in transaction t.
- The set of items $\mathcal{I}$ make up the 'alphabet' of the dataset.
- An itemset I is an element of $\mathcal{P}(\mathcal{I})$: $I \in \mathcal{P}(\mathcal{I})$.

Preliminaries: Basic Definitions

- The support of an itemset $supp_{\mathcal{D}}(I)$, is the number of transactions in which I occurs:

$$supp_{\mathcal{D}}(I) = |\{t \in \mathcal{D} \mid I \subseteq t\}|$$

- Itemsets are frequent with respect to some minimum support θ.
- This restricts the number of possible itemsets.
- Given a minimum support θ, the set of all frequent itemsets $\mathcal{F}$ is defined as:

$$\{I \in \mathcal{F} \mid supp_{\mathcal{D}}(I) \geq \theta\}$$

Preliminaries: Minimum Description Length Principle

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

$$L(H) + L(\mathcal{D} \mid H)$$

- $L(H)$ is the length of the description of H in bits.
- $L(\mathcal{D} \mid H)$ is the length of the description of the data in bits, encoded by using H.

Preliminaries: Minimum Description Length Principle

- MDL is a practical version of the Kolmogorov Complexity.
- The Kolmogorov Complexity cannot be computed.
- The Kolmogorov Complexity of an object is the length of the shortest program that produces the object as output.
- *Example*: The string ‘abababab’ can be described as: $4\times$ ‘ab’

Preliminaries: Compression and Machine Learning

MDL applied to item sets: Code Tables!

- Use item sets to describe the data through code tables

Definition: Code Table

Let $\mathcal{I}$ be a set of items and $\mathcal{C}$ be a set of codes. A code table CT over $\mathcal{I}$ and $\mathcal{C}$ is a table with two columns such that: The first column contains subsets over $\mathcal{I}$, all singleton item sets must be present. The second column contains codes from $\mathcal{C}$ and every code is allowed to occur at most once.

- The standard code table CT_{ST} only contains the singleton item sets.

Preliminaries: Compression and Machine Learning

Input: Transaction $t \in \mathcal{D}$ and code table CT, with CT and $\mathcal{D}$ over a set of items $\mathcal{I}$.
Output: A cover of t using non-overlapping elements of CT.

```
S ← smallest element X of CT in Standard Cover Order for which X ⊆ t
if t \ S = ∅ then
    Res ← {S}
else
    Res ← {S} ∪ StandardCover(t \ S, CT)
end if
return Res
```

Preliminaries: Compression and Machine Learning

- The item sets in the code tables are sorted to avoid trying all combinations to cover a transaction
- The sorting order is called: **Standard Cover Order**
  1. Sort descending on item set size $|I|$
  2. Sort descending on support
  3. Sort ascending lexicographically

Preliminaries: Compression and Machine Learning

- Actual codes are not needed, code lengths are used to compute the compressed size.
- The more an item set is used, the shorter its code length is.
- The usage count of an item set I is

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

- This implies a probability distribution of $I \in CT$

$$\mathbb{P}(I \mid \mathcal{D}) = \frac{usage(I)}{\sum_{Y \in CT} usage(Y)}$$

- The code length $L(code_{CT}(I))$ then is

$$L(code_{CT}(I)) = -log(\mathbb{P}(I \mid \mathcal{D}))$$

Preliminaries: Compression and Machine Learning

Lemma 1

For any $t \in \mathcal{D}$ its encoded size in bits $L(t \mid CT)$ is:

$$L(t \mid CT) = \sum_{I \in cover(CT,t)} L(code_{CT}(I))$$

The encoded size of $\mathcal{D}$ when encoded by CT, $L(\mathcal{D} \mid CT)$, is:

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

The length of a code table $L(CT \mid \mathcal{D})$ is:

$$L(CT \mid \mathcal{D}) = \sum_{I \in CT} L(code_{ST}(I)) + L(code_{CT}(I))$$

The total encoded length $L(\mathcal{D}, CT)$ then is:

$$L(\mathcal{D}, CT) = L(\mathcal{D} \mid CT) + L(CT \mid \mathcal{D})$$

Preliminaries: Compression and Machine Learning: `Krimp`

- Many algorithms use a pre-mined set of candidate item sets $\mathcal{F}$.
- $\mathcal{F}$ is traversed in **Standard Candidate Order**
  1. Sort descending on support
  2. Sort descending on item set size $|I|$
  3. Sort ascending lexicographically

Preliminaries: Compression and Machine Learning: `Krimp`

Function *Krimp*$(\mathcal{D}, \mathcal{F})$

  Data: Dataset $\mathcal{D}$, candidate set $\mathcal{F}$, both over a set of items $\mathcal{I}$

  Result: Code table CT

  $CT \leftarrow$ **Standard Code Table**$(\mathcal{D})$;

  $\mathcal{F}_0 \leftarrow \mathcal{F}$ in **Standard Candidate Order**;

  for $F \in \mathcal{F}_0 \setminus \mathcal{I}$ **do**

    $CT_c \leftarrow (CT \cup F)$ in **Standard Cover Order**;

    if $L(\mathcal{D}, CT_c) < L(\mathcal{D}, CT)$ **then**

      $CT \leftarrow CT_c$;

    end

  end

  return CT;

Preliminaries: Compression and Machine Learning: `Slim`

- Mining a set of (frequent) item sets can take a lot of time.
- When dropping the the minimum support θ, the number of frequent item sets explode.
- It is possible to generate candidates more efficiently!
- **Insight**: Every item set is the union of two other item sets.
- Generate candidate item sets directly from code tables.

Preliminaries: Compression and Machine Learning: `Slim`

Function $Slim(\mathcal{D})$

  Data: Dataset $\mathcal{D}$

  Result: Code table CT

  $CT \leftarrow$ **Standard Code Table**$(\mathcal{D})$;

  for $F \in \{X \cup Y : X, Y \in CT\}$ *in* ***Gain Order*** **do**

    $CT_c \leftarrow (CT \oplus F)$ in **Standard Cover Order**;

    if $L(\mathcal{D}, CT_c) < L(\mathcal{D}, CT)$ **then**

      $CT \leftarrow post - prune(CT_c)$;

    end

  end

  return CT;

Preliminaries: Compression and Machine Learning: `Groei`

- Use a number of code tables to gain insight in the data from different aspects.
- All output code tables contain the same number of non-singleton item sets.
- This produces a structure function $\kappa_{\mathcal{D}}$
- Allows for the study of the correlation structure at different levels of granularity.

Preliminaries: Compression and Machine Learning: `Groei`

Function $Groei(\mathcal{D}, b)$

Data: Dataset $\mathcal{D}$, Beam-width b

Result: Code tables $\mathcal{CT} = \{CT_i, \ldots, CT_k\}$

$k \leftarrow 1$;

$\mathcal{CT}_1^{cand} \leftarrow \texttt{Generate}(CT_\alpha^{\mathcal{D}})$;

$\mathcal{CT}_1^{best} \leftarrow \{CT \mid \text{best } b \text{ tables from } \mathcal{CT}_1^{cand}\}$;

repeat

    $k \leftarrow k + 1$;

    $\mathcal{CT}_k^{cand} \leftarrow \texttt{Generate}(\mathcal{CT}_{k-1}^{best})$;

    $\mathcal{CT}_k^{best} \leftarrow \{CT \mid \text{best } b \text{ tables from } \mathcal{CT}_k^{cand}\}$;

until $L(\mathcal{D} \mid \mathcal{CT}_k^{best}) \geq L(\mathcal{D} \mid \mathcal{CT}_{k-1}^{best})$;

return $\mathcal{CT}_k^{best}$;

Preliminaries: Summary

- The `Cover` algorithm allows to compress the data by using code tables.
- `Krimp` uses a pre-mined set of item sets.
- The number of candidates explodes when the minimum support is lowered.
- `Slim` avoids this explosion by directly generating candidates from the code table.
- `Groei` outputs a number of code tables which together produce a structure function.

Preliminaries: Clustering: Hard

Let $\mathcal{D}$ denote the dataset, K the number of clusters and $\mathcal{D}_i \subseteq \mathcal{D}$ a cluster. A hard partitioning of data set $\mathcal{D}$ tries to find K partitions such that:

1. $\forall i \in [1, K] : \mathcal{D}_j \neq \emptyset$
   All partitions must be non-empty.
2. $\bigcup_{i=1}^{K} \mathcal{D}_i = \mathcal{D}$
   All partitions together must contain all transactions.
3. $\forall i, j \in [1, K] : i \neq j \Rightarrow \mathcal{D}_i \cap \mathcal{D}_j = \emptyset$
   There is no overlap between the partitions.

Problem Description and Research Questions

- Observations and Definitions
- Problem Description
- Research Questions

Problem Description: Observations

- `Groei` produces a number of code tables.
- However, all code tables are of the same complexity and are not the best compressing tables.
- Get rid of the structure function and make candidate generation more efficient.
- We do not want a disjoint clustering, a non-disjoint clustering is required.
- Each code table corresponds to a cluster.
- It is also possible for a transaction to belong to all clusters with a degree of membership $u_{i,j} \in [0,1]$,the membership coefficient of the jth object in the ith cluster.
- The membership coefficient must satisfy the following two constraints:

$$\forall j : \sum_{i=1}^{K} u_{i,j} = 1 \qquad \text{and} \qquad \forall i : \sum_{j=1}^{N} u_{i,j} < N$$

Problem Description: Definitions

The Membership Coefficient is the probability that tuple $d_j \in \mathcal{D}$ belongs to cluster C_i:

$$\mathbb{P}(d_j \in C_i) = \frac{2^{-CT_i(d_j)}}{\sum\limits_{l}^{C} 2^{-CT_l(d_j)}}$$

The Encoded Cluster Length of a cluster C_i is determined by its code table CT_i, and is the Code Table Encoded Length over all the transactions in the database $\mathcal{D}$:

$$L(C_i \mid CT_i) = \sum_{j=1}^{N} L(d_j \mid CT_i)$$

Problem Description

Let $\mathcal{D}$ denote a database, and let $\mathcal{I}$ denote the set of items in the database. The database $\mathcal{D}$ is then a subset of $\mathcal{P}(\mathcal{I})$, $\mathcal{D} \subseteq \mathcal{P}(\mathcal{I})$. So, every tuple $t \in \mathcal{D}$ is also an element of $\mathcal{P}(\mathcal{I})$. The goal is then to find the code table CT that best compresses $\mathcal{D}$.

$$\min CT(\mathcal{D})$$

Problem Description

Let $\mathcal{D}$ denote a database, and let $\mathcal{CT}$ denote a set of code tables such that $CT_1, CT_2 \in \mathcal{CT}$. Then either:

- $CT_1, CT_2 \in \mathcal{CT}$ form an antichain;
- or, $\exists \mathcal{D}_1, \mathcal{D}_2 \subset \mathcal{D}$ given both partitions are large enough that:

$$CT_1(\mathcal{D}_1) \leq CT_1(\mathcal{D}_2) \qquad \text{and} \qquad CT_2(\mathcal{D}_2) \leq CT_2(\mathcal{D}_1)$$

Problem Description

let $\mathcal{D}$ denote a database, let $\mathcal{CT}$ denote a set of code tables, let Q denote some quality threshold, and let C denote the number of code tables possible for $\mathcal{D}$

$$\forall i \in [1, C] : \left[CT_i(\mathcal{D}) > Q \Rightarrow CT_i \notin \mathcal{CT} \right]$$

Research Questions

- Do the obtained clusters capture the characteristics of the underlying data distribution?
- Are the clusters dissimilar enough to each describe a specific characteristic of the database?
- Are the obtained clusters able to identify a multi-valued relationship, if present?
- Is the runtime low enough for interactive usage?

Clustering Algorithms

- `GroeiNoS`
- Candidate Generation
- `Slim` Candidate Generation (`GroeiSlimNoS`)

Clustering Algorithms: `GroeiNoS`

Function $GroeiNoS(\mathcal{D}, b)$

Data: Dataset $\mathcal{D}$, Beam-width b

Result: Code tables $\mathcal{CT} = \{CT_i, \ldots, CT_k\}$

$k \leftarrow 1;$

$\mathcal{CT}_1^{cand} \leftarrow \texttt{Generate}(CT_\alpha^{\mathcal{D}});$

$\mathcal{CT}_1^{best} \leftarrow \{CT \mid \text{best } b \text{ tables from } \mathcal{CT}_1^{cand}\};$

repeat

$\quad k \leftarrow k + 1;$

$\quad \mathcal{CT}_k^{cand} \leftarrow \texttt{Generate}(\mathcal{CT}_{k-1}^{best});$

$\quad \mathcal{CT}_k^{best} \leftarrow \{CT \mid \text{best } b \text{ tables from } \boxed{\mathcal{CT}_k^{cand} \cup \mathcal{CT}_{k-1}^{best}\};}$

until $L(\mathcal{D} \mid \mathcal{CT}_k^{best}) \geq L(\mathcal{D} \mid \mathcal{CT}_{k-1}^{best});$

return $\mathcal{CT}_k^{best};$

Clustering Algorithms: Candidate Generation

Function $Generate_{basic}(\mathcal{CT})$

  Data: Code tables $\mathcal{CT} = \{CT_1, \ldots, CT_n\}$

  Result: Code tables $\mathcal{CT} = \{CT_1, \ldots, CT_k\}$

  $\mathcal{CT}^{cand} = \{\}$;

  for $I \in \mathcal{F}$ **do**

    for $CT \in \mathcal{CT}$ **do**

      if $I \notin CT$ **then**

        $\mathcal{CT}^{cand} = \mathcal{CT}^{cand} \cup \{(CT \cup I)$ Standard Cover Order $\}$;

      end

    end

  end

  return $\mathcal{CT}^{cand}$;

Clustering Algorithms: Candidate Generation

Function $Generate_{slim}(\mathcal{CT})$

  Data: Code tables $\mathcal{CT} = \{CT_1, \ldots, CT_n\}$

  Result: Code tables $\mathcal{CT} = \{CT_1, \ldots, CT_k\}$

  $\mathcal{CT}^{cand} = \{\}$;

  for $CT \in \mathcal{CT}$ **do**

    for $F \in \{X \cup Y : X, Y \in CT\}$ **do**

      $\mathcal{CT}^{cand} = \mathcal{CT}^{cand} \cup \{(CT \cup F)$ Standard Cover Order $\}$;

    end

  end

  return $\mathcal{CT}^{cand}$;

Experiments

- Setup
- Datasets
- Compression
- Clustering
- Classification
- Multi-Valued Dependencies

Experiments: Setup

- Maximum iterations: 250
- Cut-off time: 12 hours
- All experiments performed on same machine
- Default beam-width = 10
- Default algorithm: `GroeiSlimNoS`

Experiments: Datasets

$\mathcal{D}$	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{I}\rvert$	ρ	θ	$\lvert\mathcal{F}\rvert$
anneal	898	71	20.1%	100	2.55×10^4
breast	699	16	62.4%	1	9.92×10^3
chess	3196	75	49.3%	2500	1.15×10^4
ionosphere	351	157	22.3%	125	1.03×10^4
iris	150	19	26.3%	1	5.43×10^3
led7	3200	24	33.3%	1	1.53×10^4
mushroom	8124	119	19.3%	2500	2.37×10^3
mammals	2183	121	20.5%	850	9.26×10^3
pageblocks	5473	44	25.0%	1	6.36×10^4
pima	768	38	23.7%	1	2.88×10^4
wine	178	68	20.6%	10	8.81×10^3
wine	178	68	20.6%	20	1.45×10^3
wine	178	68	20.6%	30	3.99×10^2

Experiments: Compression

- General Compression
- Runtime
- Lowering the support
- Beam-width

Experiments: Compression

- Measure the relative compression by the best compressing code table:

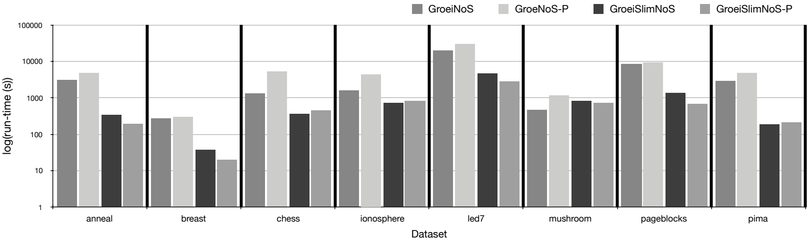
$$L\% = \frac{L(\mathcal{D}, CT)}{L(\mathcal{D}, ST)} \times 100\%$$

- The lower the value of $L\%$, the more structure is captured.

Experiments: Compression

$\mathcal{D}$	θ	Groei-F	Groei	GroeiNoS	GroeiSlim	GroeiSlimNoS
anneal	100	51,3%	45,2%	45,2%	**43,0%**	**43,0%**
breast	1	23,2%	16,5%	16,5%	**15,6%**	**15,6%**
chess	2500	65,4%	65,1%	65,1%	65,1%	65,1%
ionosphere	125	78,3%	71,8%	71,8%	**71,1%**	**71,1%**
iris	1	46,4%	45,5%	45,5%	45,5%	45,5%
led7	1	39,6%	28,3%	28,3%	**27,4%**	**27,4%**
mammals	850	66,7%	65,3%	65,3%	**65,0%**	**65,0%**
mushroom	2500	66,1%	**65,7%**	**65,7%**	66,2%	66,2%
pageblocks	1	7,3%	5,0%	5,0%	5,0%	5,0%
pima	1	39,1%	33,9%	33,9%	**31,0%**	**31,0%**
wine	10	**71,7%**	72,2%	72,2%	73,0%	73,0%
wine	20	**74,7%**	75,3%	75,3%	75,1%	75,1%
wine	30	**76,8%**	77,5%	77,5%	77,6%	77,6%

Experiments: Compression: Runtime



Experiments: Compression: Lower Support

$\mathcal{D}$	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{I}\rvert$	ρ	θ	$\lvert\mathcal{F}\rvert$
chess	3196	75	49.3%	500	8.46×10^9
ionosphere	351	157	22.3%	35	2.26×10^9
mushroom	8124	119	19.3%	1	1.56×10^{10}
mammals	2183	121	20.5%	200	9.38×10^7

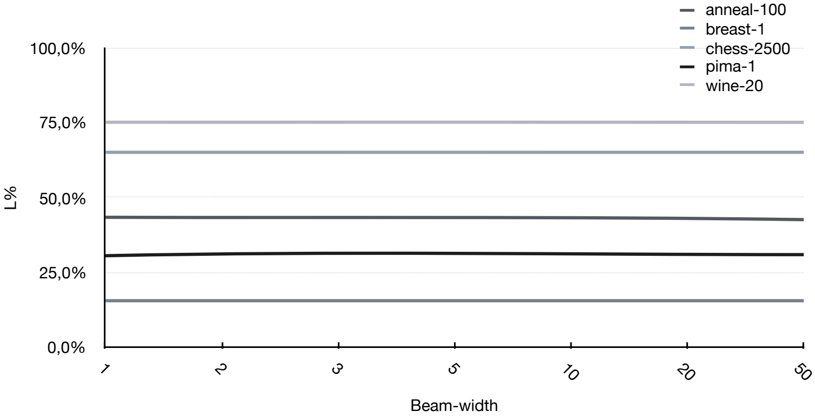
Experiments: Compression: Lower Support

		`GroeiSlimNoS`	
$\mathcal{D}$	θ	b = 1	b = 3
chess*	500	27,0%	27,0%
mushroom**	1	23,5%	23,7%
ionosphere	35	56,8%	56,9%
mammals*	200	47,0%	46,8%

(*) Terminated because of iteration limit.

(**) Terminated because of time limit.

Experiments: Compression: Beam-width



Experiments: Clustering

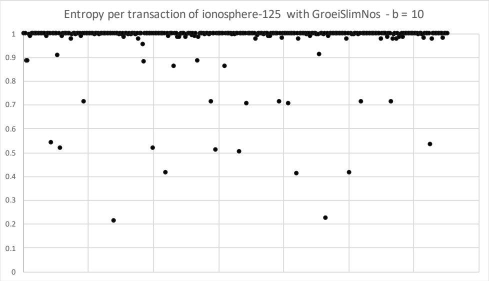
- Entropy of transactions
- Dissimilarity between code tables
- Probability distribution

Experiments: Clustering: Entropy

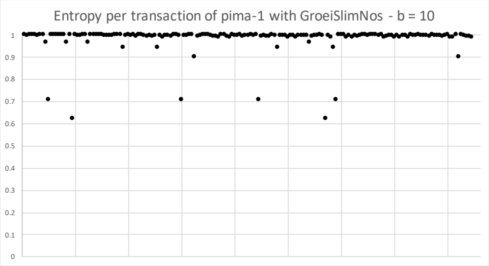
- Measures the homogeneity of transactions with respect to all clusters.
- The higher the entropy, the more uniformly the transaction is compressed by all code tables.

$$E = -\sum_{i=1}^{C} \mathbb{P}(t \in D_i) \log_b \mathbb{P}(t \in D_i),$$

Experiments: Clustering: Entropy

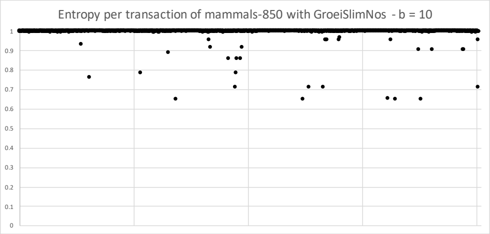


(a) ionosphere-125



(b) pima-1

Experiments: Clustering: Entropy



(a) mammals-850

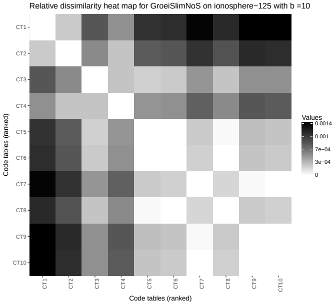
(b) anneal-100

Experiments: Clustering: Dissimilarity

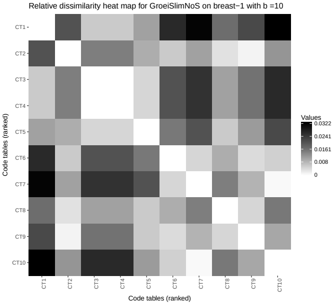
- Measure the pairwise relative dissimilarity in compressed cluster size.
- Code tables are ranked from best compressing to worst compressing.

$$DS(CT_x, CT_y, \mathcal{D}) = \max\left\{ \frac{CT_y(\mathcal{D}) - CT_x(\mathcal{D})}{CT_x(\mathcal{D})}, \frac{CT_x(\mathcal{D}) - CT_y(\mathcal{D})}{CT_y(\mathcal{D})} \right\}$$

Experiments: Clustering: Dissimilarity

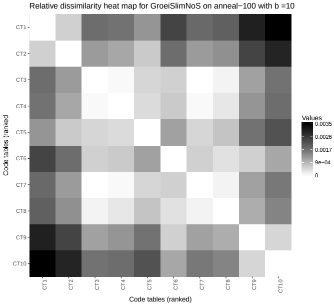


(a) ionosphere-125

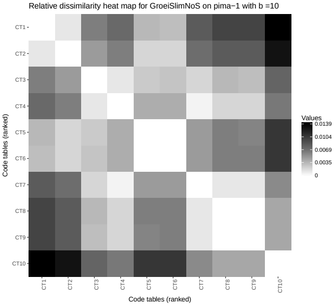


(b) breast-1

Experiments: Clustering: Dissimilarity



(a) anneal-100

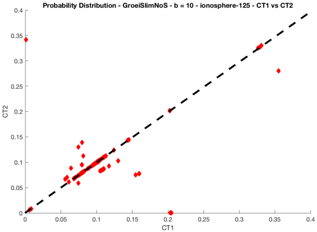


(b) pima-1

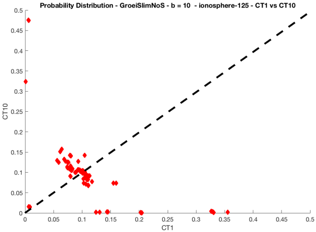
Experiments: Clustering: Distribution

- It also possible to look at a few code tables and the transactions.
- This will highlight the differences between the code tables on the level of transactions.

Experiments: Clustering: Distribution: Ionosphere-125

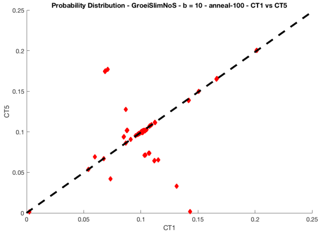


(a) $CT1$ versus $CT2$.

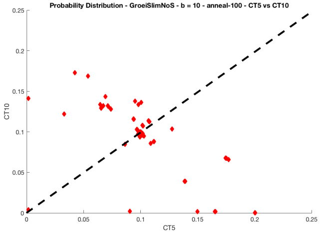


(b) $CT1$ versus $CT10$.

Experiments: Clustering: Distribution: Anneal-100



(a) $CT1$ versus $CT5$.



(b) $CT5$ versus $CT10$.

Experiments: Classed Datasets

- Classification
- Purity

Experiments: Classed Datasets: Classification

Dataset $\mathcal{D}$	Number of classes $\|cl\|$
ionosphere	2
letterrecognition	26
mushroom	2
pendigits	10
wine	3

Experiments: Classed Datasets: Classification

- 10-fold cross validation is used for the experiments.
- 1 fold for validation, 9 folds for training.
- Training: run the algorithm on each class for each fold.
- Assign to the class that compresses the transaction the best.

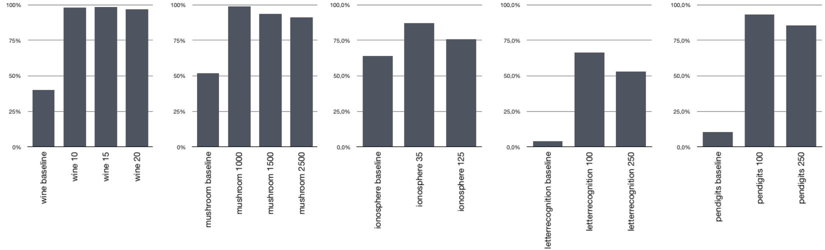
$$\mathbb{P}(d \in \mathcal{D}_i) \propto \frac{1}{CT_i(d)}$$

$$C(d) = \arg\max_{D_i \in \mathcal{D}} \mathbb{P}(d \in \mathcal{D}_i) = \arg\max_{D_i \in \mathcal{D}} \frac{1}{CT_i(d)}$$

Experiments: Classed Datasets: Classification

- Measure the ratio of cases that have been classified correctly.
- Baseline: accuracy when assigning to majority class.

$$A = \frac{\text{Number of cases classified correctly}}{\text{Total number of cases}}$$



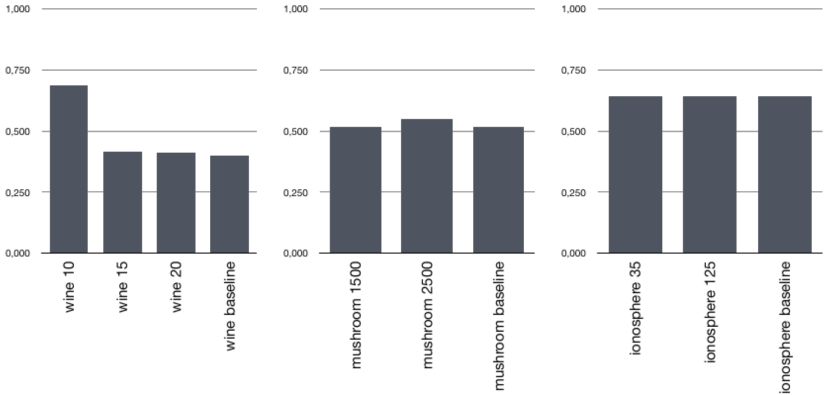
Experiments: Classed Datasets: Purity

- How well do the obtained clusters characterise the classes?
- Let $\mathscr{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$ the set of clusters, and $\mathscr{C} = \{c_1, \ldots, c_m\}$ denote the set of classes where each class c_j is the set of all cases which belong to c_j, then the purity is:

$$purity(\mathscr{D}, \mathscr{C}) = \frac{1}{N} \sum_{i=1}^{k} \max_j |\mathcal{D}_i \cap c_j|$$

- Baseline is ratio of classes belonging to the majority class.
- beam-width = number of classes

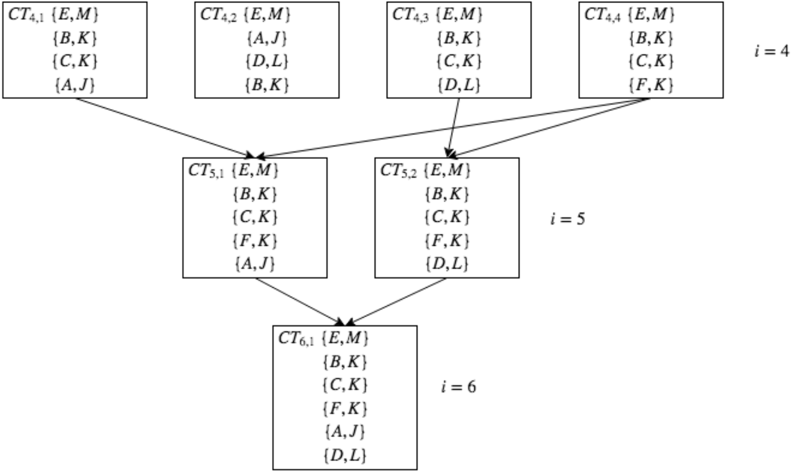
Experiments: Classed Datasets: Purity



Experiments: Multi-Valued Dependencies

(β, γ, ψ)	Number of occurrences
$(A, J, *)$	4
$(B, K, *)$	4
$(C, K, *)$	4
$(D, L, *)$	4
$(E, M, *)$	2
$(F, K, *)$	2
$(G, J, *)$	2
$(H, L, *)$	2
$(I, J, *)$	1

Experiments: Multi-Valued Dependencies



Discussion

- Compression is on-par or better than `Groei-F` in most cases.
- Big improvement in runtime.
- Is able to handle item sets with lower support settings but this can still be improved.
- All code tables capture the general patterns
- The code tables also capture specific patterns
- Classification experiments show that further lowering of the support is required to better capture the structure.
- The algorithm is able to identify multi-valued dependencies.

Conclusion

- Non-disjoint clustering sheds light on the data from different perspectives.
- Both commonalities and differences are identified.
- Future work: Further improve candidate generation (USE THE GAIN ORDER!)