Utrecht University

Department of Information and Computing Sciences
Algorithmic Data Analysis Research Group

Assessment of Unsupervised Models: In pursuit of an evaluation measure

Master Thesis

Pantea Haghighatkhah

Supervisors:
prof. dr. Arno Siebes
dr. Ad Feelders

ICA-6090117

Utrecht, August 2019

Abstract

One of the categories of machine learning is unsupervised learning. The assessment of models of this category is particularly challenging since the user lacks evidence regarding the original data set and the correct conclusions that must be made from the data set. This thesis is a pursuit for finding proper measures that can aid us in gaining insight and understanding the models. We introduce measures to evaluate the model's performance generally and with respect to specific subsets. Hence, we find out what parts of the data were learned well by the model and what parts were overlooked. We also introduce a procedure for producing synthetic data with controlled levels of randomness to examine the models with varying amounts of noise in the data. Finally, we apply our methods to various data sets and numerous models learned from them and conclude eminence of some of these models. We also point out the reasons for the poor performance of the models on some of the synthetic data sets.

Preface

I started this project about nine months ago with all its challenges. Throughout these nine months I gained an invaluable insight about the unsupervised models and their evaluation with the help of my supervisor prof. dr. Arno Siebes, who I would like to thank for all the time he invested in my thesis project. Moreover, I would like to thank dr. Ad Feelders who took over the position of second supervisor and guided me in the later stages of my thesis. At the end I want to thank my family, friends for their help and support throughout these two years and most importantly I want to thank Utrecht University community and alumni for the Utrecht Excellence Scholarship that was given to me for joining this Master's programme and made all of this possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

There are various types of data mining and machine learning methods currently used. One of the categories especially used on the data sets to extract the unknown for the user is *Unsupervised Learning*. This type of learning strives to discover the unknown patterns hidden in the data. As a result, there are no means of classification such as labels available for the process of learning. The nature of this category of learning methods causes ambiguity in assessing the quality of the unsupervised model. The user of such method does not have any indications to anticipate the patterns that the learning process must find. This impacts one's ability to evaluate the performance of these models especially in a quantitative manner. Another issue arises when multiple models are learned on the same data source and we need to compare them scientifically and fairly in order to employ the best model among them. Another issue can be that while carrying out unsupervised learning, we are not aware of the coverage of the learned model, in other words, we do not know which parts of the data is learned and which parts are missed by the model.

As the learning goal and the data type varies in each project, evaluation of the resulting models from learning algorithms becomes increasingly challenging. This may be due to lack of standardized method for assessment of such models and varying definition of success which can be subjective to the project in which the algorithm is being applied.

In general we strive to answer these question: "Has the model overfitted/underfitted to the data?" "What parts of the data has it overfitted/underfitted to?" Ideally to answer these questions perfectly, one must have full knowledge of all existing patterns in the data i.e. its structure and consequently have the perfect knowledge of the noise in the data set. However, this knowledge is hardly available in real life scenarios and the motivation to use unsupervised models is to find out about these data patterns. In this project we have introduced some measures and approaches that can help assessing models when the ground truth (the true structure) is unknown. Multiple experiments are performed to aid us in deciding whether these measures indeed point to the best model or not.

For the purpose of this project we are targeting pattern set mining algorithms. This category of unsupervised learning algorithms returns a set of patterns found in the data that best define the dataset. The number of patterns that capture the essence of the data should be limited and the pattern set must summarize the data concisely. The aim of this project is to be able to quantitatively (and possibly qualitatively) measure performance of a model learned by such algorithm.Compression algorithms fall in the category of pattern set mining algorithms as they seek the true distribution of the data which expresses the true structure of the data. The algorithm that I will use for this purpose is *Krimp*, a pattern set mining algorithm.

We aim to investigate the extent to which a Krimp model has captured the structure among the noise of the data. We propose to carry out this investigation by evaluating the model with respect to various subsets of the original data set including random and non-random subsets. The non-random subsets are chosen in a way to capture coherent data together and as a result represent a substructure of the data. We consider the size of the encoded (sub)sets as an indication of the performance. Choosing various subsets non-random manner allows for investigation of various substructures of the data. In contrast, randomly sampled sets let us explore the effectiveness of the model on the data set as a whole. The method is tailored to consider both of these aspects while evaluating a model.

While forming various (sub)sets of the data, we introduce controlled noise to the data. In order to achieve this we attempted to generate our data from ground truth and then added the noise at varying levels to the generated data. This provides us with the means to test overfitting of the model. Overfitting occurs when the model fits to the noise present in the data (i.e. the parts of the data that do not follow the general patterns). As we exaggerate the noise in these data sets, we can examine if the model's performance deteriorates considerably or it will keep a steady performance along various noise levels.

Chapter 2

Preliminaries

In this chapter we introduce the methods and notions that we will make use of in our procedure and experiments.

2.1 Krimp

As we have mentioned in the previous chapter, our research in heavily dependent on an unsupervised learning algorithm, which is the Krimp algorithm [1] in our case. Krimp is a compressor that aims to compress a given data as it partially learns the data set's distribution. It achieves this by finding the set of patterns that define the data set in the most concise form. This algorithm is tailored to process discrete categorical data sets where there are a number of attributes and each attribute has a domain of values to choose from. The set of all possible values of all attributes are called the set of items noted as $\mathcal{I}$. Each data point is represented by a row which is an itemset (set of items) $\{i_1, i_2, \ldots, i_r\}$ such that every item in the set is unique and belongs to $\mathcal{I}$. Particularly, each item in the row is the instantiation of one of the attributes.

A pattern found by Krimp is in form of an itemset of an arbitrary size and it is associated with a code. All the patterns found by Krimp and their associated codes compose a code table (CT) which is the final model that is learned by Krimp from the data set. The code table returned by this learning algorithm has a specific order. The patterns returned are sorted based on their length, i.e. the longer pattern comes before the shorter pattern in the code table.

To compress a row of the data set using the code table, you start at the top of the code table and as soon as you encounter a pattern that is a subset of the row i.e. it occurs in the row, you remove all the items of the pattern from the row and add the code associated with the pattern to the compression of that row. You continue until the row is empty. To ensure compressability of every row, the code table also includes the individual items at the end that are called the singletons.

2.2 Subset

One of the choices to be made is the method used for forming subsets from the original dataset. The options considered for this choice are subset querying and bootstrap.

As we do not have access to a data source and only hold one collection of the data points, it is not possible to illustrate our measures with absolute certainty. It is needless to say that our estimations will carry a certain error and in order to decrease this deviation we make use of the Bootstrap. Bootstrapping is a method to reduce the error of measurements of the data. It is a technique of resampling in which we make B samples (with replacement) of size N (given the size of the original data set is also N)[2]. Our measurements are obtained by applying the calculations to these samples. As the number of samples B increases the standard error of our estimated measurements decreases and gets closer to the true error. In addition, it is a possibility to learn the models on each of these Bootstrap samples, and compare their performances on the original set with the performance of the original model on the samples. This way we can at the same time find models that improve performance comparing to the original model while we are evaluating it.

2.3 Measures

2.3.1 Compression length

One of the useful measures to gain insight about the performance of a Krimp model is the compression length of the data set. This refers to the size of the data set after it is compressed by the code table. The reasoning behind using this as a measure for the purpose of evaluating a model comes from the Krimp concept [1]. It reasons that the best model compresses the data best. In other words the best model describes the data the shortest. As mentioned, running the Krimp Algorithm on the data set yields a code table as the model. The code table dictates substitution of subsets of rows with a code. Applying this on every row in the dataset results in presumably the shortest encoding (compression). One can therefore use the length of the encoded dataset as some indication of performance of the model.

2.3.2 Compression ratio

However, the compression length heavily depends on the size of the starting data set. To assess the model's performance on the substructures we may only use it to compress various subsets of the data. If size of one of these subsets is considerably smaller or larger than the other subsets ,then its compression length is not comparable to the rest. In this case, it is more reasonable to compare the compression ratios. This measure is defined as the ratio between the size of uncompressed data to the size of compressed data.

2.3.3 Cross-compression

The two measures described above alone are not enough to provide knowledge about performance of models relative to each other. Specifically, we are observing each model in isolation. It is learned on a data set and it is used to compress the same data set. This procedure provides us with no information regarding the general performance of the model. It may be the case that the model has overfitted and it performs poorly on some other data (collected from the same source). The exact opposite can also be the case, namely, the model may have learned the data insufficiently, however, we cannot detect this by only observing its compression length and compression ratio calculated in isolation. Hence, it is viable to create one or multiple points of reference.

Considering this, for each bootstrap set we apply Krimp to derive a model. We can use each model to compress every data set (including the original data set) and use the original model to compress the bootstraps. This procedure is called cross compression. It allows us to observe the performance of the model on the marginally different data sets that are still adequate representative of the original data set. While also giving us the chance to observe the performance of marginally different models on such data sets. This way we can create a performance matrix of all the models which provide us with many points of reference to be able to compare the original model with.

Another advantage of this procedure beside providing us with insight for evaluation of a single model is that it can possibly find a model that ignores the noise in the data better and consequently captures the structure more effectively. Therefore, it makes a shorter encoding of the original dataset.

Chapter 3

Approach

In this section we outline multiple strategies for the assessment process of code tables learned by the Krimp algorithm and formalize the procedures and measures we will take. Our approach can be split in two main branches based on the criteria used to form the data sets. Namely, the branch that focuses on random samples and calculating the aforesaid measurements for each model derived from these samples. The second branch which, focuses on directed subsets and evaluation of models with respect to these subsets. By directed subset we imply that the subsets are not formed at random and a correlation exists among the data points of the subset.

3.1 Approach1: Bootstrap (random samples)

A preliminary step for all strategies is to make bootstrap sets. Let N be the size of the original dataset, then each bootstrap sample $\mathbf{x}^{*i}$ ($i \in \{1, ..., B\}$) is made by sampling from the original dataset s times with replacement (resampling), therefore $|\mathbf{x}^{*i}| = s$. The choice for the number of bootstraps (B) is dependent on the size of original dataset as well as the desired estimation error of the measurements. However, in practice it usually ranges between 50 to 200 [3].

Here we discuss the calculations of compression length and compression ratio when using bootstraps and how we plan to extract information regarding the performance of models on the data set as a whole or in its subsets containing substructure.

3.1.1 Approach 1.1: Estimating compression length and ratio

This is the simplest approach which only requires calculation of the desired statistic on each bootstrap sample $\mathbf{x}^{*i}$. First, the code table is obtained from the data set using Krimp. This code table is composed in a way to minimize the size of compressed data set. Let $\mathcal{D}$ be the dataset such that $|\mathcal{D}| = N$ and let d be a data point within $\mathcal{D}$ ($d \in \mathcal{D}$) . Moreover, let $C(d, M)$ denote the encoding of d with respect to the model M which is a code table produced by Krimp. Then the total compression length of $\mathcal{D}$ is denoted by $tcl(\mathcal{D}, M)$:

$$tcl(\mathcal{D}, M) = \sum_{d \in \mathcal{D}} |C(d, M)| \tag{3.1}$$

Now that we have the formula for calculation of the length of dataset encoding we can use it to calculate the compression ratio of the model using the following formula:

$$CR(\mathcal{D}, M) = \frac{|\mathcal{D}|}{tcl(\mathcal{D}, M)} \tag{3.2}$$

We apply the measure introduced in Formula 3.1 and Formula 3.2 to every bootstrap sample $\mathbf{x}^{*i}$ ($i \in 1, \dots, B$) derived from D. Hence, our estimate of the total compression length for the data and the compression ratio are as follows:

$$\widehat{tcl}(\mathcal{D}) = \frac{1}{B} \sum_{i=1}^{B} tcl(\mathbf{x}^{*i}, M) \tag{3.3}$$

$$\widehat{CR}(\mathcal{D}, M) = \frac{1}{B} \sum_{i=1}^{B} CR(\mathbf{x}^{*i}, M) \tag{3.4}$$

Applying this process will provide us with apprehension with respect to the effectiveness of the original model. The model is compressing various data sets that are formed at random, it may encounter data sets with more exaggerated noise as well as other data sets with more structure. Hence, the variations in its performance can bring insight about its effectiveness.

3.1.2 Approach 1.2: The cross-compression matrix

The bootstrap samples are sets representing the original data set. As a result, running the learning algorithm (Krimp in this case) may be informative about the capabilities of the algorithm with respect to the original data set. Hence, in this approach, once we have the bootstrap samples $\mathbf{x}^{*1}$, ..., $\mathbf{x}^{*B}$, we run the Krimp algorithm on each $\mathbf{x}^{*i}$ sample and obtain model M_i corresponding to the sample $\mathbf{x}^{*i}$. The slight variations in the bootstrap samples results in slight variations in the models found.

Next step is to find out which of the models have captured the structure within the data better. Supposedly, the models should have similar performances since they are learned from sets all representing the same data set with slight alterations. However, these slight alterations can result in a set that incorporates less of the noise or less of the structure within the original data set. Our first attempt to investigate this is to apply each model to every bootstrap sample and form a matrix. Let M_0 denote the model learned on the original data set $\mathcal{D}$ which we refer to as $\mathbf{x}^{*0}$ for convenience. Let l_{ij} denote the total compression length of $\mathbf{x}^{*i}$ once the model M_j compresses it:

$$l_{ij} = tcl(\mathbf{x}^{*i}, M_j).$$

We can hence form a matrix of cross-compression as follows:

	M_0	M_1	$\dots$	M_{B-1}	M_B
$\mathbf{x}^{*0}$	l_{00}	l_{01}	$\dots$	$l_{0(B-1)}$	l_{0B}
$\mathbf{x}^{*1}$	l_{10}	l_{11}	$\dots$	$l_{1(B-1)}$	l_{1B}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathbf{x}^{*B-1}$	$l_{(B-1)0}$	$l_{(B-1)1}$	$\dots$	$l_{(B-1)(B-1)}$	$l_{(B-1)B}$
$\mathbf{x}^{*B}$	l_{B0}	l_{B1}	$\dots$	$l_{B(B-1)}$	l_{BB}

Table 3.1: l_{ij} denotes the total compression length resulting from encoding of $\mathbf{x}^{*i}$ by M_j.

For each model M_k where $k \in \{1, \dots, B\}$ we will have a set of compression lengths associated with it denoted by $L_k = \{l_{0k}, l_{1k}, l_{2k}, \dots, l_{Bk}\}$. We can do the same calculations for compression ratio using that $r_{ij} = CR(\mathbf{x}^{*i}, M_j)$ and make a similar matrix 3.2 replacing l_{ij} with r_{ij}. This will leave us with a set of compression ratios for each model M_k denoted by $R_k = \{r_{0k}, r_{1k}, r_{2k}, \dots, r_{Bk}\}$ similarly to the L_k.

	M_0	M_1	$\dots$	M_{B-1}	M_B
$\mathbf{x}^{*0}$	r_{00}	r_{01}	$\dots$	$r_{0(B-1)}$	r_{0B}
$\mathbf{x}^{*1}$	r_{10}	r_{11}	$\dots$	$r_{1(B-1)}$	r_{1B}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathbf{x}^{*B-1}$	$r_{(B-1)0}$	$r_{(B-1)1}$	$\dots$	$r_{(B-1)(B-1)}$	$r_{(B-1)B}$
$\mathbf{x}^{*B}$	r_{B0}	r_{B1}	$\dots$	$r_{B(B-1)}$	r_{BB}

Table 3.2: r_{ij} denotes the compression ratio resulting from encoding of $\mathbf{x}^{*i}$ by M_j.

3.2 Approach 2: Directed subsets

The idea behind this main branch is to calculate the previously introduced measures on subsets of the data that are not randomly chosen. The data points present in the directed subsets will have a certain pattern in common which is a way to capture a portion of the general structure within the subset. Models learned from the directed subsets are then compared against the original model (learned from the original data set). It is expected that if a model has learned the structure of the data well, it performs as well or perhaps even better on the directed subset. This expectation is due to the higher coherence or in other words a rather more clear structure within the directed subset. An ideal model has the same performance on any subset of the data as it has on the original dataset.

Directed subsets are derived from the original data set by querying the dataset for a pattern. The result will be the data points which include the pattern in the query. To explain this more formally, we first need to clarify what is meant by a *pattern*.

Let $\mathcal{A} = \{A_1, A_2, \ldots, A_s\}$ be the set of attributes that defines a data point (row in the data). Let $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ be the set of all possible values that the attributes in $\mathcal{A}$ can hold. Please note that each attribute $A_x \in \mathcal{A}$ can hold exactly one value from all the possibilities available for that value. To clarify this, we consider an example of a data set that has attributes "*Eye_color, Height, Gender*". Each of the attributes can have multiple values. *Eye_color* $\in$ {Blue, Black, Brown, Green }, *Height* $\in$ {Tall, Average, Short} and Gender $\in$ {Male, Female}. The set of all values is then {Eye_color = Blue, Eye_color = Black, Eye_color = Brown, Eye_color = Green, Height = Tall, Height = Average, Height = Short, Gender = Male, Gender = Female}. As you can see, each of the attributes has its own specific domain and even it can be the case that the value types of different attributes varies. However, it is not possible for an attribute to have multiple values at the same time. If it is, a new attribute which is the combination of the original attributes must be introduced and each combination of the values should form a value for this new attribute.

Each data point is then defined as an instantiation of these attributes or in other words set of values assigned to these attributes. Let d be a data point, it is then defined as $d = \{i_1, i_2, \ldots, i_s\}$ where $A_1 = i_1, A_2 = i_2, \ldots, A_s = i_s$. The dataset $\mathcal{D}$ is then defined as a collection of such data points. A pattern $\mathcal{P}$ is then defined as a set of restrictions on at least one of the attributes. This restriction is in form of definite values for a (nonempty) subset of attributes. For the sake of simplicity, from now on we take attributes to have binary distinct discrete values. Therefore, we can represent a data point as a subset of $\mathcal{I}$ i.e $d \subset \mathcal{I}$. Consequently a pattern $\mathcal{P}$ can be formed also as a subset of $\mathcal{I}$. We say that a row d has pattern $\mathcal{P}$ if and only if $\mathcal{P} \subset d$. Now that we have introduced the notion of attributes and patterns we proceed to define the process of subset querying more formally.

A query $Q(\mathcal{D}, \mathcal{P})$ denotes results of querying pattern $\mathcal{P}$ from the dataset $\mathcal{D}$ where $\mathcal{P}$ is a non-empty list of attributes. Each data point returned in the result of the query must contain all the attributes in the pattern. This is captured formally in the following equation:

$$Q(\mathcal{D}, \mathcal{P}) = \{d \in \mathcal{D} | \mathcal{P} \subset d\} \tag{3.5}$$

It is important to note that the query Q separates the dataset in two (mutually exclusive) subsets of the dataset, namely, $Q(\mathcal{D}, \mathcal{P})$ and $\mathcal{D} \setminus Q(\mathcal{D}, \mathcal{P})$.

3.2.1 Querying the data

In this approach instead of making numerous random sets, we make numerous coherent subsets of the data. This is to ensure existence of a (sub)structure within the set. By doing this we aim to identify parts of the data that are not captured well by the model. Or even the parts of the data that the model overfits to. Hence the procedure to chose these subsets must be chosen such that it accommodates our needs.

In this approach we firstly need the list of most important patterns in the data or in other words mine the pattern sets of the data. These patterns come in various supports $supp(\mathcal{P})$:

$$supp(\mathcal{P}) = \frac{|\{d \in \mathcal{D} | \mathcal{P} \subset d\}|}{|\mathcal{D}|}.$$

Given the definition above, the support is then a number in the range $[0, 1]$. The support of a pattern can be an area of focus when it comes to querying the dataset. The support of a pattern relates to the significance of the pattern or rather the structure it represents in the dataset. The higher $supp(\mathcal{P})$, the more general the returning set of $Q(\mathcal{P}, \mathcal{D})$ would be. On the other hand, if the support of a pattern is low, querying such pattern returns a subset of the data that is smaller and hence includes a clear substructure that can be easily distinguished from the noise. To clarify this further, one may think of the pattern as a magnifying glass. The lower its support, the stronger the magnifying glass which zooms in on a smaller part of your data. This can have its advantages and disadvantages. The advantage is that the subset will not contain many substructures but rather only a few, thus the structure is more distinguishable. On the other hand, if the subset is too small it may miss all the structure and only contain noise. If the pattern has a large support, it selects a larger portion of the data set and hence results in a more random and chaotic set to be returned by the query.

The patterns that we use must be mined using a certain method. There is a large variety of options to choose a mining method from. We have multiple requirements from the patterns. Besides significance, all together they should be able to cover the entire data set or in other words all parts of the data set must be represented by these patterns. Also it is essential that querying these patterns yields various substructures of the data. This is to limit the intersections between these subsets and ensure examining mutually exclusive sections of the data.
For this purpose we find the *Miki*'s in our data set [4]. *Miki* stands for *Maximally Informative k-itemset*. The K-itemset is a set of size k including items. Each item can be considered as a pattern of size one. They are chosen such that they maximize the information conveyed about the data set. The method illustrated in [4] finds k patterns that best explain the data set. This is decided based on the measure of joint entropy which indicates the amount of information captured by the set of patterns (in this case).

The entropy of an itemset is then calculated from its joint entropy. Let $miki = \{I_1, \dots I_k\}$ and let $I_j \in miki$ such that where $I_j = i_j \in \{0, 1\}$ where 0 implies absence of item I_j in a row and $I_j = 1$ implies otherwise. The joint entropy is then defined as:

$$H(miki) = -\sum_{i_1 \in \{0,1\}} \cdots \sum_{i_k \in \{0,1\}} P(i_1, \ldots, i_k) \log_2 P(i_1, \ldots, i_k)_2 \tag{3.6}$$

In each iteration every item that already does not exist in *miki* is added to the set and the joint entropy is calculated using Formula 3.6. The item that increases the joint entropy the most is then chosen as the next item in the *miki*. This approach makes a greedy choice in each step of the iteration and does not calculate the most informative set. That is due to fact that solving this problem perfectly is NP-hard.

The parameter k can be chosen by the user. One should consider that if the value k is not sufficiently large, then the evaluation is partial and does not suffice as a complete assessment since some parts of the data set will be missed out. On the other hand, unnecessarily large k will result in an itemset including items that do not provide any extra information i.e. do not increase the mutual entropy of the miki and hence originate in the same sections of the data. Consequently querying these items (patterns) will yield redundant subsets. This is ineffective and in the worst case the similar performance of a model on the redundant subsets can be mistaken for stability of its performance. Therefore it may result in an unjust assessment of the model by yielding ill-advised measurements in the process of evaluation with respect to *miki* in the data.

The authors of [4] consider integer values of 2 to 7 for the choice of k as they claim this to be sufficient for their selected data sets. The goal of this method is to maximize joint entropy of the itemset to be returned. Clearly, the more items added to the set, the higher the amount of information provided by the set and thus the higher the joint entropy of the set. However, it is expected that the added information by new items diminishes, as the size of the itemset grows. A possible approach to find the proper k is to start from the lowest value 2 and calculate the joint entropy of the set. Each step we find and add the next item such that its presence in the set increases the entropy the most. After several iterations the joint entropy will increase only by a small percentage. In other words, the joint entropy stabilizes. We define a minimum growth of entropy for each next item that is added to the set and as soon as this criteria is not met, we stop increasing k. We chose that an adequate increase in the joint entropy is 15% for increment of k by 1.

For each data set, we will take all its bootstraps $\mathbf{x}^{*1} \ldots \mathbf{x}^{*B}$ and we will query each bootstrap by every pattern in the miki. Note that the individual items in the miki are considered as patterns. Since the miki contains k items, from each bootstrap $\mathbf{x}^{*i}$ we can make k subsets by querying each of these items. Hence, for every item in the miki (denoted by $\mathcal{P}$) we have a set of subsets of the bootstraps:

Each column in the Table 3.3 represents the subsets of bootstraps of the data that are queried by the same pattern $\mathcal{P}_i$. The subsets in a column e.g. $\mathcal{P}_i$, are representative of the same substructure of the original data set. To gain insight about the performance of a

	$\mathcal{P}_1$	$\mathcal{P}_2$	$\dots$	$\mathcal{P}_k$
$\mathbf{x}^{*0}$	$Q(\mathbf{x}^{*0}, \mathcal{P}_1)$	$Q(\mathbf{x}^{*0}, \mathcal{P}_2)$	$\dots$	$Q(\mathbf{x}^{*0}, \mathcal{P}_k)$
$\mathbf{x}^{*1}$	$Q(\mathbf{x}^{*1}, \mathcal{P}_1)$	$Q(\mathbf{x}^{*1}, \mathcal{P}_2)$	$\dots$	$Q(\mathbf{x}^{*1}, \mathcal{P}_k)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathbf{x}^{*B}$	$Q(\mathbf{x}^{*B}, \mathcal{P}_1)$	$Q(\mathbf{x}^{*B}, \mathcal{P}_2)$	$\dots$	$Q(\mathbf{x}^{*B}, \mathcal{P}_k)$

Table 3.3: Each bootstrap set can be queried by $\mathcal{P}_i \in miki$

model in various section of the data (represented by patterns), we collect the compression ratio of the model for every subset. Then density graph of the compression ratios for subsets within each column of Table 3.3 form an empirical distribution.

The distribution of compression ratio of the model on one sections of data is not insightful when considered in isolation. Since there is no possibility to measure how far off the distribution is i.e how bad the model is performing on the subset. However, we know if a model has learned the data evenly in all subsections, then its performance is uniformed in all of the sections of the data. This fact is then used in the later sections to help us formalize our evaluation measure.

3.3 A sanity check: Pattern Statistics

We generate numerous bootstrap sets with varying amounts of randomness. It will be explained in Chapter 4 how we restrain and alter the amount of randomness in the data such that it does not hinder the main patterns in the synthetic data. In our data generation method we start from the ground truth which is the main patterns and while generating the synthetic data (bootstraps) we apply randomness of a desired level. As a result of applying this method, we will have access to a pool of bootstraps with a variety of noise levels.

It is then interesting to investigate which patterns are found most frequently in these data sets. To do this, we should run our learning algorithm Krimp on each of the generated sets to extract the patterns. We put together all the patterns that are found by Krimp from all of the data sets. Then the statistic we want to track and study is the number of occurrences of each pattern in each data set. Separating the data sets to groups of different randomness levels, allows for collection of their associated statistic in groups of data sets with the same amount of noise. For each group and each pattern to study, we obtain a distribution for number of occurrences of the pattern.

This experiment help us investigate the performance of our data generation algorithm and observe if the ground truth that we start with can be found in the generated data. The most frequent patterns in that are found in the entire generated data sets can be ranked by their number of occurrences in the entire data sets generated. The patterns of the highest ranks should match with the ground truth i.e. the patterns in the original code table.

3.4 Evaluation measures

To obtain a formal evaluation of a model, we aim to find a quantitative measure of its performance. This measurement is then considered as an assessment of the the model from various aspect. Each part of the main structure in the data set can be considered essential and therefore, it is expected to be captured by the model. To conduct a fair evaluation, one must examine the model with respect to all significant structures that it is expected to learn.

Using the first approach — calculating the compression length/ratio for each bootstrap set— we can obtain the histograms of each measurement for every model — (M_0, M_1, ..., M_B) learned from the bootstrap samples — yielding the empirical distributions denoted as $p_0, p_1, \ldots, p_B$.

Moreover, we have the compression ratios of each subset bootstrap (queried by miki patterns) when compressed by the models. As a result for each model we have a distribution of the compression ratios per subset denoted by $q_1, q_2, \ldots, q_k$. Each distribution educate us about the measurements of the model in one section of the data distinguished by a miki item. This will help us obtain an understanding about evenness of the model's learning throughout various segments of the data. A better model strives to uniformly learn these segments of the data. The variations in the distributions of the compression ratio of a model in different data subset, can indicate overfitting in some areas and hence underfitting in some of the other areas.

Ideally, if the algorithm learns a pattern perfectly, the resulting model will compress the subsets of the data made by querying patterns equally well and even as well as it compresses the whole data set. Thus the distributions of the compression lengths have negligible differences ($p_1, \ldots p_B$). Similarly, if a model has captured a pattern $\mathcal{P}_i$ poorly and it has managed to capture some of it by chance, it should have apparent variations in its compression ratio distributions of various data set groups.

3.4.1 Kullback-Leibler divergence

To quantify the inadequacy of the model in expressing patterns $\mathcal{P}_i$, $i \in \{1, ..., k\}$, we use Kullback-Leibler divergence method [5] to measure the average difference of q_i from $p_0, \ldots, p_B$.

The distance between two distributions P and Q defined by Kullback-Leibler divergence is as following:

$$D_{KL}(P||Q) = -\sum_{x \in X} P(x) \log \frac{Q(x)}{P(x)}$$

The KL divergence measure is a way to quantify the difference between two given distribution based on the information notion or in other words, the entropy. The cross entropy of distribution P and distribution Q ($H(P, Q)$) is the average length (in bits) of a sample of P distribution that is encoded using the optimal compression code of Q. In case $P = Q$ you can expect the cross entropy to be equal to $H(P)$ or entropy which is in average the minimum number of bits needed to encode samples from P distribution. The D_{KL} measure then captures $H(P, Q) - H(P)$ which is essentially the extra bits of information needed to encode P sample using Q distribution instead of P. This in a way formulates the distance between Q and P distributions. The motivation behind the concept of this difference is also presented below as we expand and rewrite the formula.

$$\begin{aligned} D_{KL}(P||Q) &= -\sum_{x \in X} P(x) \log \frac{Q(x)}{P(x)} \\ &= -\sum_{x \in X} p(x) \log q(x) + \sum_{x \in X} p(x) \log p(x) \\ &= H(P, Q) - H(P) \end{aligned}$$

3.4.2 Evaluation with respect to the data set as a whole

While D_{KL} is a suitable measure for comparing between two distributions, we should find an appropriate approach to incorporate it to reach our goal of quantifying the performance of the model. So far our approaches have obtained distribution of compression rates and compression lengths for each bootstrap. This gives us a distribution in p_m ($m \in \{0, \ldots, B\}$) representing the model and its performance on the entire dataset. These bootstraps are created at different randomness levels $l \in \{2, \ldots, 20\}$) (later discussed in details). We can group the measurements per randomness level (for every model m) and form the distributions $P_m = \{p_2^m, p_3^m, \ldots, p_{20}^m\}$.

We also have collected distributions of these measures when the model is compressing the specific subsets of the data (separated by $\mathcal{P}_i$). Let $Q = \{q_1, ..., q_k\}$ denote the collection of all such distributions.

Here we define our evaluation measures as following:
As the randomness level increases in different levels, a perfect model must be able to keep a steady performance regardless of the noise in the data. Therefore, the first measure for assessing the model with respect to the entire data is the sum of pair-wise KL-Divergence of $p_l^m, l \in \{2, ..., 20\}$. For each model m this is formulated as:

$$Divergence\_all(m) = \sum_{p \in P_m} \sum_{p' \in P_m} D_{KL}(p||p') \tag{3.7}$$

3.4.3 Evaluation with respect to all partitions of the data

The other aspect of the model that we are concerned with is the stability of its performance through different subsets of the data each representing a substructure. To capture this as a measure, we again use the KL-Divergence as a measure of difference between distributions. The better the model, the steadier the value of its compression ratio for the subsets, thus the lower the KL-Divergence of its compression ratio distributions. We add up the pair-wise differences of the model's compression ratio distributions for each subset of the data and consider it as our second evaluation measure. The compression rate distribution of model m on the subset queried by $\mathcal{P}_i$ is denoted by q_i^m and the set of all of these distributions is represented by $Q_m = \{q_1^m, \dots, q_k^m\}$, hence we formulate the above evaluation measure as following:

$$Divergence\_subsets(m) = \sum_{q \in Q_m} \sum_{q' \in Q_m} D_{KL}(q||q') \tag{3.8}$$

Computing the score by the Formula 3.7 and Formula 3.8 provides us with an overall quantification of the model on the data set. It brings insights about whether the model has managed to capture the structures well or it learned the data partially.

3.4.4 Evaluation with respect to a single partition of the data

Recall that we have bootstraps from various levels of randomness. If we query each of these levels by patterns in our miki, we will create bootstrap subsets per randomness level. Hence, creating distributions of compression ratios per randomness level, per pattern. Taking the average divergence between distributions of the different randomness levels but the same pattern, provides insight to robustness of the model in finding a certain substructure and compressing it in presence of varying amounts of noise. For each pattern $\mathcal{P}_i$ several subsets $S_l^1, ..., S_l^B$ are formed where l is the randomness level and $l \in \{2, \dots, 20\}$. For model m, the compression ratios of these subsets are calculated and result in $CR(S_l^b)$ where l ranges in $\{2, \dots, 20\}$ and b ranges in $\{1, \dots B\}$. For each choice of l and i ($\mathcal{P}_i$) we have a distribution of compression ratios denoted as q_l^i. The difference between distributions of the compression ratio of subsets made by the same pattern and different randomness levels should be minimized as the model's performance increases. In that case, it can easily compress the same subsection of the data including a substructure when its noise varies. Let $L = \{2, \dots, 20\}$, i be the pattern index and m be the model index, then the formulation of this measure is as following:

$$Divergence(m, i) = \sum_{l \in L} \sum_{l' \in L} D_{KL}(q_l^i||q_{l'}^i) \tag{3.9}$$

The difference of Measure 3.9 from the previous two approaches is that this measure is a way assess the model with respect to only one substructure of the data rather than the whole data. The overall performance of the model on the entire data might be a blend of a poor and a good performance. While with this measure we can decide what parts have been learned the least and the most.

Chapter 4

Data Generation

4.1 Data Generation

For the purpose of experimenting the evaluation methods, it is essential to generate the data synthetically such that we have full knowledge of the ground truth. Hence, we attempt to generate the data according to the code table of some known data sets. The data generation algorithm then gets the code table as the ground truth, the synthetic data generated using the algorithm then is designed in a way to comply to a ground truth.

Furthermore, as mentioned previously, we need to add controlled amount of randomness to the generated data for the purpose of testing the models in various noise levels. This is controlled by some parameters that can be passed to the data generation algorithm and is explained further in this section.

4.2 Data Tree

Besides the occurrence of patterns in the data set, their co-occurrence is also of importance. There are certain patterns that occur more often together and on the other hand there are patterns that rarely happen together. The associations of the patterns must be preserved through the process of data generation otherwise the generated data will not include similar associations as the original and hence the co-occurrences can be different. Consequently, applying Krimp learning may in such generated data set result in newly introduced patterns that originally did not (partially) exist in the ground truth.

As we reasoned, the knowledge of patterns co-occurrences must be applied to the process of data generation in order to generate synthetic data that can accurately represent the original data set and keep its integrity. For this purpose we introduce the notion of *Data Tree.* The tree keeps track of the co-occurrences of the patterns. The production procedure of the Data Tree is explained in the following subsection.

4.2.1 Data Tree Generation

Running Krimp algorithm on $\mathcal{D}$ results in the code table $CT_{\mathcal{D}}$. Then the compression of $\mathcal{D}$ by $CT_{\mathcal{D}}$ is represented by $Cover(\mathcal{D}, CT_{\mathcal{D}})$ hence we also call it *cover* of $\mathcal{D}$. Each row of the cover is the compression of the corresponding row of the data set, meaning that it includes the collection of patterns that compresses the row.

The tree is made of nodes and each node holds the following information within: *counter*, *pattern*, *parent* and *children*. Initially the tree only contains the root node and such that includes $root.counter = 0$, $root.parent = null$ and $root.children = \{\}$. In order to generate the Data Tree, we start by reading each row of the cover, which yields a list of patterns. We order this set according to the ordering of the code table $CT_{\mathcal{D}}$ (this ordering can be done with respect to other criteria too). Let us denote this ordered list by $R = \{P_1, P_2, \ldots P_n\}$ where each element of the list is a pattern from the code table that exists in the corresponding row. Needless to say, all patterns are mutually exclusive and together they have all the items that exists in the row. After sorting the path that is to be added to the tree, we find the right position in the tree that it can be added. To clarify this, consider an arbitrary row $ABCD$ in the cover which is to be added to the tree. If the path ABC already exists in the data tree there is no need to add a separate path to the root, we can just add a child with pattern D to the last node of the path ABC. So in Algorithm 1 in lines 5 to 20 we attempt to find the largest part of R (starting from the beginning of R) that already exists in the data tree and only add the remaining to the right node in the tree. In parallel we also update the number of times we traverse the nodes in the tree for the purpose of probability calculations. Once all the rows in the cover are added to the tree we return the *root* node representing the data tree (Algorithm 1).

The tree as generated by the algorithm we presented, captures the entire data set and the co-occurrences of patterns. Taking a random walk in the tree to a leaf results in a path and putting together items in the patterns of the nodes in the path, results in generating a row of the cover. Then putting the items in these patterns together in a set forms the corresponding row in the data set. Hence, taking a random walk in the tree results in a random row of the data set. The motivation behind this procedure for generating the rows instead of resampling the data set directly is to be able to add randomness to the relevant segment of the data e.g. the noise.

It is however important to generate a row according to its probability. Some co-occurrences of patterns are more probable and hence rows including such combinations must happen relatively more often. In order to apply this consideration to the process of generating random rows, we make use of the count measure that is being updated in the node. While taking a random walk, in order to decide which child to choose as the next node, we generate a random number and use the count parameter of the children as their probability measure we can decide what node is the next in our path. Hence, our random walk will comply to these probabilities so that the bootstrap that is made using the tree

Algorithm 1 GENERATE DATA TREE **Input:** (Cover, code table) **Output:** (root)

```
1: procedure GenerateTree($Cover, CT$)
2:     Let $root$ be a node
3:     Let $root.parent = null$, $root.counter = 0$, $root.children = \{\}$
4:     for each line of $Cover$ denoted by $R$ do
5:         Sort $R$ according to $CT$
6:         Let $current = root$
7:         $current.count + +$
8:         for $i$ from 1 to $R.length$ do
9:             Let $n$ be the a node in $current.children$ such that it has pattern $R[i]$
10:            if $n == null$ then
11:                for $j$ from $i$ to $R.length$ do
12:                    Let $n'$ be a new node with $n'.pattern = R[j]$
13:                    $n'.parent = current$
14:                    $n'.count = 1$
15:                    $current.children.add(n')$
16:                    $current = n'$
17:                end for
18:            else
19:                $current = n$
20:                $current.count + +$
21:            end if
22:        end for
23:    end for
24:    return $root$
25: end procedure
```

is an accurate representation of the data set.

4.2.2 Introducing randomness

It is desirable to add controlled amount of variations to the data such that it does not change main structure and only influences the noise present in the data. It is expected that the randomness added to the noise of the data i.e. randomizing the noise present in the data, does not hinder the main structure of the data and hence has low effects on the learning of the data. Therefore, once this synthetic data is compressed its compression length should not differ significantly from the original data set. It is also valuable to note that if the randomness is applied carelessly such that it ruins the structure of the data will cause uneven and considerably poor performance of the original model.

Even if the variations are added properly to the data set, as we increase the randomness it enters the parts of the data that is deemed as the structure, it is expected that the code table extracted from the original data set is unable to compress such noisy data well. It is hence interesting and essential to investigate various levels of randomness in the data.

In the process of generating the data tree, we add rows of the cover to the root node. The patterns in the rows are sorted based on a measure of importance i.e. the same order as the code table. The adding process of the cover first tries to traverse the tree as long as patterns of nodes visited is the same as pattern in the cover in the right order. As it does so, it also increments the count parameter in each node by one. Therefore, the algorithm keeps track of the number of times a certain sequence of patterns have occurred. The longer the sequence, the less number of occurrences it has. Thus, as we get closed to the leaves of the tree the count parameters decrease. The minimum for the count parameter is one.

To control the level of randomness we use the notion of minimum support (minsup) as a threshold to prune the data tree. After generating the data tree, we traverse the tree and for every node n if $n.count < minsup$, then n and all its descendants get eliminated from the data tree. Thus, *minsup* works as a threshold for pruning the least likely subsequences. We can experiment with different values for this parameter to see the effect of it on the tree post-pruning. Needless to say, as *minsup* increases, the size of the pruned tree decreases (Algorithm 2).

We have chosen to work with rectangular data sets, meaning that all rows of the data set have the same length r. After pruning the tree, the random walks in the tree can be possibly result in an itemset smaller than r. This is then the room for adding randomness to the synthetic data as we extend every insufficiently small row that we generate so that it complies to the rectangular size of the data set before it is added to the synthetic data set. Also note that the rows generated from the data tree never have more than r items, where r is the size of the rows in the original data set that the data tree is generated from.

To fix the rows with less than r items, we add singletons according to their probability in the cover. It can be the case that some items mostly co-occur with other items and hardly occur as singletons, taking the probability associated with the code length of the singletons in the code table allows us to comply to these situations.

Another way to accommodate for randomness is to prune the nodes of the tree that hold a singleton as pattern (Algorithm 3). Such patterns are added to the code table to make sure that all items are included in the code table and as a result every row can be compressed. It is hence reasonable, to think of the singletons in a row more of noise than the structure and use this reasoning as an other pruning strategy. After pruning singletons, the tree that we are left with only includes patterns consisting two or more items.

Again this results in the possibility that some of the walks in the tree result in rows with less than r items. Again to fix this we randomly add singletons for the row according to their probability. This way we are randomizing the noise in each row of the data.

It is essential to note that some of the items cannot co-occur as it was explained in the Chapter 2 each column represents an attribute and its value is chosen from its domain.

Algorithm 2 MINSUP PRUNING **Input:** (root, minsup) **Output:** (root of the pruned tree)

```
procedure MinsupPruning($root, minsup$)
    Let $q$ be a queue with $root$ node in it.
    while $q$ is not empty do
        Pop the first node in $q$ and call it $n$
        for each $child$ in $n.children$ do
            if $child.count < minsup$ then
                remove $child$ from $n.children$
            else
                add $child$ to the end of $q$
            end if
        end for
    end while
    return $root$
end procedure
```

Each attribute must have only one value assigned to it, thus the items that are in the same domain cannot be both present in a row. To address this, prior to the data generation we calculate the domain of each column (attribute) from the original data set and we take them into consideration while adding singletons to the purpose of fixing the length of the randomly generated row. Needless to say, the part of the row that is generated from the data tree, does not require domain check since the data tree is generated from the cover of the original data which includes valid patterns, as a result itemsets generated from the tree are valid.

Algorithm 3 SINGLETON PRUNING **Input:** (root) **Output:** (root of the pruned tree)

```
procedure SingletonPruning($root$)
    Let $q$ be a queue with $root$ node in it.
    while $q$ is not empty do
        Pop the first node in $q$ and call it $n$
        for each $child$ in $n.children$ do
            if $child.pattern$ is a singleton then
                remove $child$ from $n.children$
            else
                add $child$ to the end of $q$
            end if
        end for
    end while
    return $root$
end procedure
```

4.3 Sorting criterion

As mentioned previously, the patterns in rows from the cover are first sorted before being added to the tree. The first criterion that can be used for sorting these patterns is the order of the code table. The code table is sorted based on the pattern length. It is also interesting to experiment with sorting by some other criteria such as code-length of the pattern, linear or non-linear combination of pattern length and code length of patterns.

For the linear combination of the code length and pattern length we have considered the following as measures:

- $code\_length$
- $pattern\_length$
- $code\_length - pattern\_length$

Moreover, we also also apply the following non-linear criteria to the process of tree generation and data generation.

- $code\_length/pattern\_length$
- $code\_length \times pattern\_length$
- $pattern\_length/code\_length$

In the experiments (Chapter 5) we have illustrated and compared their effects on the generated data and its compression.

Chapter 5

Experiments

5.1 Data Generation

To generate synthetic data, we made use of some of the well-known data sets in *UCI* database.

- iris
- breast
- heart
- mushroom
- led7
- ionosphere
- tic-tac-toe
- pima
- wine

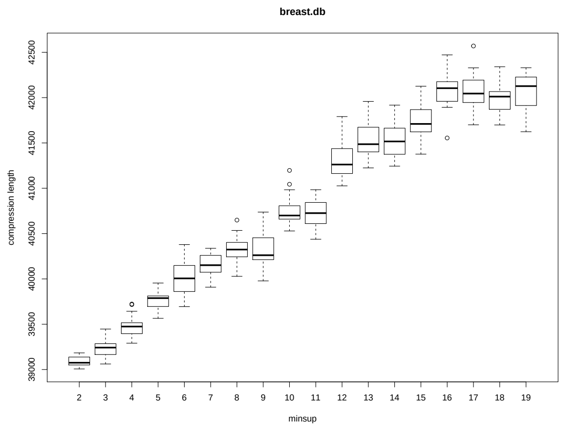
After mining each data set using Krimp, we used their code tables and covers as the ground truth for generation of our synthetic data. The data tree made for each pair of code table and cover is created as explained in Algorithm 1 and then pruned according to both strategies in Algorithm 2 and Algorithm 3.

In our experiment with the *minsup* pruning strategy we use the thresholds for minimum supports of $\{2, 3, \ldots 20\}$ and for each minimum support we generate 25 synthetic data sets. As a result in total we have 475 bootstraps for each data set and these bootstraps contain varying levels of randomness. We also generate 50 bootstraps for the singleton pruning strategy to hold a comparison with the other method.

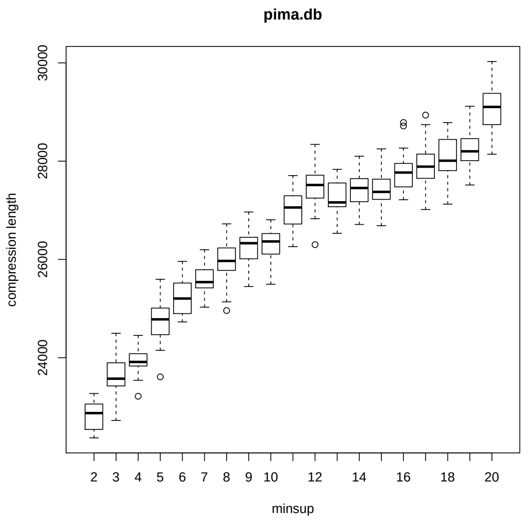
For each bootstrap we have generated the same number of rows as its original data set. Each row is fixed according to the specifications of the data set which are the domain and the row size. As a result the generated bootstraps are close representatives of the original data set.

5.2 Compression & Code table

It is expected that as the *minsup* threshold grows we prune more parts of the data tree and hence, partially remove the structure of the data. Therefore, it is normal to observe an increase in the compression length as this threshold grows. All data sets behave similar to Figure 5.1 showing an upward trajectory indicating increase of the compression length as the *minsup* increases.



(a) breast data set



(b) pima data set

Figure 5.1: The boxplot of compression lengths for 25 trials per minsup.

There are cases in which the growth of compression length is more subtle and a sharp increase is not observed. For example, the *iris* data set is one of such cases (Figure 5.2). The increase of *minsup* results in larger portions of the row being random. In the case of *iris* data set, due to the short length of the rows (five items) and hence short patterns, the permutations of items added to the end of rows as randomness is very limited and it can easily coincide with a pattern. Although we prune more from the structure as we increase *minsup*, however, because of short length of patterns and thus limited combination of the items, while randomly fixing the rows by adding singletons we sometimes end up generating the patterns that we have pruned in the row. As a result, increasing the *minsup* does not have as severe of an effect as we observe in the other data sets (Figure 5.2).

Using the singleton pruning strategy, we have made bootstraps for each data set. The compression length of the bootstraps has approximately unimodal distribution in various data sets (Figure 5.3). These bootstraps are nice presenters of the data set. However, the randomness that is added to such synthetic data, is less in control and is very dependent on the structure of the original data set. If there are more singletons at the end of rows in the cover, the randomness we add is more since larger part of each row with be eliminated. Otherwise, the randomness is less and in both cases it is not quantified.

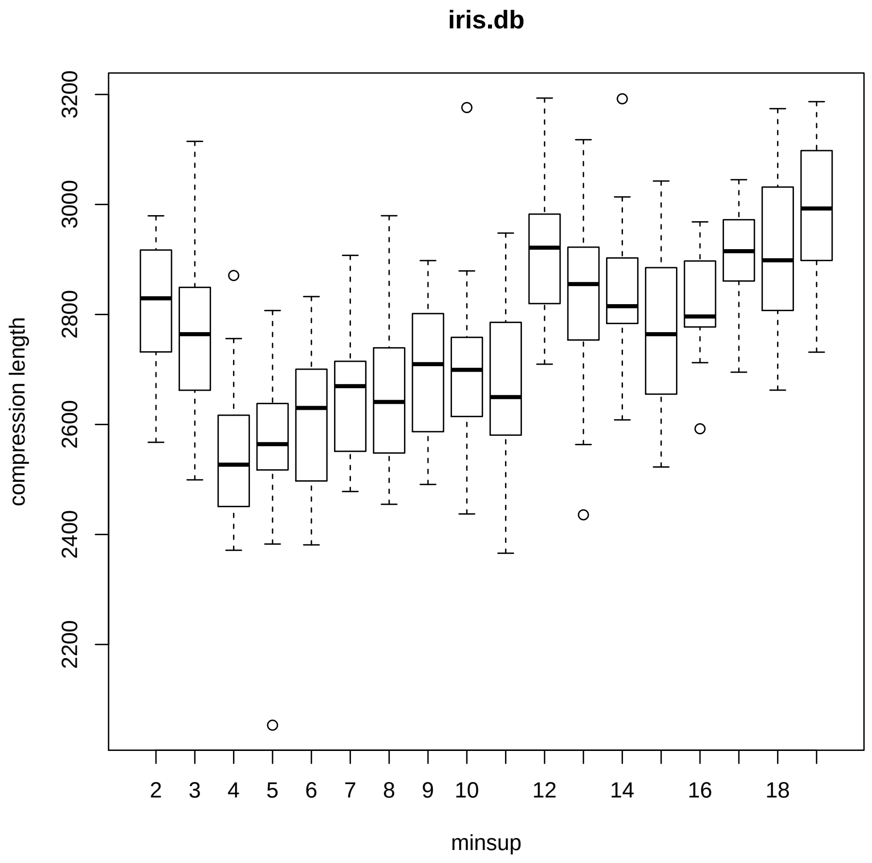
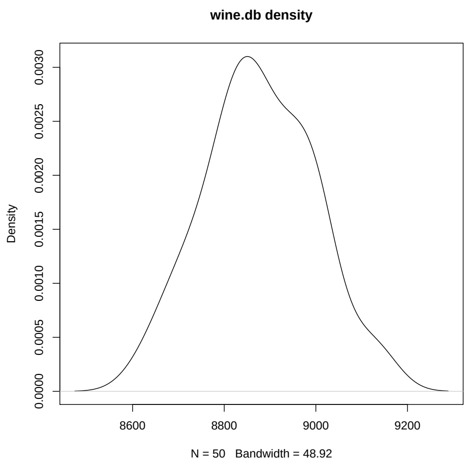
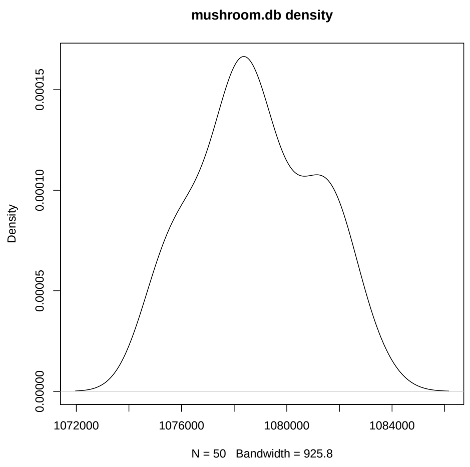


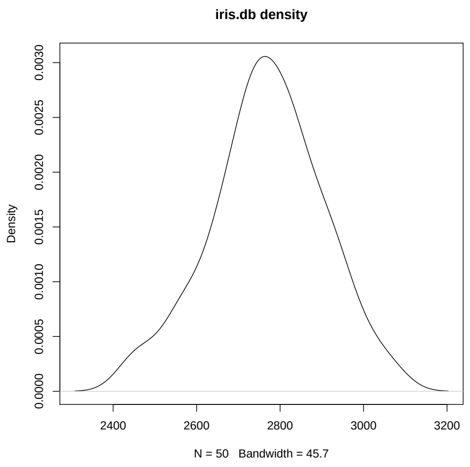
Figure 5.2: The minimum support pruning strategy does not influence the compression length of the iris data. The short pattern and short row lengths makes possibility of accidentally making patterns by adding random singletons higher.



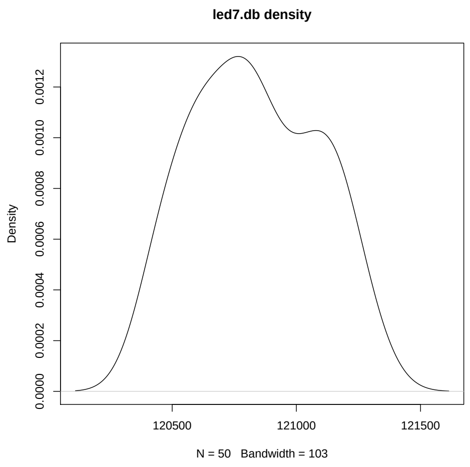
(a) wine data set



(b) mushroom data set



(c) iris data set



(d) led7 data set

Figure 5.3: The distribution of the compression length when data is generated using the singleton pruning method is fairly unimodal.

5.3 Cross-Compression

After generating the synthetic data sets, we used Krimp to find the code tables of each of the synthetic data sets. In this step, we use these code tables to cross compress all the synthetic data generated from the original data set.

Then we proceed to compressing all the synthetic data sets. This is only done for the synthetic data with *minsup* threshold, as the variations in noise makes it interesting to study. In the case of cross compressing the data generated with the singleton pruning strategy we will not be able to extract any knowledge of how each model handles the noise and which model does a better job at finding the structure regardless of higher noise.

The cross compression is performed by applying 475 code tables to 475 synthetic data sets. This returns a sizeable of 475×475 indicating the compression length given the data set and the code table it has been compressed by.

Every synthetic data set is generated using $minsup \in \{2, 3, \ldots, 20\}$. For each minimum support we generate 25 trials or in other words 25 data sets. Each of these synthetic data sets have a corresponding code table as well. Our goal is to see the effect of variations in minimum support in ability of the extracted code table to compress the data sets in different levels of randomness. Hence, for the code table at minimum support of s and the data set at minimum support of s' we want to gather information about the average compression lengths or the distribution of the compression length. For each choice of s and s' there are 25×25 compression lengths available.

Another parameter that we introduced was the sorting method that we will use for construction of the data tree. Sorting of paths before adding them to the tree has a direct effect on what pattern gets to be eliminated in the pruning procedure as the last patterns in the path have higher chances in having lower *count* variables i.e. lower support and hence they will be eliminated together with all their descendants. Since the descendants also have equal or lower *count*. Changes in sorting method allows for experimenting with where we want to add the randomness to the data. If the portions of the data that we randomize is closer to the true noise, the changes in compression lengths in the cross-compression matrix is more subtle. As opposed to this, if we add the randomness to the portions of the data that was structure then we observe more significant increase in the compression lengths in the cross-compression matrix as the randomness level increases. Using this reasoning we attempt to study the sorting methods and find the most effective of them.

One of the examples for the cross-compression is illustrated in Figure 5.4. The synthetic data generated for this cross compression is made by trees created with the pattern length sorting criterion. The $y-$axis illustrates the code tables learned on data of various *minsup*. The $x-$axis indicates the data of various *minsup*. It is important to note that the size of the circle at (i, j) where i is ct-led7-syn-i and j is led7-syn-j, illustrates the average compression size of the $25 \times 25 = 625$ corresponding trials.

There are a number of aspects in this balloon graph that is aligned with our reasoning regarding the data tree and embedding of randomness to the data. The main axis of the

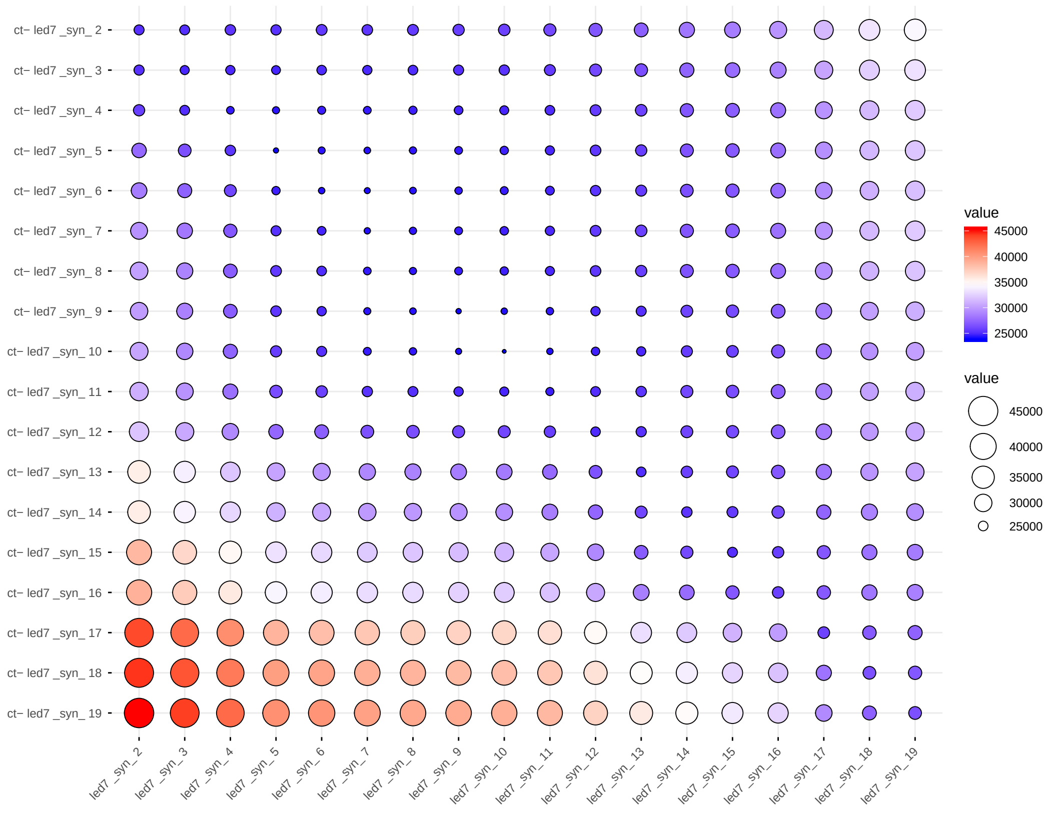


Figure 5.4: The cross compression of *"led7"* data set. All the synthetic data generated for this cross compression is produced using the **pattern length** sorting criteria. The $y-$axis shows code tables learned from data of various *minsup* and the $x-$axis represents the data sets of various *minsup*. The size and color of each circle indicates the average compression length of 625 trials, same as the color schema. The main diagonal indicates the compression length of the models compressing their own data.

matrix contains the compression length of data sets when compressed by their own code tables. Naturally, this results in the lowest compression lengths in their column (other code tables compressing the same data) and row (other data sets being compressed by the same code table). This is sensible since the code table of each data set is extracted in a way that it best compresses its data set. Also the model learns the data and attempts to fit it best, including the noises introduced by us. Therefore, as you observe in the graph, the closer each circle is to the main diagonal, the closer it is to its intended data set or code table and thus the smaller size (avg compression length) it has.

Another trend visible in Figure 5.4 is the effect of increased randomness on synthetic data sets and on their code tables. As the minimum support increases and consequently the randomness increases, more of the structure is lost in the code table. This results in the data sets being harder to compress and impacts the learning of the code tables. Therefore, circles in the bottom left corner and the top right corners have larger sizes.

In the case of *led-7* the bottom left corner is doing worse i.e. the average compression length increases considerably. This is a result of insufficient learning of the structure in the data with higher amount of noise. This is caused by overfitting to the significant noise.

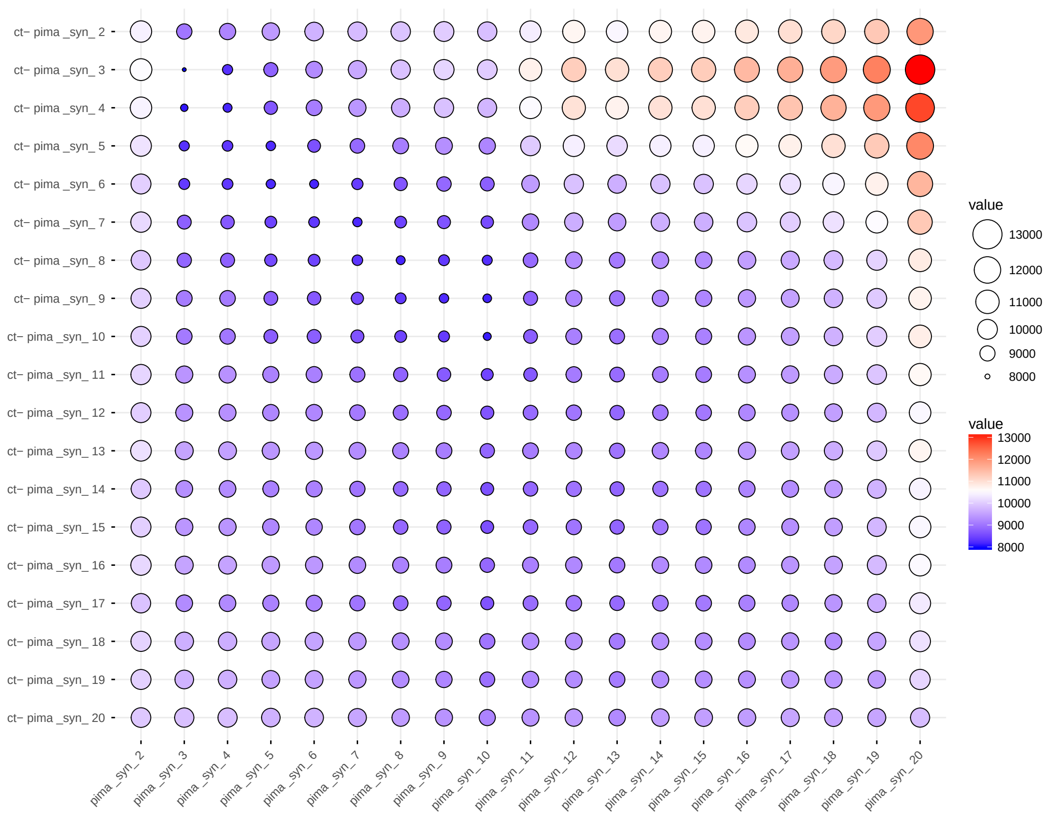
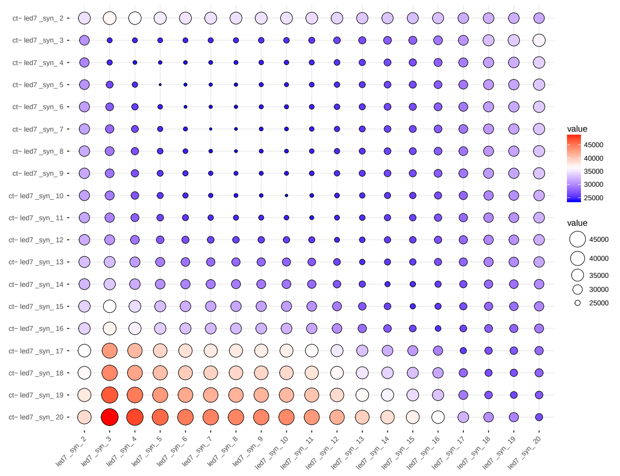


Figure 5.5: The cross compression of *pima* data set. The synthetic data is generated using the pattern length criterion for sorting data tree paths. The $y-$axis shows code tables of various minsup and the $x-$axis represents the data sets of various *minsup*. The size and color of each circle indicates the average compression length of 400 trials.

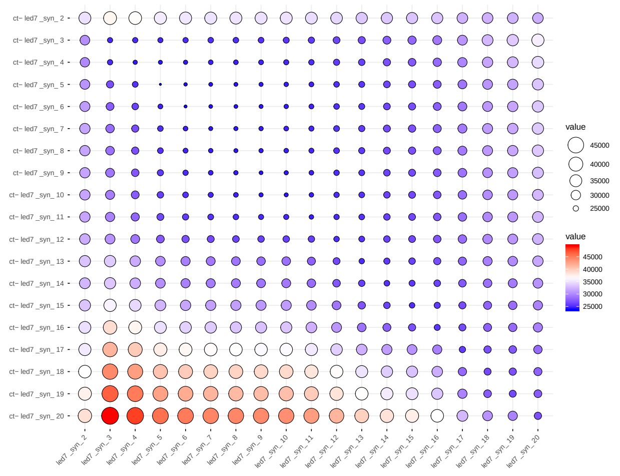
Similar trends are visible in the cross compression results of other data sets in Figure 5.5 for the *pima* data sets. The difference is that the increased noise in the data worsened the performance of models of lower *minsup* more noticeably compared to the performance of models of higher *minsup* on low noise data. It implies less overfitting in the latter models.

5.3.1 Various sorting criteria

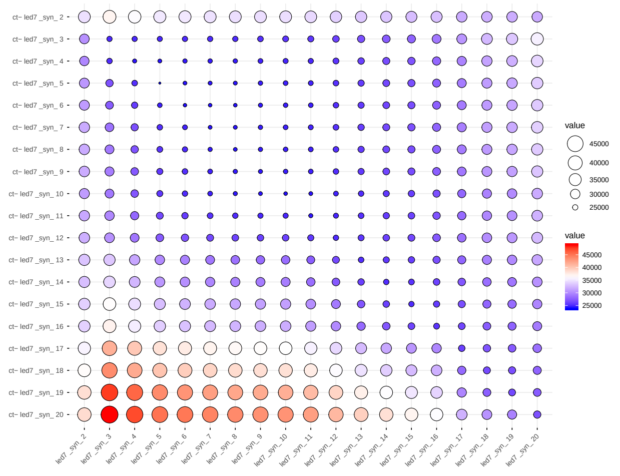
All the cross-compression results observed so far are made using the pattern length sorting criterion. This criterion is the same criterion that sorts the code table returned by Krimp. So we apply the same ordering as the code table to the patterns in one line of the cover. As mentioned in Chapter 4 we also consider other methods to sort the paths. Here we illustrate their effect on the compression length averages when cross-compressing data sets.



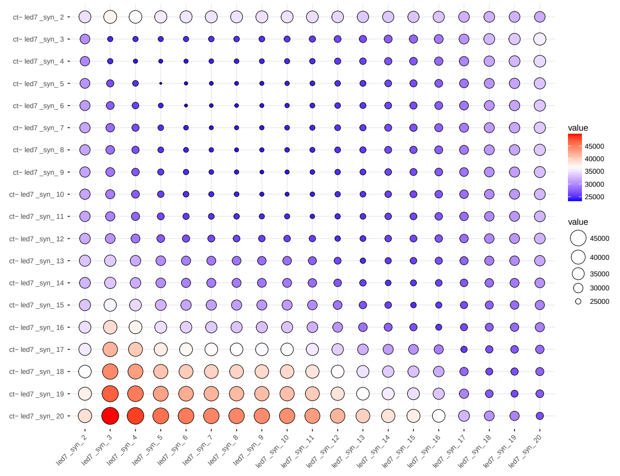
(a) *pattern_length*/*code_length*



(b) *code_length* − *pattern_length*



(c) *code_length*/*pattern_length*



(d) *pattern_length* × *code_length*

Figure 5.6: Various sorting methods used in generation of data.

As you can observe from Figure 5.6 the change in the sorting method results in minor variations in the four methods mentioned in Table 5.1.

However, they all differ from the *pattern_length* criterion presented in Figure 5.4 with respect to the first column of the cross-compression matrix. The *pattern_length* sorting method makes relatively more uniform averages for compression lengths. This is the result of more subtlety in adding randomness in the data set meaning that the randomness is truly added to the noise, and as we increase the level of randomness it exaggerates the noise to the point that it impacts the structure of the data. The consequence of using the sorting methods in the Table 5.1 on the rest of the data sets is similar. Hence, we have decided to use the *pattern_length* as our main sorting method used for the rest of the experiments as they can be computationally expensive and repeating them sixe times

$pattern\_length/code\_lengt$	$code\_length - pattern\_length$
$code\_length/pattern\_length$	$pattern\_length \times code\_length$

Table 5.1: This group of criteria for sorting result in highly similar effects on changes of the average compression ratio (circles) through the cross-compression matrix.

is not desired.

5.3.2 KL-Divergence for compression length distribution

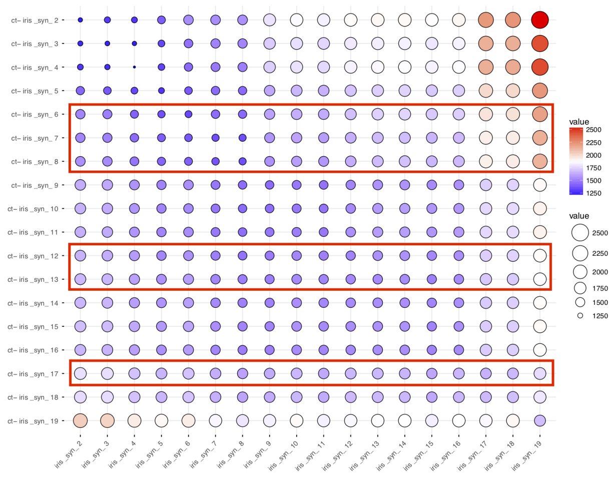
One of the interesting experiments that can be done for the purpose of comparisons of models with regards to the data set as a whole is checking the stability of the model in various stages of randomness.

As mentioned previously, ideally, if a model learns the data well, it will be able to compress it just as well when the noise is increased. However, inevitably as the noise increases in the data it will eventually have significant impact on the model's ability to compress the data since the models we learn are imperfect. Hence, we strive to learn models that get closer to the ideal case and as a result minimize the deviations of model performance as the noise in the data increases.

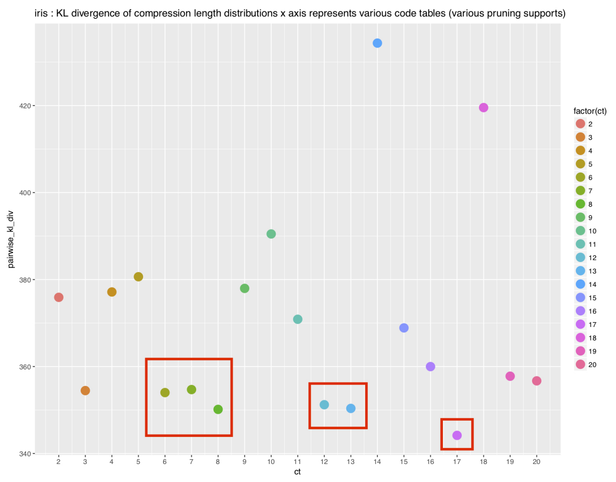
As planned in the Chapter 3 for each code table, we apply the KL-Divergence measure on the distributions of compression lengths of the data with varying levels of randomness. We then plot the results of pairwise KL-Divergence for models learned on each randomness level i.e learned on data sets of each $minsup \in \{2, ..., 20\}$.

In the iris data sets you can notice that the models with $minsup = 17$ clearly have the lowest variation in their averages of compression lengths per randomness level of data sets. In this case, the pairwise KL-Divergence of the compression length of these models also happen to be the lowest. While the averages in the models of *minsup*s 6, 7 and 8 seem to vary more than models of *minsup* 12, 13 and 17, but their distributions seem to diverge almost similarly.

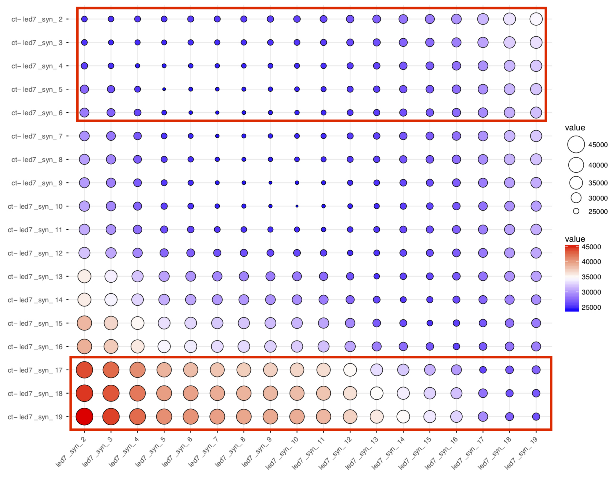
In the led7 data set there are numerous models that have low variance distributions. Some of the models are acquired from the data sets of low randomness. While some other are obtained from the data sets with higher noise.



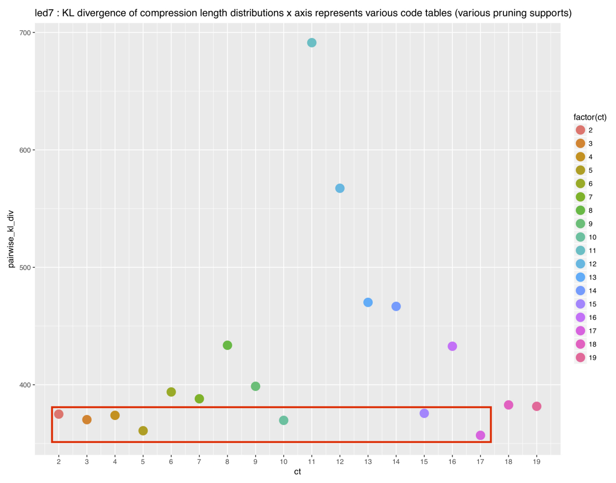
(a) iris cross-compression matrix



(b) Pairwise iris KL-divergence



(c) led7 cross-compression matrix



(d) Pairwise led7 KL-divergence

Figure 5.7: In Figure 5.7a and Figure 5.7c we have illustrated the models with the lowest variations in their distributions by the red boxes. Similar to Figure 5.4 the circles in these figures also shows the average of compression lengths of 625 instances. The groups of data sets (of the same *minsup*) with the lowest pairwise KL-Divergence are indicated by the red boxes in Figure 5.7b and Figure 5.7d. The $x-$axis in Figure 5.7b illustrates the *minsup* variable for the models.

5.4 Most frequent patterns statistic

As we suggested in Chapter 3 we collect the statistics on the patterns that occur in any of the synthetic data sets generated. Namely, we collect the number of occurrences of each pattern in each data set, in each group of synthetic data with the same *minsup* parameter and also in total (summation in all data sets). For each pattern the distribution of its occurrence is extracted from data sets of the same *minsup* level. This allows us to compare the distributions across the randomness levels using KL-Divergence.

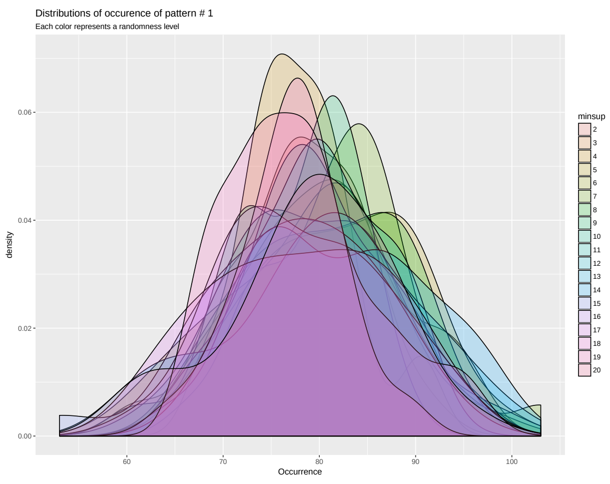
If a pattern has been occurring considerably often in a large portion of the data sets, it is a significant and important pattern. As a confirmation we have collected these patterns and compared them with the patterns of the original code table from which all this data has been generated. Indeed, the most frequent patterns in the synthetic data is the ones with the lowest code length i.e. the highest occurrence in the original cover. This is completely sensible since the code table is used as the ground truth and the synthetic data generated using the ground truth must follow it.

As you can clearly observe in Figure 5.8, the distributions start to vary more noticeably as the rank of their corresponding patterns increases already in higher ranks ,namely, # 1, # 2, # 3, # 4. It is however the question if the same trend is observed when the rank goes lower. If that is the case the KL-divergence measure that we use as an indicator for the difference of distributions must increase.

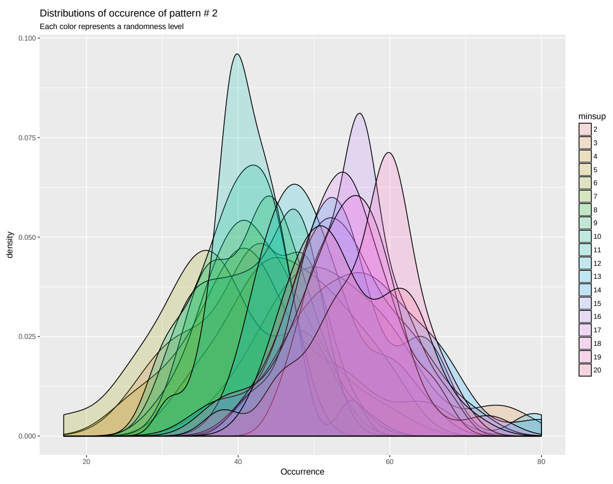
Given two distributions, KL-divergence returns a measurement of their difference. Each pattern has 25 distributions associated with it. We sum up their pairwise KL-divergence measurements and use it as a measure indicating the deviations of distributions. Doing this for the 60 most frequent patterns found in all 475 data sets (of iris in this case), we can find out if a general trend holds for the patterns.

As illustrated in Figure 5.9 as the total frequency of the patterns decreases the more random their occurrences becomes in the generated data sets and hence their pairwise KL-divergence increases more sharply. In fact, the number of patterns in the original code table of iris data set i.e. the ground truth that the iris synthetic data set is generated from is 31. Therefore, the rest of the patterns in our ranking are found randomly from the noise and as you can see as the ranking goes above 30 the pairwise KL-divergence exhibits clearly more deviations from the general trend as a sign that those patterns only are captured from the noise.

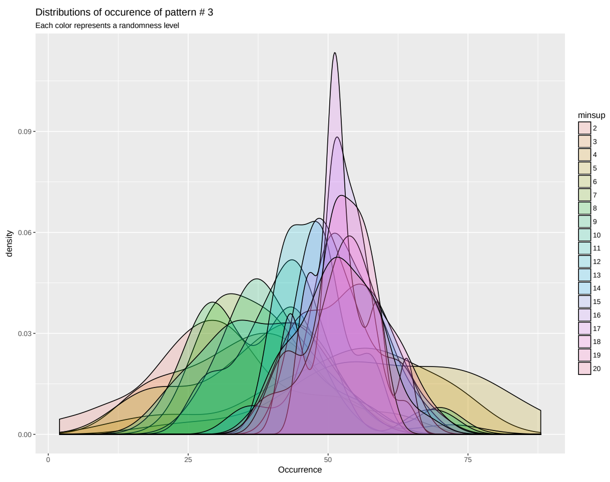
The same tendency is observed in the rest of the synthetic data generated. In some of the data sets it is the case that the pairwise KL-Divergence stays similar throughout the higher ranks and it starts increasing sharply from a certain rank.



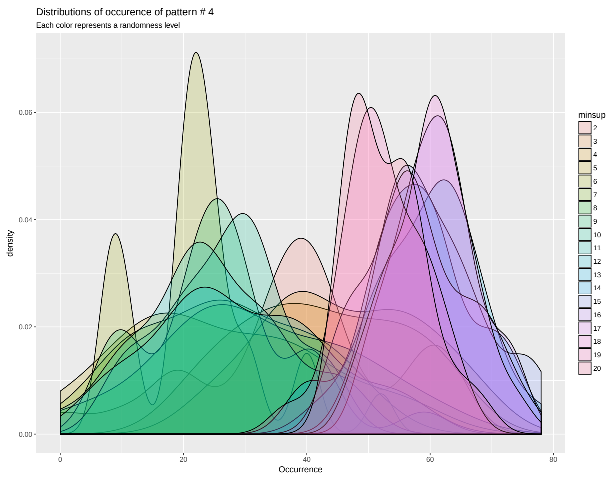
(a) rank #1



(b) rank #2



(c) rank #3



(d) rank #4

Figure 5.8

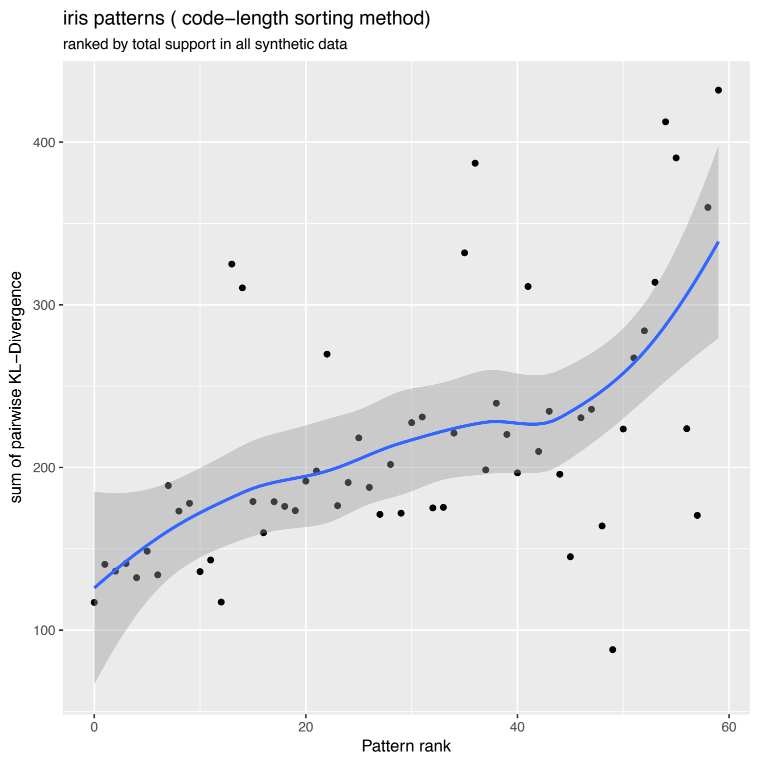


Figure 5.9: The trend of increased variations holds for all the significant patterns in the iris data set.

5.5 Directed subsets

In this section we attempt to separate the data sets into meaningful subsets that include correlated data points and use the code tables on these data sets to retrieve information regarding the performance with respect to each subset of the data. The reasoning for this approach is fully explained in Chapter 3.

To do this we find the *miki* for each data set then for each item in the *miki* retrieved, we make the corresponding subset of the data by including all rows of the data that contains the item. We also keep subset of the data that excludes the item to be able to study the effect of it.

As we apply the *miki* algorithm on the original data sets, we also keep inspecting the entropy as the k parameter i.e. the number of items in the *miki*, increases. If the growth of entropy goes below the threshold of 15% we stop increasing k. It turns out that the proper value for k is the row length of the data sets. This is reasonable as each of position in the row is an attribute with a certain domain. When we know value of an attribute, knowing about the values that it has not acquired does not provide us with any new information. As a result it is sensible that as the number k grows beyond the row length the additional information is insignificant and below the threshold we have in mind.

Having the right set of *miki* for each original subset, we can use it to create subsets from every synthetic set generated. For every item $i \in miki$ we query every synthetic set. Hence item i results in generation of 475 subsets out of the synthetic data. Doing this for every item in the *miki* leaves us with $k \times 475 \times 2$ for each data set ($\times 2$ because we both have subsets including and excluding an item).

When the separation of the data is over, the next step is to use all the models found from the cross compression part of our experiments on every subset that we have formed. We have 25 models per *minsup* level. We apply each of the 475 models to these subsets and extract the compression ratio for each subset.

As an indication of excellent performance we considered the pairwise KL-divergence of the compression ratio distribution for the various subsets of data. To be more specific, the distribution of compression ratios are made using the compression ratio collected from the subsets of the various randomness levels.

It is expected from a perfect model to exhibit similar performance across all subsets of the data set. We evaluate this by comparing the compression ratio among all the subsets of the data.

Firstly, we observe the performance of the models of various *minsup*, in each individual subset. Each item in *miki* is represented by a number in the range $[0, k)$, this number represents the rank of the item as you can see in the legend of Figure 5.10 and Figure 5.11 and the rank is based on the increase of entropy e.g. the item represented by 0 increased the entropy of *miki* the most once added to the set.

screening the variations in distributions made on each subset is done with the motivation to understand which subsets have been learned better and by which of the models.

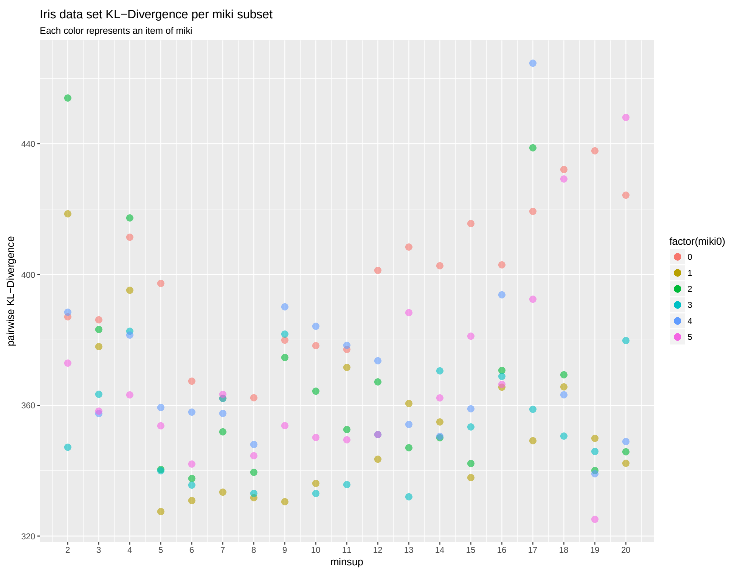


Figure 5.10: Each point represents the summation of pairwise KL-Divergence for models of a certain *minsup* for a specific subset (represented by a color).

Knowing the pairwise KL-divergence of distributions associated with every subsets of the data provides us with knowledge regarding how easily the subset is learned by the models in general. For example, if the subset is more easily learn-able then it is expected to results in low kl-divergence in various models. This is made more visible in Figure 5.11 as you can trace each item by the line. In this case subset of item #1 seems to have low divergences for its compression ratio distributions of all models.

We also noticed that it can be informative to observe the boxplots of the points presented in Figure 5.10 when grouped by models' *minsup*. This compares the quality of learning among various subsets. The boxplots are illustrated in Figure 5.1 and it is observable that the average quality of learning is similar and only the variations can tell us something about the evenness of learning within these subsets.

If we shift our focus from the quality of learning in subsets to the general quality of learning in the models. One of the ideas to check the capability of the model to learn the subsets uniformly was to calculate the divergence of the compression ratio distributions of all these subsets. In other words for each model we have accumulated all of its compression ratios per subset. As a result we capture a distribution for each subset. The KL-divergence is then calculated on these distributions. This way we will be able to tell how similar the distribution of compression ratios are in various subsets of the data i.e. how evenly the model has learned the data.

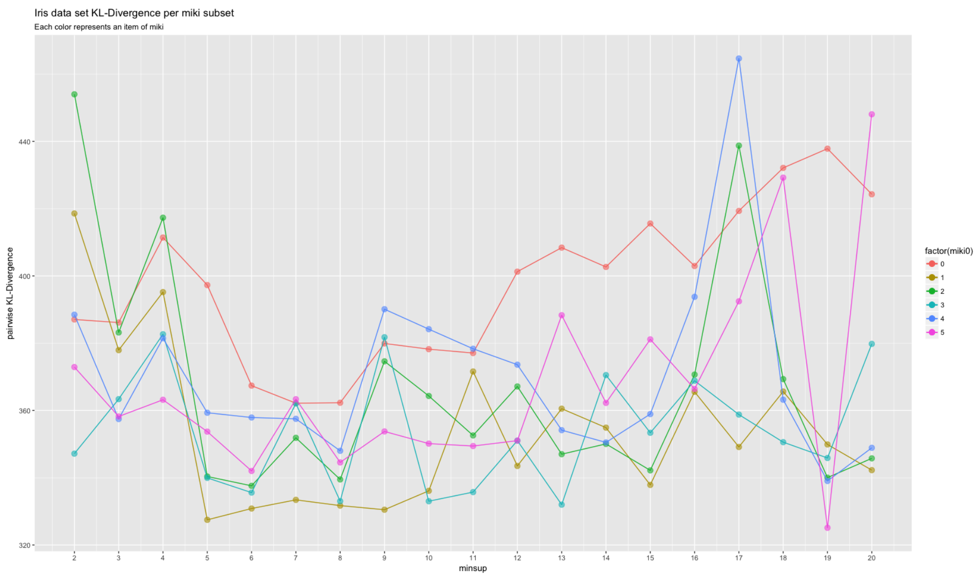


Figure 5.11: Each line represent a subset of the iris data set.

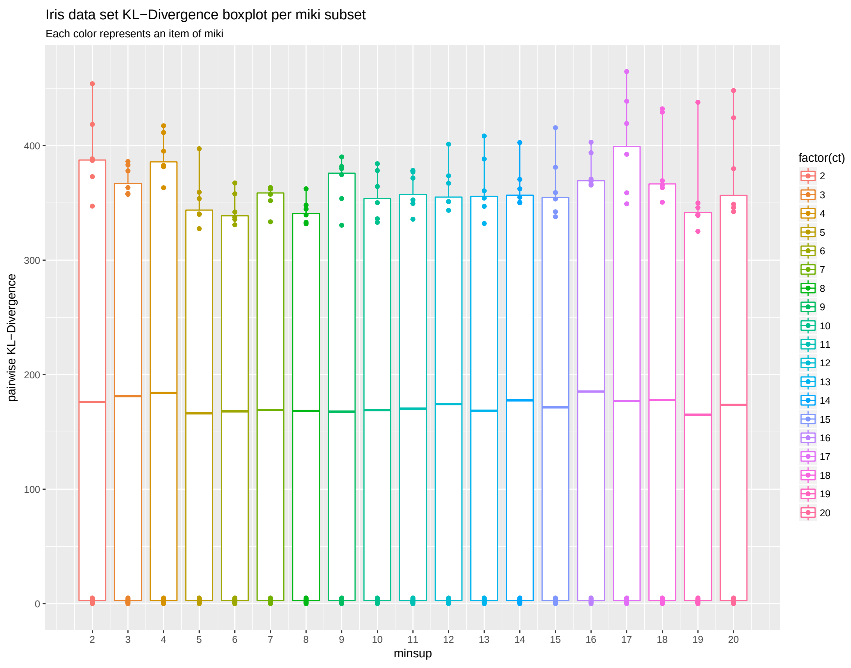
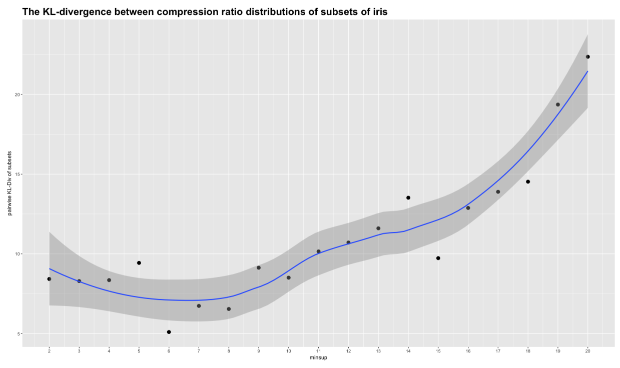
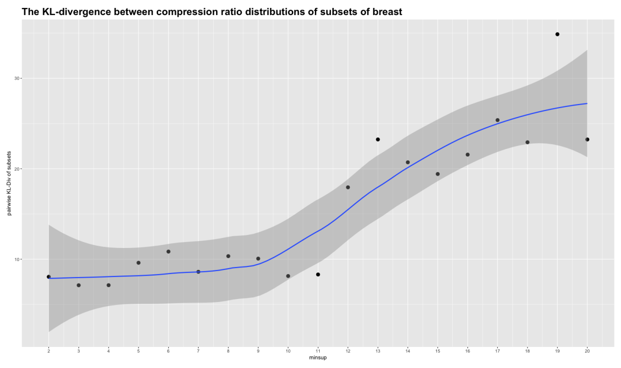


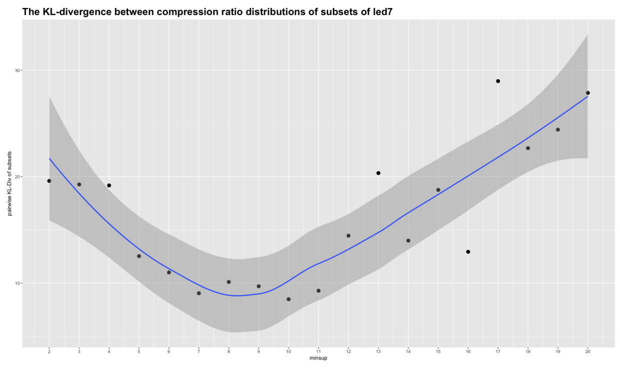
Figure 5.12: The boxplots for each *minsup* in the Figure 5.10. Each color represents a model made using a certain *minsup*.



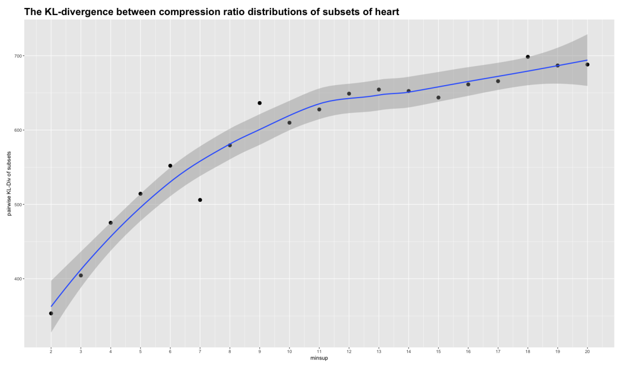
(a) iris



(b) breast



(c) led7



(d) heart

Figure 5.13: The changes in uniformity of learning subsets of data for the models as their *minsup* increases.

Chapter 6

Summary and Conclusions

We have shown many possibilities to study the data and also the performance of models learned on the data. Each data set that we have used comes with certain characteristics and density; hence it is expected that our experiments yield varying result per data set and so our conclusions for the best model for each data set can point to a different model i.e. models from different randomness levels. We made an attempt to find a universal measure that can be used on the unsupervised models to extract their proficiency in expressing the data. This allows for comparison among models extracted from the same data and the possibility to rank them regardless of the algorithm that is used to learn them and whether it was unsupervised, semi-supervised or of any other category. Consequently, the measure yields a number for any two given models that facilitates the comparison between them.

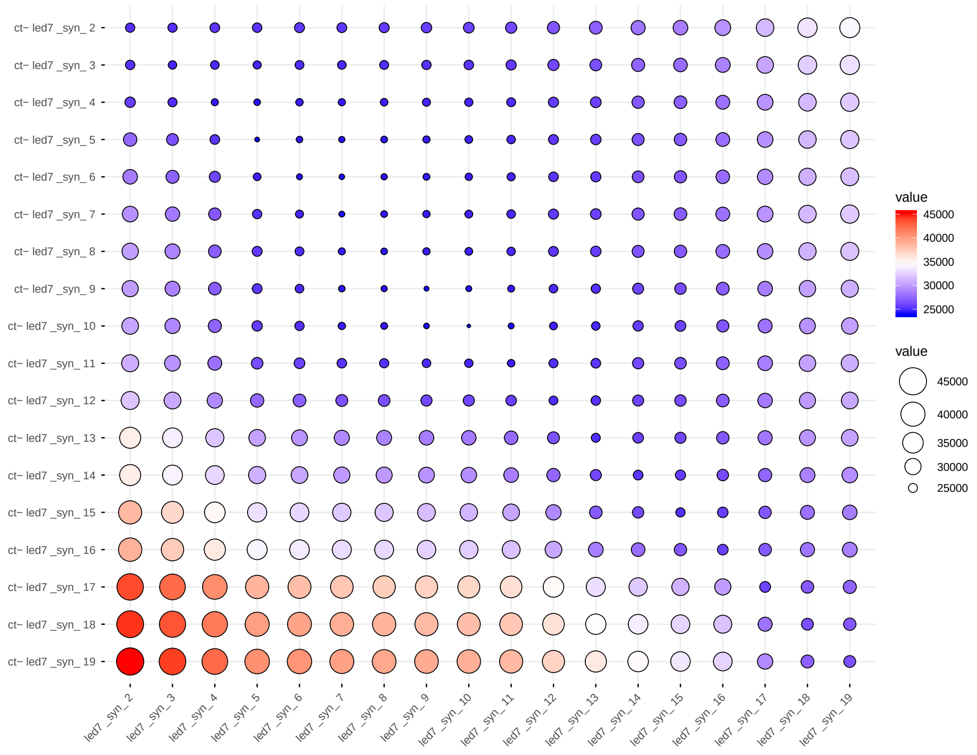
In our experiments we introduced some possibilities that have the potential to serve as a universal measure, however, they each focus on one essential aspect of the learning process which must be evaluated. As a result the assessment process merely depends on the aspect that is deemed important to evaluate. One may regard the general performance of the model on the entire data important, while another values the evenness of a learning algorithm and consequently the model. To clarify this with an example, data set $\mathcal{D}$ may contain a significant subset $\mathcal{D}_A$ and a rather smaller subset $\mathcal{D}_B$ that are mutually exclusive. Consider a model M_1 that learned the subset $\mathcal{D}_A$ very well, it may even overfit to this subset while it captures $\mathcal{D}_B$ poorly. Also consider a model M_2 that has learned the two subsets almost equally well but not as well as M_1 has captured $\mathcal{D}_A$. In this scenario it can be the case that the total performance of M_1, e.g. its total compression length, is much better than that of M_2.

This does not necessarily imply eminence of M_1 over M_2. If the user of the model values the uniformity of the model's performance over various subsets of the data, then the high total performance of the model is not informative and does not lead to any conclusions for the user. In other words, the user wants to avoid overfitting and underfitting in the subset of the data and naturally the more uniformly these subsets are learned the less is the overfitting and underfitting of the data. The opposite can also be the case where the total performance is valued over the uniformity of the performance in the subsets of the data, and hence models such as M_1 are preferred to M_2. As a result there is no standard procedure or measure to answer our question about superiority of a model over another. For each problem to be solved by a model, we have a certain goal to achieve. The goal may differ from problem to problem hence different methods must be used to measure how far the resulting model has landed from the initial goal for solving the problem.

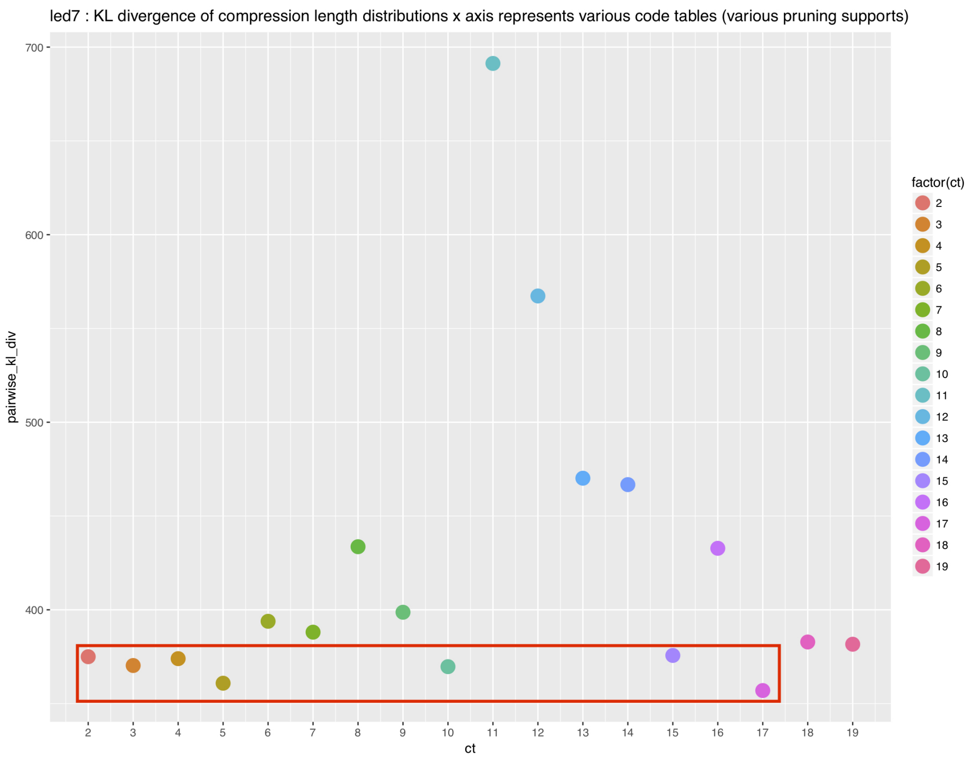
In this project we have investigated multiple methods for assessment of each of the valuable aspects of a model for its user. In the first approach we extracted information about the performance of the models with respect to the data as a whole by constructing the compression matrix. The compression matrix illustrated the influence of increasing the randomness of the data on the models learned from it. The variations in ability of the model in compressing the data of different *minsup* parameters is closely studied in this approach. As we reasoned before, the variation of performance is a strong indicator for the quality of the model since it implies stability. So in addition to the balloon plots (e.g. Figure 5.4) we also considered the KL-divergence of the distributions of performance of the models (e.g. Figure 5.7d). Let us take led7 as an example data set that we want to study closely. In Figure 5.4 we observe that the code tables of *minsup* 2 to 12 have high performance as their average code length is low. When we only look at the steadiness of this performance we obtain Figure 5.7d that tells us the models 2, 3, 4, 5, 8 and 10 beside having a high performance they also have stability in their performance as the randomness of the data set increases.

In our second approach we study the model with respect to meaningful subsets of the data (almost mutually exclusive). This is done with the motivation to investigate how well subsets of the data are learned by the model. To this end we extracted the *most informative k-itemset* to be able to separate the data set into correlated subsets consisting of substructures of the data. We have used all code tables to compress these subsets of the data and then we extracted information regarding the stability of the model's performance in each subset e.g. in Figure 5.11 or Figure 6.2a for the led7 data set. The subsets with item 5 of the miki in, the led7 data set is considerably harder to learn in the data sets with $minsup \in \{2, 3\}$. Consequently, the models learned on these data sets are not the top performers in the cross-compression matrix and counter intuitively the models of higher noise e.g. $minsup \geq 4$ managed to reach lower averages for compression of the data (Figure 6.1a). Moreover, as the stability of the compression ratio deteriorates for subsets of item 1 for $minsup \geq 17$, item 1 yields significantly larger subset of the data as it also ranks higher in increase of entropy once added to the miki. This heavily affects the learnability of the data set as you notice in the bottom left corner of the compression matrix in Figure 6.1a i.e the models retrieved from data sets of $minsup \geq 17$ hardly learn the major part of the data (subset including item 1) and that results in poor compression length of the models.

The subsets of the data are similarly learnable in randomnesses of 4, 5, 6, 7 and 8 as the KL-divergence of the subsets' compression ratios are close i.e. the lines are more concentrated in Figure 6.2a. Considering the performance of the models with respect to these subsets, you can observe that the models learned on the data sets of *minsups* 5, 6, 7, 8 result in a more even capturing of the subsets (Figure 6.2b). These models also show a significant total performance in the cross compression matrix as their total compression

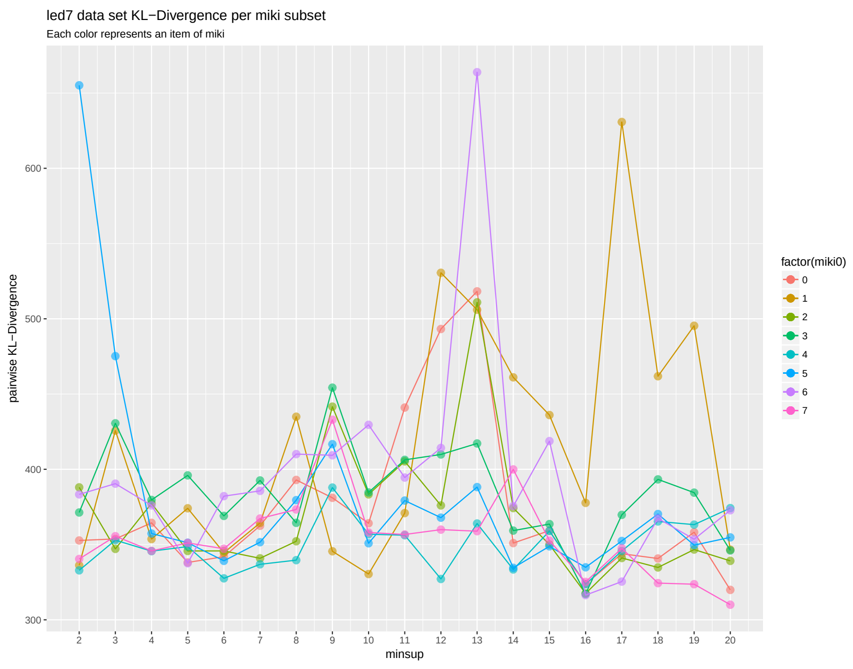


(a) Cross compression

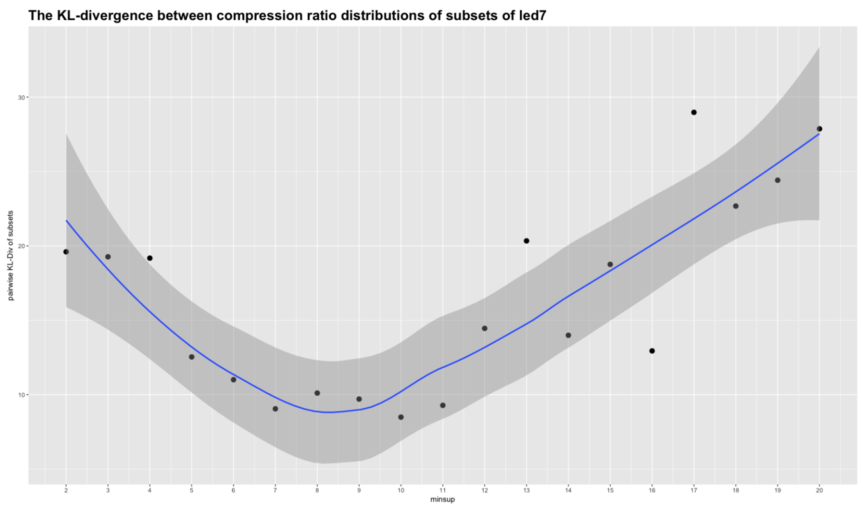


(b) KL-divergence of the compression lengths

Figure 6.1: First approach



(a) Learnability of each subset as randomness increases.



(b) KL-divergence

Figure 6.2: Second approach. The subsets of the data are similarly learnable for randomnesses of 4, 5, 6, 7 and 8 as the KL-divergence of the subsets' compression ratios are close i.e. the lines are more concentrated in Figure 6.2a. Then considering the performance of the models with respect to these subsets, you can observe that the models learned on the data sets of *minsups* 5, 6, 7, 8 result in a more even capturing of the subsets (Figure 6.2b).

lengths are low throughout varying amounts of randomness. Among these models, the models of $minsup = 5$ have a stable overall performance as well, as you can see in the red box indicating the models with the lowest KL-Divergence of compression length distributions in Figure 6.1b.

Having rankings of the models with respect to various criteria, e.g. evenness of learning or total performance, aids the user to pick the model that best fits the goal of the project. Although, in this case some of the models come very close in all criteria and can be used interchangeably as they have almost the same performance in all aspects.

This approach is also usable in practice when one data set is available and we would like to evaluate available models to be able to rank them with respect to various criteria and choose the best. The bootstraps can be made without extra randomness and proceed to the approaches that we introduced to assess the models. The added randomness in a sense aids us in assessing the robustness of the models to noise. As long as we do not want to use the code table of the data to add the noise, we can use the original data set to generate our bootstraps. Furthermore, the miki itemset can also be found on the data given and then the resulting miki can be used to query all the generated bootstraps to form subsets. This provides us with means to apply our second approach.

Bibliography

[1] Jilles Vreeken, Matthijs Van Leeuwen, and Arno Siebes. Krimp: mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23(1):169–214, 2011. 3, 4

[2] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Number 57 in Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, Boca Raton, Florida, USA, 1993. 4

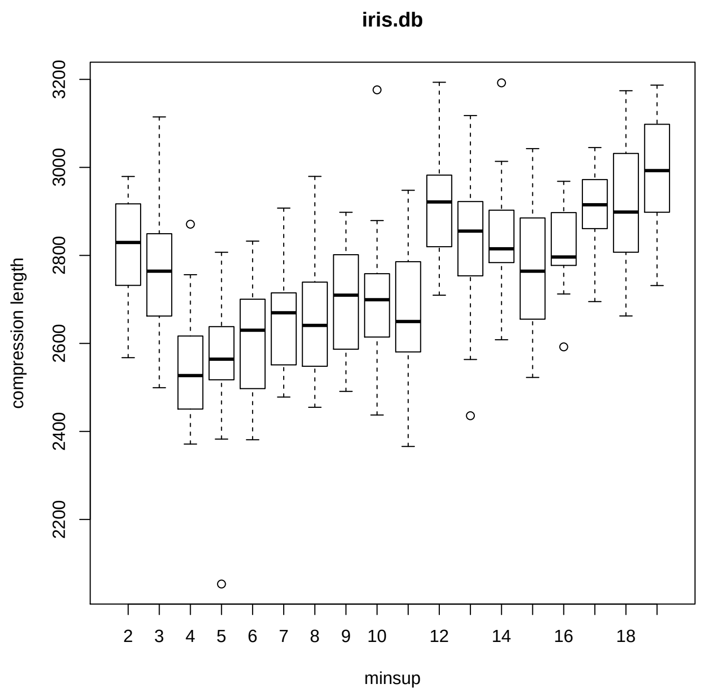
[3] Bradley Efron. Better bootstrap confidence intervals. *Journal of the American statistical Association*, 82(397):171–185, 1987. 7

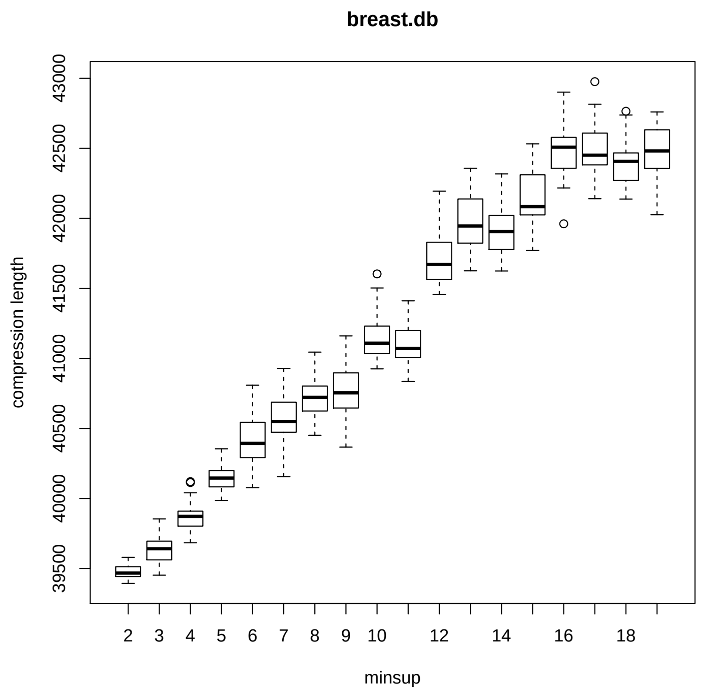
[4] Arno J Knobbe and Eric KY Ho. Maximally informative k-itemsets and their efficient discovery. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 237–244. ACM, 2006. 11, 12

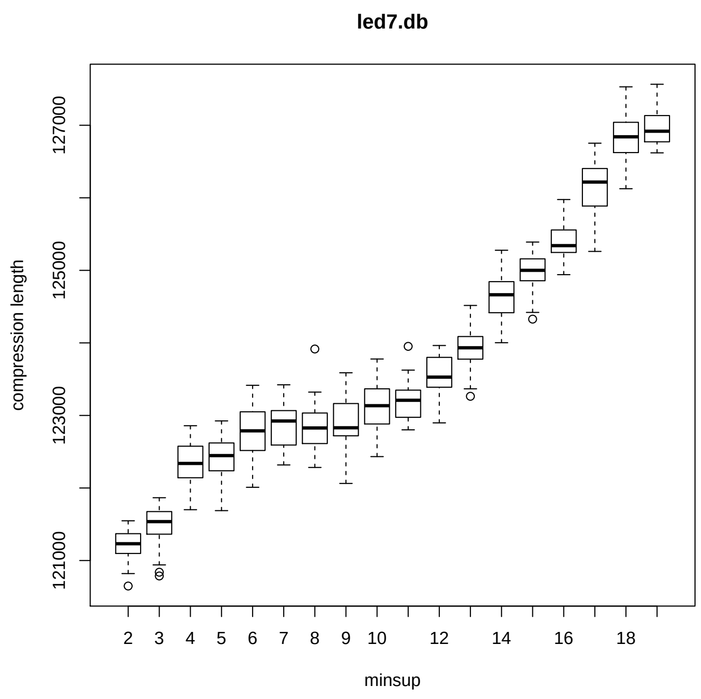
[5] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951. 14

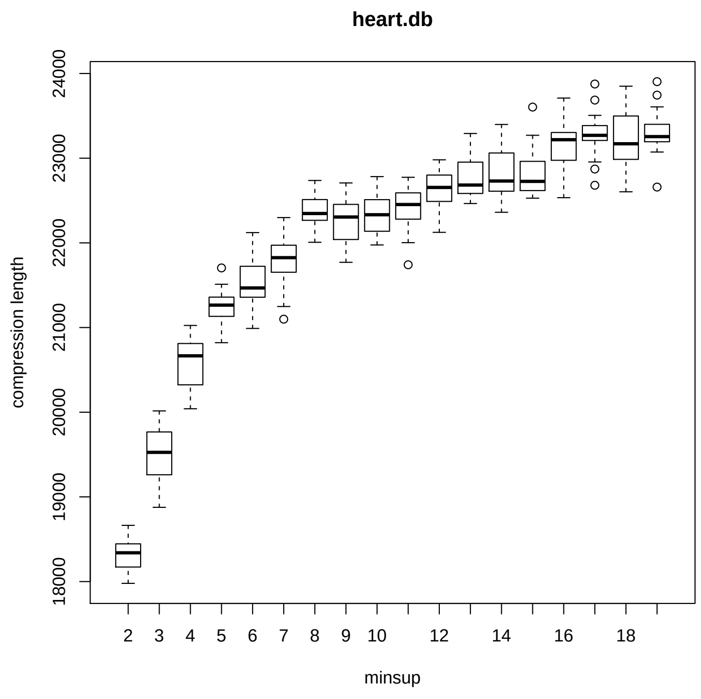
Appendix A

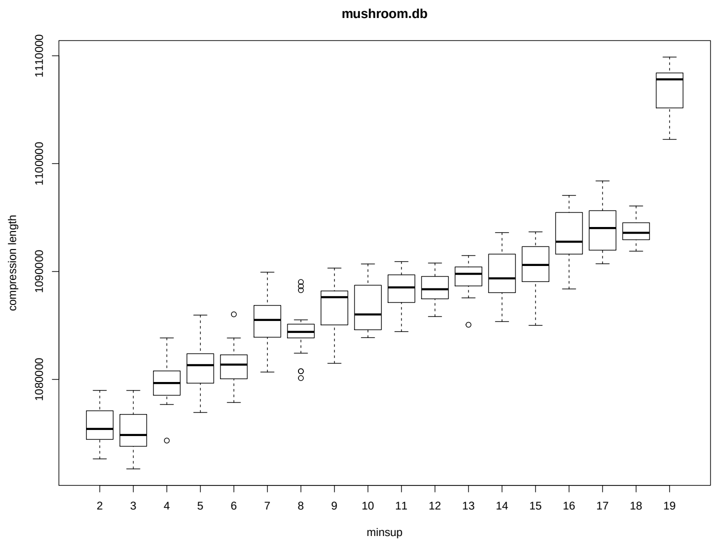
Synthetic Data Generation

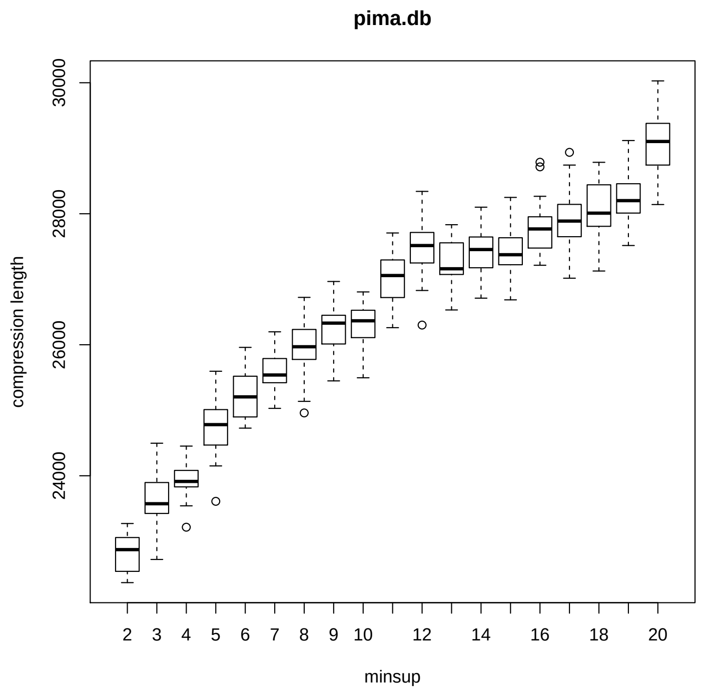

iris.db
compression length
3200
3000
2800
2600
2400
2200
2 3 4 5 6 7 8 9 10 12 14 16 18
minsup

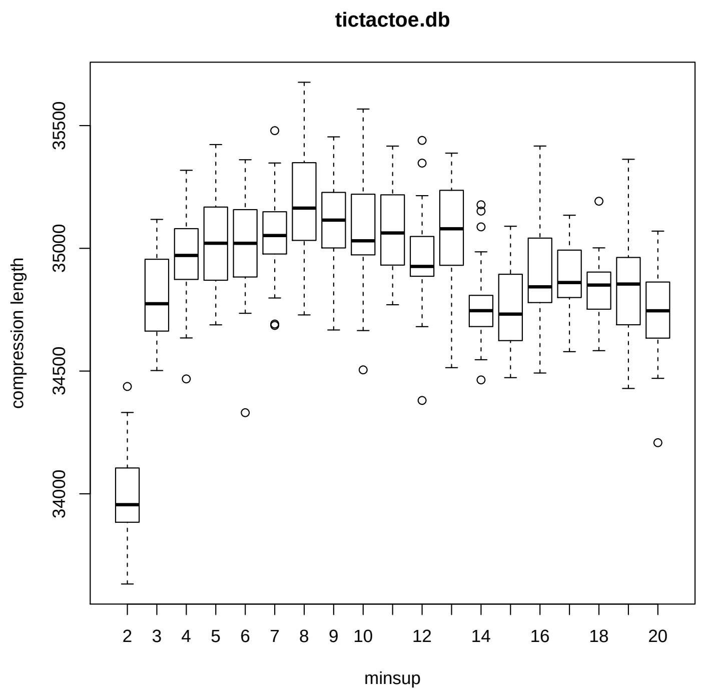

breast.db
compression length
43000
42500
42000
41500
41000
40500
40000
39500
2 3 4 5 6 7 8 9 10 12 14 16 18
minsup

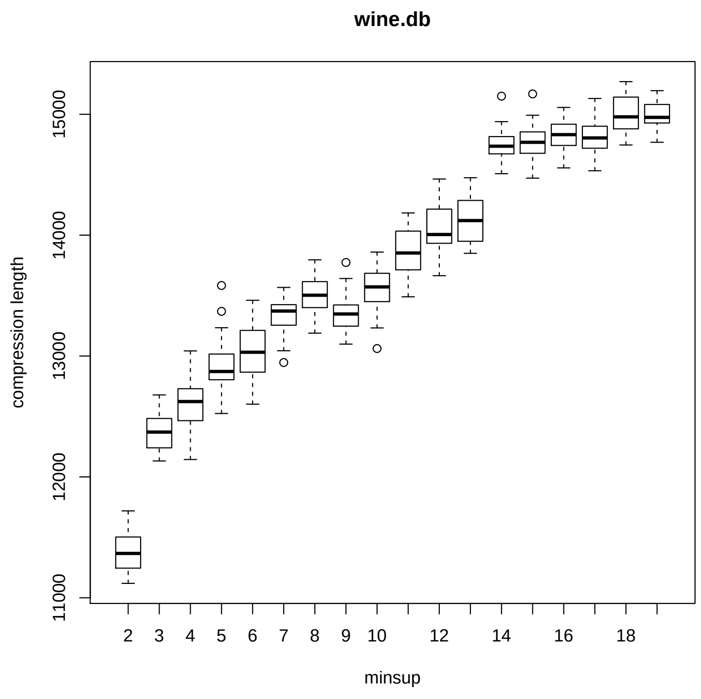

led7.db
compression length
121000
123000
125000
127000
2 3 4 5 6 7 8 9 10 12 14 16 18
minsup

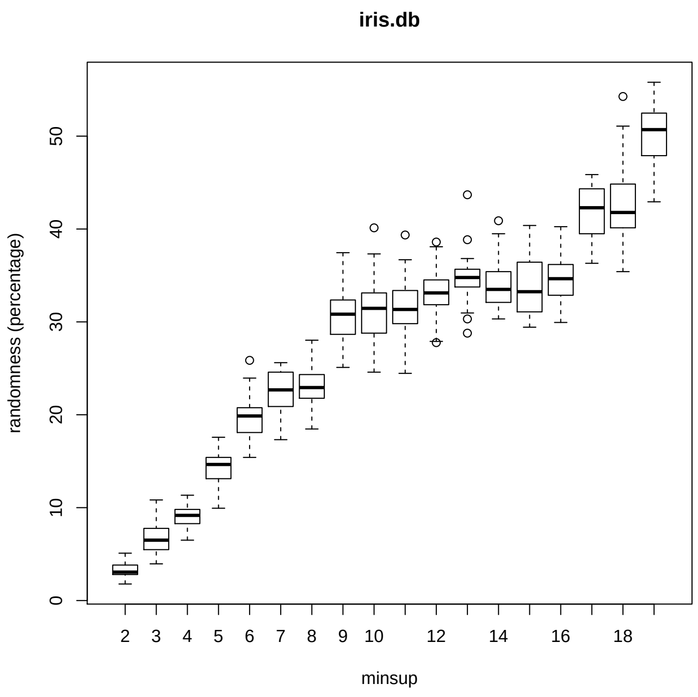

heart.db
compression length
18000
19000
20000
21000
22000
23000
24000
2 3 4 5 6 7 8 9 10 12 14 16 18
minsup

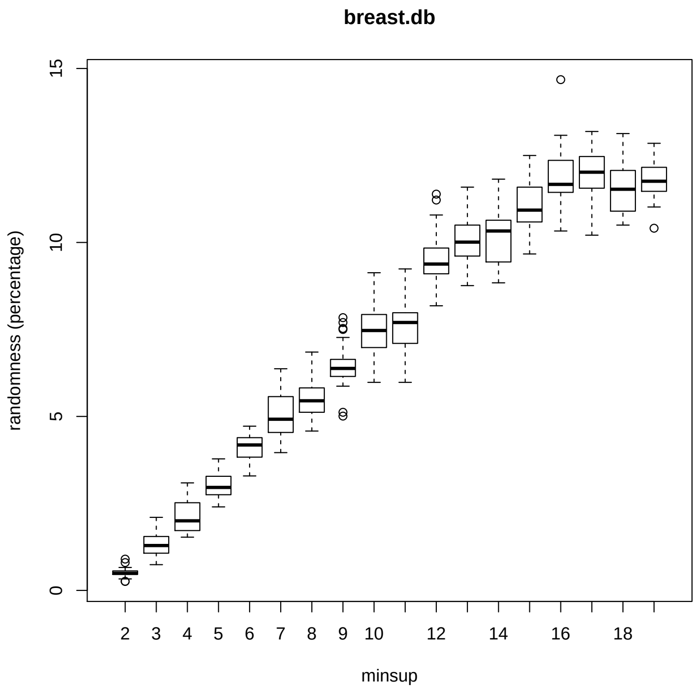

mushroom.db
compression length
minsup
1080000
1090000
1100000
1110000
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

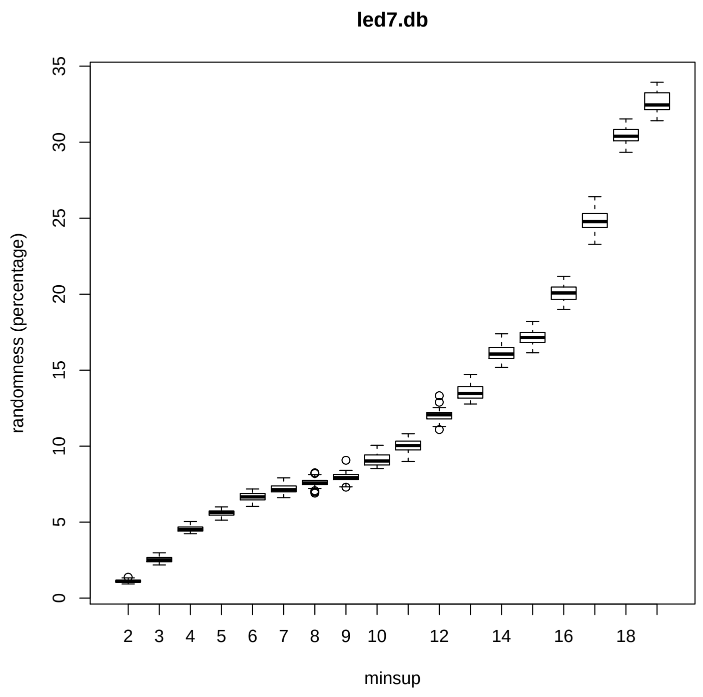

pima.db
compression length
minsup
24000
26000
28000
30000
2
3
4
5
6
7
8
9
10
12
14
16
18
20

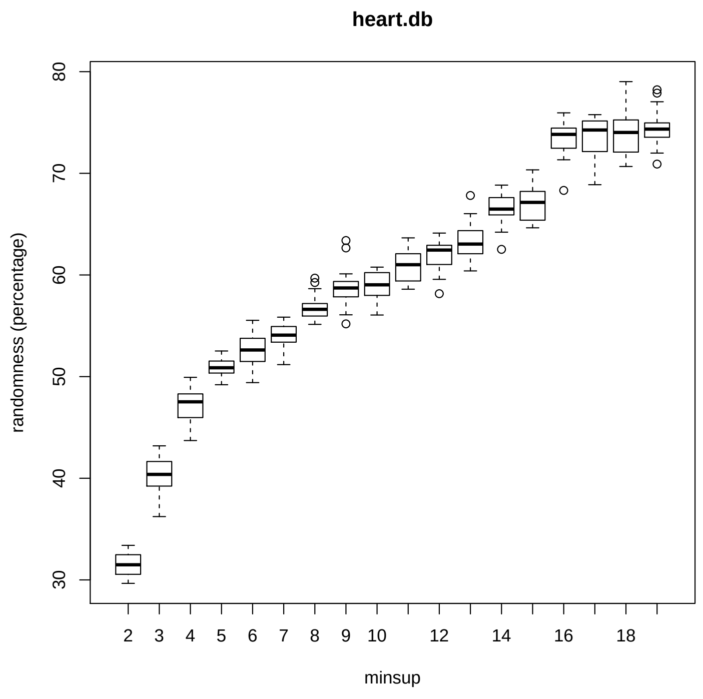

tictactoe.db
compression length
minsup
34000
34500
35000
35500
2
3
4
5
6
7
8
9
10
12
14
16
18
20

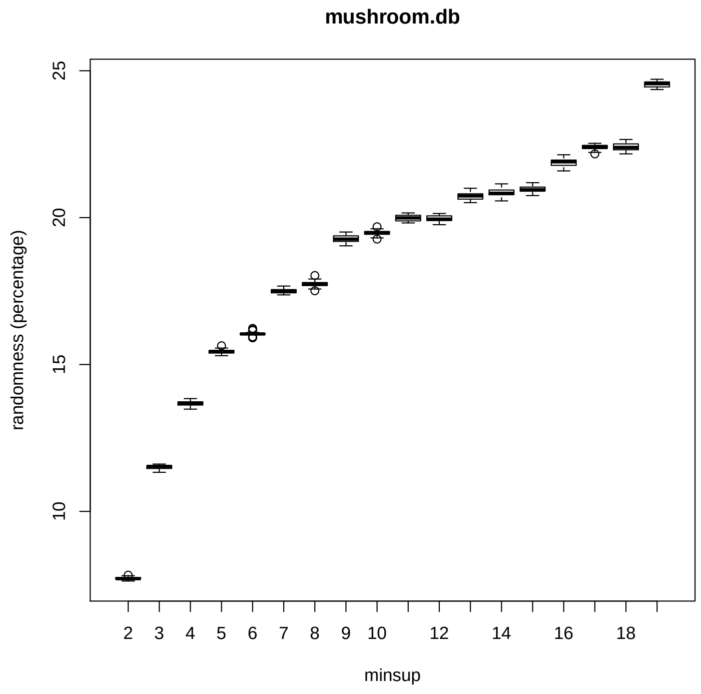

wine.db
compression length
minsup
11000
12000
13000
14000
15000
2
3
4
5
6
7
8
9
10
12
14
16
18

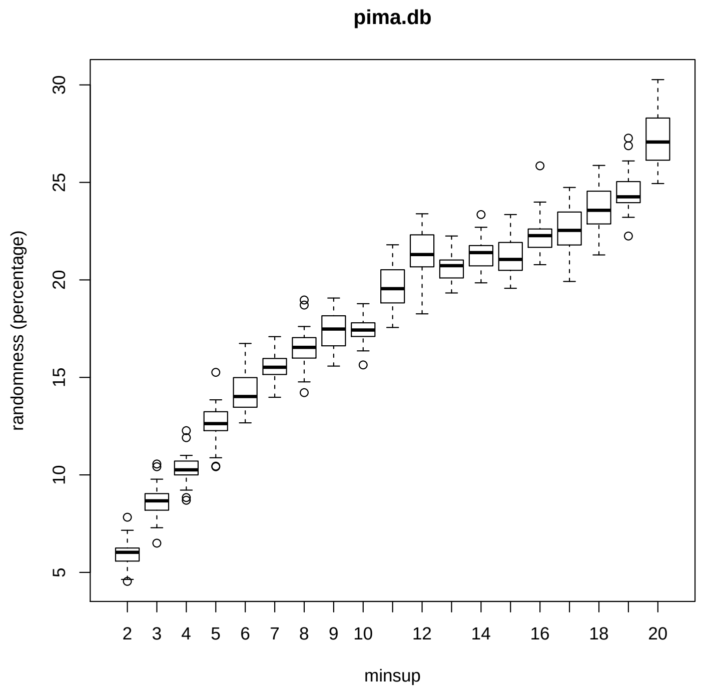

iris.db
randomness (percentage)
minsup
0
10
20
30
40
50
2
3
4
5
6
7
8
9
10
12
14
16
18

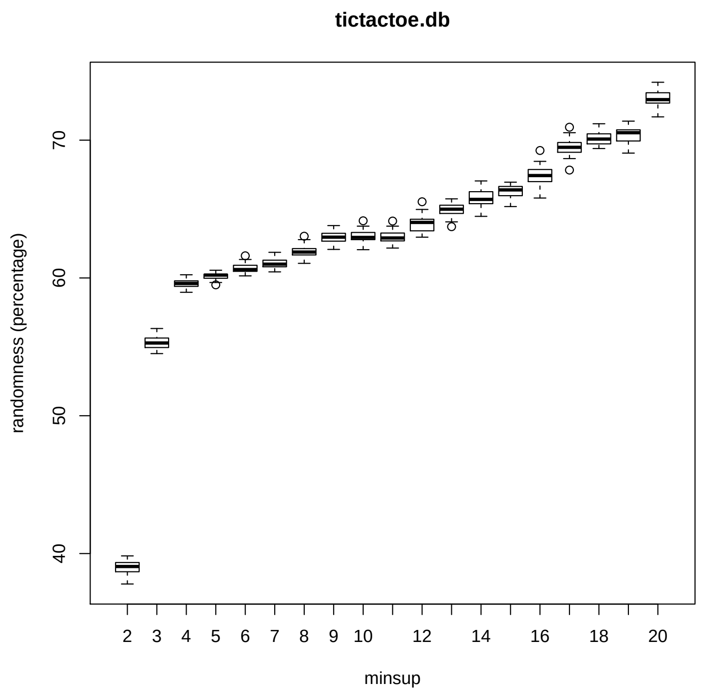

breast.db
randomness (percentage)
minsup
0
5
10
15
2
3
4
5
6
7
8
9
10
12
14
16
18

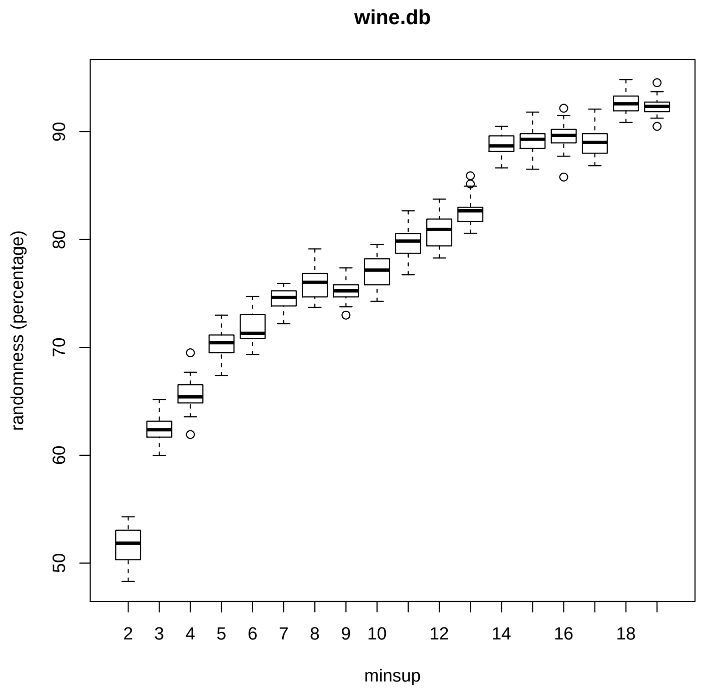

led7.db
randomness (percentage)
minsup
0
5
10
15
20
25
30
35
2
3
4
5
6
7
8
9
10
12
14
16
18


heart.db
randomness (percentage)
minsup
30
40
50
60
70
80
2
3
4
5
6
7
8
9
10
12
14
16
18


mushroom.db
randomness (percentage)
minsup
10
15
20
25
2 3 4 5 6 7 8 9 10 12 14 16 18


pima.db
randomness (percentage)
minsup
5
10
15
20
25
30
2 3 4 5 6 7 8 9 10 12 14 16 18 20


tictactoe.db
randomness (percentage)
40
50
60
70
minsup
2 3 4 5 6 7 8 9 10 12 14 16 18 20


wine.db
randomness (percentage)
50
60
70
80
90
minsup
2 3 4 5 6 7 8 9 10 12 14 16 18

Appendix B

Experiments

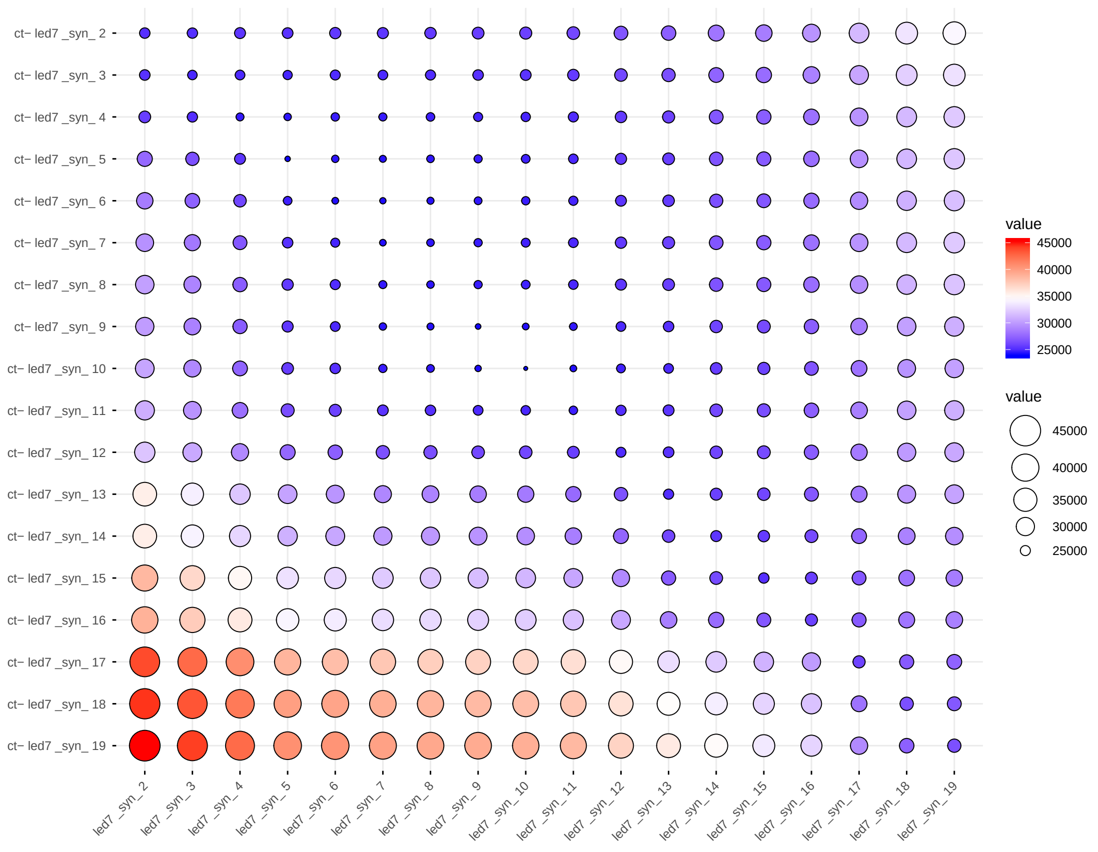


Figure B.1: Led7 data set: Pattern Length sorting method used for data generation.

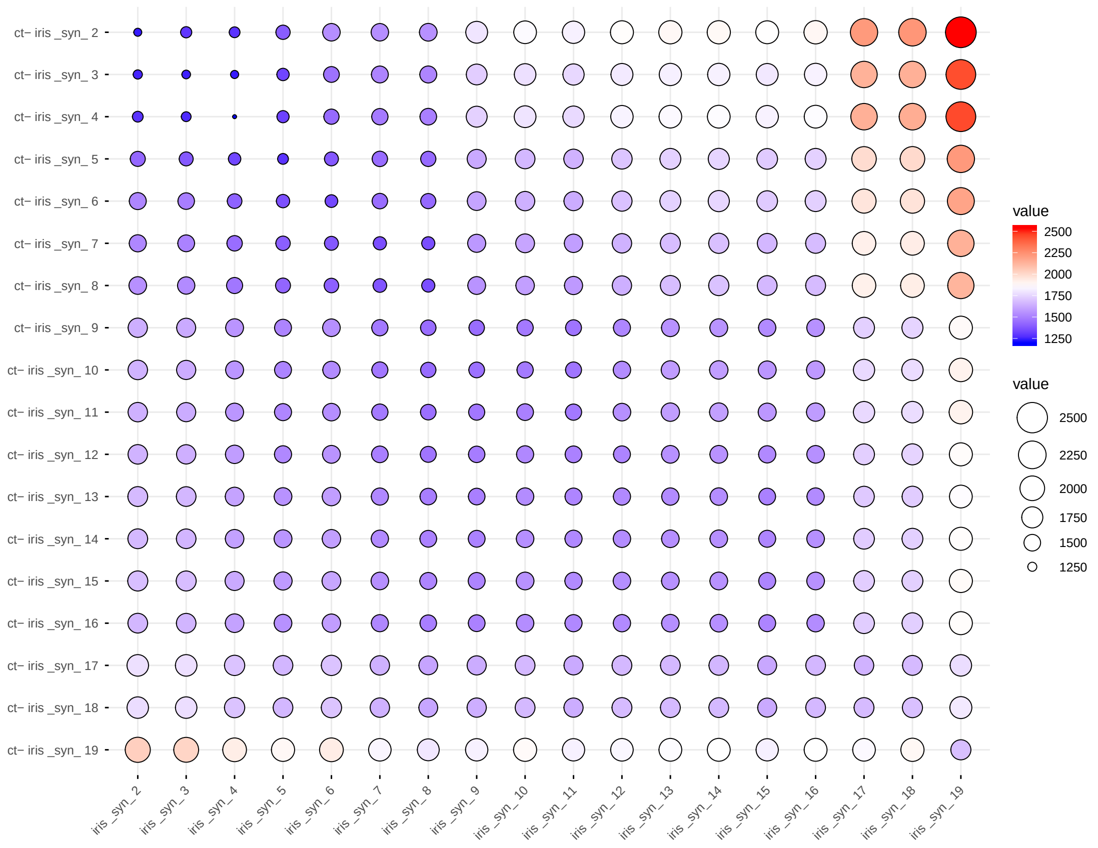


Figure B.2: Iris data set: Pattern Length sorting method used for data generation.

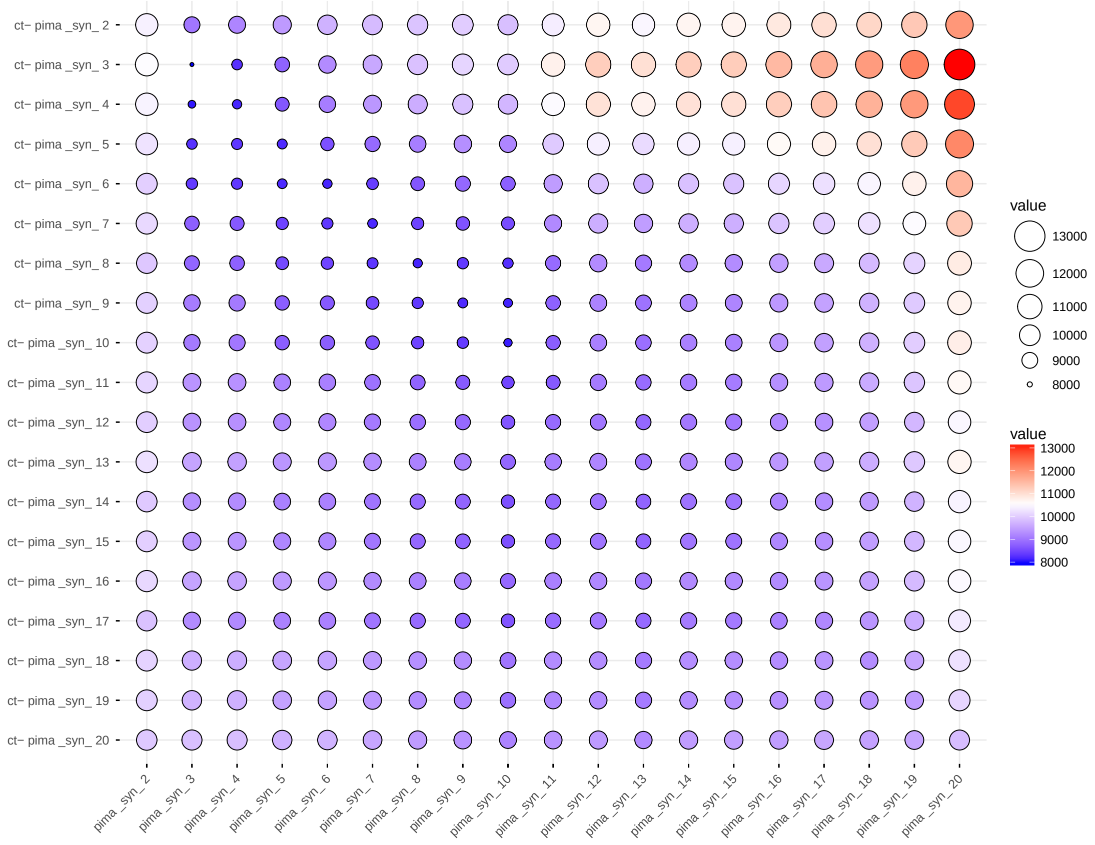


Figure B.3: Pima data set: Pattern Length sorting method used for data generation.

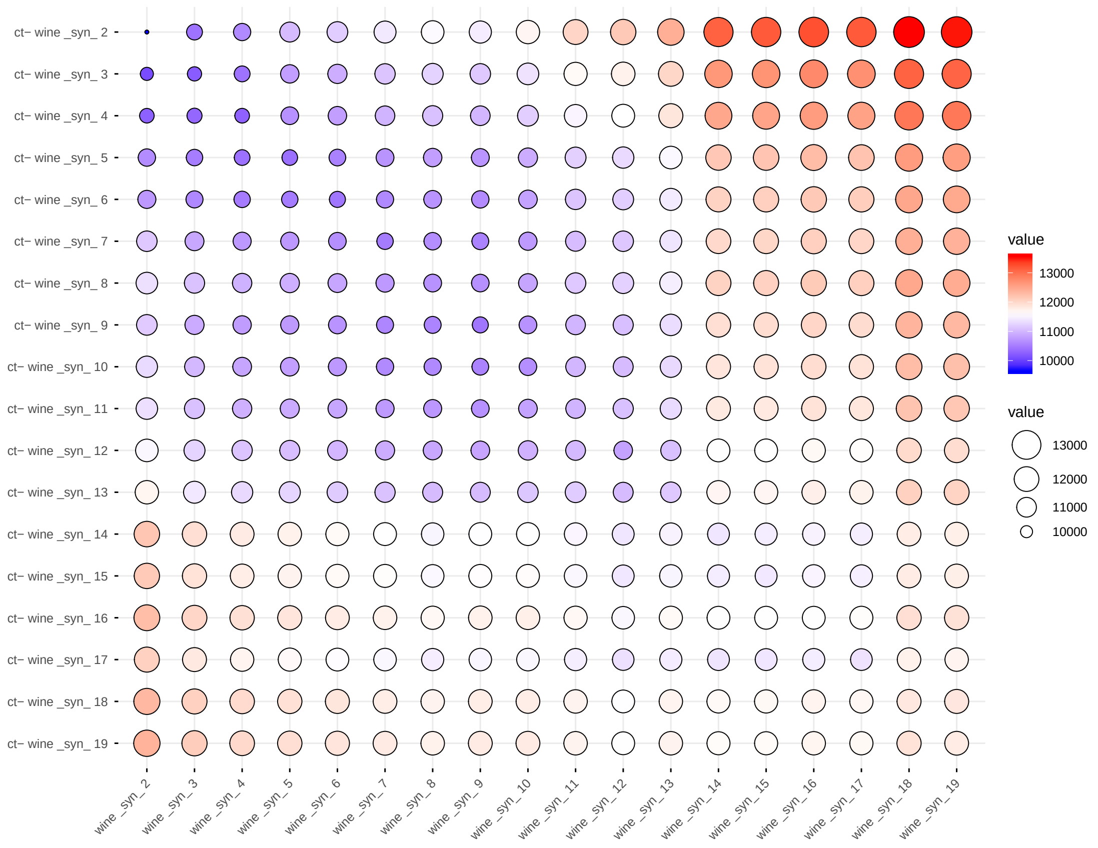


Figure B.4: Wine data set: Pattern Length sorting method used for data generation.

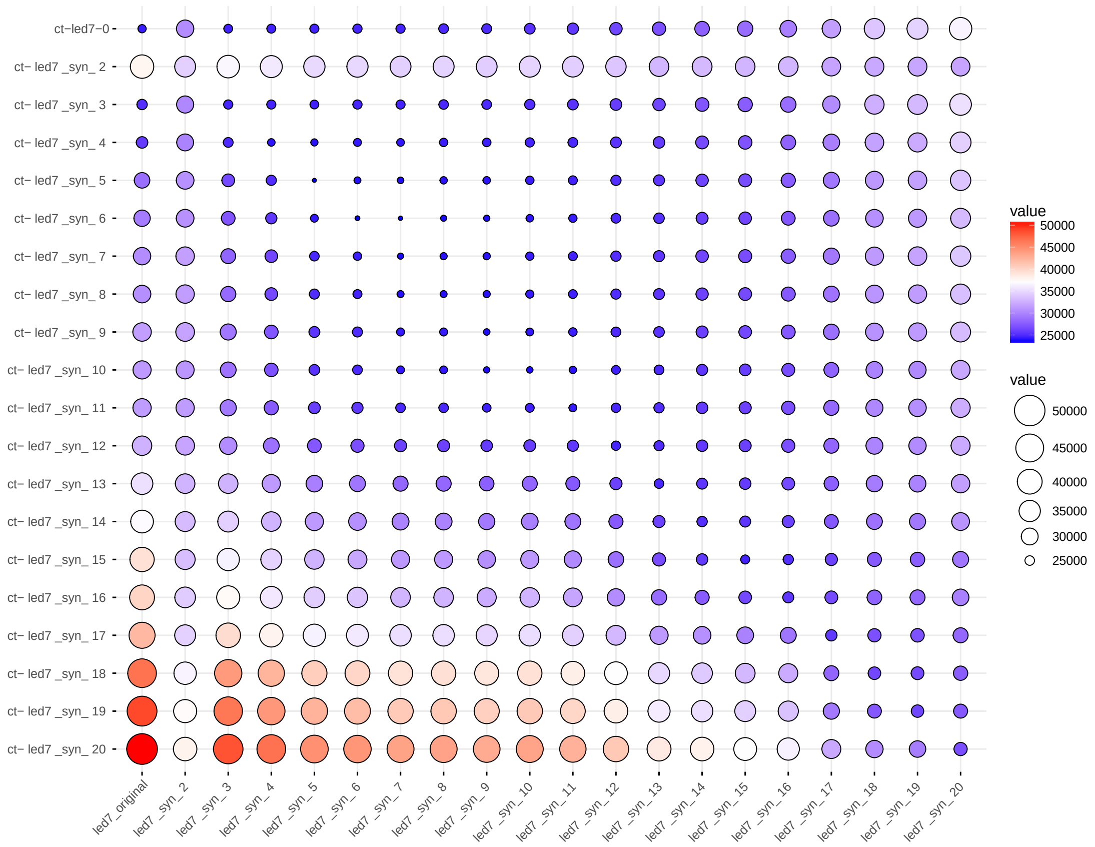


Figure B.5: Led7 data set: Code Length sorting method used for data generation.

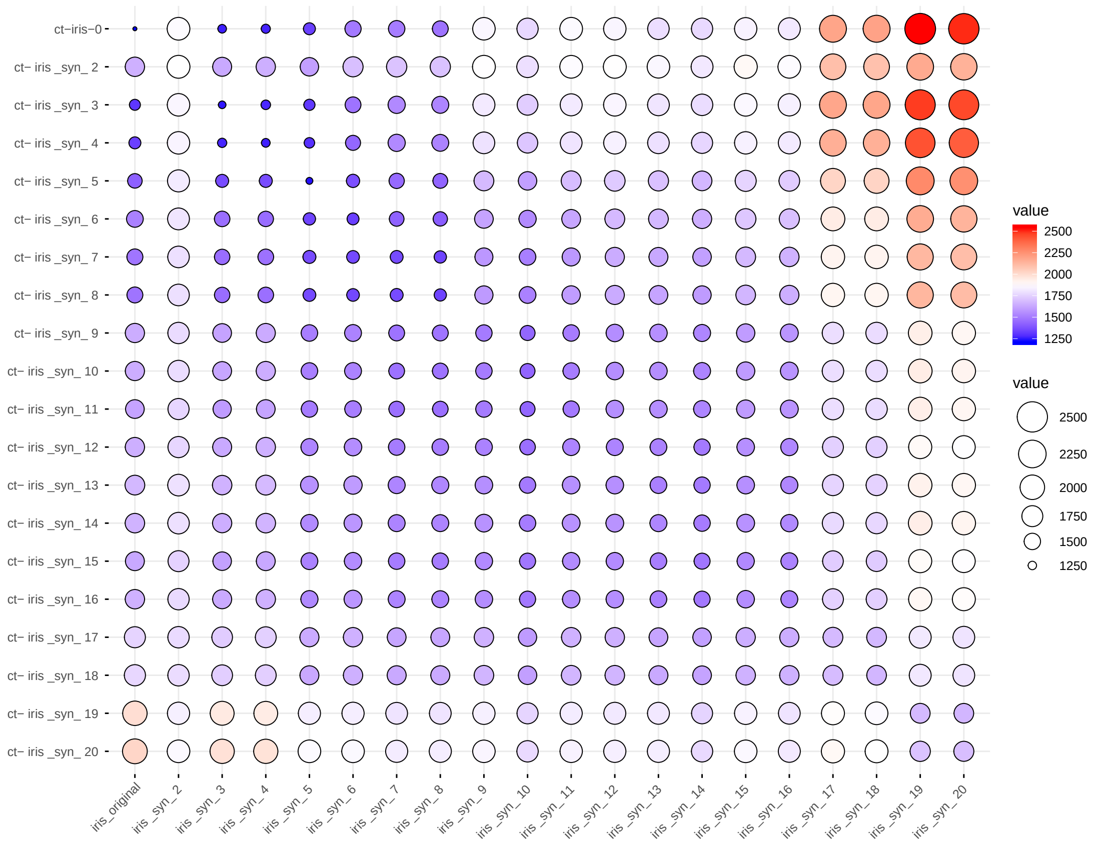


Figure B.6: Iris: Code Length sorting method used for data generation.

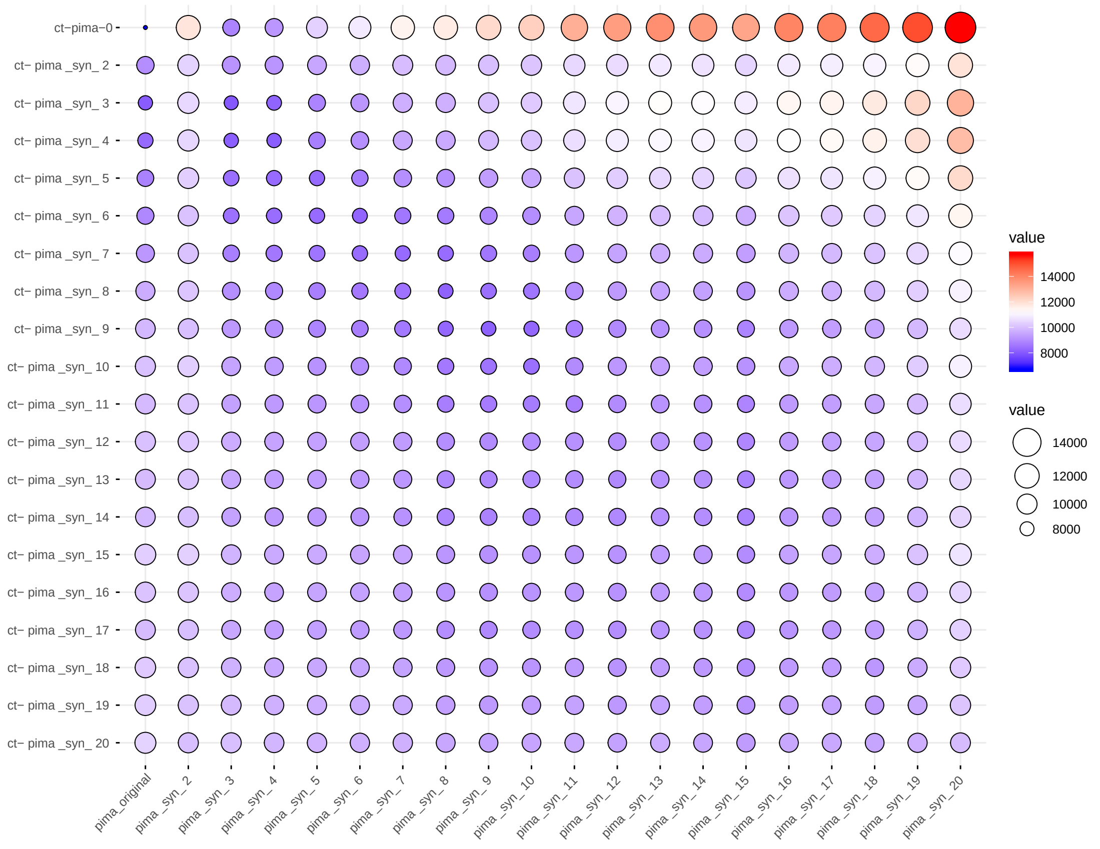


Figure B.7: Pima data set: Code Length sorting method used for data generation.

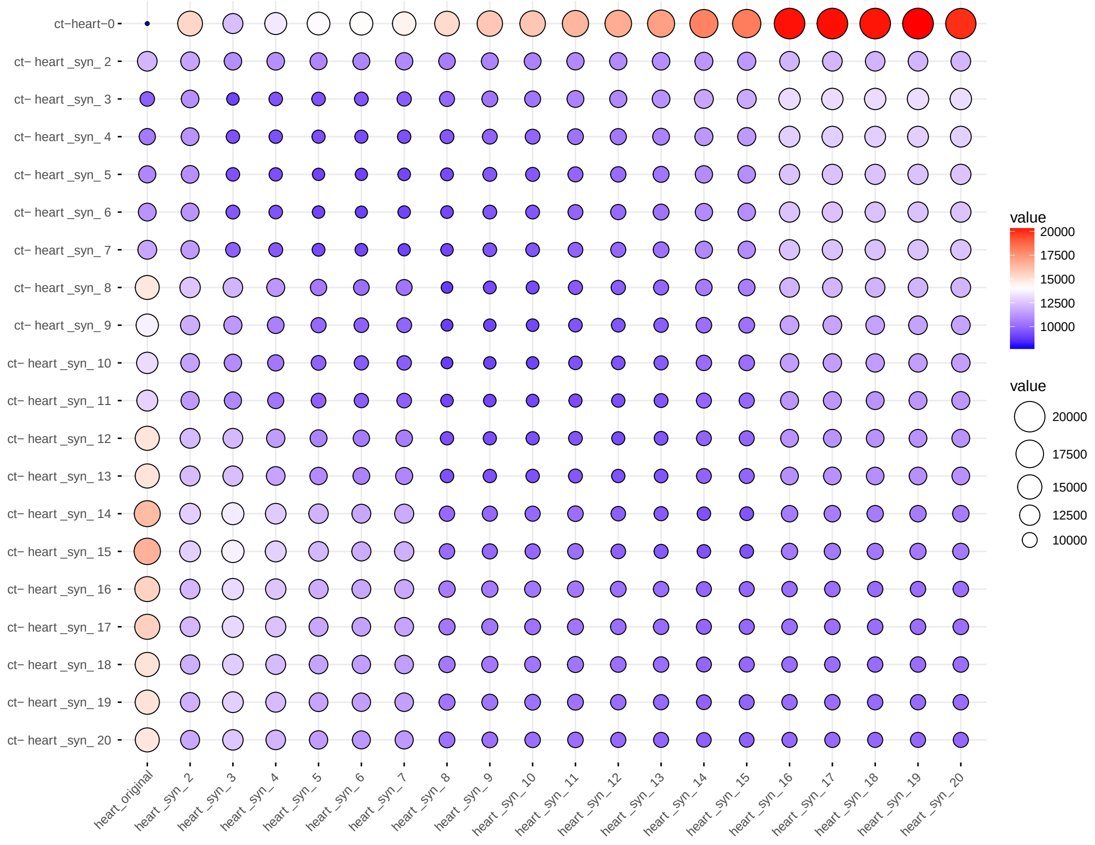


Figure B.8: Heart data set: Code Length sorting method used for data generation.

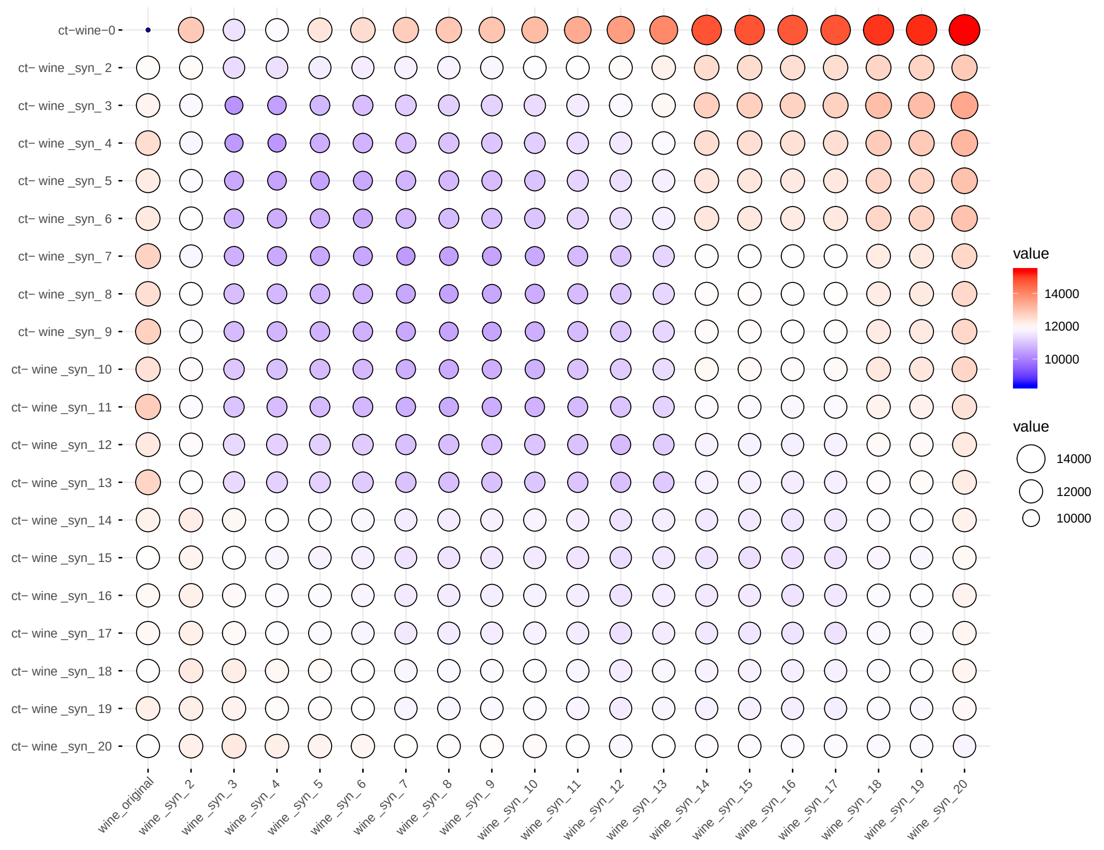


Figure B.9: Wine data set: Code Length sorting method used for data generation.

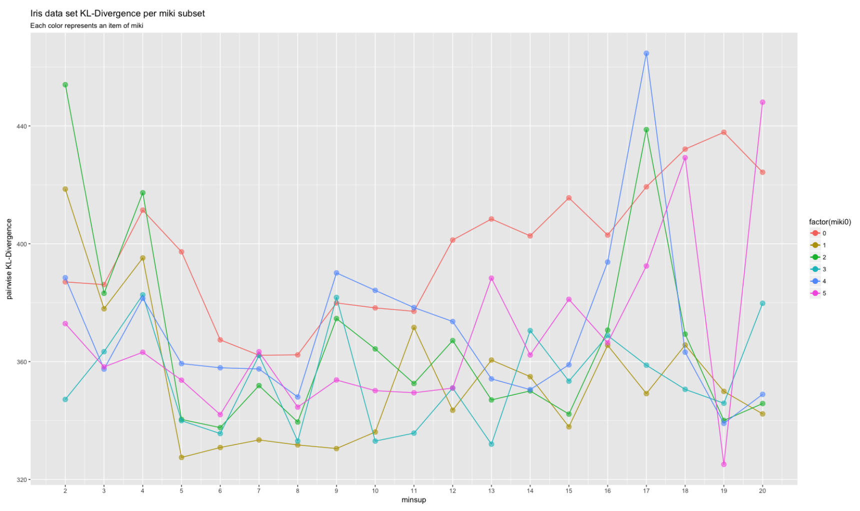


Figure B.10: Iris data set: Divergence of distributions per miki subset, representing learnability of the subset.

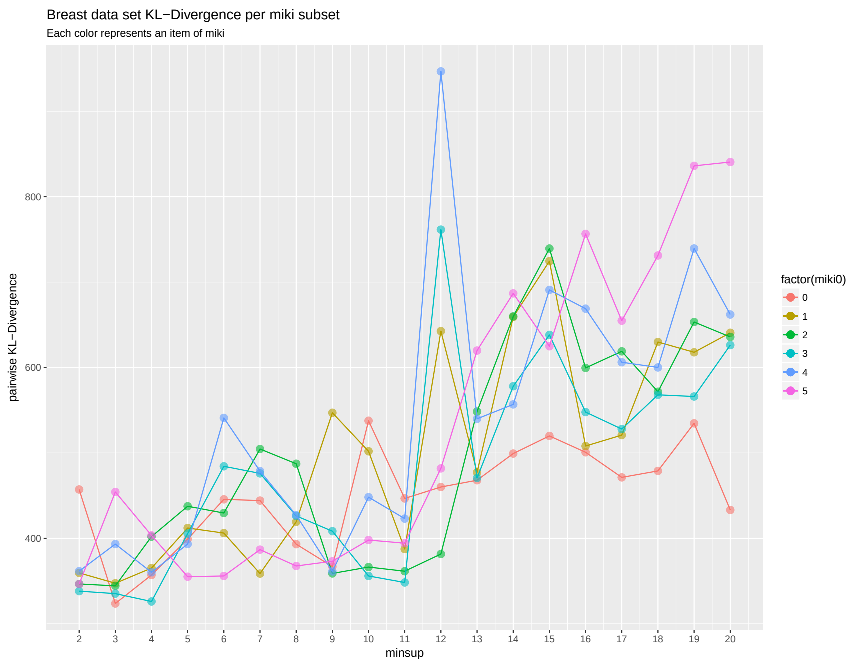


Figure B.11: Breast data set: Divergence of distributions per miki subset, representing learnability of the subset.

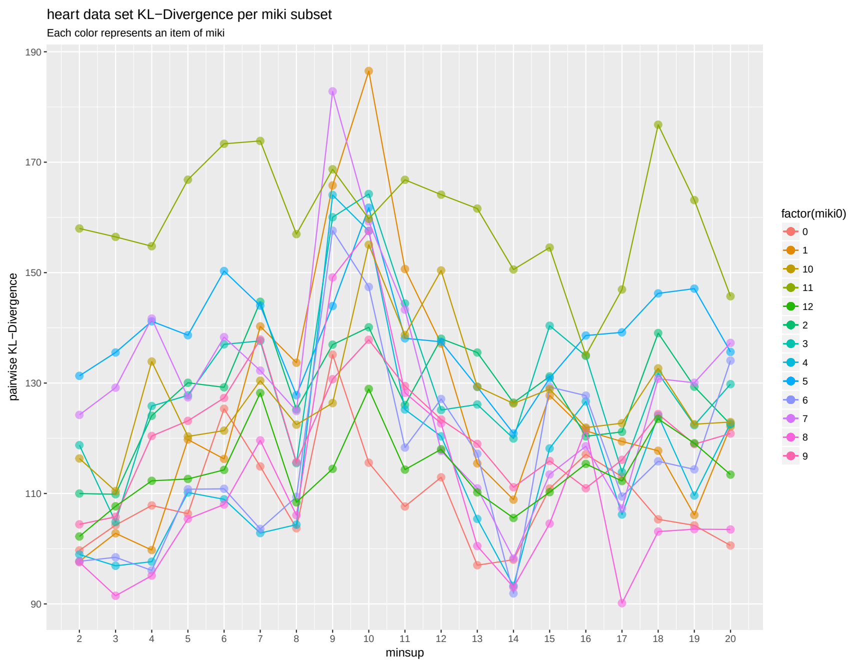


Figure B.12: Heart data set: Divergence of distributions per miki subset, representing learnability of the subset.

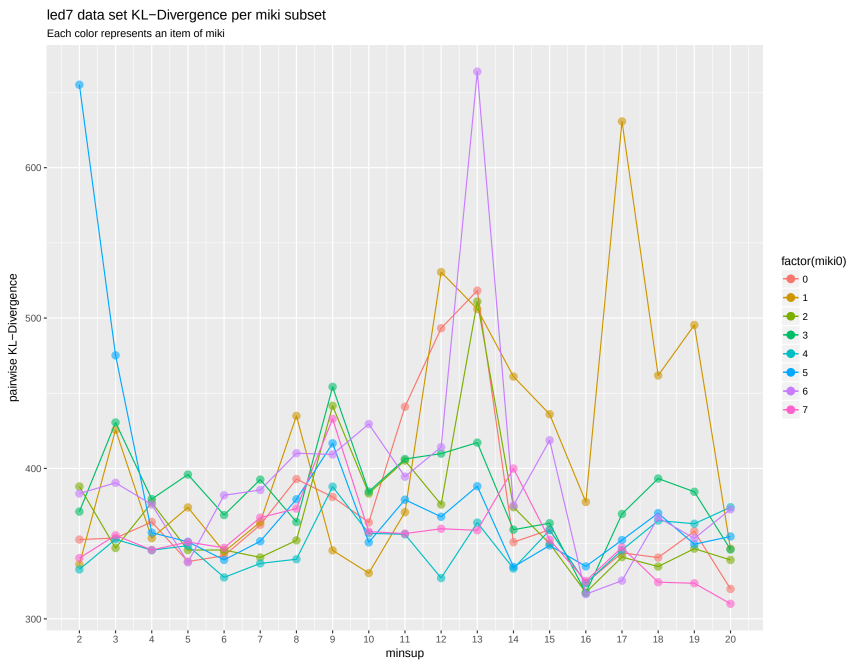


Figure B.13: Led7 data set: Divergence of distributions per miki subset, representing learnability of the subset.

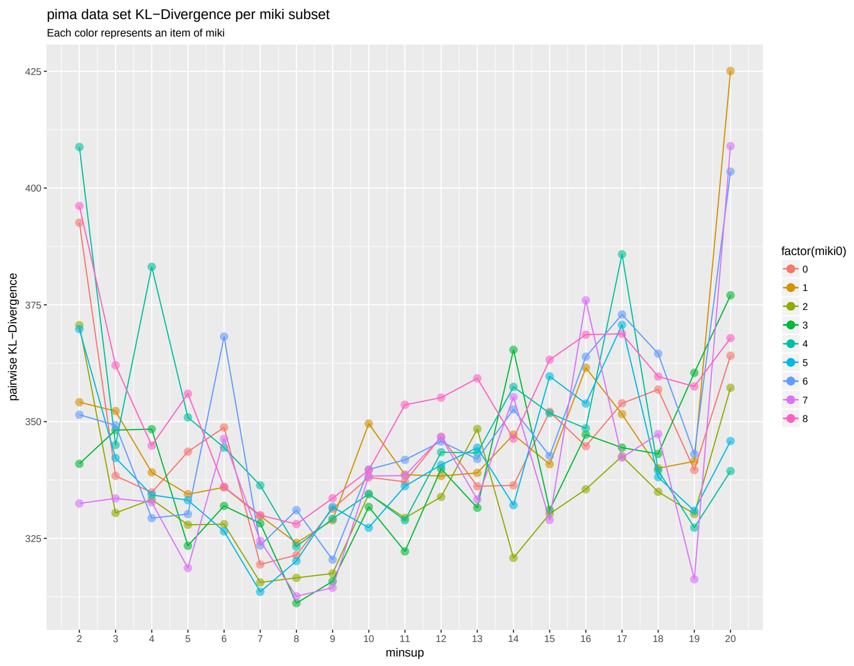


Figure B.14: Pima data set: Divergence of distributions per miki subset, representing learnability of the subset.

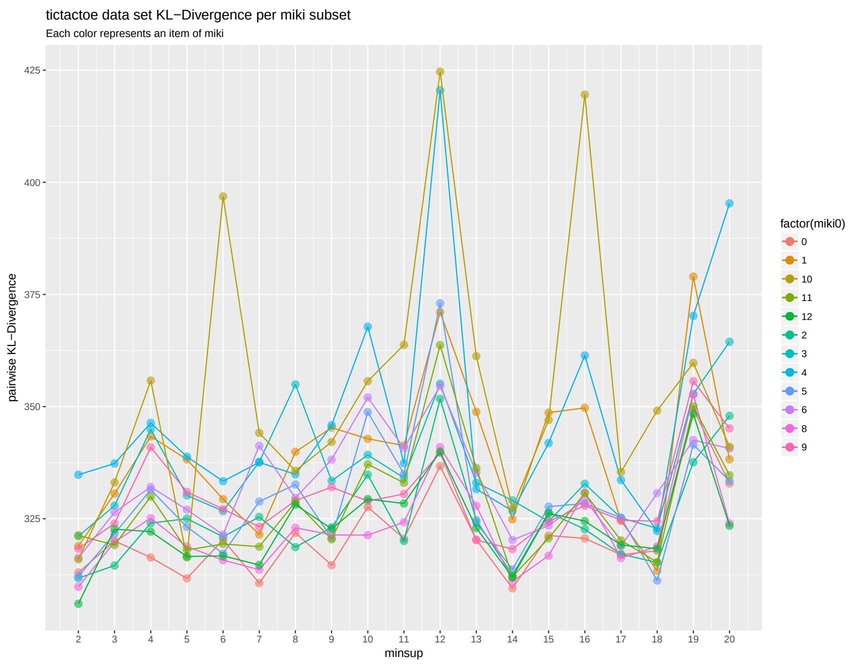


Figure B.15: Tictactoe data set: Divergence of distributions per miki subset, representing learnability of the subset.

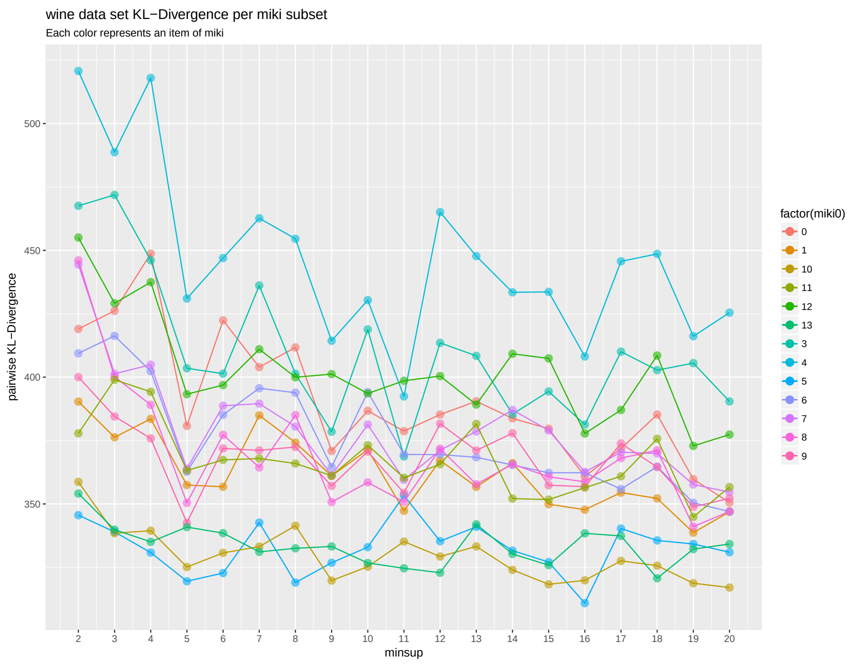


Figure B.16: Wine data set: Divergence of distributions per miki subset, representing learnability of the subset.

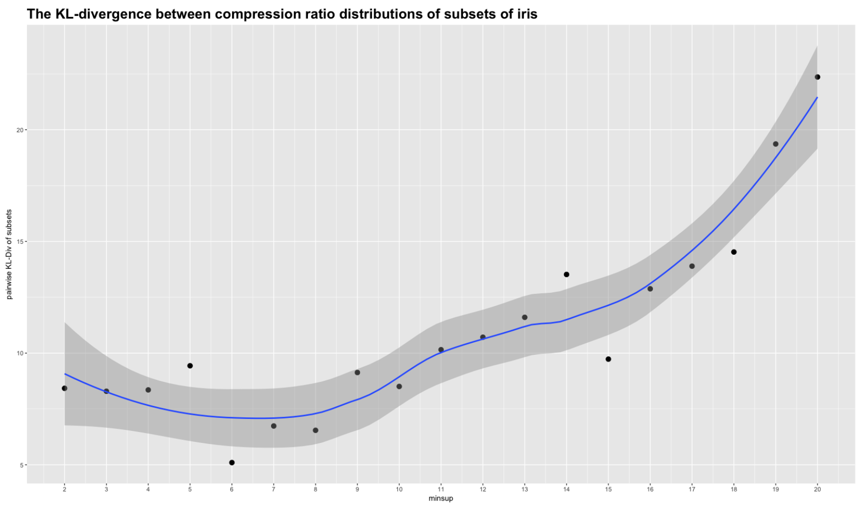


Figure B.17: Iris data set: Measure of evenness of learning among all subsets. The higher the divergence measure in the graph, the higher the chaos and uneven learning of the subsets.

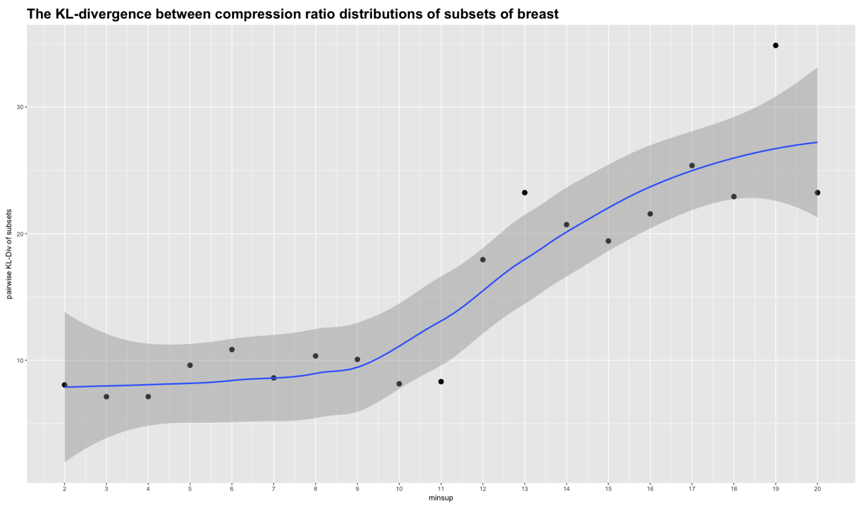


Figure B.18: Breast data set: Measure of evenness of learning among all subsets. The higher the divergence measure in the graph, the higher the chaos and uneven learning of the subsets.

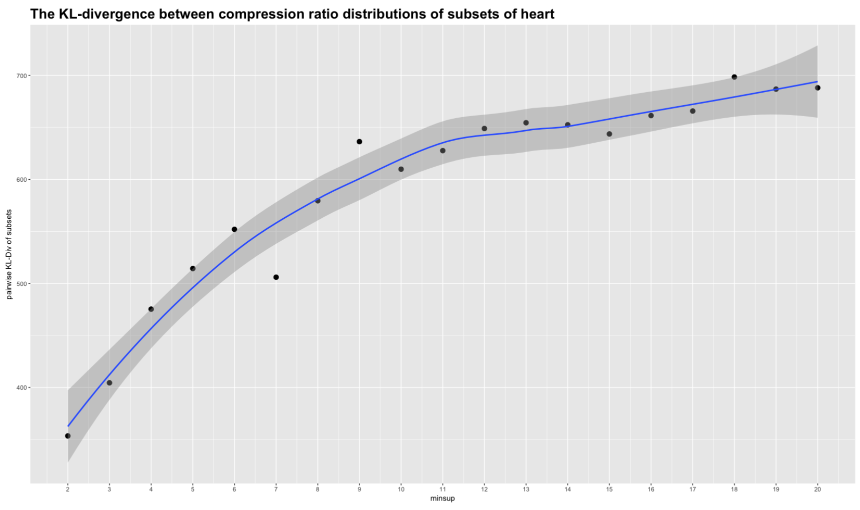


Figure B.19: Heart data set: Measure of evenness of learning among all subsets. The higher the divergence measure in the graph, the higher the chaos and uneven learning of the subsets.

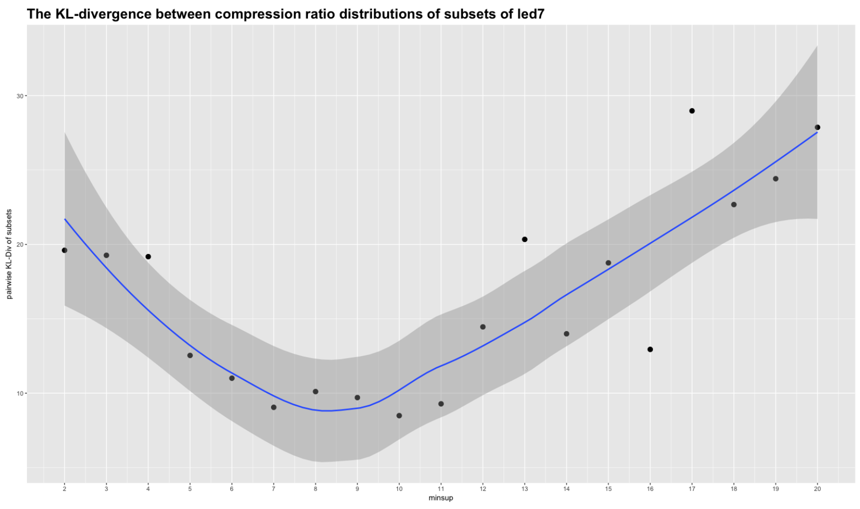


Figure B.20: Led7 data set: Measure of evenness of learning among all subsets. The higher the divergence measure in the graph, the higher the chaos and uneven learning of the subsets.

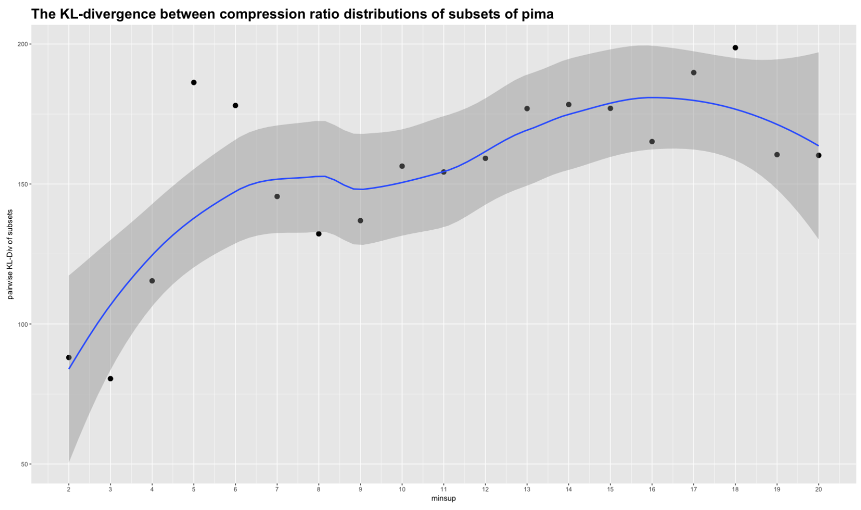


Figure B.21: Pima data set: Measure of evenness of learning among all subsets. The higher the divergence measure in the graph, the higher the chaos and uneven learning of the subsets.

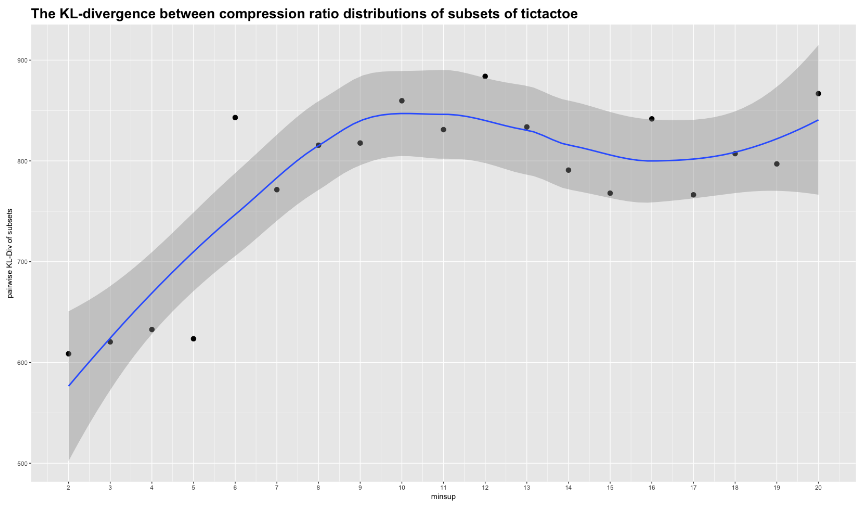


Figure B.22: Tictactoe data set: Measure of evenness of learning among all subsets. The higher the divergence measure in the graph, the higher the chaos and uneven learning of the subsets.

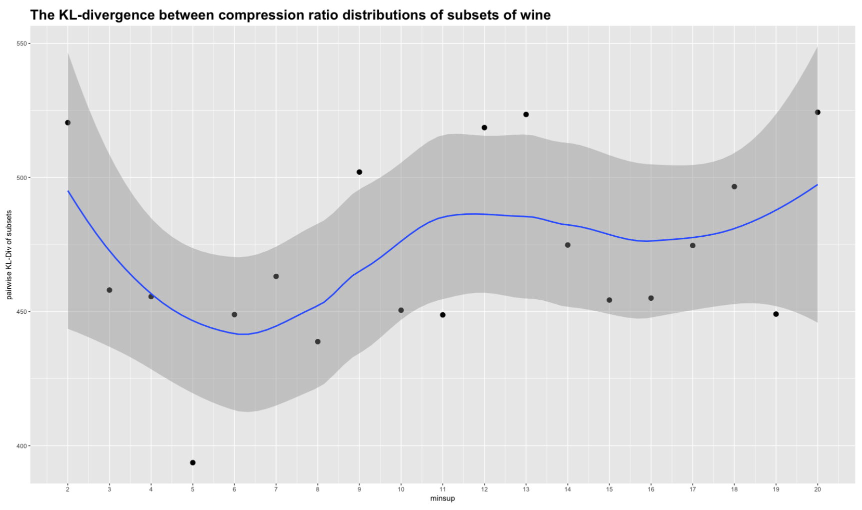


Figure B.23: Wine data set: Measure of evenness of learning among all subsets. The higher the divergence measure in the graph, the higher the chaos and uneven learning of the subsets.