UTRECHT UNIVERSITY

MASTER THESIS

Ensemble of Code Tables

Author:
Jaspreet SINGH
ICA_3754022

Supervisors:
Prof. dr. Arno SIEBES
dr. Ad FEELDERS

A thesis submitted in fulfillment of the requirements
for the degree of Master of Science

in the

Department of Information and Computing Sciences
Division of Artificial Intelligence
Algorithmic Data Analysis Research Group

July 10, 2018

UTRECHT UNIVERSITY

Abstract

Faculty of Science
Department of Information and Computing Sciences

Master of Science

Ensemble of Code Tables

by Jaspreet SINGH
ICA_3754022

In this master thesis non-disjoint clustering algorithms are presented, which are based on the Minimum Description Length (MDL) principle. The algorithms capture the underlying distribution from different perspectives by compressing the data using a series of code tables. A cover algorithm describes how to compress the database using a code table. Every code table is iteratively grown until compression does not improve any more. Experiments show that the algorithms are able to identify structure in the data because the data gets compressed to some extent by the code tables. Clustering experiments show that the general structure is captured by all obtained code tables and that the different groups of patterns that are dissimilar to the general patterns, are captured by different code tables. This confirms that the code tables view the data from different perspectives. The classification experiments show that, given the class labels, the code tables are dissimilar enough to capture the different characteristics of the classes. Without the class labels it is able to find the difference between the classes when the support is sufficiently low. It is also possible to identify multi-valued dependencies in the data. This is the case when code tables in a single iteration are anti-chains and later end up in the same code table.

Acknowledgements

First of all, I would like to thank my thesis supervisor professor Arno Siebes for always providing constructive feedback and giving guidance. One of my friends, Oscar Bartman also deserves praise because he read through the project proposal to identify spelling- and grammar mistakes. I am also grateful that my parents, my sisters and my friends supported me, and kept me motivated throughout this process.

Contents

1 Introduction

The aim in machine learning always has been to approximate the underlying data distribution as well as possible using some model. The world of machine learning can be divided into supervised learning and unsupervised learning. In supervised learning the models are trained using labelled data (examples) and are evaluated to determine their performance. In unsupervised learning there is no labelled data, so it is impossible to train and evaluate models. The main goal in unsupervised learning is to describe interesting patterns in the data. One such way of describing the interesting patterns in the data is to use clustering. In clustering, groups of objects are identified that are similar in some sense. This commonly performed in exploratory data analysis. The domain where overlap between clusters is allowed is called fuzzy clustering. Here, all the data points have a degree-of-membership towards all the clusters. Another problem in clustering is that the number of clusters is not known beforehand and that models have to be tuned to give the best results.

The `Krimp`-algorithm introduced by Siebes et. al. has been extended to also be able to perform clustering [VVLS11]. The algorithm was originally introduced to tackle to problem of pattern mining by using the concept of the Minimum Description Length principle. This is: 'The best set of patterns that describe the data the best are those that compress the data the best'. The authors observed that the obtained model could also be used for classification, which is a supervised learning task. The difference between classification and clustering is that in classification the clusters are already given by the classes, while clustering the goal is to find these classes. Later, the extension to disjoint clustering was made by Leeuwen et. al. [LVS09]. This is done by randomly partitioning the database into k components and swapping transactions until the database cannot be compressed any further, or by first applying `Krimp` to the entire database and then creating k copies and iteratively eliminating itemsets in the code tables that reduce the compressed size the most. Siebes et. al. observe that a series of models are able to give a better description of the data than one single model is able to [SK11]. This resembles the idea of boosting, where a series of weaker models is used to get a better description of the data. The models in this case are a series of code tables, and give a structure function. The structure functions suggests a natural partitioning of the data. They observe that different numbers of code tables give insight into the data on different levels.

The aim of this thesis project is to find clusters and to allow for overlap between clusters by extending `Krimp` and the works that follow it. Allowing overlap is achieved by using techniques more similar to bagging than boosting. Both bagging and boosting are a form of ensemble learning, which means that a series of weaker models have similar performance as a strong model. An advantage of bagging is that it tries to de-correlate the data by using bootstrap sampling, which decreases the bias and variance of the model. A better description of the data can be achieved by allowing overlap between clusters because the variety in the data is captured in the overlap between clusters.

In order to gain insight in the problem domain a literature study is presented in Chapter 2. Relevant works to the problem are discussed and summarised. In

Chapter 3 a formal problem definition is given based on the literature study. Also the the relevance of the project to science, technology and society is argued. In Chapter 4 the methodology for evaluation is presented. In Chapter 5 the newly developed algorithms are presented. These are experimentally evauated and the results of the evaluation are presented in Chapter 6. The obtained results are then discussed in Chapter 7. Finally, the conclusion is given in Chapter 8.

2 Literature Study

In this chapter a literature review is presented where relevant works with respect to this thesis project are summarised. First, works related to ensemble learning are discussed. Second, the notion of compression and works related to clustering with ensemble structures are discussed. At last, the works related to machine learning by means of compression are summarised. In particular, the `Krimp`-algorithm and the works following up upon the the `Krimp`-algorithm are summarised [VVLS11].

2.1 Ensemble Learning

Ensemble learning methods were originally created to reduce variance, and thereby increase accuracy in decision-making systems [Pol12]. A metaphor to describe this is: rather than consulting a single expert with lots of knowledge, consult a series of experts where each of the experts focuses on a certain aspect of the problem. From a statistical point of view, any classification error consists of two components: bias and variance. Bias is captured in the accuracy of a classifier and the variance in the precision of the classifier when trained on various training sets. Often there is a trade-off between these two components: classifiers with low bias tend to have high variance and vice versa. It is known that averaging has a smoothing effect, this means that smoothing reduces variance. Ensemble systems create a series of classifiers with similar bias. By combining the outputs (averaging), the variance is reduced. Combining the outputs does not generally lead to a better performance but the likelihood that a bad classifier is chosen is reduced.

An ensemble learning system consists of three components: data sampling and selection, training the classifiers and combining the classifiers. Data sampling and selection is important for the notion of diversity. If every classifier is trained on relatively diverse data, the outputs will be independent or preferably negatively correlated. The most common way to achieve is by training the classifiers on different subsets of training data. Another way is by using different subsets of features to train each classifier. Training the individual classifier is the core of ensemble-based systems. Combining the outputs can be done in many ways. This depends somewhat on the type of classifier used.

2.1.1 Boosting

The idea behind boosting is that a combination of simple classifiers, obtained by a weak learner, are able to perform better than any of the simple classifiers alone [FF12]. A weak learner is a learning algorithm which classifies cases correctly with probability strictly higher than 0.5. It needs to be slightly better than random guessing. A strong learner on the other hand yields a classifier with high accuracy. The concept of weak and strong learners is rooted in the theory of PAC (probably approximately correct) learning.

Definition 1 (Strong learner)**.** *Let $f : \mathcal{X} \to \{-1, +1\}$ be a hypothesis (classification rule), such that $f \in \mathscr{F}$, where $\mathscr{F}$ is some class of functions from $\mathcal{X}$ to $\{-1, +1\}$. Let $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ be a set of pairs such that $y_i = f(\mathbf{x}_i)$ and $\mathbf{x}_i$ are samples from some distribution P. Given enough data, a strong learner is able to produce an arbitrarily good classifier with high probability: for every $P, f \in \mathscr{F}, \epsilon \geq 0$ and $\delta \leq 0.5$, with probability no less than $1 - \delta$, it outputs a classifier $h : \mathcal{X} \to \{-1, +1\}$ which satisfies $\mathbb{P}_P[h(\mathbf{x}) \neq f(x)] \leq \epsilon$*

Definition 2 (Weak learner)**.** *Let $f : \mathcal{X} \to \{-1, +1\}$ be a hypothesis (classification rule), such that $f \in \mathscr{F}$, where $\mathscr{F}$ is some class of functions from $\mathcal{X}$ to $\{-1, +1\}$. Let $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ be a set of pairs such that $y_i = f(\mathbf{x}_i)$ and $\mathbf{x}_i$ are samples from some distribution P. Given enough data, a weak learner is able to produce an arbitrarily good classifier with high probability: for every $P, f \in \mathscr{F}$ and a particular pair $\epsilon_0 \geq 0$ and $\delta_0 \leq 0.5$, with probability no less than $1 - \delta$, it outputs a classifier $h : \mathcal{X} \to \{-1, +1\}$ which satisfies $\mathbb{P}_P[h(\mathbf{x}) \neq f(x)] \leq \epsilon_0$*

The underlying idea behind boosting is that a strong learner can be obtained by combining weak learners. This means that weak and strong learnability are equivalent in the sense that one can obtain a strong learner from a combination of weak learners.

A popular boosting algorithm, proposed by Freund and Schapire, is the adaptive boosting (AdaBoost) algorithm [FS95]. AdabBoost uses weighted versions of the same training data instead of selecting random samples. Because the same data is used repeatedly, the dataset does not need to be very large. AdaBoost learns a set of classifiers by using a weak learner. The weak classifiers are obtained sequentially by using reweighed versions of the training set. These weights depend on the misclassification errors of the previous classifiers. This allows for focus on the patterns that were not classified well by the previous weak classifiers.

2.1.2 Bagging and Random Forests

Bagging is another form of ensemble learning, which stands for Bootstrap Aggregation [Pol12]. Given a dataset $\mathcal{D}$ with N records, bagging trains T independent classifiers. Each of the classifiers is trained by sampling N instances with replacement from $\mathcal{D}$. The diversity between the classifiers comes from the variation within the bootstrapped samples on which each of the classifiers is trained in combination with using a weak classifier. The output of the classifiers is combined by using a simple majority vote.

Random forests are an extension of bagging, and were developed to be a competitor of boosting [CCS12]; [Bre01]. From a computational perspective Random Forests are appealing because they are relatively fast to train, predication is also fast, have very few parameters to tune, and can directly be used for high-dimensional problems. From a statistical point of view Random Forests are interesting because they can be used for unsupervised learning, measuring variable importance and differential class weighting.

A Random Forest is an ensemble model consisting out of trees. Like before, the goal is to find a function $f(\mathbf{x})$ for predicting y. The prediction function is determined by a loss function $\mathcal{L}(y, f(\mathbf{x}))$ such that the expected value of the loss is minimised with respect to the joint distribution X and Y.

$$\mathbb{E}_{XY}(\mathcal{L}(y, f(\mathbf{x})))$$

The prediction function f consists out of a collection of base classifiers $h_1(\mathbf{x}), \ldots, h_N(\mathbf{x})$. The base classifiers formally are trees $h(\mathbf{x}, \theta)$ but θ is omitted from notation. The output of the function f is the majority output from all base classifiers.

2.2 Clustering

In clustering the goal is to separate an unlabelled data set into a set of natural hidden data structures [XW05]. Clustering is also called unsupervised classification because the class labels are unknown. This is applied in exploratory data analysis. Clustering algorithms partition the data into a number of groups. A cluster can be seen as a group containing homogeneous objects in some sense and is separated from other groups. This means that patterns in the same cluster must be similar and patterns in different clusters must be different. Hard partitioning is the most popular form of clusters, here an object can only belong to a single cluster. The goal is to divide a dataset $\mathcal{D}$ into k disjoint groups.

Definition 3 (Hard Clustering)**.** *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. $\forall i \in [1, K] : \mathcal{D}_j \neq \emptyset$
2. $\bigcup_{i=1}^{K} \mathcal{D}_i = \mathcal{D}$
3. $\forall i, j \in [1, K] : i \neq j \Rightarrow \mathcal{D}_i \cap \mathcal{D}_j = \emptyset$

The borders between clusters can be softened using some kernel. To go even further, it is also possible for a pattern to belong to all clusters with a degree of membership $u_{i,j} \in [0, 1]$, which is the membership coefficient of the jth object in the ith cluster and is called fuzzy clustering. The membership coefficient must satisfy the following two constraints:

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

Fuzzy c-means (FCM) clustering is one of the most widely used fuzzy clustering algorithms [Dun73]. The algorithm partitions the set of data points $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ into a set of c fuzzy clusters. The algorithm returns a set of cluster centres $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_c\}$ and a partition matrix U. The matrix contains a weight $u_{i,j}$ for every data point $\mathbf{x}_i$ and cluster $\mathbf{c}_j$, such that $u_{i,j}$ represents the degree of which data point $\mathbf{x}_i$ belongs to cluster $\mathbf{c}_j$. Each of the cluster centres c_k are computed using a weighting function w_k for every data point x:

$$c_k = \frac{\sum_x w_k(x)^m x}{\sum_x w_k(x)^m}$$

where m is any positive real number. This is the mean of all data points, weighted by their degree of belonging to cluster k. The goal is to minimize the following objective function

$$\arg\min_{C} \sum_{i=1}^{n} \sum_{j=1}^{c} w_{i,j}^{m} \parallel \mathbf{x}_i - \mathbf{c}_j \parallel^2 \qquad \text{where} \qquad w_{i,j} = \frac{1}{\sum_{k=1}^{c} \left(\frac{\|\mathbf{x}_i - \mathbf{c}_j\|}{\|\mathbf{x}_i - \mathbf{c}_k\|} \right)^{\frac{2}{m-1}}}$$

The FCM algorithm works as by first choosing a number of clusters, then assign random weights in the first partition matrix $U^{(0)}$. Then, until convergence is reached: computer the centre c_k for every cluster and for every data point compute $w_k(x)$. In the FCM algorithm, m is the current iteration the algorithm is in. Convergence is defined as the change in coefficient values being no more than ϵ.

The algorithm described above is based on the k-means clustering algorithm, which computes a hard clustering and is also referred to as Lloyd's algorithm [Llo06]. The algorithm partitions a set of n data points $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ into k sets $S = \{s_1, \ldots, s_k\}$. The object function then is:

$$\arg\min_S \sum_{i=1}^{k} \sum_{\mathbf{x} \in S} \| \mathbf{x} - \mu_i \|^2$$

This assigns each data point to the cluster whose mean has the least squared Euclidean distance is the lowest. The downside of the methods is that both use all features of the data and thus suffers from the curse of dimensionality. This means that the distance between data points will increase when the number of dimensions grow. A method of which the performance does not rely on the number of dimension is preferred.

2.2.1 Clustering and Ensemble Structures

Vega and Ruiz survey a number of clustering ensemble algorithms [VPRS11]. The observation is made that clustering ensemble algorithms are made up of two steps: generation and consensus. The generation step generates a number of clusterings. These can be obtained quickly when using weak clustering algorithms. The next step is to combine these generated clusterings. A consensus function determines the assigned cluster for each data point by aggregating the obtained clusterings in the previous step. This can be done by either object co-occurrence or median partitioning. In object co-occurrence the cluster of a data point is the majority of the clusters it belongs to in the ensemble structure. In the second approach the median partition is defined as:

$$P^* = \arg\max_{P_i \in \mathcal{P}} \sum_j \Gamma(P_i, P_j)$$

Here Γ is some similarity function between partitions. This is the partition that maximizes the similarity with all clusterings in ensemble structure. This is generalised in algorithm 1:

Algorithm 1: A general template for a clustering algorithm by using ensemble structures

```
Data: Dataset $\mathcal{D}$, number of clusterings $m$, consensus function $\Gamma$ and weak
      clustering algorithm $C$
Result: A clustering $P^*$
1 $\mathcal{P} \leftarrow \emptyset$;
2 for $i \leftarrow 0$ to $m$ do
3     $D_i \leftarrow selectData(\mathcal{D})$;
4     $\mathcal{P} \leftarrow \mathcal{P} \cup C(\mathcal{D}_i)$;
5 end
6 $P^* \leftarrow \arg\max_{P_i \in \mathcal{P}} \sum_j^m \Gamma(P_i, P_j)$;
```

Clustering and Boosting

Frossyniotis et. al. propose an iterative multiple clustering approach that is called boost-clustering, which iteratively recycles the data set and provides multiple clusterings and results in a single partition [FLS04]. A distribution over the data points is computed and a new training set is randomly sampled from the original dataset. Afterwards, a basic clustering algorithm is used to partition the newly samples data set. The final clustering is obtained by means of weighted voting, a weight is assigned to every partition according to some quality measure.

Smeraldi et. al. propose Cloosting, iterative clustering by using AdaBoost [Sme+11]. It is argued that the homogeneity within the cluster and separation between the clusters are obtained by, first, the use of regularised AdaBoost to reject outliers. AdaBoost has a tendency to put higher weights on data points which are difficult to classify. These data points often happen to be outliers in high-noise cases. This issue is alleviated by using the regularised version of AdaBoost. Second, weak learners allow for specialisation of a particular region in the feature space. At last, the decision boundaries are smoothed with a Gaussian kernel. The algorithm works by starting with K random partitions. Then, until convergence is reached: First train K strong classifiers to recognise the elements of each cluster using regularised AdaBoost. Second, compute the score S_k for each data point for each cluster using some score function S_k. Assign each point to the cluster which maximizes the score.

Algorithm 2: A general template for a clustering algorithm by using boosting

```
Data: Dataset $\mathcal{D}$, number of clusterings $m$, consensus function $\Gamma$ and weak
      clustering algorithm $C$
1 $\mathcal{P} \leftarrow \emptyset$;
2 for $i \leftarrow 0$ to $m$ do
3     $D_i \leftarrow sampleData(\mathcal{D})$;
4     $\mathcal{P} \leftarrow \mathcal{P} \cup C(\mathcal{D}_i)$;
5 end
6 $P^* \leftarrow \arg\max_{P_i \in \mathcal{P}} \sum_j^m \Gamma(P_i, P_j)$;
```

Clustering and Bagging

Dudoit and Fridlyand describe a method to apply bagging to clustering called BagClust [DF03]. First apply a clustering algorithm P to the entire data set $\mathcal{D}$ to obtain the clusters for each data point. Second, form a bootstrap sample on the data set, $\mathcal{D}^b$ denotes the bth bootstrap sample. Then apply P on $\mathcal{D}^b$ to obtain the cluster for each data point. Afterwards, permute the cluster labels in the bootstrap sample $\mathcal{D}^b$ such that there is maximum overlap between $\mathcal{D}^b$ and $\mathcal{D}$. Repeat this procedure B times and then assign a bagged cluster label to each data point.

The approach by Minaei et. al. is similar to the previous one [MBTP04]. First, create a reference clustering using some algorithm P. Then draw b bootstrap samples $\mathcal{D}^b$ of $\mathcal{D}$ and cluster each one using P to obtain a clustering P^b. At last, use some consensus function Γ on the obtained clusterings to obtain a final clustering. The labels are assigned according to the initial clustering.

The major difference between both methods is the sampling method that are used. The boosting methods use random samples or samples derived from the last

Algorithm 3: A general template for a clustering algorithm by using bagging

```
Data: Dataset $\mathcal{D}$, number of clusterings $m$, consensus function $\Gamma$ and weak
      clustering algorithm $C$
Result: A clustering $P^*$
1 $\mathcal{P} \leftarrow \emptyset$;
2 $P_{ref} \leftarrow C(\mathcal{D})$;
3 for $i \leftarrow 0$ to $m$ do
4     $D_i \leftarrow bootstrapSample(\mathcal{D})$;
5     $\mathcal{P} \leftarrow \mathcal{P} \cup C(\mathcal{D}_i)$;
6 end
7 $P^* \leftarrow \arg\max_{P_i \in \mathcal{P}} \sum_j^m \Gamma(P_i, P_j)$;
8 $P^* \leftarrow assignLabels(P_{ref}, P^*)$
```

iteration while bagging uses bootstrap samples. Bagging however requires a reference clustering P_{ref} to assign the labels. The algorithm templates 1, 2 and 3 can all be easily extended by inserting additional steps as long as the output is a clustering.

2.3 Compression Based Data Mining

Many data mining problems can be related to the Kolmogorov Complexity [Grü05]. This means they can be practically solved by means of compression. MDL is a practical version of the Kolmogorov Complexity. MDL and the Kolmogorov Complexity make use of the principle of Induction by Compression. For two-part MDL, this is described as:

Definition 4. *Given a set of models $\mathcal{H}$, the best model $H \in \mathcal{H}$ is the model that minimises*

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

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

Two-part MDL is used because we want to find the set of frequent item sets that yield the best compression. This is called the compressor. In order to use MDL we need to answer the following three questions:

- What are the models $\mathcal{H}$?
- How does $H \in \mathcal{H}$ describe a database?
- How is all this encoded in bits?

2.3.1 `Krimp`: mining items sets that compress

Siebes et al. use the notion of the Minimum Description Length (MDL) to create code tables, which are used to mine the interesting sets of patterns from data [VVLS11]. The main idea is the use of a code table to compress the database.

Definition 5 (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.*

In order to encode a transaction $t \in \mathcal{D}$ over $\mathcal{I}$, a cover function $cover(CT, t,)$ is needed to identify which elements of CT are used to encode the transaction t. The output of this function is a disjoint set of element of CT that cover t:

Definition 6 (Cover)**.** *Let $\mathcal{D}$ be a database over a set of $\mathcal{I}$, t drawn from $\mathcal{D}$, let $\mathcal{CT}$ be the set of all possible code tables over $\mathcal{I}$, and CT a code table such that $CT \in \mathcal{CT}$. Then, $cover : \mathcal{CT} \times \mathcal{P}(\mathcal{I}) \mapsto \mathcal{P}(\mathcal{P}(\mathcal{I}))$ is a cover function iff it returns a set of item sets such that:*

1. *$cover(CT, t)$ is a subset of CS*
2. *if $X, Y \in cover(CT, t)$, then either $X = Y$ or $X \cap Y = \emptyset$*
3. *The union of all item sets in $cover(CT, t)$ equals t*

A database $\mathcal{D}$ can be encoded with code table CT by replacing each transaction $t \in \mathcal{D}$ by the code of the items sets in the cover of t, that is:

$$t \rightarrow \{code_{CT}(X) \mid X \in cover(CT, t)\}$$

The codes in the code table should be chosen in such a way that the used item set which is used the most has the shortest length to get the best compression. The actual codes are never used, only the lengths of codes of the item sets. The optimal lengths of the codes can be computed by using the Shannon entropy.

Theorem 1. *Let P be a distribution on some finite set $\mathcal{D}$, there exists an optimal prefix code $\mathcal{C}$ on $\mathcal{D}$ such that the length of the code for $d \in \mathcal{D}$, which is denoted as $L(d)$, is given by*

$$L(d) = -\log(\mathbb{P}(d))$$

The goal is to find the best code table and not to actually compress the dataset.

Definition 7. *Let $\mathcal{D}$ be a transactional database over the set of items $\mathcal{I}$, $\mathcal{C}$ a prefix code, cover a cover function, and CT a code table over $\mathcal{I}$ and $\mathcal{C}$. The usage of an item set $X \in CT$ is then defined as:*

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

From this we can derive a probability distribution for all item sets X in the code table CT.

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

A code table is code-optimal iff all the codes for all item sets are optimal, that is:

$$\forall X \in CT : L(code_{CT}(X)) = code_{CT}(X) = -\log(\mathbb{P}(X \mid \mathcal{D}))$$

Lemma 1. *Let $\mathcal{D}$ be transactional database over $\mathcal{I}$, CT be a code table over $\mathcal{I}$ and code-optimal for $\mathcal{D}$, and usage the usage function for cover:*

1. *For any $t \in \mathcal{D}$, the encode length in bits $L(t \mid CT)$ is:*

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

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

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

To use the MDL principle we need $L(H)$ and $L(\mathcal{D} \mid H)$, the latter can be computed by using Lemma 1. To compute $L(H)$, we make use of the standard code table ST. This is the optimal coding for $\mathcal{D}$ when only the frequencies of the singleton item sets are known, and is the simplest independent description of the data.

Definition 8. *Let $\mathcal{D}$ be transactional database over $\mathcal{I}$, CT be a code table over $\mathcal{I}$ and code-optimal for $\mathcal{D}$. The size of CT $L(CT \mid \mathcal{D})$ in bit is:*

$$L(CT \mid \mathcal{D}) = \sum_{X \in CT: usage_{\mathcal{D}}(X) \neq 0} = L(code_{ST}(X)) + L(code_{CT}(X))$$

The set of item sets of a coding table is called the coding set CS. The goal is to find the smallest coding set from a set of item sets $CS \subseteq \mathcal{F}$ such that for the corresponding code table CT, $L(\mathcal{D}, CT)$ is minimal. This is called the *Minimal Coding Set Problem*. This means that we have to find the optimal code table and cover function.

Since the search space is too large to search exhaustively, a heuristic is applied. This heuristic considers the code table in a fixed order and is called the Standard Cover Order. The elements in the code table are first decreasingly sorted by cardinality, second decreasing on support, and at last increasing on lexicography. This makes the ordering total. The standard cover algorithm works as follows: for a transaction t, the code table is traversed in the standard cover order. An item set $X \in CT$ is included in the cover of t iff $X \subseteq t$. Afterwards X is removed from t. This is repeated for $t \backslash X$ until everything is covered. Before starting the `Krimp`-algorithm, the set of candidate item sets $\mathcal{F}$ is sorted by the Standard Candidate Order. We first sort $\mathcal{F}$ decreasing on support, second decreasing on cardinality, and at last increasing on lexicography.

- Start with the standard code table ST
- Keep adding candidates from the sorted set $\mathcal{F}$ one by one. Each time, take the item set that is maximal according to the Standard Candidate Order. Then cover the database using the standard cover algorithm. If the obtained encoding compresses the data better, keep it. Else, discard it.

Algorithm 4: The `Krimp` algorithm

```
1  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 $CT$
2      $CT \leftarrow$ Standard Code Table($\mathcal{D}$);
3      $\mathcal{F}_0 \leftarrow \mathcal{F}$ in Standard Candidate Order;
4      for $F \in \mathcal{F}_0 \backslash \mathcal{I}$ do
5          $CT_c \leftarrow (CT \cup F)$ in Standard Cover Order;
6          if $L(\mathcal{D}, CT_c) < L(\mathcal{D}, CT)$ then
7              $CT \leftarrow CT_c$;
8          end
9      end
10     return $CT$;
```

The `Krimp`-algorithm can also be used for classification. The assumption made here is that database is an independent and identically distributed (i.i.d.) drawn sample from some underlying distribution. The assumption is similar to the Naive

Bayes assumption, the item sets in CT are independent if any co-occurrence of two item sets $X, Y \in CT$ in the cover of a transaction is independent. This means that $L(t \mid CT) = -\log(P(t \mid \mathcal{D}))$.

Lemma 2. *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two bags of transactions over $\mathcal{I}$, which are samples from two different distributions, and t an arbitrary transaction over $\mathcal{I}$. Let CT_1 and CT_2 be the optimal code tables for $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively, then:*

$$L(t \mid CT_1) > L(t|CT_2) \Rightarrow P(t \mid \mathcal{D}_1) < P(t \mid \mathcal{D}_2)$$

This means the best choice is to assign t to the distribution which yields the shortest code length.

2.3.2 Compression picks the significant item sets

Leeuwen et al. argue that `Krimp` picks the item sets that matter [LVS06]. From enormous amounts of candidates it only selects a low amount of item sets. The selected item sets obtain a high compression ratio on the data and, furthermore, it shows similar performance with today's top classifying methods even though `Krimp` has not been designed for classification. The conclusion made is that `Krimp` is well suited for capturing the characteristics of the data.

An observation made is that while the compression of the item sets is already very good, but by allowing overlap in the cover of transaction, fewer item sets would be needed for a full cover of the database. This would, however, make covering and properly selecting the item sets computationally more complex. The `Krimp`-algorithm also scales particularly well. This is because the time to consider a single candidate set scales linearly with its support. An improvement in the runtime is obtained when only using the closed frequent item sets. The reason for this is that the closed representation is good at preserving the most important item sets, so this means there are less candidate sets that have to be considered by the `Krimp`-algorithm.

2.3.3 Characterising the Difference

Vreeken et al. extend the notion of capturing the data distribution by using compression with a generic dissimilarity measure on databases [VVLS07]. This approach can identify the patterns that characterise the difference between two distributions.

The MDL principle implies that when we have an optimal encoding for a database $\mathcal{D}_1$, $\mathcal{D}_1$ will have a shorter encoding than some other database $\mathcal{D}_2$. So let MDL_i be the optimal compressor induced from $\mathcal{D}_i$, and let t be a transaction in $\mathcal{D}_1$, then $\mid MDL_1(t) - MDL_2(t) \mid$ is small if t is equally likely to be generated by the underlying distributions of $\mathcal{D}_1$ and $\mathcal{D}_2$. This difference is large if t is more likely to be generated by the underlying distribution from one database than the other. This means that if the differences in code length is large, then on average the smallest code length will be $MDL_1(t)$. This should also hold for the code tables generated by the `Krimp`-algorithm, $CT_1(t) - CT_2(t)$ measures how characteristic t is for $\mathcal{D}_1$. Now we need a way to aggregate the difference over all transactions. This is defined as the Aggregated Code Length Difference:

$$ACLD(\mathcal{D}_1, CT_2) = \frac{CT_2(\mathcal{D}_1) - CT_1(\mathcal{D}_1)}{CT_1(\mathcal{D}_1)}$$

Note that ACLD is not symmetric, it is not a proper measure. To make this symmetric, the maximum value of two Aggregated Code Length Differences is taken: $\max\{ACLD(\mathcal{D}_a, CT_b), ACLD(\mathcal{D}_b, CT_a)\}$. This can be rewritten in terms of the sizes of the compressed database to get the following definition for dissimilarity measure:

Definition 9 (Dissimilarity Measure)**.** *For all databases x and y, define the code table dissimilarity measure DS between x and y as:*

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

There are three methods for analysing the difference. First, the code table covers of databases can be compared. This gives information on which patterns are important in one database are either over or under-expressed in another database. This shows the characteristics of the differences in structure between the two databases. To perform this analysis, first the `Krimp`-algorithm is run to obtain a code table for a database $\mathcal{D}_2$, which is used to cover $\mathcal{D}_1$. The identification of the differences is then done by finding those patterns in the code table that have a large shift in frequency between the two database covers. The same can be done the other way around to gain even more insight. The second approach is to analyse how specific transactions are covered by different code tables. This gives information, in detail, where differences are identified by the code tables. This is useful in case specific transactions are of interest. This analysis is performed by computing the respective code tables for two databases. After computing the individual code length differences, pick those transactions that fit well in one database and not in the other. After selecting a transaction, cover it with both code tables separately and visualise which patterns are used for this. It is covered by longer and more frequent patterns if it belongs to a certain distribution. Last, knowledge can be extracted about specific differences and similarities between the distributions from the code tables. This analysis is performed by directly comparing the patterns in both code tables. For each pattern in a code table, its own length is compared to its encoded length in the other code table. If the distributions are similar, the encoded lengths should be similar. The patterns for which the encoded lengths differ significantly characterise the difference between the two distributions.

2.3.4 Identifying the components

Leeuwen et al. observe that databases are a mixture of different distributions and that each of these distributions are characteristic components of a database [LVS09]. The goal is to discover an optimal partitioning of the database. The characteristics of the different components are heterogeneous, while the individual components are homogeneous. The formal problem statement is as follows: Find a partitioning of the database such that

$$\sum_{i \in \{1,\ldots,k\}} L(CT_i, db_i)$$

is minimised. The optimal number of components is determined by MDL by picking the smallest encoded size over every possible partitioning and all possible code tables. For any partitioning, the best code tables are the optimal code tables. Given a set of code tables, each transaction goes to the code table that compresses the transaction best. This gives two ways to solve the problem: find an optimal partitioning or try to find an optimal set of code tables.

Model-driven component identification is concerned with identifying components by finding an optimal set of code tables. The idea is that a code table that captures the entire distribution of a database models the underlying component distributions implicitly. The code tables for specific components can be extracted from the original code table. The algorithm first obtains a code table for the entire database by using the `Krimp`-algorithm. Then, for all possible values of $k \in [1, |\mathcal{D}|]$, identify the best k components. The solution is the one that minimises the total encoded size. To identify the k components, start with k copies of the original code table. Iteratively eliminate the code table element that reduces the compressed size the most until compression cannot be improved any more. Iteratively remove each element in the code table temporarily to determine the best possible elimination. To compute the total compressed size, each transaction is assigned to the code table that compresses the transaction the best. For each component, recompute the optimal code lengths and the total encoded size. A property of this method is that the number of patterns required to define the components is never higher than the number of item sets in the original code table.

Data-driven component identification is concerned with identifying the component by finding the MDL-optimal partitioning of the data. Provided a particular source distribution has more transactions in one database than in another, transactions of that distribution will be encoded shorter by a code table (compressor) induced on that data. This property can be exploited to find the components of a database. Given a partition of the data, a code table can be induced for each part. Each transaction is assigned to the code table which encodes it the shortest. This is done iteratively until all transactions remain in the same part. This gives insight of the found clusters through code tables. The algorithm starts with a random partitioning and lets MDL pick the best result.

The optimal number of components is automatically determined by MDL for both methods. No parameters have to be set. When dealing with very large databases, the data-driven methods will provide good results quickly. For analysis of reasonable amount of data, the model-driven method has the advantage of the data-driven method of characterising the components very well in a modest number of patterns. Both methods are suitable to identify the components without prior knowledge, without the need of a distance metric and without a specification of the number of components.

2.3.5 A structure function for transaction data

Siebes and Kersten use a set of code tables to characterize the data instead of a single code table [SK11]. The fundamental assumption made in data analysis is that a dataset $\mathcal{D}$ consists out of a structural component and an an accidental component. The structural component is the data to be captured by the model and the accidental component is the data as generated from that model. In practice often there are good models instead of the optimal model, which captures all the structure in the data. Siebes and Kersten characterise a dataset by a series of models, where each of the models captures an aspect of the structural component, in this case the models are code tables. Each of these code tables is the best from a set of alternative models for the dataset. All the models in this set have the same complexity. The complexity of a code table is the number of non-singleton item sets in the leftmost column. The best model based on the MDL principle from the set of all models is the one that best compresses the dataset. The non-on singleton item sets in the code table describe the essential correlation structure in the data. By varying the complexity of the sets of alternative models, a function is defined from the set of natural numbers to the set of models for the data set. This is called the structure function, denoted by $\mathcal{K}_{\mathcal{D}}$. The structure function $\mathcal{K}_{\mathcal{D}}$ allows for inspection of the correlation structure of the data at different levels of granularity.

The definition of the code tables differs from the definition used before. First, it is not required that all singleton item sets are present in the code table. Second, the code tables produced by the `Krimp`-algorithm have stronger restrictions on the order of item sets, these do not hold here. The code table only containing the singleton item sets from $\mathcal{P}(\mathcal{I})$ is denoted by CT_α. The Shannon code is based on $usage_{\mathcal{D}}(\{i\}) = sup(\{i\})$. The code table which contains all unique transaction for some dataset, ordered first on length and then lexicographically, is denoted by CT_ω. The Shannon code is based on $usage_{\mathcal{D}}(t) = sup(t)$. Given a CT, a database D^{CT} can be constructed for which CT is a code table. This is possible because for every $I \in CT$, $\mathbb{P}(I)$ is known.

Theorem 2. *For each $D \in \mathcal{D}$ there exists at least one $CT \in \mathcal{CT}$ such that CT is a code table for D. And, for each $CT \in \mathcal{CT}$ there exists at least one $D \in \mathcal{D}$ such that CT is a code table for D.*

This gives the following definition:

Definition 10. *Let $D \in \mathcal{D}$ and $CT \in \mathcal{CT}$, D defines a subset of $\mathcal{CT}$ by*

$$\mathcal{CT}|_D = \{CT \in \mathcal{CT} \mid CT \text{ is a code table for } D\},$$

and, CT defines a subset of $\mathcal{D}$ by

$$\mathcal{D}|_{CT} = \{D \in \mathcal{D} \mid CT \text{ is a code table for } D\}$$

There exists an equivalence relation on $\mathcal{CT}$. We say that $CT_1, CT_2 \in \mathcal{CT}$ are equivalent, $CT_1 \equiv CT_2 \Leftrightarrow$ every row in CT_1 is also a row in CT_2, and for $I, J \in CT_1$, $I \cap J \neq \emptyset \wedge I$ occurs before J in $CT_1 \Rightarrow I$ occurs before J in CT_2. This follows from the fact that if two code tables were equivalent, then all the transactions would have the same cover for both code tables. This means that databases are completely characterised by their code tables.

Theorem 3. *Let $D_1, D_2 \in \mathcal{D}$, then*

$$\mathcal{CT}|_{D_1} = \mathcal{CT}|_{D_2} \Leftrightarrow \forall I \in \mathcal{P}(I) : sup_{D_1}(I) = sup_{D_2}(I)$$

This theorem shows that, whatever structure D has, it is captured by at least one of the code tables. This also holds the other way: code tables are characterised by the databases they model.

Theorem 4. *Let $CT_1, CT_2 \in \mathcal{CT}$, then*

$$\mathcal{D}|_{CT_1} = \mathcal{D}|_{CT_2} \Leftrightarrow CT_1 \equiv CT_2$$

Some code tables are satisfied by smaller sets of databases. This means that exists a natural partial order on code tables based on these sets.

Definition 11. *Let $CT_1, CT_2 \in \mathcal{CT}$:*

$$CT_1 \preceq CT_2 \Leftrightarrow \mathcal{D}|_{CT_1} \supseteq \mathcal{D}|_{CT_2}$$

In general it is hard to check $\mathcal{D}|_{CT_1} \supseteq \mathcal{D}|_{CT_2}$. Since the assumption is made that CT_1 and CT_2 are models of the same dataset the check can be done by using the following theorem:

Theorem 5. *Let $D \in \mathcal{D}$ and let $CT_1, CT_2 \in \mathcal{CT}|_D$, $CT_1 \preceq CT_2$ iff:*

$$\forall J \in CT_2 : cover(CT_1, J) \textit{ succeeds } \wedge \forall I \in CT_1 \exists J \in CT_2 : I \in cover(CT_1, J)$$

This means that we obtain a more restrictive model by either extending one of the item sets or by adding a new item set. A consequence is that there exists a smallest and largest element for $\mathcal{CT}_D$.

Theorem 6. *Let $D \in \mathcal{D}$, then for all $CT \in \mathcal{CT}_D$:*

$$CT_\alpha^D \preceq CT \preceq CT_\omega^D$$

Putting all this together we get the following insights on the structure of $\mathcal{CT}$:

Corollary 1. *Let $D_1, D_2 \in \mathcal{D}$, then:*

1. $\mathcal{CT}|_{D_1} \cap \mathcal{CT}|_{D_2} \neq \emptyset \Leftrightarrow CT_\alpha^{D_1} \equiv CT_\alpha^{D_2}$
2. $CT_\omega^{D_1} \in \mathcal{CT}|_{D_2} \Leftrightarrow \mathcal{CT}|_{D_1} \subseteq \mathcal{CT}|_{D_2}$

The series of models that characterise D lie somewhere between CT_α and CT_ω, since they respectively are an under-specification and over-specification. There are two methods to make a code table more restrictive. We can either extend the item sets or add an item set. The second methods suggests that there is a natural partitioning of $\mathcal{CT}|_\mathcal{D}$.

Definition 12. *For $k \in \mathbb{N}$:*

$$\mathcal{CT}|_\mathcal{D}^k = \{CT \in \mathcal{CT}|_\mathcal{D} \mid CT \textit{ has exactly } k \textit{ non-singletons } \}$$

This definition means that all code tables in $\mathcal{CT}|_\mathcal{D}^k$ have the same complexity. This also means that $\mathcal{CT}|_\mathcal{D}^k \preceq \mathcal{CT}|_\mathcal{D}^{k+1}$. The structure function is then defined as:

Definition 13 (Structure function). *For $D \in \mathcal{D}$, the partial function $\mathcal{K}_\mathcal{D} : \mathbb{N} \to \mathcal{CT}_D$ is defined as:*

$$\mathcal{K}_D(k) = \underset{CT \in \mathcal{CT}|_D^k}{\arg\min} L(CT, D)$$

The goal is then to find an algorithm that computes this structure function and to show how the structure function provides insight into the structure of the data.

There is not enough structure on $\mathcal{CT}|_D$ to compute $\mathcal{K}_\mathcal{D}$ efficiently, for this reason the heuristic algorithm `Groei` is introduced to approximate $\mathcal{K}_\mathcal{D}$. First, the candidate set is restricted to all-non singleton frequent item sets and all singleton item sets. The algorithm is a beam search on $\mathcal{CT}|_\mathcal{D}^k$. For every k, starting from $k = 1$, search for the b best $CT \in \mathcal{CT}|_\mathcal{D}^k$. If the best of these b solutions is better than the best for $k - 1$, continue as before with $k + 1$, else halt. For $k = 0$, the 'best' table is CT_α^D. In the end we have b tables of complexity k. The algorithm tries to grow more and more complex tables to get insight in the structure of the data.

2.3.6 Directly Mining Descriptive Patterns

Smets and Vreeken introduce `Slim`, an any-time algorithm for mining sets of item sets directly from the data. It is argued that the search space grows to be infeasible for `Krimp` for large and dense databases because it first needs to mine the frequent item sets. Also, `Krimp` only considers candidates once and in fixed order. The implication of this is that candidates are rejected that can be useful later on. The key idea of the paper is to find good code table candidates by mining good item sets in the cover space.

The cover of a data set $\mathcal{D}$ by a code table CT can be viewed as a $|\mathcal{D}| \times |CT|$ binary matrix $\mathbb{C}$, where every row corresponds to a transaction $t \in \mathcal{D}$ and each column corresponds to the elements $X \in CT$. The value of a cell is 1 when $X \in cover(t)$, and 0 otherwise. This cover matrix is also called the cover space.

In optimising compression, the quality of a candidate X is the gain in total compression when X is added to CT. Every candidate X corresponds to the combination of code table elements, which is identified by itemset $Y \in \mathbb{C}$. Every candidate X can be construction by taking the union of various code table elements $X_i \in CT$ identified by $i \in Y$. The candidate X is then defined as $X = \bigcup_{i \in Y} X_i$.

It is possible to estimate the compression gain and then only calculate the exact gain for the best estimated candidate. The itemsets that will most likely give the best gain are the highly frequent sets which consist of only a few items. Let X and Y be item sets in CT, and let CT' be the code table after adding the union of these two item sets, i.e. $CT' = CT \oplus X \cup Y$. The *usage* of $X \cup Y$ can be estimated as $|usage(X) \cap usage(Y)|$. Let ΔL denote the difference in encoded size between CT and CT'. This is the gain in bits for candidate $X \cup Y$.

Definition 14 (Compression Gain)**.**

$$\begin{aligned}\Delta L(CT \oplus X \cup Y, \mathcal{D}) &= L(CT, \mathcal{D}) - L(CT \oplus X \cup Y, \mathcal{D}) \\ &= \Delta L(\mathcal{D} \mid CT \oplus X \cup Y, \mathcal{D}) + \Delta L(CT \oplus X \cup Y, \mathcal{D} \mid \mathcal{D})\end{aligned}$$

Let $x = usage(X)$ for item set $X \in CT$, and $s = \sum_{X \in CT} x$. Similarly, x' and s' are used for CT'. Finally, let $xy' = usage(X \cup Y)$ for CT'. The difference between encoding $\mathcal{D}$ by either CT or CT' is then defined as:

Definition 15.

$$\Delta L(\mathcal{D} \mid CT \oplus X \cup Y) = s \log s - s' \log s + xy' \log xy' - \sum_{C \in CT c \neq c'} (c \log c - c' \log c')$$

And the difference in the complexity of the models is:

Definition 16.

$$\begin{aligned}\Delta L(CT \oplus X \cup Y \mid \mathcal{D}) = \log xy' - L(X \cup Y \mid ST) + |CT| \log s - |CT'| \log s' \\ + \sum_{\substack{C \in CT \\ c' \neq c \\ c'c \neq 0}} \log c' - \log c + \sum_{\substack{C \in CT \\ c' \neq c \\ c'c \neq 0}} \log c' - L(C \mid ST) \\ + \sum_{\substack{C \in CT \\ c' \neq c \\ c'c \neq 0}} L(C \mid ST) - \log c\end{aligned}$$

The assumption made is that only the *usage* for X, Y and $X \cup Y$ will change. For calculating the estimated gain in compressed size, the following rules are used:

$xy' = |usage(X) \cap usage(Y)|$, $x' = x - xy'$, $y' = y - xy'$ and $s' = s - xy'$. This makes it easy to compute an accurate estimate of ΔL for $X \cup Y$.

From this, the `Slim` algorithm has been created (Algorithm 5). First the standard code table is constructed from the data set. All pairwise combinations of $X, Y \in CT$ are considered as candidates in Gain Order (descending on $\Delta L(CT \oplus (X \cup Y), \mathcal{D})$) every iteration. A candidate is added to CT in Standard Cover Order. This process is repeated until the compression does not improve any more. It is not necessary to explicitly compute all the possible candidates. By applying branch-and-bound to find $X \cup Y$ with highest estimated gain, it is not needed to consider any element V or W with lower usage than the current best candidate $X \cup Y$ because the elements are traversed on usage.

Algorithm 5: The `Slim` algorithm

```
1 Function Slim($\mathcal{D}$)
      Data: Dataset $\mathcal{D}$
      Result: Code table $CT$
2     $CT \leftarrow$ Standard Code Table($\mathcal{D}$);
3     for $F \in \{X \cup Y : X, Y \in CT\}$ in Gain Order do
4         $CT_c \leftarrow (CT \oplus F)$ in Standard Cover Order;
5         if $L(\mathcal{D}, CT_c) < L(\mathcal{D}, CT)$ then
6             $CT \leftarrow post - prune(CT_c)$;
7         end
8     end
9     return $CT$;
```

3 Problem Description and Research Questions

In this chapter the problem description is presented with the formal problem definition. The research questions with their respective rationale are given in order to be able to solve the problem and evaluate the model. Furthermore, the relevance of this thesis project with respect to science, technology and society is argued.

3.1 Problem Description

In order to give the problem description, definitions for the encoded length $L(X)$ for the encoded length of tuples (data point) d, clusters C and the database $\mathcal{D}$ are needed. The definitions given in Chapter 2 cannot be applied directly because data points d belong to all clusters C with some degree of membership. The degree-of-membership is expressed as the probability that data point d_j belongs to cluster C_i, $\mathbb{P}(d_j \in C_i)$. This probability is proportional to length of tuple d_j when encoded by the respective code table of cluster C_i, CT_i.

Definition 17 (Membership Coefficient)**.** *The Membership Coefficient is the probability that tuple $d_j \in \mathcal{D}$ belongs to cluster C_i:*

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

This definition is based on the principle of induction, and in particular, the principle of the Universal Prior [RH11]. This Universal Prior uses the Kolmogorov complexity and is expressed as: $w_v^U = 2^{-K(v)}$. Since the Kolmogorov complexity is not computable, code tables with the cover algorithm are used instead because they are based on the Minimum Description Length principle, which is more practical. The encoded length of a data point d_j, given some cluster C_i, is based on the the code table CT_i and the probability that d_j belongs to C_i. This is inspired by the idea of the Bayesian mixture.

For the following definitions: let d denote a tuple in $\mathcal{D}$; let N denote the number of tuples in $\mathcal{D}$; let C denote the number of clusters; let CT_i denote the code table which belongs to cluster C_i;

Definition 18 (Expected Encoded Length)**.** *The Encoded Length of a tuple d is:*

$$\mathbb{E}[L(d)] = \sum_i^C L(d \mid CT_i) = \sum_i^C \mathbb{P}(d \in CT_i) CT_i(d)$$

Now it is possible to define the encoded length of some cluster C_i by its respective code table CT_i in terms of all the data points in the database D. This is the sum of the length all data points encoded by the code table CT_i of cluster C_i.

Definition 19 (Encoded Cluster Length)**.** *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 $\mathcal{D}$:*

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

The length of the encoded database is then the length of all the code tables of all clusters and the Encoded Length of all data points.

Definition 20 (Expected Encoded Database Length)**.** *The total encoded size of the database $\mathcal{D}$ using a set of code tables $\mathcal{CT}$ is:*

$$\begin{aligned}\mathbb{E}[L(\mathcal{D} \mid \mathcal{CT})] &= \sum_{i=1}^{C} L(CT_i) + \sum_{j=1}^{N} L(d_j) \\ &= \sum_{i=1}^{C} \Big(L(CT_i) + L(C_i \mid CT_i) \Big)\end{aligned}$$

Proof.

$$\begin{aligned}\mathbb{E}[L(\mathcal{D} \mid \mathcal{CT})] &= \sum_{i=1}^{C} L(CT_i) + \sum_{j=1}^{N} L(d_j) && \text{(Definition 20)} \\ &= \sum_{i=1}^{C} L(CT_i) + \sum_{j=1}^{N}\sum_{i=1}^{C} L(d_j \mid CT_i) && \text{(By Definition 18)} \\ &= \sum_{i=1}^{C} L(CT_i) + \sum_{i=1}^{C}\sum_{j=1}^{N} L(d_j \mid CT_i) && \text{(Commutativity Rule)} \\ &= \sum_{i=1}^{C} \Big(L(CT_i) + \sum_{j=1}^{N} L(d_j \mid CT_i) \Big) && \text{(Distributive Rule)} \\ &= \sum_{i=1}^{C} \Big(L(CT_i) + L(C_i \mid CT_i) \Big) && \text{(By Definition 19)}\end{aligned}$$

□

Recall that the goal is to approximate the underlying distribution as well as possible. This is, find the set of code tables that compresses the database the best. The ideal situation is when there is an infinite amount of data available. In that case the underlying distribution is approximated the best.

$$\sum_{i=1}^{\infty}\sum_{c=1}^{C} \mathbb{P}(d_i \in C_i) CT_c(d_j)$$

However, this is not computable in finite time. This problem can be tackled from several angles. The first one is the most straightforward approach. That is:

Problem Defintion. *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 $\mathcal{D}$.*

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

This is the problem definition that is tackled by `Krimp` and `Slim`. This definition can be extended to deal with partitions of a database and multiple code tables.

Problem Defintion. *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:*

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

An anti-chain is a subset of some partially ordered finite set S such that any two distinct elements in the subset are incomparable. These code tables differ in a single item set. If it is possible to find two tables according to this definition then the dataset contains a multi-valued dependency.

The last and hardest problem to tackle is that only the code tables which compress the database than some quality threshold Q are accepted to be part of the set of code tables $\mathcal{CT}$.

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

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

This problem is hard to tackle because the number of possible code tables for a single database $\mathcal{D}$ is potentially extremely large, $\mathcal{O}(|\mathcal{F}|)$.

First, a good description of the underlying data distribution needs to be found. This is according to the definition given by Leeuwen et. al. [LVS09]. The goal is to characterise the the underlying distribution - the best characterisation is the most concise description of the data. The probability (weight) that a data point belongs to a certain cluster is proportional to the encoded length of the data point by the code table of that cluster. The weight is thus determined by this probability because a higher probability will give a higher weight to a data point and vice versa. When this is compared to k-means and c-means fuzzy clustering, the concept of a middle point (centre) is given by the code table because it summarises a certain characteristic [Dun73]; [Llo06]. The concept of distance to the centre is represented by the ability of a code table to compress a data point. The shorter the encoded length, the better it is represented by the cluster, and hence the relative probability that it belongs that cluster is high. The aim here is not to minimise the encoded database length because the best encoding is a disjoint clustering.

Current clustering approaches do not scale well to high-dimensional datasets because most approaches use some kind of distance measure based on the attributes and require domain knowledge [CV05]. The performance goes down when applying them to high-dimensional datasets because of the curse of dimensionality. The distance between data points becomes larger when dealing with high-dimensional datasets. The approach which allows overlap between clusters is called fuzzy clustering and suffers from the same issue when using a distance measure based on the attributes. Approaches based on the notion of compression do not suffer from the curse of dimensionality because no feature information is used other than the raw values are used. Furthermore, no domain knowledge is required. The idea behind compression-based measures/metrics is that the similarity is determined by how well objects can be compressed given the information in other objects. The compression methods are based on the notion of the Kolmogorov complexity, which is not computable [GV03]; [LP08]; [FM07]. A more practical version of the Kolmogorov complexity is the notion of the Minimum Description Length (MDL) [Grü07]. The MDL principle has been successfully applied in data mining tasks [VVLS11], and in particular, for the purpose of clustering [LVS09]; [Böh+06].

The current approaches for clustering by means of compression based on ensemble learning, are based on boosting [FF12] techniques. With boosting, data is sampled without replacement from a dataset, whereas in bagging data is sampled with replacement [Pol12]; [CCS12]. An exception to this is AdaBoost, which reweighs the same data iteratively based off the performance of the previous base classifier, making it somewhat similar to bagging [FS95]. The difference between Random Forests and AdaBoost is that with Random Forests the attributes are sampled while AdaBoost uses the entire data set [Bre01]. To allow overlap, an approach similar to AdaBoost and Random Forests must be taken because AdaBoost uses the misclassified samples and Random Forests use a bootstrap sample. Siebes and Kersten's approach finds the best b code tables that describe the database best of the same complexity [SK11]. They observe that adding an item set to a code table makes it more restrictive - and suggests a natural partitioning on the dataset when using a series of code tables. However, they have not made any efforts to investigate this further with the use of the `Groei`-algorithm. The approach then becomes to iteratively grow code tables, like the `Groei`-algorithm, but letting go of the restriction that all code tables need to be of the same complexity. A consequence of this is that the best b code tables need be tracked with varying complexities, rather than the b code tables with the same complexity.

3.2 Research Questions

The goal of the project is to allow overlap in the partitions by using ensembles of code tables to get a better description of the data. This suggests that clustering has to be performed from the beginning, not afterwards, we have to keep track of many potential clusters from the beginning. The aim is to get a better description for the data, while also having dissimilarities within the ensemble of code tables to show the ability to identify specific database components. Another aim is to keep the number of components low; this allows for an expert to interpret the found relations by hand. It also good to have low run time, this will make the method suitable for interactive usage by data analysts/data scientists.

Research Question 1. *Are the obtained clusters able to identify a multi-valued relationship, if present?*

Research Question 2. *Do the obtained clusters capture the characteristics of the underlying data distribution?*

Research Question 3. *Are the clusters dissimilar enough to each describe a specific characteristic of the database?*

Research Question 4. *Is the runtime low enough for interactive usage?*

3.3 Relevance for Science, Technology and Society

By allowing overlap in the partitions, the margins between the partitions become soft. For every entry in the database there is a probability for every cluster to which it belongs. This can be seen as a degree of membership for every cluster. The main contribution to science is that this method allows for a better approximation of the underlying empirical distribution of the databases. Another contribution to science is that this form of soft-clustering exposes the relations between clusters if they are present. In high-dimensional domains, where the partitioning of attributes is not as clear, the code tables are able to give insight in the characterisation of clusters and relations between clusters. This method further motivates the need for non-attribute-based measures and metrics to allow for generic and scalable algorithms. High-dimensional domains are hard to visualize and interpret for humans. The code tables each give a good characterisation of the components they model in terms of item sets.

The technology that can be built upon the contributions of this thesis project will allow to give a probabilistic interpretation of the patterns in the dataset with respect to the found clusters. This means that current data mining tools will be able to give better insight into patterns present in datasets.

Having overlap in the partitions of high-dimensional data will introduce a degree of membership to clusters instead of hard membership. The generic approach can be applied to a large variety of domains which are characterized by having datasets with many attributes. Some example domains with high-dimensional data are: medicine, consumer behaviour research, DNA research and multimedia analysis.

4 Methodology

In this chapter the methodology for experimentation is discussed. First the data that will be used for the experiments will be discussed. Second, the methodology for evaluating the experiments is argued.

4.1 Data

For the experimental validation of the developed model, a variety of data sets will be used. For the sake reproducibility, the data sets used are freely available. Furthermore, the data sets are used in most data mining research for evaluation, this makes it easier to compare to other work.

The LUCS-KDD data repository contains a variety of data sets [Coe04], most of these come from the UCI machine learning repository [Lic13] but are normalised and discretised. Data sets from the UCI machine learning repository have widely used in machine learning research. The Frequent Itemset Mining Dataset Repository (FIMI) contains classic benchmark data sets [Fim]. The data sets in the FIMI repository are obtained from click-stream data, retail market basket data and traffic accidents data. The mammals data set contains the presence/absence of European mammals within geographical areas of 50×50 kilometres [Mam]. This data set is useful for visualising the obtained clusters.

4.2 Methodology

The main goal in machine learning is to describe the underlying data distribution using some model. In this case, the model is the set of code tables as clusters in combination with the cover algorithm. The code tables allow for compression of the data set and show the ability to give a good description of the underlying distribution while using a relatively small amount of itemsets. The number of code tables are the number of clusters found. Each of the clusters should describe a unique characteristic of the underlying distribution. This is evaluated using a purity measure and a dissimilarity measure. The purity describes how well a characteristic is captured by a cluster and the dissimilarity measures the separation between clusters.

In order to evaluate whether the obtained model truly captures the underlying distribution, it must be able to correctly classify new cases from the same domain to the found clusters. This allows for the measurement of the accuracy.

The model should be robust against changes in: data set size, the number of itemsets present and the density of the data set. Robust in the sense that the runtime and the performance measures mentioned above should not be affected by too much. The runtime should not vary more than the order of change in the size of the data set and dimensionality. This ensures scalability and stability of the model and allows the model to be used in practical applications.

4.2.1 Clustering Performance

Code tables contain the itemsets with their respective code lengths. Since there will be multiple code tables, each of the itemsets is compressed by the table which compresses it the best, i.e. giving it the shortest code length. This result can be compared to the results of previous works [VVLS11]; [SK11]; [LVS09]. Siebes et al. use the relative total compressed size of the data set $\mathcal{D}$ [VVLS11]:

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

For some data sets the optimal number of clusters is known. The objective to get as close as possible to this optimal value. A similar approach to Lonardi et al. can be taken to parameter-free data mining [KLR04]. This is done by choosing a compression-based dissimilarity measure.

A survey conducted on the methods for evaluation of disjoint clustering techniques show that purity and entropy are most commonly used in the fields of fuzzy- and kernel-based clustering for validation [Ami+09]. In clustering the objects within a cluster must be closely related, this is measured by the entropy. On the other hand, clusters must be as distinct as possible, and this is expressed by the purity.

Purity is a commonly used clustering measure, it indicates how well a characteristic of the underlying data distribution is captured by the clusters. This is the weighted sum of the purity of the individual clusters. The baseline purity is the ratio of cases belonging to the majority class. Let $\mathscr{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$ the set of clusters, and let denote $\mathscr{C} = \{c_1, \ldots, c_m\}$ 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:

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

Since each of the code tables represents a cluster, the obtained tables also allow for calculation of the dissimilarity between clusters using the code table dissimilarity measure devised by Leeuwen et. al. [VVLS07] because it is based on compression. A relatively high dissimilarity indicates a better separation of the clusters. The most interesting dissimilarity will be between the clusters where there is overlap, this dissimilarity will be lower than non-overlapping clusters.

The partition entropy E, defined by Chuang et. al, measures the homogeneity of the clusters [Chu+06]. The lower the entropy is, the more homogeneous the objects are in each of the clusters.

$$E = \frac{-\sum_j^N \sum_i^C u_{ij} \log u_{ij}}{N} = \frac{-\sum_j^N \sum_i^C \mathbb{P}(d_j \in D_i) \log \mathbb{P}(d_j \in D_i)}{N}$$

The entropy E_c of a single cluster c is then defined as:

$$E_c = -\sum_j^N u_{cj} \log u_{cj} = -\sum_j^N \mathbb{P}(d_j \in D_c) \log \mathbb{P}(d_j \in D_c)$$

4.2.2 Classification Performance

The difference between supervised classification and clustering is that in supervised learning the clusters are given as the class labels. It is possible to evaluate the performance of classifying new cases. Since we are dealing with overlapping clusters, the probability that a case d belongs to cluster $\mathcal{D}_i$ is proportional to its encoded length $CT_i(d)$, that is:

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

The assigned class $C(d)$ is then:

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

The accuracy describes the systematic errors made by the found model, this gives information on the bias of the model. This performance measure can be computed using a confusion matrix (Table 4.1). Each cell in the matrix represents how many

actual class → predicted class ↓	$\mathcal{D}_1$	...	$\mathcal{D}_n$
$\mathcal{D}_1$			
...			
$\mathcal{D}_n$			

TABLE 4.1: A confusion matrix. The predicted classed are the rows and the actual classes are the column. Every cell represents the number of cases that were classified as D_i (row), while the true class is D_j. Note that the cells where $i = j$ are the number of cases which are classified correctly.

cases that are actually in class $\mathcal{D}_j$ are predicted as being in class $\mathcal{D}_i$. The diagonal in the confusion matrix contains the number of cases per class/cluster that have classified correctly. The sum over the a row i indicates how many cases have been identified as belonging to class $\mathcal{D}_i$. The sum over a column j indicates how many actually belong to class $\mathcal{D}_j$.

The true positives (TP) are cases that have been correctly identified as belonging to class D_j. The true negatives (TN) are those cases that have been correctly identified not belonging to class D_j. False negatives (FN) are those that actually belong to class D_j but have been predicted to belong to some other class. False positives (FP) are those classes that have been predicated as belonging to class $\mathcal{D}_i$ but actually belong to some other class. For class $\mathcal{D}_j$ and confusion matrix M we get the following definitions:

$$TP_{\mathcal{D}_j} = M_{jj} \qquad TN_{\mathcal{D}_j}(M) = \sum_{i=1}^{n} M_{ii} - M_{jj}$$

$$FN_{\mathcal{D}_j} = \sum_{i=1}^{n} M_{ij} - M_{jj} \qquad FP_{\mathcal{D}_j}(M) = \sum_{i=1}^{n} M_{ji} - M_{jj}$$

The performance measures can now be defined in terms of the data in confusion matrix M. The accuracy A is the fraction of cases identified correct for class D_j.

$$A_{\mathcal{D}_j} = \frac{TP_{\mathcal{D}_j} + TN_{\mathcal{D}_j}}{TP_{\mathcal{D}_j} + TN_{\mathcal{D}_j} + FP_{\mathcal{D}_j} + FN_{\mathcal{D}_j}}$$

4.2.3 Scaling and Stability

Changes in the data set size, the number of itemsets present and the density of the data set might affect the performance of the model. It is important that the model scales to larger datasets because this allows it to be used in practical applications.

An increase in the size of the dataset and the number of itemsets will lead to a greater runtime. The increase in runtime is dominated by the increase in the number of items because the set of candidate itemsets grows exponentially in the number of items $\mathcal{O}(\mathcal{F}) = \mathcal{O}(2^{|\mathcal{I}|})$. An itemset is called frequent if and only if it's support is above some threshold θ. The support of an itemset $\mathcal{I}$ in dataset D is the number of transactions that contain $\mathcal{I}$ [VVLS11]:

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

The set of all frequent itemsets is then defined as:

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

The density of a data set can also influence the performance of a model. A data set can be viewed as a matrix containing boolean values where every transaction is a row and and every item is represented as a column. A transaction t contains item i if and only if the cell corresponding to $(t, i) = 1$. The density $\rho \in [0, 100]$ of data set D is then the percentage of one's in the matrix:

$$\rho(\mathcal{D}, \mathcal{I}) = \frac{|\{(t,i) \mid t \in \mathcal{D} \wedge i \in \mathcal{I} \wedge (t,i) = 1\}|}{|\mathcal{D}| \times |\mathcal{I}|} \times 100\%$$

A value closer to 100 indicates a dense data set whereas a value closer to 0 indicates a sparse data set. A sparse data set is harder to compress because the itemsets in the code table will be relatively shorter and relatively more itemsets are needed to cover a single transaction. This observation is made from Table 3 and Table 4 in [VVLS11] with the Retail data set.

5 Algorithms

In this chapter the clustering algorithms are presented. The basis for these algorithms is the Groei algorithm presented by Siebes and Kersten [SK11]. The first algorithm is a variation of the Groei algorithm where the the restriction is lifted that all code tables need to be of the same complexity. The second algorithm differs in the candidates are generated, and is inspired by the Slim algorithm [SV12].

Unlike the Groei algorithm, which tries to insert candidate item sets into every possible place in a code, or Groei-F, which only inserts candidate item sets at the end of the code tables, these algorithms insert candidates into the code tables based on their lengths (Standard Cover Order). This similar to the Krimp and Slim algorithm.

5.1 GroeiNoS - Groei No Structure Function

The algorithm presented in this subsection is a variation of the original Groei algorithm, where the the restriction is lifted that all code tables need to be of the same complexity and is called GroeiNoS (Groei No Structure Function). The psuedo code is given in Algorithm 6.

Algorithm 6: The GroeiNoS algorithm.

```
1  Function GroeiNoS($\mathcal{D}, b$)
       Data: Dataset $\mathcal{D}$, Beam-width $b$
       Result: Code tables $\mathcal{CT} = \{CT_i, \dots, CT_k\}$
2      $k \leftarrow 1$;
3      $\mathcal{CT}_1^{cand} \leftarrow$ Generate($CT_\alpha^{\mathcal{D}}$);
4      $\mathcal{CT}_1^{best} \leftarrow \{CT \mid$ best $b$ tables from $\mathcal{CT}_1^{cand}\}$;
5      repeat
6          $k \leftarrow k + 1$;
7          $\mathcal{CT}_k^{cand} \leftarrow$ Generate($\mathcal{CT}_{k-1}^{best}$);
8          $\mathcal{CT}_k^{best} \leftarrow \{CT \mid$ best $b$ tables from $\mathcal{CT}_k^{cand} \cup \mathcal{CT}_{k-1}^{best}\}$;
9      until $L(\mathcal{D} \mid \mathcal{CT}_k^{best}) \geq L(\mathcal{D} \mid \mathcal{CT}_{k-1}^{best})$;
10     return $\mathcal{CT}_k^{best}$;
```

The GroeiNoS algorithm starts by generating candidate tables from CT_α and selecting the best b code tables. In the following iterations, new candidates are generated from the best b candidates from the previous iteration. The best code tables in iteration k are the best b candidates generated in that iteration $\mathcal{CT}_k^{cand}$ and the best candidates from the previous iteration $\mathcal{CT}_{k-1}^{best}$. The original Groei algorithm, given in Algorithm 7, is very similar to GroeiNoS, except for that it only selects the best candidates from the set of candidates generated in the current iteration $\mathcal{CT}_k^{cand}$.

The GroeiNoS and Groei algorithms are template algorithms where the candidate generation algorithm, Generate, can be implemented as desired. More post-processing steps can be added after selecting the best set of code tables at every

Algorithm 7: The `Groei` algorithm.

```
1  Function $Groei(\mathcal{D}, b)$
       Data: Dataset $\mathcal{D}$, Beam-width $b$
       Result: Code tables $\mathcal{CT} = \{CT_i, \ldots, CT_k\}$
2      $k \leftarrow 1$;
3      $\mathcal{CT}_1^{cand} \leftarrow$ Generate$(CT_\alpha^{\mathcal{D}})$;
4      $\mathcal{CT}_1^{best} \leftarrow \{CT \mid$ best $b$ tables from $\mathcal{CT}_1^{cand}\}$;
5      repeat
6          $k \leftarrow k + 1$;
7          $\mathcal{CT}_k^{cand} \leftarrow$ Generate$(\mathcal{CT}_{k-1}^{best})$;
8          $\mathcal{CT}_k^{best} \leftarrow \{CT \mid$ best $b$ tables from $\mathcal{CT}_k^{cand}\}$;
9      until $L(\mathcal{D} \mid \mathcal{CT}_k^{best}) \geq L(\mathcal{D} \mid \mathcal{CT}_{k-1}^{best})$;
10     return $\mathcal{CT}_k^{best}$;
```

iteration (after line 8). One example of a post-processing step is pruning the code tables to eliminate superfluous item sets in the tables.

Algorithm 8: The original candidate generation algorithm.

```
1  Function $Generate_{original}(\mathcal{CT})$
       Data: Code tables $\mathcal{CT} = \{CT_1, \ldots, CT_n\}$
       Result: Code tables $\mathcal{CT} = \{CT_1, \ldots, CT_k\}$
2      $\mathcal{CT}^{cand} = \{\}$;
3      for $I \in \mathcal{F}$ do
4          for $CT \in \mathcal{CT}$ do
5              if $I \notin CT$ then
6                  for $i \in [0, |CT \backslash CT_{ST}| >$ do
7                      $\mathcal{CT}^{cand} = \mathcal{CT}^{cand} \cup \{(CT \cup I)$ Insert at Position $i\}\}$;
8                  end
9              end
10         end
11     end
12     return $\mathcal{CT}^{cand}$;
```

In the implemention of the basic `Groei` algorithm, candidates are generated by inserting them in the place where they orignally belong in the code table, this is in Standard Cover Order [VVLS11]. The `Groei` algorithm, created by Siebes and Kersten, generates even more candidates by creating a candidate for inserting an item set at every possible position in the code table. The time complexity of `Generate`$_{original}$

is $\mathcal{O}(|\mathcal{F}| \times \sum_{i}^{|\mathcal{CT}|} (|CT_i| \times |\mathcal{D}| \times |\mathcal{I}|))$. This algorithm is given in 8.

Algorithm 9: The basic candidate generation algorithm.

```
1  Function $Generate_{basic}(\mathcal{CT})$
       Data: Code tables $\mathcal{CT} = \{CT_1, \dots, CT_n\}$
       Result: Code tables $\mathcal{CT} = \{CT_1, \dots, CT_k\}$
2      $\mathcal{CT}^{cand} = \{\}$;
3      for $I \in \mathcal{F}$ do
4          for $CT \in \mathcal{CT}$ do
5              if $I \notin CT$ then
6                  $\mathcal{CT}^{cand} = \mathcal{CT}^{cand} \cup \{(CT \cup I)$ Standard Cover Order $\}$;
7              end
8          end
9      end
10     return $\mathcal{CT}^{cand}$;
```

Algorithm 10: Candidate generation where the itemset is inserted at the end of the code table.

```
1  Function $Generate_{fast}(\mathcal{CT})$
       Data: Code tables $\mathcal{CT} = \{CT_1, \dots, CT_n\}$
       Result: Code tables $\mathcal{CT} = \{CT_1, \dots, CT_k\}$
2      $\mathcal{CT}^{cand} = \{\}$;
3      for $I \in \mathcal{F}$ do
4          for $CT \in \mathcal{CT}$ do
5              if $I \notin CT$ then
6                  $\mathcal{CT}^{cand} = \mathcal{CT}^{cand} \cup \{(CT \cup I)$ End of Non-Singleton Sets $\}$;
7              end
8          end
9      end
10     return $\mathcal{CT}^{cand}$;
```

5.2 GroeiSlim - Candidate Generation Inspired by Slim

The previous methods use a pre-mined set of (frequent) item sets $\mathcal{F}$ as the potential new item sets. This can become infeasible for the previous when datasets get larger and more dense. The Slim algorithm solves this problem by directly creating candidates from the data [SV12]. This way a set of pre-mined (frequent) item sets is not needed and the candidates are created and evaluated on-the-fly.

In some sense the Groei and Slim algorithms work similarly. Groei uses a beam-search and a set of pre-mined candidates to grow a set of code tables and Slim either extends or creates a new item set from the item sets available in the code table. The Slim algorithm can be altered and used as a candidate generation method for Groei. This creates the GroeiSlim algorithm. The generation method works by keeping all the possible code tables in memory and then letting Groei pick the best set of candidates. The generation algorithm $\texttt{Generate}_{slim}$ is given in Algorithm 11. The original Slim algorithm can be considered as GroeiSlim with beam-width $b = 1$. This only keeps track of the best code like Slim because only the best candidate is considereda at every iteration.

Algorithm 11: The smart (slim) candidate generation algorithm.

```
1 Function $Generate_{slim}(\mathcal{CT})$
      Data: Code tables $\mathcal{CT} = \{CT_1, \ldots, CT_n\}$
      Result: Code tables $\mathcal{CT} = \{CT_1, \ldots, CT_k\}$
2     $\mathcal{CT}^{cand} = \{\}$;
3     for $CT \in \mathcal{CT}$ do
4         for $F \in \{X \cup Y : X, Y \in CT\}$ do
5             $\mathcal{CT}^{cand} = \mathcal{CT}^{cand} \cup \{(CT \cup F)$ Standard Cover Order $\}$;
6         end
7     end
8     return $\mathcal{CT}^{cand}$;
```

The worst time complexity is $\mathcal{O}(|\mathcal{F}|^3 \times |\mathcal{D}| \times |\mathcal{I}| \times |\mathcal{CT}|)$ but according to the authors of `Slim` and by inspecting previous results, this is rather pessimistic because code tables are often magnitudes smaller than the size of the set of all possible item sets.

6 Experiments

In this chapter the results from performing the evaluation methods as described in Chapter 4 are applied on the algorithms presented in Chapter 5. First the setup and the datasets used for experimentation are presented. Afterwards results concerning compression, clustering and classification are presented. Finally, results are shown whether it is possible to find multi-valued dependencies.

6.1 Setup

The experiments have been performed on a MacBook Pro (13-inch, 2015) with a 2.9 GHz Intel Core i5 processor and 8GB of 1867MHz DDR3 RAM Memory. The cut-off time for compressing a single dataset is set to 24 hours and the maximum number of iterations is 250. The beam-width b is set to 10, unless mentioned otherwise.

6.2 Datasets

For the experimental validation of the developed models, a variety of data sets is used. All datasets are freely available. The datasets are commonly used for the evaluation of machine learning models.

The datasets used, come from a variety sources. The LUCS-KDD data repository contains a variety of data sets [Coe04], most of these come from the UCI machine learning repository [Lic13] but are normalised and discretised. The Frequent Itemset Mining Dataset Repository (FIMI) contains classic benchmark data sets [Fim]. The mammals dataset contains the presence/absence of European mammals within geographical areas of 50×50 kilometres [Mam].

The datasets with their respective statistics are shown in Table 6.1. This table shows general information on the number of rows, alphabet size, density, the minimum support, and the number of frequent item sets with respect to the minimum support θ.

$\mathcal{D}$	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{I}\rvert$	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
Lower support:					
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

TABLE 6.1: The statistics of all the data sets used in for the experiments. The first column $\mathcal{D}$ denotes the name of the data set, the second column $|\mathcal{D}|$ denotes the number of rows in the data set, the third column $|\mathcal{I}|$ denotes the size of the alphabet, the fourth column ρ is the density of the data set expressed in the 1-s%, the fifth column shows the minimum support used, and the sixth column $|\mathcal{F}|$ is the size of the item set collection based on the minimum support.

6.3 Compression

Compression measures to what extent a structure has been captured by the model. The lower the compression ratio, the more structure is captured. In this section the results with respect to the compression ratio are reported. First the results will be presented when using a fixed beam-width on all algorithms and compressing all datasets. Afterwards, the runtime and convergence is reported. Results with lower support settings are also presented because the overall runtime of `GroeiSlimNoS` is magnitudes smaller than the runtime of `Groei`. At last, the effect of varying the beam-width on the compression is presented.

6.3.1 Compression

The compression of the best code tables is compared for each algorithm. The baseline used is the `Groei-F` algorithm because it only inserts an item set at a single point in the code table, and so does this version of `Groei`. Each of the algorithms is compared with and without pruning. The compression is expressed as the total compressed size relative to the compression obtained by using the standard code table.

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

The results without pruning are shown in Table 6.2 and the results with pruning are shown in Table 6.3. The beam-width for all experiments shown in these tables is set to $b = 10$.

$\mathcal{D}$	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%

TABLE 6.2: Results of the running the various compression algorithms with beam-width $b = 10$. The remaining columns show the relative compression ratio for each of the algorithms. The bold cells are the best results for each row.

$\mathcal{D}$	θ	`Groei-P`	`GroeiNoS-P`	`GroeiSlim-P`	`GroeiSlimNoS-P`
anneal	100	44,5%	44,5%	**42,4%**	**42,4%**
breast	1	16,0%	16,0%	**15,6%**	**15,6%**
chess	2500	**65,0%**	**65,0%**	65,1%	65,1%
ionosphere	125	71,7%	71,7%	**70,9%**	**70,9%**
iris	1	45,5%	45,5%	45,5%	45,5%
led7	1	27,7%	27,7%	**27,3%**	**27,3%**
mammals	850	65,3%	65,3%	**63,7%**	**63,7%**
mushroom	2500	**65,7%**	**65,7%**	66,3%	66,3%
pageblocks	1	**4,9%**	**4,9%**	5,0%	**4,9%**
pima	1	32,2%	32,5%	**30,8%**	**30,8%**
wine	10	**72,0%**	**72,0%**	72,2%	72,2%
wine	20	74,8%	74,8%	74,2%	**74,1%**
wine	30	77,1%	77,1%	77,1%	77,1%

TABLE 6.3: Results of the running the various compression algorithms with pruning and beam-width $b = 10$. The two leftmost columns show the dataset $\mathcal{D}$ with their respective minimum supports θ. the remaining columns show the relative compression ratio for each of the algorithms. The bold cells are the best results for each row.

Both tables show that, in almost all cases, compression improves or performs on-par with `Groei-F`, except for the 'wine' dataset. However, this dataset is relatively small compared to the remainder of the datasets, as shown in Table 6.1. The `Groei(NoS)` and `GroeiSlim(NoS)` algorithms show similar performance across the board. Even though the differences are small, the `GroeiSlim(NoS)` variants have a slight edge for most datasets. As expected, the pruning variants further improve the performance of all algorithms.

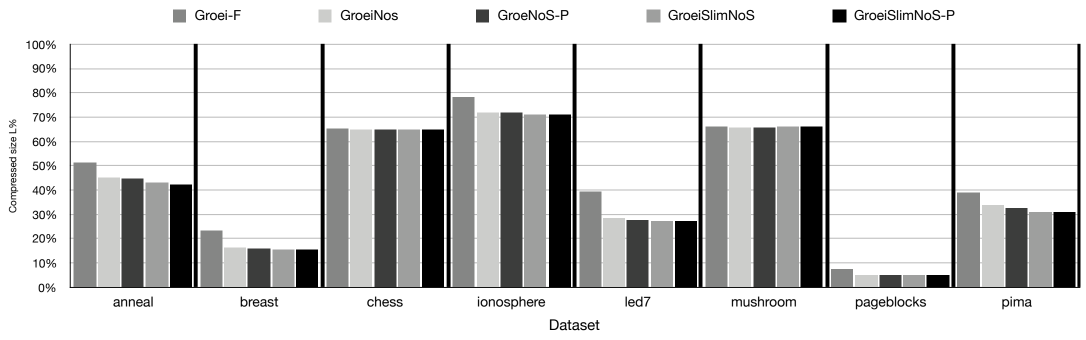


FIGURE 6.1: The compressed size of the datasets in $L\%$.

The results of Table 6.2 and Table 6.3 are summarised in Figure 6.1. This figure confirms the claims above, `Groei-F` is beaten in almost every case, and `Groei(NoS)` and `GroeiSlim(NoS)` show similar performance.

6.3.2 Run Time

There is no information on the run time of `Groei-F` because the implementation was not available and the runtime is also not reported. Figure 6.2 shows the runtime of the 'NoS' algorithms on various datasets. The time scale is shown on a logarithmic scale because of the large differences between the runtimes. Even though compression ratio's are similar, there is a large difference in the runtime between the regular `Groei` algorithm and the `GroeiSlim` variants.

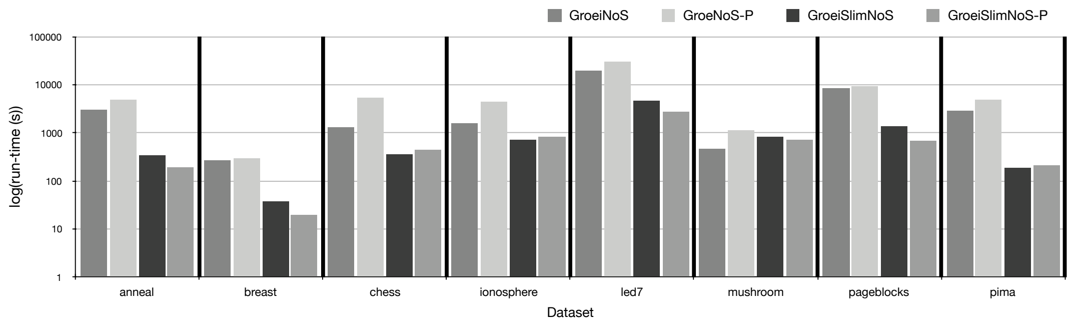


FIGURE 6.2: Run time in log(run-time) (run-time in seconds) of each algorithm ran on the datasets.

The large differences in the run times mainly come from the candidate generation procedures because the remainder of the template algorithm is the same. Even though the worst case scenario for the `GroeiSlim` algorithms is that the code tables contain all possible item sets, and all the non-singleton item sets have to extended, it is unlikely that this will be the case. This is confirmed by the run times shown in Figure 6.2.

For each iteration, the `Groei` algorithms have to go over the entire mined (frequent) item set collection for every candidate. At every iteration k, every one of the b best candidate tables from the previous iteration $k-1$ has to be extended with every item in the entire item set collection $|\mathcal{F}|$ and every newly generated candidate code table has to cover the dataset $|\mathcal{D}|$ to recompute the encoded size $L(\mathcal{D},\mathcal{CT})$. With the `GroeiSlim` algorithms, every item set I has to be combined with every other item set J in every code table, this results in $|CT|^2$ evaluations per code table. The code tables start with small item sets because only the alphabet is available in CT_α. At every iteration, either an existing item set is extended, or a new one is created using the singleton items. The amount of evaluations with the `GroeiSlim` algorithms will be larger than the number of evaluations with the `Groei` algorithms only if $|CT| > \sqrt{|\mathcal{F}|}$.

6.3.3 Convergence

Figure 6.3 shows the amount of iterations before convergence is reached for each dataset. The `GroeiSlim` algorithms need more iterations before reaching convergence than the `Groei` algorithms. Even though more iterations are needed, the run time is much shorter for the `GroeiSlim` algorithms. This is confirmed by Figure 6.4, which shows the compressed size over the iterations.

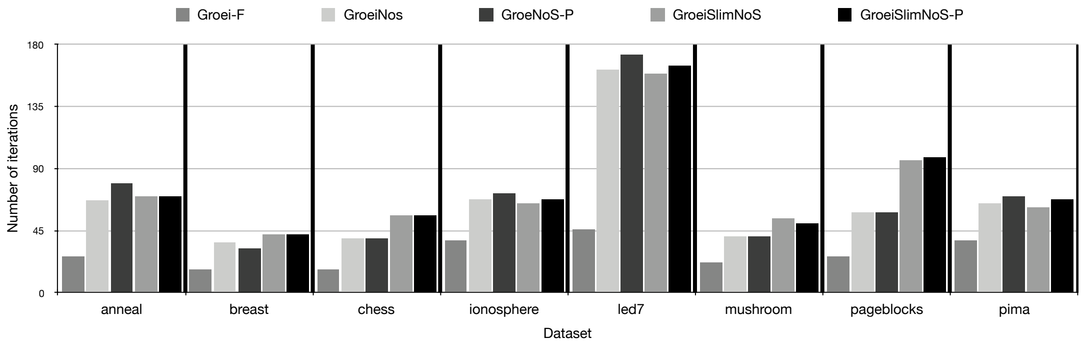


FIGURE 6.3: The amount of iterations needed before convergence is reached on various datasets.

The convergence of the algorithms run on the led7 with settings from Table 6.2, is shown in Figure 6.4. The convergence showed the same relative curves for all tested datasets. The dashed curves show the convergence of the compressed database size $L(\mathcal{D} \mid CT_{best})$ of the `GroeiSlimNoS` algorithm and the solid lines show the convergence of the `GroeiNoS` algorithm. The dotted line is the reported compression by Siebes and Kersten of the `Groei-F` algorithm [SK11].

The reason that the `GroeiSlim` requires more iterations before convergence is reached, is that in the earlier iteration `GroeiSlim` only creates small candidates because it needs to start by merging singleton item sets. On the other side, because the `Groei` algorithms require a pre-mined itemset collection, it is already possible in these cases to identify the larger item sets with high supports values.

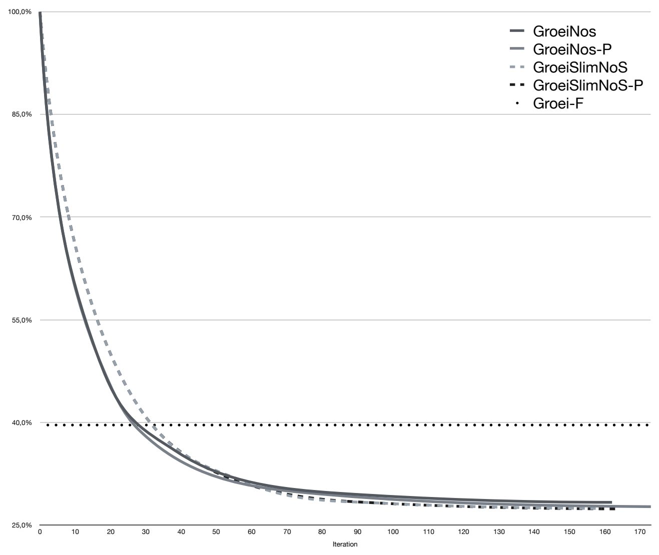


FIGURE 6.4: Convergence of the compression on the led7 dataset when run by the various algorithms. The parameters have the same values as shown in the tables above.

6.3.4 Lowering the support

Since `Slim` is able to compress datasets with very low support in a reasonable amount of time, `GroeiSlim` should also be able to the same. Table 6.4 shows the compression when support is lowered for some of the datasets. The `Groei` algorithms did not complete the compression task because even a single iteration took a very large amount of time. The reason for this is that the number of frequent item sets with low support is extremely large (mushroom $\theta = 1$ has 1.56×10^{10} frequent item sets). This is why the `Groei` algorithm is only tested with higher support settings.

		`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%

TABLE 6.4: `GroeiSlimNoS` compression results on a selection of datasets with lower support settings.
(*) Terminated because of iteration limit.
(**) Terminated because of time limit.

Table 6.4 shows that the compression ratio improves when the support is lowered. However, as is noted, for some of the datasets the set time limit or maximum number of iterations was reached. Compression could have been better if these limits were not imposed.

6.3.5 Beam-Width

Previous experiments have been conducted with a fixed beam-width. Figure 6.5 shows the effect on the compression ratio when the beam-width is altered. For all datasets the beam-width did not influence the relative compression ratio. The changes in compressed size are minimal.

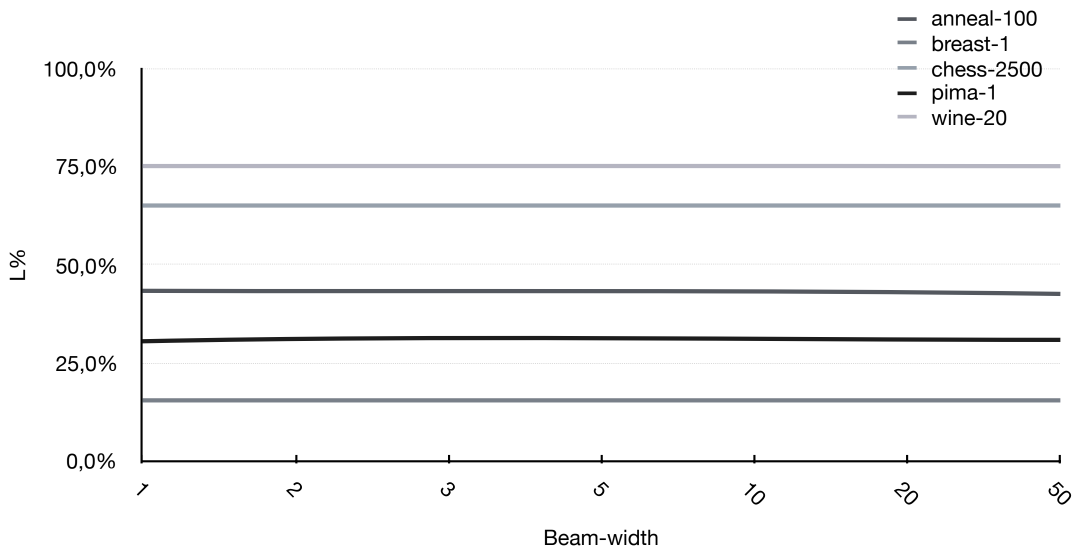


FIGURE 6.5: Running `GroeiSlimNoS` with varying beam-widths on different datasets. The vertical axis shows the relative compressed size $L\%$ and the horizontal axis shows the beam-width.

6.4 Clustering

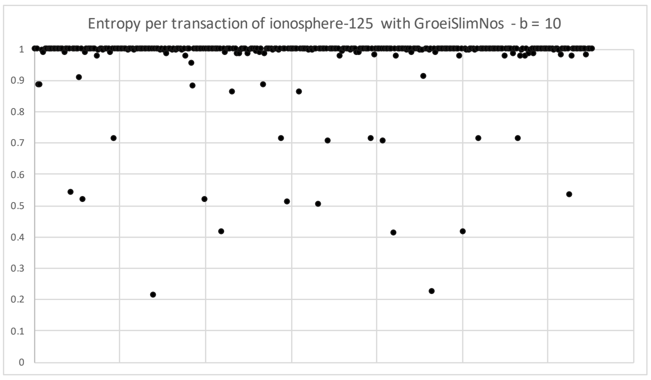
Clustering performance cannot be directly measured. One can compute the entropy of all the transactions over the identified clusters to measure the homogeneity of the items. Since a single code table models a single cluster, the dissimilarity between the clusters can be computed and used to measure the relative difference between clusters. The differences between the code tables can be visualised by computing the probability distribution over all code tables for every transaction. This shows whether each code table captures different characteristics of the data set.

6.4.1 Entropy

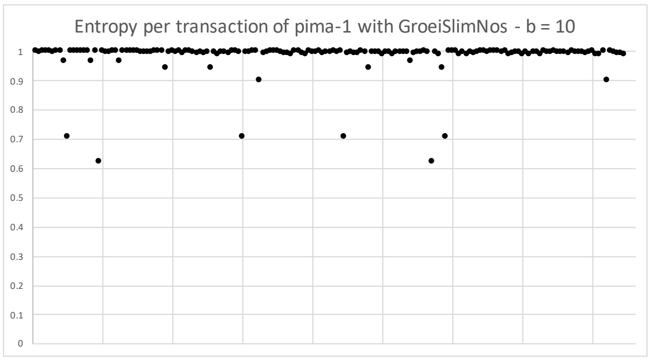
The entropy of a transaction over all clusters measures the uniformity of the distribution over all code tables. The more uniform the distribution, the closer the entropy will be to the value of 1 and vice versa. The entropy of a single transaction t is defined as:

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

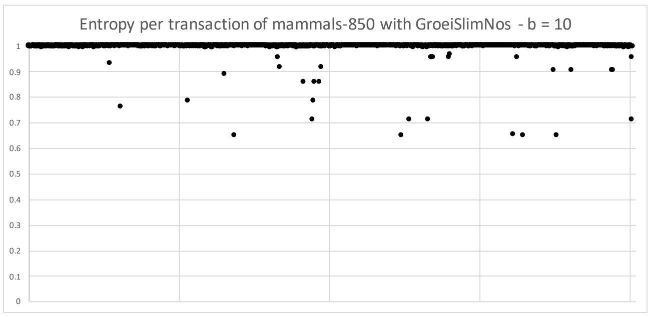
where b is the beam-width.



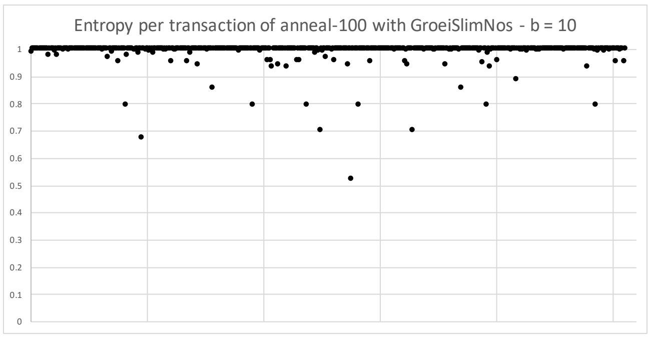
(A) ionosphere-125



(B) pima-1



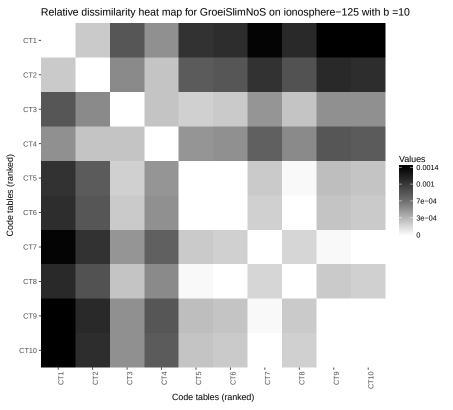
(C) mammals-850



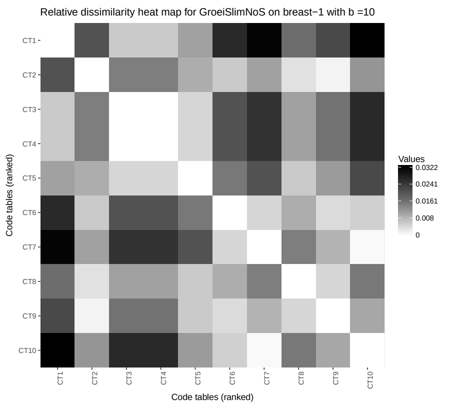
(D) anneal-100

FIGURE 6.6: The entropy for various datasets over all unique transactions using `GroeiSlimNos` with $b = 10$. The higher the entropy, the more uniform the distribution is over all code tables. The lower the entropy, the larger the difference between encoded size between the code tables of a transaction.

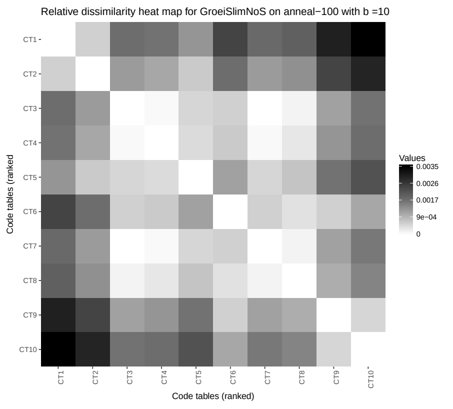
Figure 6.6 shows the entropy of the unique transactions of various datasets using `GroeiSlimNos` with $b = 10$. In all plots there is a large amount of transactions with



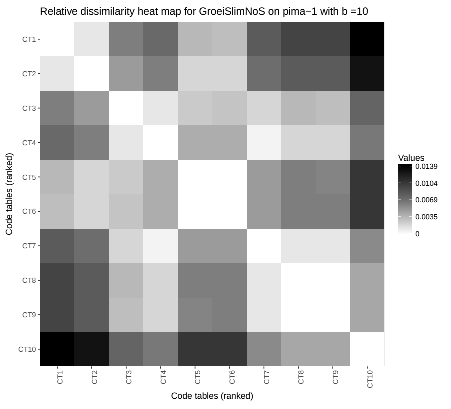
(A) ionosphere-125



(B) breast-1



(C) anneal-100



(D) pima-1

FIGURE 6.7: Relative dissimilarity heat map for the code tables outputted by `GroeiSlimNoS` on various datasets with $b = 10$. Lighter shades indicate relatively lower dissimilarities while darker shades indicate relative higher dissimilarities.

an entropy around the value of 1. All these transactions are equally characterised by the code tables. In all cases, there is smaller group of transactions that have lower entropy. These transactions are characterised by a smaller group of tables. The lower the entropy, the fewer code tables characterise the transaction. These experiments show that `GroeiSlimNoS` identifies common patterns in all code tables, and also identifies patterns that are specific to smaller amount of tables. In the plots it is also visible, that in (B)-(D) the entropy does not drop below 0.5. This means that the transactions in general are characterised by a larger amount of tables which compress the transaction similarly.

6.4.2 Dissimilarity

The dissimilarity between two code tables on the same dataset is the relative difference in the compressed database size $L(\mathcal{D} \mid CT)$, note that this is different from the overall compressed size $L(\mathcal{D}, CT)$. The first definition does not include the length of the code table self, and the latter definition does respectively. The obtained code tables $\mathcal{CT} = \{CT_1, \ldots, CT_b\}$ are always sorted by in ascending order on total compressed size $L(\mathcal{D}, CT)$. This does not necessarily imply that the compressed database size $L(\mathcal{D} \mid CT)$ will also be sorted in ascending order.

	ionosphere-125				pima-1			
CT rank	$\|CT\|$	$L(\mathcal{D} \mid CT)$	$L(CT)$	$L(\mathcal{D}, CT)$	$\|CT\|$	$L(\mathcal{D} \mid CT)$	$L(CT)$	$L(\mathcal{D}, CT)$
1	62	55724	4119	59844	52	6720	1607	8328
2	62	55739	4107	59846	52	6729	1599	8328
3	61	55770	4078	59848	51	6761	1570	8332
4	61	55755	4096	59851	51	6770	1562	8332
5	60	55784	4068	59851	52	6743	1592	8335
6	60	55785	4068	59853	52	6742	1592	8335
7	60	55798	4056	59854	51	6775	1563	8339
8	60	55787	4068	59855	51	6784	1554	8339
9	60	55801	4054	59855	51	6784	1555	8339
10	59	55800	4056	59856	50	6813	1528	8341

TABLE 6.5: Compressed size per code table for the ionosphere-125 and pima-1 datasets with `GroeiSlimNoS` $b = 10$. $|CT|$ shows the number of non-singleton item sets in the code table.

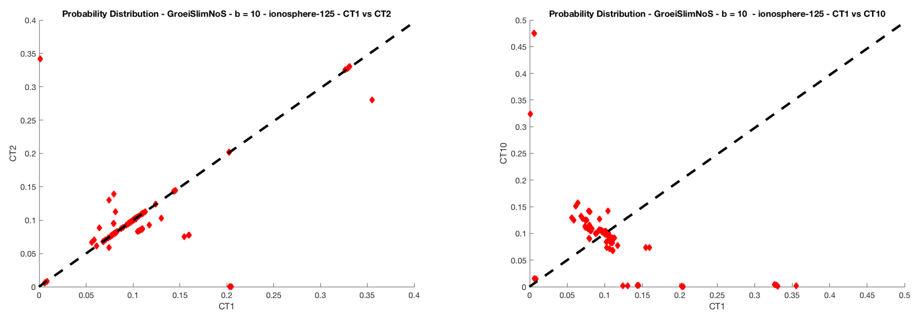
Figure 6.7 shows that the obtained code tables are of varying similarity for a selection of datasets, this implies that the code tables are not equivalent. The dissimilarity measure is applied in the following way: since the dataset has no single code table, and the dataset is the same for every pair of code tables, the normalisation step is done by dividing by the encoded database length of both code tables. For dataset $\mathcal{D}$ with code tables CT_x and CT_y, the code table dissimilarity measure DS between CT_x and CT_y is:

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

For (A) and (B) in Figure 6.7, the higher ranked tables are more similar to each other, and the same holds for the lower ranked tables. However, this is not the case for (C) and (D). Here the code tables get more dissimilar as the difference in rank increases.

Table 6.5 shows the absolute compression values for the code tables. It shows that the algorithm is able to find different code tables for the same dataset that compress the data on a similar level. The code tables obtained for ionosphere-125 by using `GroeiSlimNos` with $b = 10$ are inspected manually to identify the commonalities and differences between the tables. For CT_1 and CT_2, the first 11 item sets are the same. The 12th item set in CT_1 is not present in CT_2. The absence of this itemset in CT_2 influences the usage count of the 18th item set in CT_2 because it is a subset of the 12th item set. Another difference between the two is that CT_2 contains an additional smaller itemset. When looking at the difference between CT_1 and CT_{10}, the first 11 item set are the same, similar to the previous case. In this case though, the code tables differ in 4 item sets. This difference influences the usage counts of 13 item sets in CT_{10}. The general pattern that shows is that the first 11 item sets are the same for all code tables, and then the differences start occurring. The different characteristics of the dataset are captured by the smaller item sets in the tables.

The difference between code tables CT_1 and CT_5 of the pima-1 dataset is that CT_1 contains two more item sets at positions 22 and 45. Like with the ionosphere-125 dataset, the common patterns are captured in the larger item sets, which reside at the top of the tables. The difference between CT_1 and CT_{10} is that CT_1 contains 3 more item sets, these sets reside at the top of the table, this is why difference in compressed size is larger. This is also visible in Figure 6.7. To check whether each of the code tables capture different characteristics, the probability distribution of each



(A) $CT1$ versus $CT2$. (B) $CT1$ versus $CT10$.

FIGURE 6.8: Probability distribution of Code tables in the form of a scatter plot on the ionosphere-125 dataset using the `GroeiSlimNoS` algorithm with $b = 10$.

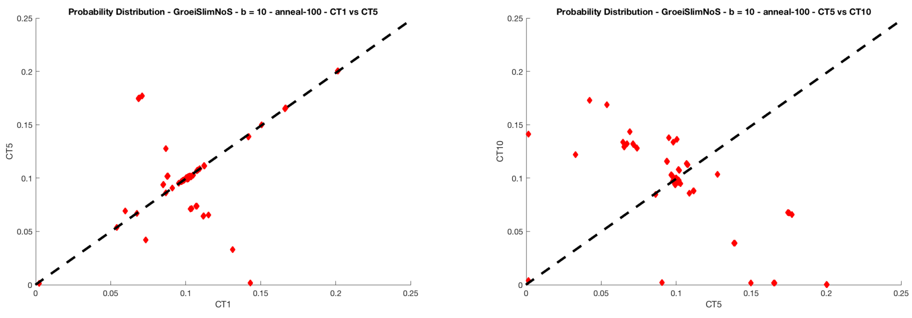
transaction over the obtained set of tables must be inspected.

6.4.3 Distribution

To show whether the set of code tables truly captures different characteristics of the datasets, the probability distribution of each row in the dataset over the set of code tables is inspected. This information is shown as a set of scatter plots and shows the pairwise probability of all data points of a dataset. When two code tables equally characterize a data point (have the same compression), the point will lie on the diagonal line. The larger the deviation from this line, the better the one code table characterizes the data point than the other. This all is shown in Figures 6.8 and 6.9.

Figure 6.8 shows the probability distribution between two code tables in each figure for the ionosphere-125 dataset using the `GroeiSlimNoS` algorithm with $b = 10$. In (A) $CT1$ is compared to $CT2$. Figure 6.7 shows that these two code tables are relatively similar. Even though the code tables show database rows that lie on the diagonal line, there are also transactions that are compressed very poorly by one, and better by the other. The code tables also show similarity in the sense that both also compress some database rows such that the probability $\mathbb{P}(d \in CT_i) > \frac{1}{b}$. In (B) $CT1$ is compared to $CT10$. Figure 6.7 shows that these two code tables are relatively dissimilar. Even though $CT10$ is the 'worst' performing code table, it is still able to compress a portion of the transaction better than $CT1$. Compared to (A), more transactions are scattered around the $\mathbb{P} = \frac{1}{b}$ area, and less transactions lie on the diagonal line. The amount of extremes also is larger, more transactions are compressed relatively very well by one and bad by the other table.

Figure 6.9 shows the probability distribution between two code tables in each figure for the anneal-100 dataset using the `GroeiSlimNoS` algorithm with $b = 10$. In (A) $CT1$ is compared to $CT5$. Figure 6.7 (C) shows that these two code tables are neither very similar or dissimilar. This figure shows a combination of results shown in Figure 6.8. There are some transactions that have equal compressed length, and some transactions are compressed well by one and bad by the other. This is less extreme than when the code tables are very dissimilar. In (B) $CT5$ is compared to $CT10$. Figure 6.7 (C) shows that these two code tables are moderately dissimilar. Compared to (A), the transactions are more grouped around the $\frac{1}{b}$ area. The most notable difference between the two is that in (B) the 'scatteredness' of the transactions that do not lie close to the diagonal is larger.



(A) $CT1$ versus $CT5$. (B) $CT5$ versus $CT10$.

FIGURE 6.9: Probability distribution of Code tables in the form of a scatter plot on the anneal-100 dataset using the `GroeiSlimNoS` algorithm with $b = 10$.

Having compared various degrees of relative similarity and varying entropy levels between code tables, in all situations it shows that there is a group of transactions that are captured equally well by both code tables. In all situations, there are also transactions that are captured well by one code table table and less well by the other. As the tables become more dissimilar, this phenomenon occurs more often.

6.5 Classification

The goal here is not to build the best classifier but rather to capture the underlying data distribution. A reasonable classification performance will indicate that the underlying data distribution is sufficiently captured by the obtained code tables. Another method of confirming this, is by first clustering by setting the beam-length equal to the amount of classes present in the dataset and afterwards classifying the cases by each of the clusters to obtain the purity. This will show how well the different characteristics are captured.

6.5.1 Classification Performance

The classification experiments have been performed using classed datasets with varying amount of classes and 10-fold cross validation. For each of the experiments, the classification accuracy is measured. The number of classes per dataset are shown in Table 6.6 and the results are presented in Figure 6.10.

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

TABLE 6.6: Number of classes per dataset.

Figure 6.10 shows that the accuracy is well above the established baseline for all cases. Lower levels of support show that the accuracy increases in most cases, and thus the underlying distribution is better captured. The claim is not that a set of code tables obtained by `GroeiSlimNoS` deliver results that are on-par with state-of-the-art

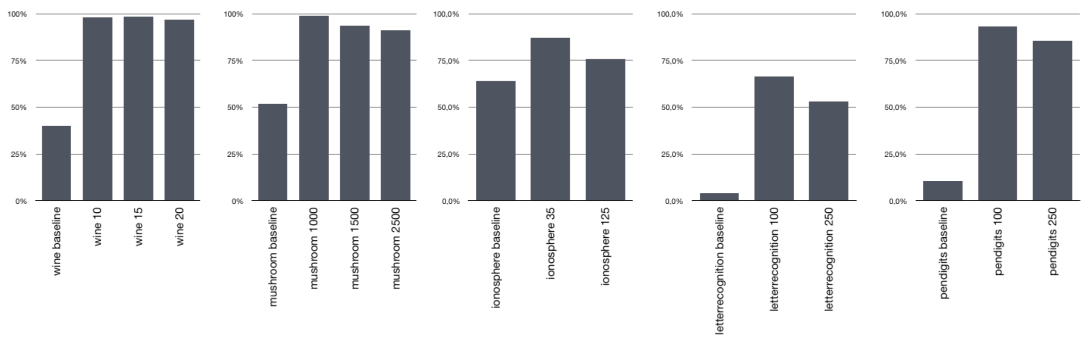


FIGURE 6.10: Classification performance (Accuracy) with respect to each of the baselines of the classed datasets with `GroeiSlimNoS` and $b = 10$. The baseline for the datasets is the accuracy when assigning to the majority class.

classifiers, but rather that the results show that the algorithm captures the underlying data distribution of each of the classes. It is also interesting to measure whether the algorithm is able to capture the the different classes without prior knowledge of the class labels. This ability expressed as the purity.

6.5.2 Purity

Since class labels show the true data distribution, one can measure how well a clustering algorithm is able to find this distribution by using purity as the performance measure. In this case, the beam-width is set equal to the amount of classes for the classed databases ($b = |cl|$). Since the minimum support θ controls the amount granularity that is to be captured by the code tables, this is also varied. In the experiments, the code table which best compresses a transaction, is the code table that assigns the transaction to a class.

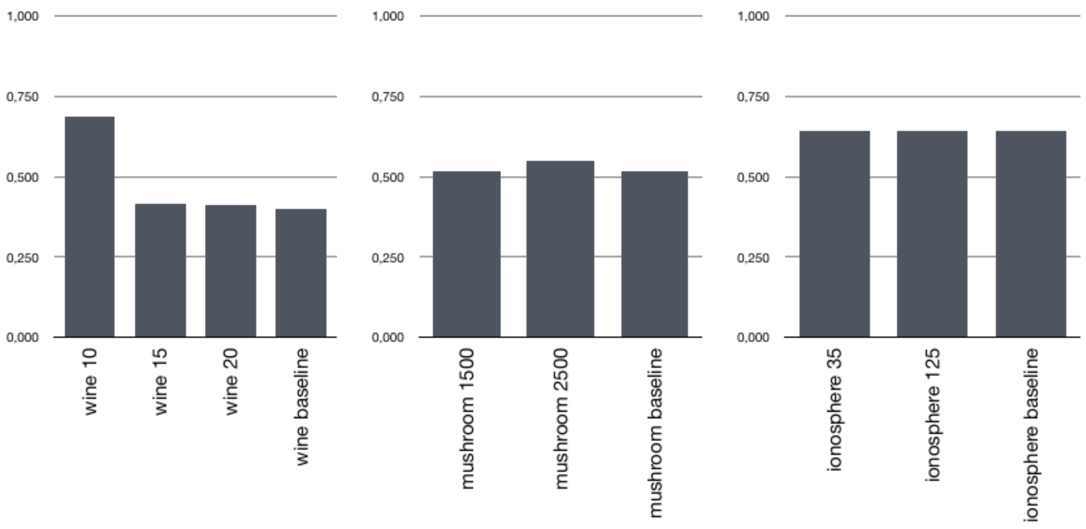


FIGURE 6.11: Purity of different datasets with varying support levels using `GroeiSlimNoS` and $b = 10$. The baseline is the ratio of classed belonging to the majority class.

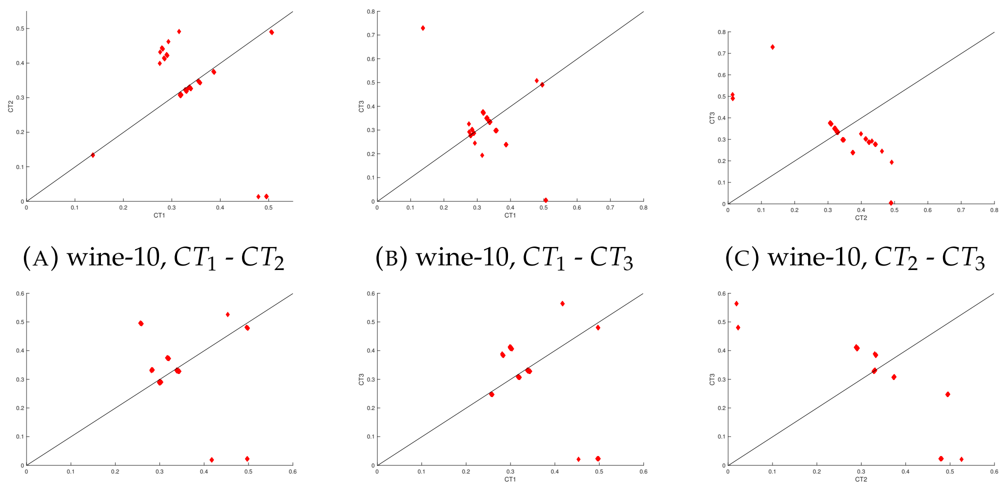
Figure 6.11 shows that in all cases the purity is at least at the level of the baseline. This indicates that the general structure of the data is captured. When the support is low enough, like with the wine-10 dataset, the purity increases. With the classification experiments, Figure 6.10 shows that the accuracy decreases slightly when

clusters ↓, classes →	cl_1	cl_2	cl_3	clusters ↓, classes →	cl_1	cl_2	cl_3
CT_1	**468**	216	63	CT_1	378	**405**	306
CT_2	72	27	**342**	CT_2	**117**	36	45
CT_3	99	**288**	27	CT_3	**144**	90	81

TABLE 6.7: Tables showing $|\mathcal{D}_i \cap C_j|$ in each cell, in terms of code tables and class labels using `GroeiSlimNoS`. For each cluster i (represented as a code table), the class for which $|\mathcal{D}_i \cap C_j|$ is maximal is shown in **bold**. Left: wine-10, Right: wine-15

lowering the support from 15 to 10 for the wine dataset. In this case though, the purity increases. The difference between the classification and purity experiments is that in the classification experiments the structure of the individual classes is captured separately, while in the purity experiments the structure of the entire dataset is captured with the number of clusters being equal to the amount of classes of the respective datasets. This is why the purity and classification accuracy are not comparable.

Table 6.7 shows the amount of transactions that overlap between the clusters and classes for the wine-10 and wine-15 datasets. With the wine-15 dataset (right), CT_1 compresses most transaction the best and therefore is allowed to assign a class label to most transactions. With the wine-10 dataset, CT_1 assigns 46.6% of the transactions to a class, and with the wine-15 dataset this is 68.0% of the transactions. A consequence of this is that the separation between the classes is better visible for the wine-10 dataset. This is confirmed by Figure 6.12. This figure shows the difference in the obtained code tables between wine-10 and wine-15. The most notable difference between the two rows is that in the upper row (wine-10) the transactions appear more scattered, while in the bottom row (wine-15) the transaction are more cluttered. Because of the lower support level, more differences between transactions are captured in the code tables. When comparing CT_1 and CT_2 ((A) and (D)), the transactions that are better compressed with CT_2 are further away from the diagonal, thus better characterised by the code table for the wine-10 dataset. This is also clearly visible in Table 6.7 when comparing the first two rows of both tables. CT_2 characterises an entirely different class than CT_1 does. The same shows for code tables CT_2 and CT_3 ((C) and (F)). However, code tables CT_2 and CT_3 cannot clearly separate the data between classes cl_1 and cl_2 for the wine-10 dataset. This is also visible in Figure 6.12 (B), most transactions are close to the diagonal, meaning that there is not much of a difference between the encoded length of transactions. This explains why still many transactions are classified to cl_2 by CT_1 for wine-10.



(A) wine-10, CT_1 - CT_2 (B) wine-10, CT_1 - CT_3 (C) wine-10, CT_2 - CT_3

(D) wine-15, CT_1 - CT_2 (E) wine-15, CT_1 - CT_3 (F) wine-15, CT_2 - CT_3

FIGURE 6.12: Probability distribution of Code tables in the form of a scatter plot on the wine-10 and wine-15 datasets using the `GroeiSlimNoS` algorithm. The top row shows the code tables obtained from the wine-10 dataset and bottom row shows the code tables obtained from the wine-15 dataset.

6.6 Identifying Multi-Valued Dependencies in Data

It is unknown whether the previously used datasets contain multi-valued dependencies. For the next set of experiments, three custom-made datasets will be used. One contains multi-valued dependencies, one also contain multi-valued dependencies but the data will be more noisy and one dataset will contain no such dependencies. The experiments are performed using the `GroeiSlim` algorithm because it only selects the best code tables that are generated in the same iteration and improve the compression of the previous best.

The Multi-Valued Dependency (MVD) dataset consists out of thee columns, β, γ and ψ. There is an obvious multi-valued dependency between β and γ and this is shown in Table 6.8.

(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

TABLE 6.8: The number of occurrences per triple in the MVD dataset. An asterisk (*) indicates that this can be any value. The value of ψ is unique for every entry of (β_i, γ_i).

Figure 6.13 shows the selected candidate tables with their respective non-singleton item sets for every table from every iteration when running `GroeiSlim` on the MVD dataset. Only item sets from the columns β and γ are selected in all iterations. Although not all pairs shown in Table 6.8 are present, the multi-valued dependency is clearly visible in the code tables.

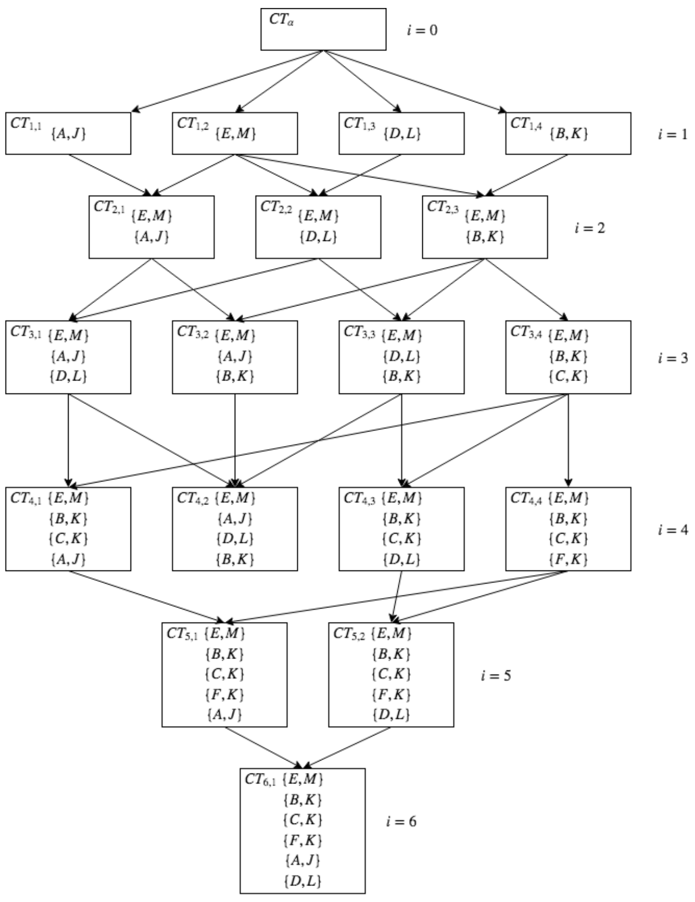

CT_α
$i = 0$
$CT_{1,1}$ $\{A,J\}$
$CT_{1,2}$ $\{E,M\}$
$CT_{1,3}$ $\{D,L\}$
$CT_{1,4}$ $\{B,K\}$
$i = 1$
$CT_{2,1}$ $\{E,M\}$ $\{A,J\}$
$CT_{2,2}$ $\{E,M\}$ $\{D,L\}$
$CT_{2,3}$ $\{E,M\}$ $\{B,K\}$
$i = 2$
$CT_{3,1}$ $\{E,M\}$ $\{A,J\}$ $\{D,L\}$
$CT_{3,2}$ $\{E,M\}$ $\{A,J\}$ $\{B,K\}$
$CT_{3,3}$ $\{E,M\}$ $\{D,L\}$ $\{B,K\}$
$CT_{3,4}$ $\{E,M\}$ $\{B,K\}$ $\{C,K\}$
$i = 3$
$CT_{4,1}$ $\{E,M\}$ $\{B,K\}$ $\{C,K\}$ $\{A,J\}$
$CT_{4,2}$ $\{E,M\}$ $\{A,J\}$ $\{D,L\}$ $\{B,K\}$
$CT_{4,3}$ $\{E,M\}$ $\{B,K\}$ $\{C,K\}$ $\{D,L\}$
$CT_{4,4}$ $\{E,M\}$ $\{B,K\}$ $\{C,K\}$ $\{F,K\}$
$i = 4$
$CT_{5,1}$ $\{E,M\}$ $\{B,K\}$ $\{C,K\}$ $\{F,K\}$ $\{A,J\}$
$CT_{5,2}$ $\{E,M\}$ $\{B,K\}$ $\{C,K\}$ $\{F,K\}$ $\{D,L\}$
$i = 5$
$CT_{6,1}$ $\{E,M\}$ $\{B,K\}$ $\{C,K\}$ $\{F,K\}$ $\{A,J\}$ $\{D,L\}$
$i = 6$

FIGURE 6.13: The selected candidate tables lattice with the non-singleton item sets for every table from every iteration when running `GroeiSlim` on the MVD dataset. Every edge indicates that the table in iteration i can be generated from the parent in iteration $i-1$.

Table 6.8 shows the anti-chains present in the lattice of code tables in Figure 6.13. It shows that all code tables that share a common parent table are anti-chains. This is not surprising because the child is generated when either a new item set is created from the alphabet and added or an existing itemset is extended in the parent. Let $\mathcal{CT}_i$ denote all candidates from iteration i, then all sets of anti-chains $\mathbb{S}_i$ for a single iteration i are then be defined as:

$$\mathbb{S} = \{S \mid \mathcal{CT}'_i \subseteq \mathcal{CT}_i \wedge S = \bigcap_{CT_{i,x} \in \mathcal{CT}'_i} CT_{x,i} \wedge |S| = i-1\}$$

All anti-chains share the exact same non-singleton item sets except for one. `GroeiSlim` is successfully able to create anti-chains and identify the multi-valued dependency in this case.

Iteration i	Anti-chains
0	$\{\}$
1	$\{CT_{1,1}, CT_{1,2}, CT_{1,3}, CT_{1,3}\}$
2	$\{CT_{2,1}, CT_{2,2}, CT_{2,3}\}$
3	$\{CT_{3,1}, CT_{3,2}\}, \{CT_{3,1}, CT_{3,3}\}, \{CT_{3,2}, CT_{3,3}, CT_{3,4}\}$
4	$\{CT_{4,1}, CT_{4,2}\}, \{CT_{4,1}, CT_{4,3}, CT_{4,4}\}, \{CT_{4,2}, CT_{4,3}\}$
5	$\{CT_{5,1}, CT_{5,2}\}$
6	$\{\}$

TABLE 6.9: The sets of anti-chains present in every iteration as shown Figure 6.13. All pairs of code table in a single set are anti-chains.

When random noise is added to the MVD dataset, the Noisy Multi-Valued Dependency (NMVD) dataset is obtained. It contains exactly the same entries as the MVD dataset but random entries are added. Table 6.10 shows that only the item sets listed in Table 6.8 are in the code table and that random noise is not captured in the table. `GroeiSlim` is still able to detect de multi-valued dependency in the presence of noise.

Item sets
$\{E, M\}$
$\{A, J\}$
$\{B, K\}$
$\{C, K\}$
$\{F, K\}$
$\{G, J\}$
$\{I, J\}$
$\{D, L\}$
$\{H, L\}$

TABLE 6.10: The item set in the final code table obtained by running `GroeiSlimNoS` on the NMVD dataset.

The random noise (RN) dataset only contains random entries. In this case no dependencies in the data should be found. By running `GroeiSlim` on this dataset, the CT_α code table is obtained. This table does not contain any non-singleton item sets. This is because random data does not contain any structure, and the algorithm tries to capture structure. These experiments do not only validate that the algorithm is resistant against noise, but also that it is able to capture interesting patterns in the data.

7 Discussion

The compression results of the experiments performed in Chapter 6 show that the algorithms improve on the `Groei-F` in most cases. The most notable difference is the improvement in runtime. Runtime improves by several orders of magnitude. This means that the `GroeiSlim(NoS)` is able to handle datasets with lower supports. It still is not able to handle supports as low as the `Slim` is tested with. These support levels allow for more interactive and practical usages of the algorithms. Varying the beam-width does not influence the ability to compress by much. The beam-width is however important for clustering purposes, this means the beam-width cannot be set to 1 if the purpose is to characterise the dataset from different perspectives.

The results from the clustering experiments shed light on the distribution uniformity per transaction, the differences between the obtain code tables, and the distribution of transactions over pairs of code tables in order to determine whether the obtained code tables truly capture the underlying distribution from different perspectives. The results of experiments regarding entropy show that the code tables are equally able to capture the general distribution of the datasets. It also shows that a portion of the transactions does not have a uniform distribution over the obtained code tables. This is because some code tables compress a transaction better than others, thus characterising the dataset from a different perspective. All the obtained code tables are unique. As the rank increases, the relative dissimilarity increases because the compression ratio decreases. The obtained code tables differ in the number of non-singleton item sets, the encoded database length $L(\mathcal{D} \mid CT)$ and the total encoded length $L(\mathcal{D}, CT)$. Manual inspection shows that larger item sets with high support are common between tables and the difference between the clusters is expressed by a few item sets. These item sets characterise the difference between clusters. The results that show the probability distribution over pairs of code tables confirm that that the probability distribution is truly different and that even a small difference between code tables can lead to non-uniform distributions. The results also show that there is a group of transactions that are compressed uniformly by the code tables and the other group is compressed with varying probabilities towards the various clusters. This is also visible in the entropy results. The transactions that have a uniform distribution over the code tables show the larger patterns in the data and the transactions with lower entropy and non-uniform distributions over the code tables show the the different characteristics of the obtained clusters with respect to the code tables.

When comparing non-disjoint clusters to disjoint clusters, the non-disjoint clusters should not be interpreted in the same way as disjoint clusters. While disjoint clustering techniques explicitly try to find strictly separated groups of patterns, they fail to show the commonalities in the data. These commonalities become visible in the patterns that are similar in all code tables when applying the `GroeiSlimNoS` algorithm. Next to identifying these commonalities, the algorithm is also able to find the different groups in the data.

The classification results show that the difference between the classes is captured when the actual class labels are given. Accuracy increases when the support is lowered because more details of the datasets is captured in the code tables. In all cases, the performance is above the set baseline. The experiments regarding the purity show that the purity is at the baseline and improves when the support gets low enough because the finer differences get exposed. This exposes one of the points where the algorithm can still be improved on, which is in order to get better results in reasonable amounts of time, the support must be lowered. In order to deal with lower support settings, the amount of evaluated candidate tables must be lowered. Now for every candidate, the entire database is covered. By reducing the amount of candidate tables which have to be evaluated, the amount of database covers that have to be performed lowers too. This can be done by evaluating the generated itemsets in Gain Order and only generating candidate tables with the $c \times b$ first item sets. Here $c \in \mathbb{N}^+$ is some constant greater than 1 and b is the beam-width. This will reduce the amount of database covers in the worst case from $\mathcal{O}(|\mathcal{F}| \times b)$ to $\mathcal{O}(c \times b)$ evaluations per iteration. There is a possibility that the compression performance will go down when this is applied.

The `GroeiSlim` algorithm successfully shows the ability to identify multi-values dependencies in data. The algorithm shows that in the presence of a multi-valued dependency, the data in these columns is added as an itemset in the table and anti-chains are generated within the same iteration. These are then present in the larger tables. The amount of code tables also reduces as more data of the same dependency ends up in a single table. The algorithm also is resistant against noise in the data and does not identify a multi-valued dependency when it is not present.

8 Conclusion

In this master thesis project, algorithms have been presented in order to perform non-disjoint clustering. These are inspired by the `Groei` algorithm. Experiments show that the `GroeiSlimNoS` algorithm is able to identify the different characteristics in the data, and that it both captures the general patterns and also patterns that differ from the general patterns but can still be grouped together with other patterns. This gives insight in the data from different perspectives. The obtained code tables are different and capture different aspects of the dataset. The compression results show that all code table are able to capture the structure in the data and improve on the compression ratios obtained by the `Groei-F` algorithm.

The `GroeiSlim` algorithm is successfully able to find multi-valued dependencies in data while being resistant to noise. This is done by creating anti-chains in every iteration and the item sets from the different anti-chains end up in the same code table in later iterations. This pattern of candidate table generation suggests that such dependencies might be present in the data.

There is still room for improvement. Although the runtime is much lower than the `Groei` algorithm, the support levels used in the experiments for the `Slim` algorithm do not give results in reasonable amounts of time to allow the algorithm to be used interactively. For this, the amount of database covers must be lowered. This can be done evaluating only a limited amount of candidates in Gain Order. By being able to handle lower supports, more interesting groups of patterns can be found in the data. This can be investigated in future works. Other improvements are investigating whether a parameter-free variant can be created such that the optimal number of clusters is automatically determined.

Bibliography

[Ami+09] Enrique Amigó et al. “A comparison of extrinsic clustering evaluation metrics based on formal constraints”. In: *Information retrieval* 12.4 (2009), pp. 461–486.

[Bre01] Leo Breiman. “Random Forests”. In: *Machine Learning* 45.1 (2001), pp. 5–32.

[Böh+06] Christian Böhm et al. “Robust information-theoretic clustering”. In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2006, pp. 65–75.

[CCS12] Adele Cutler, D. Richard Cutler, and John R. Stevens. “Random Forests”. In: *Ensemble Machine Learning: Methods and Applications*. Ed. by Cha Zhang and Yunqian Ma. Boston, MA: Springer US, 2012, pp. 157–175. ISBN: 978-1-4419-9326-7.

[Chu+06] Keh-Shih Chuang et al. “Fuzzy c-means clustering with spatial information for image segmentation”. In: *computerized medical imaging and graphics* 30.1 (2006), pp. 9–15.

[Coe04] Frans Coenen. *The LUCS–KDD Discretised–normalised ARM and CARM Data Library*. 2004. URL: http://www.csc.liv.ac.uk/~frans/KDD/Software/LUCS_KDD_DN/.

[CV05] Rudi Cilibrasi and Paul MB Vitányi. “Clustering by compression”. In: *IEEE Transactions on Information theory* 51.4 (2005), pp. 1523–1545.

[DF03] Sandrine Dudoit and Jane Fridlyand. “Bagging to improve the accuracy of a clustering procedure”. In: *Bioinformatics* 19.9 (2003), pp. 1090–1099.

[Dun73] J. C. Dunn. “A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters”. In: *Journal of Cybernetics* 3.3 (1973), pp. 32–57.

[FF12] Artur J. Ferreira and Mário A. T. Figueiredo. “Boosting Algorithms: A Review of Methods, Theory, and Applications”. In: *Ensemble Machine Learning: Methods and Applications*. Ed. by Cha Zhang and Yunqian Ma. Boston, MA: Springer US, 2012, pp. 35–85. ISBN: 978-1-4419-9326-7.

[Fim] *Frequent Itemset Mining Dataset Repository*. URL: http://fimi.ua.ac.be/data/.

[FLS04] D Frossyniotis, Aristidis Likas, and Andreas Stafylopatis. “A clustering method based on boosting”. In: *Pattern Recognition Letters* 25.6 (2004), pp. 641–654.

[FM07] Christos Faloutsos and Vasileios Megalooikonomou. “On data mining, compression, and kolmogorov complexity”. In: *Data mining and knowledge discovery* 15.1 (2007), pp. 3–20.

[FS95] Yoav Freund and Robert E Schapire. “A desicion-theoretic generalization of on-line learning and an application to boosting”. In: *European conference on computational learning theory*. Springer. 1995, pp. 23–37.

[Grü05] Peter Grünwald. "A tutorial introduction to the minimum description length principle". In: (2005).

[Grü07] Peter D Grünwald. *The minimum description length principle*. MIT press, 2007.

[GV03] Peter D Grünwald and Paul MB Vitányi. "Kolmogorov complexity and information theory. With an interpretation in terms of questions and answers". In: *Journal of Logic, Language and Information* 12.4 (2003), pp. 497–529.

[KLR04] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. "Towards parameter-free data mining". In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2004, pp. 206–215.

[Lic13] M. Lichman. *UCI Machine Learning Repository*. 2013. URL: http://archive.ics.uci.edu/ml.

[Llo06] S. Lloyd. "Least Squares Quantization in PCM". In: *IEEE Trans. Inf. Theor.* 28.2 (2006), pp. 129–137. ISSN: 0018-9448.

[LP08] Ming Li and MB Paul. *P. Vit anyi, An introduction to Kolmogorov complexity and its applications*. 2008.

[LVS06] Matthijs van Leeuwen, Jilles Vreeken, and Arno Siebes. "Compression picks item sets that matter". In: *Knowledge Discovery in Databases: PKDD 2006* (2006), pp. 585–592.

[LVS09] Matthijs van Leeuwen, Jilles Vreeken, and Arno Siebes. "Identifying the components". In: *Data Mining and Knowledge Discovery* 19.2 (2009), pp. 176–193.

[Mam] *Mammals data set*. URL: http://www.european-mammals.org.

[MBTP04] Behrouz Minaei-Bidgoli, Alexander Topchy, and William F Punch. "Ensembles of partitions via data resampling". In: *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on*. Vol. 2. IEEE. 2004, pp. 188–192.

[Pol12] Robi Polikar. "Ensemble Learning". In: *Ensemble Machine Learning: Methods and Applications*. Ed. by Cha Zhang and Yunqian Ma. Boston, MA: Springer US, 2012, pp. 1–34. ISBN: 978-1-4419-9326-7.

[RH11] Samuel Rathmanner and Marcus Hutter. "A philosophical treatise of universal induction". In: *Entropy* 13.6 (2011), pp. 1076–1136.

[SK11] Arno Siebes and René Kersten. "A structure function for transaction data". In: *Proceedings of the 2011 SIAM International Conference on Data Mining*. SIAM. 2011, pp. 558–569.

[Sme+11] F. Smeraldi et al. "CLOOSTING: CLustering Data with bOOSTING". In: *Multiple Classifier Systems*. Ed. by Carlo Sansone, Josef Kittler, and Fabio Roli. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 289–298.

[SV12] Koen Smets and Jilles Vreeken. "Slim: Directly mining descriptive patterns". In: *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM. 2012, pp. 236–247.

[VPRS11] Sandro Vega-Pons and José Ruiz-Shulcloper. "A survey of clustering ensemble algorithms". In: *International Journal of Pattern Recognition and Artificial Intelligence* 25.03 (2011), pp. 337–372.

[VVLS07] Jilles Vreeken, Matthijs Van Leeuwen, and Arno Siebes. "Characterising the difference". In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2007, pp. 765–774.

[VVLS11] Jilles Vreeken, Matthijs Van Leeuwen, and Arno Siebes. "Krimp: mining itemsets that compress". In: *Data Mining and Knowledge Discovery* 23.1 (2011), pp. 169–214.

[XW05] Rui Xu and Donald Wunsch. "Survey of clustering algorithms". In: *IEEE Transactions on neural networks* 16.3 (2005), pp. 645–678.