RealKrimp — Finding Hyperintervals that Compress with MDL for Real-Valued Data

Jouke Witteveen[1], Wouter Duivesteijn[2], Arno Knobbe[3], and Peter Grünwald[4]

[1] ILLC, University of Amsterdam, The Netherlands
[2] Fakultät für Informatik, LS VIII, TU Dortmund, Germany
[3] LIACS, Leiden University, The Netherlands
[4] CWI and Leiden University, The Netherlands

Abstract. The MDL Principle (induction by compression) is applied with meticulous effort in the Krimp algorithm for the problem of itemset mining, where one seeks exceptionally frequent patterns in a binary dataset. As is the case with many algorithms in data mining, Krimp is not designed to cope with real-valued data, and it is not able to handle such data natively. Inspired by Krimp's success at using the MDL Principle in itemset mining, we develop RealKrimp: an MDL-based Krimp-inspired mining scheme that seeks exceptionally high-density patterns in a real-valued dataset. We review how to extend the underlying Kraft inequality, which relates probabilities to codelengths, to real-valued data. Based on this extension we introduce the RealKrimp algorithm: an efficient method to find hyperintervals that compress the real-valued dataset, without the need for pre-algorithm data discretization.

Keywords: Minimum Description Length, Information Theory, Real-Valued Data, RealKrimp.

1 Introduction

When data result from measurements made in the real world, they quite often are taken from a continuous, real-valued domain. This holds, for example, for meteorological measurements like temperature, precipitation, atmospheric pressure, etcetera. Similarly, the sensors in a smartphone (GPS, accelerometer, barometer, magnetometer, gyroscope, light sensor, etcetera) monitor data streams from a domain that is, for all practical purposes, real-valued. Most data mining algorithms, however, specialize in data from a discrete domain (binary, nominal), and can only handle real-valued data by discretization. Native support for real-valued data would be an asset to such algorithms.

An example of a popular discrete algorithm is Krimp [1], an algorithm that finds local patterns in the data. Specifically, Krimp seeks *frequent itemsets*: attributes that co-occur unusually often in the dataset. Krimp employs a mining scheme to heuristically find itemsets that *compress* the data well, gauged by a decoding function based on the Minimum Description Length Principle [2,3].

In an effort to extend the applicability of Krimp to continuous data, we introduce RealKrimp: a Krimp-inspired mining scheme with a strong foundation in

H. Blockeel et al. (Eds.): IDA 2014, LNCS 8819, pp. 368–379, 2014.

information theory, that finds interesting hyperintervals in real-valued datasets. This interestingness is expressed by an MDL-based model for compression in real-valued data. The resulting RealKrimp algorithm can be seen as a Krimp-inspired model for frequent patterns in continuous data, where the role of the frequent itemsets is played by hyperintervals in the continuous domains, that show an exceptionally high density.

2 Related Work

The Minimum Description Length (MDL) principle [2,3] can be seen as the more practical cousin of Kolmogorov complexity [4]. The main insight is that patterns in a dataset can be used to compress that dataset, and that this idea can be used to infer which patterns are particularly relevant in a dataset by gauging how well they compress: the authors of [1] summarize it by the slogan *Induction by Compression*. Many data mining problems can be practically solved by compression. Examples of this principle have been given for classification, clustering, distance function design (all in [5]), feature selection [6], finding temporally surprising itemsets [7], and defining a parameter-free distance measure on time series [8]. Clearly, the versatility of compression as a data mining tool has been recognized by the community. All the work done so far within the data mining community, however, has in common that the structure being compressed stems from a domain that is either finite [5,6,7] or at most countably infinite [5,8]. This is in sharp contrast with the use of MDL in statistics and machine learning, which has included continuous applications such as density estimation and regression from the very beginning [2]. The present paper provides a continuous-data MDL application in data mining.

In the data mining subtask of finding a small subset of dataset-describing patterns, arguably the most famous contribution is Krimp [1], as described in the introduction of this paper. An alternative and closely related approach to data summarizing is tiling [9]. Tiling seeks a group of, potentially overlapping, itemsets that together cover all the ones in a binary dataset. Similar as it may be to Krimp, tiling does not concern itself with model complexity or MDL. While Krimp approaches the binary dataset in an asymmetric fashion, only regarding the items that are present (the ones), two methods inspired by Krimp fill the void by approaching the dataset in a symmetric fashion. Pack [10] combines decision trees with a refined version of MDL, and typically selects more itemsets than Krimp. Conversely, LESS [11] sacrifices performance in terms of the involved compression ratio, in order to end up with a set of low-entropy patterns that is typically an order of magnitude smaller than the set found with Krimp.

3 Relating Codes and Probabilities

An important piece of mathematical background for the application of MDL in data mining, which is relevant for both Krimp and RealKrimp, is the *Kraft Inequality*, relating code lengths and probabilities (cf. [12]). In the following

sections, we inspect the inequality in its familiar form, where it is only applicable to at most countable spaces, and then show the derivation of a suitable code length function for Euclidean (hence uncountable) spaces.

Consider a *sample space*, i.e., a set of possible outcomes Ω. Let $\mathcal{E}$ be a partition of Ω that is finite or countably infinite. We think of $\mathcal{E}$ as the level of granularity at which data are observed. If Ω is countable, then in most MDL applications, $\mathcal{E} = \Omega$; but if $\Omega = \mathbb{R}^d$, then we will always receive data only up to a given precision determined by the data generating and processing system at hand; then $\mathcal{E}$ is some coarsening of $\mathbb{R}^d$; in practice, the modeler or miner may not know the details (such as the precision) of this coarsening but as we will see, the MDL principle can still be applied without such knowledge. A *probability mass function* p on a countable set $\mathcal{E}$ is simply a function $p : \mathcal{E} \to [0, 1]$ so that $\sum_{y \in \mathcal{E}} p(y) = 1$. We call p *defective* if, instead, $\sum_{y \in \mathcal{E}} p(y) < 1$.

Let $\mathcal{A}$ denote a finite alphabet, and let $\mathcal{C}$ denote a finite or countably infinite *prefix-free subset* of $\bigcup_{i \geq 0} \mathcal{A}^i$, i.e., a subset of the strings over $\mathcal{A}$ such that there exist no two elements z, z' of $\mathcal{C}$ such that z is a strict prefix of z'. A *description method* [3] for $\mathcal{E}$ with *code word set* $\mathcal{C}$ is defined implicitly by its *decoding function*, a surjection $D : \mathcal{C} \twoheadrightarrow \mathcal{E}$. While we allow that some $y \in \mathcal{E}$ can be encoded in more than one way, we do require unique decodability, so that the inverse function D from $\mathcal{C}$ to $\mathcal{E}$ does exist. Note that these requirements are standard in all applications of MDL [3]. We call a description method a (prefix) *code* if D is 1-to-1; a natural way to turn a given description method D into a code is to encode each $y \in \mathcal{E}$ by the *shortest* z with $D(z) = y$. The *length function* corresponding to code C, $\ell_C : \mathcal{E} \to \mathbb{Z}_{\geq 0}$, assigns to an outcome in $\mathcal{E}$ the length of its encoding under C.

Theorem 1 (Kraft). *For every code C over an alphabet $\mathcal{A}$, a (possibly defective) probability mass function p on $\mathcal{E}$ exists that makes short encoded lengths and high probabilities of outcomes correspond as follows:*

$$\text{for all } y \in \mathcal{E}\text{: } -\log_{|\mathcal{A}|} p(y) = \ell_C(y). \tag{1}$$

Proofs of this result exist [12] for the case when $\mathcal{C}$ is finite or countably infinite. One can also prove the converse of Theorem 1: for every probability mass function p on $\mathcal{E}$, there is a code C such that (1) holds. This allows a bi-directional *identification* of code length functions and probability mass functions [3]. Thus, as in most papers on MDL and Shannon information theory, we simply *define* codelength functions in terms of probability mass functions: every probability mass function p on $\mathcal{E}$ defines a code with for all $s \in \mathcal{C}$, lengths given by

$$\ell(s) = -\log p(s), \tag{2}$$

This is also the manner in which the relation between code lengths and probabilities is introduced in the KRIMP paper [1, Theorem 1].

What if outcomes are continuous? We start with the basic case with a sample space equal to $\mathbb{R}$. No code allows encoding data points $x \in \mathbb{R}$ with infinite precision, so one proceeds by encoding a discretization of x. We define the *uniform*

discretization $\mathcal{E}_k$ of $\mathbb{R}$ at level k as the partition $\{\ [n/2^k, (n+1)/2^k)\ \mid n \in \mathbb{Z}\}$ of $\mathbb{R}$. Every possible outcome x is a member of exactly one element of $\mathcal{E}_k$, denoted $[x]_k$.

Given an arbitrary distribution P on (a connected subset of) $\mathbb{R}$, identified by its density p, and a data point $x_0 \in \mathbb{R}$, the probability of $[x_0]_k$ is given by

$$P([x_0]_k) = \int_{x \in [x_0]_k} p(x)\mathrm{d}x \approx p(x_0)2^{-k} \tag{3}$$

As follows directly from the definition of (Riemann) integration, provided p is continuous, the approximation (3) gets better as $k \to \infty$. This makes it meaningful to define a length-like function, the *lengthiness*, on $\mathbb{R}$ by:

$$\ell^k(x) := -\log p(x) + k. \tag{4}$$

As k gets larger, (4) becomes a better approximation to the actual codelength $-\log P([x]_k)$ achieved at precision k. The lengthiness ℓ^k would only become a proper length function, i.e., one that satisfies Theorem 1, in the limit as k approaches infinity, where it would assign infinite length to all elements of X. However, crucially, the lengthiness does not alter its behavior with varying k, other than that it is shifted upwards or downwards. Hence, to *compare* elements of X, non-limit values for the lengthiness can be used as a length proxy.

To extend the idea above to encode data vectors $x = (a_1, \ldots, a_n)$ in $\mathbb{R}^n$ for some $n > 1$, we define $\mathcal{E}$ as a set of hyperrectangles of side width 2^{-k} and define $[x]_k$ to be the single hyperrectangle in $\mathcal{E}$ containing x. Approximating the integral over $[x]_k$ as in (3), we then get a lengthiness of

$$\ell^k(x) := -\log p(x) + n \cdot k. \tag{5}$$

We should note that one can formalize this discretization process in detail for general noncountable (rather than just Euclidean) measurable spaces and general types of discretization. (rather than just uniform; in practice our data may be discretizable in a different manner). This requires substantially more work but leads to exactly the same conclusions as to how to apply MDL to continuous-valued data; for details see [13].

4 Two-Part MDL Code for Hyperintervals

Given a set of candidate hypotheses $\mathcal{H}$ and data ω, the MDL Principle for hypothesis selection tells us to select, as best explanation for the data, the $H \in \mathcal{H}$ minimizing the two-stage description length

$$\ell_1(H) + \ell_2(\omega \mid H). \tag{6}$$

The first term, ℓ_1, is the codelength function corresponding to some code C_1 for encoding hypothesis H. For each $H \in \mathcal{H}$, the second term, $\ell_2(\cdot \mid H)$, is the

length function for a code $C_{2,H}$ for encoding the data 'with the help of H', i.e., a code such that, the better H fits ω, the smaller the codelength $\ell_2(\omega \mid H)$.

To find interesting patterns in an *uncountable* dataset ω, consisting of N *records* of the form $x = (a_1, \ldots, a_n)$, where each *attribute* a_i is taken from a real-valued domain, the data can be discretized at level k, turning (6) into

$$\ell_1(H) - \log p(\omega \mid H) + N \cdot n \cdot k, \tag{7}$$

where we approximate the actual codelength function by the 'lengthiness' (5) and the factor N appears because we discretize N data points.

In the original KRIMP paper, the resulting patterns are itemsets in finite-dimensional binary-valued space. In REALKRIMP, the patterns are bounded hyperrectangles, with edges parallel to coordinate axes. Unlike KRIMP, REALKRIMP does not demand any point to be covered by the hyperrectangle. Effectively we strive to find relevant endpoints of intervals of attributes. Hence, we refer to such hyperrectangles as *hyperintervals*:

Definition 1 (Hyperinterval). *Let* $\bar{\mathbb{R}} = [-\infty, +\infty]$ *represent the extended real numbers. Given a set of* $2n$ *extended reals* $h_1^L, h_1^U, h_2^L, h_2^U, \ldots, h_n^L, h_n^U$ *in* $\bar{\mathbb{R}}$*, the* hyperinterval $H \subseteq \mathbb{R}^n$ *encoded by the* $2n$*-tuple* $(h_1^L, h_1^U, h_2^L, h_2^U \ldots, h_n^L, h_n^U)$ *is the subset* $H = [h_1^L, h_1^U] \times \ldots \times [h_n^L, h_n^U]$ *of* $\mathbb{R}^n$ *in which the* i^{th} *dimension is restricted to* $[h_i^L, h_i^U]$.

Just as KRIMP strives to find itemsets that have a relatively high support, REALKRIMP should strive to find hyperintervals with a relatively high record density. We want to attain better compression, for increasing difference between density within a hyperinterval and density outside of the hyperinterval (signposted by the records in ω).

Description Length $\ell(\omega)$ of Data without Hypothesis Let M denote the volume of the smallest hyperinterval covering the entire dataset. Without prior information on the dataset, we do not want to discriminate between records a priori, so we assign the same length of $-\log 1/M$ to each record, using the code corresponding to a uniform distribution. That makes the complexity of the dataset equal to

$$N \cdot -\log 1/M = N \log M \tag{8}$$

where we can ignore the discretization constant $N \cdot n \cdot k$ since it needs to be added to both (8) and to (7), which are to be compared.

Description Length $\ell(\omega \mid H)$ of Data given Hypothesis Suppose that we are given a hyperinterval H lying within the interval of volume M; we denote the number of records it covers by N_{in} and its volume by M_{in}. Additionally, we write $N_{\text{out}} = N - N_{\text{in}}$ and $M_{\text{out}} = M - M_{\text{in}}$. Since $N_{\text{in}}, N_{\text{out}}, M_{\text{in}}$, and M_{out} are determined by H and here we assume H as given, we can base our code on these quantities. Each record x is now naturally equipped with the following length:

$$\ell(x) := \begin{cases} -\log 1/M_{\text{in}} \;=\; \log M_{\text{in}} & \text{for } x \in H \\ -\log 1/M_{\text{out}} \;=\; \log M_{\text{out}} & \text{for } x \notin H. \end{cases} \tag{9}$$

Note that we can code records x with these lengths only once we know, for each record x, whether $x \in H$ or not. Hence, to describe ω given H, we need to describe a binary vector $(b_1, \ldots, b_N)$ of length N where $b_i = 1$ if the i^{th} record is in H. The standard way of doing this is first to describe N_{in} using a uniform code on $\{0, 1, \ldots, N\}$, which takes $\log(N+1)$ bits irrespective of the value of N_{in}, adding another constant (independent of H) to the codelength that is irrelevant for comparisons and hence may be dropped; we then code $(b_1, \ldots, b_N)$ by giving its index in lexicographical order in the set of all N-length bit vectors with N_{in} 1s, which takes $\log \binom{N}{N_{\text{in}}}$ bits, which is itself equal, up to yet another constant term, to $N \cdot \text{Entropy}(N_{\text{in}}/N) = N \log N - N_{\text{in}} \log N_{\text{in}} - N_{\text{out}} \log N_{\text{out}}$ [3]. Combining with Equation (9), the complexity of the dataset is equal to:

$$N_{\text{in}} \log {}^{M_{\text{in}}}\!/_{N_{\text{in}}} + N_{\text{out}} \log {}^{M_{\text{out}}}\!/_{N_{\text{out}}} + N \log N \tag{10}$$

Description Length of Hypotheses I We gauge the complexity of specifying the model itself through specifying its boundaries. Consider an attribute a_i with minimal value L and maximal value U in ω. We may define a probability density function $\bar{p}$ on the maximal value of a_i within H as $\frac{x-L}{1/2(U-L)^2}$, a choice justified below. Given this maximal value u, we take a uniform probability density function on the minimal value of a_i within H, which has constant probability density $\frac{1}{u-L}$. Any combination of boundary values for a_i within H now has probability $\frac{x-L}{1/2(U-L)^2} \cdot \frac{1}{x-L} = \frac{2}{(U-L)^2}$, which is independent of the values themselves, thus justifying our choice for $\bar{p}$. Following this procedure for all attributes, the likelihood of every hyperinterval becomes ${}^{2^n}\!/_{M^2}$, which corresponds to the 'length':

$$-\log \bar{p}(H) = -\log {}^{2^n}\!/_{M^2} = 2 \log M - n \log 2 \tag{11}$$

While we could safely ignore the discretization constant when deriving the raw complexity of the dataset (8) and the complexity of the dataset given a model (10), we cannot do so for the codelength of the hyperinterval. This is because shortly, we will also look at hyperintervals that are defined only on a dimension $n' < n$. Hence we add a discretization constant twice (for the points describing the minimal and maximal values) to (11) to make it a proper 'lengthiness' function. To determine k, note that we should be more demanding towards the detail in the model as the number of records increases, so ideally (11) should increase with N when the discretization constant is taken into account. We take $-\log {}^{M}\!/_{N^n}$ for the discretization constant $n \cdot k$, turning (11) into:

$$2 \log M - n \log 2 - 2 \log \frac{M}{N^n} = 2n \log N - n \log 2 \tag{12}$$

This choice is quite natural, since it has an additional interpretation as the codelength arising from a rather different way of encoding H, namely by specifying, for each dimension $1 \leq i \leq n$, two records: one giving the lower boundary for attribute a_i in interval H, and one given the upper boundary.

Description Length of Hypotheses II: Unbounding Irrelevant Dimensions. When determining whether something exceptional is going on in a particular subset of the dataset, typically, only a sharply reduced subset of $n' \ll n$ attributes is relevant. We proceed to generalize the derived complexities and lengths, allowing RealKrimp to assess whether a dimension should be bounded or unbounded.

To gauge the informativeness of bounding a dimension, we turn to Equation (12). With our choice of discretization constant, the length for the model specification was derived as $2n \log N - n \log 2$. This was based on n dimensions, so the length per dimension, which we denote as Δ, is given by

$$\Delta = (2n \log N - n \log 2)/n = \log N^2/2$$

The quantity Δ represents the information contained in the specification of a single dimension of the hyperinterval. For a specification of $n' \leq n$ dimensions, the complexity of the model, as originally given in (12), becomes:

$$n'\Delta = n' \log \frac{N^2}{2}. \tag{13}$$

When encoding data based on such an n'-dimensional hyperinterval H with $n' < n$, the form of (10) remains intact, but we need to specify what N_{in} and M_{in} mean when some of the dimensions remain unbounded. We consider any hyperinterval to span the full range of any unbounded dimension. Hence, N_{in} is the number of records that are covered on the specified dimensions; coverage on the unbounded dimensions is implied. Also, M_{in} is calculated from a hyperinterval that, in the unbounded dimensions, spans the full range available in the dataset.

When encoding a hyperinterval H, we must encode which dimensions will be specified/unbounded. We do that by taking a uniform prior over the 2^n available models in the class. Hence, we obtain a constant complexity for each choice of specified/unbounded dimensions, making model comparison solely dependent on the lengths defined in those models. Therefore, a dimension is considered relevant, when specification of hyperinterval boundaries in that dimension delivers a reduction in description length bigger than Δ.

The hyperintervals that compress the dataset are those for which (8), the complexity of the database, is larger than the sum of (10), the complexity of the data given the hyperinterval, and (13), the complexity of specifying the hyperinterval. When this inequality holds, enough information is present in the hyperinterval to justify the cost of its specification, and we have found an underlying concept in the dataset. Subtracting $N \log N$ from both sides and rearranging terms, we find that we are interested in hyperintervals for which

$$N \log \frac{M}{N} - n' \log \frac{N^2}{2} \quad > \quad N_{\text{in}} \log \frac{M_{\text{in}}}{N_{\text{in}}} + N_{\text{out}} \log \frac{M_{\text{out}}}{N_{\text{out}}} \tag{14}$$

Here, everything that does not depend on the choice of hyperinterval is gathered on the left-hand side, leaving everything that does on the right-hand side.

Algorithm 1. The RealKrimp Algorithm

Input: A real-valued dataset ω
Output: Hyperintervals that compress ω well

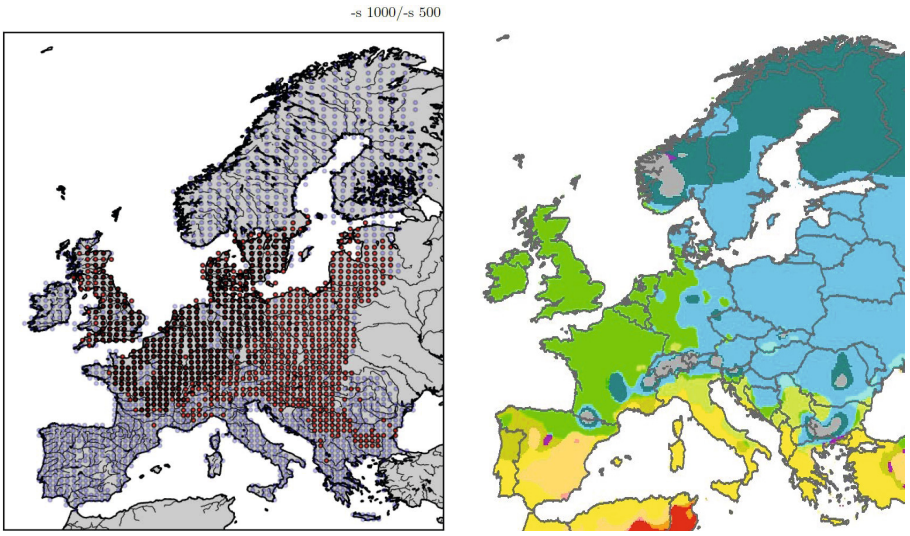
1: Sample the dataset.
2: Compute all (Euclidean) distances between records in the sample.
3: Pick two neighboring (in distance) rows in the sample.
4: Extend a hyperinterval H covering these rows based on other rows in the sample, in a compression-increasing direction (measuring compression on the entire dataset).
5: Calculate the coverage of each dimension in H.
6: In order of decreasing coverage, determine if compression improves when letting a dimension go unbounded.
7: Report the resulting hyperinterval if it is interesting according to Equation (14).
8: Until no more interesting hyperintervals can be found, restart from step 3 with two rows not covered by any of the previously reported hyperintervals.

5 The RealKrimp Algorithm

One of the most prominent problems in theory mining in general, is the *pattern explosion* problem: if we set the interestingness constraints tight, then we find only a few well-known patterns, but if we set the constraints looser, we are quickly overwhelmed with an amount of interesting patterns that is unsurveyable for any data miner. With the real-valued MDL criterion, we can also find many interesting hyperintervals very easily. When untreated, the pattern explosion problem hinders a practical application of a pattern mining method. Naturally, the Krimp paper [1] discusses the problem. Krimp strives to find a set of itemsets compressing the dataset, and obviously the candidate space is enormous. The chosen approach to the explosion, is to forego finding the *best* set of patterns — heuristically finding a pattern set that compresses *well* is good enough.

RealKrimp aims for a similar goal in an uncountable space, which amplifies the pattern explosion problem. For every hyperinterval that we could find, every boundary can have infinitely many values leading to the hyperinterval covering exactly the same records, with an arbitrarily small change in the volume of the hyperinterval. To deal with this serious problem in the applicability of real-valued MDL, in this section we introduce the RealKrimp algorithm: a mining scheme that confines its attention to those interesting hyperintervals that locally maximize the inequality of (14); no better compression is obtained by a hyperinterval that is either an extension or a restriction of the considered hyperinterval. The RealKrimp algorithm is given in Algorithm 1. Our implementation of RealKrimp is written in Python 3, and available for the general public at `http://github.com/joukewitteveen/hint`.

The first seven lines of the algorithm detail how a single hyperinterval can be found. Since we are interested in finding a set of well-compressing hyperintervals, the algorithm subsequently loops back to step 3 in an attempt to heuristically find additional compressing hyperintervals. The endurance with which the algorithm proceeds to attempt this is governed by a user-set parameter. Additionally, to gloss over small complexity bumps in the hyperinterval space, a perseverance



(a) The whole dataset (blue), a hyperinterval (red), and a sub-hyperinterval (dark red)

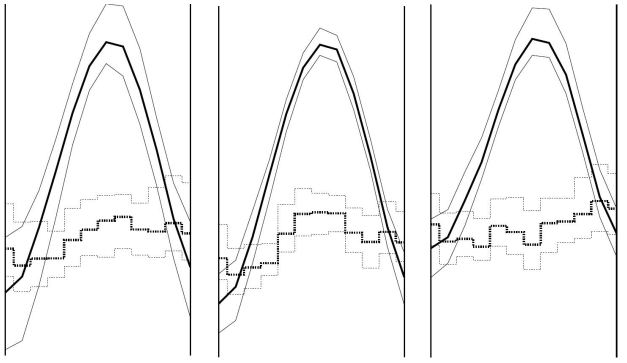
(b) Köppen classification of Europe [16]

Fig. 1. Spatial distribution over Europe of hyperintervals found by RealKrimp, juxtaposed with the Köppen classification of Europe

parameter can be set that allows the algorithm to escape local optima in step 4. Lastly, by varying the sample size employed in step 1, substantial influence can be exercised over the total runtime of the algorithm. To discuss all these details in full and properly incorporate them into the pseudocode would substantially bloat the discussion in this section, at the expense of either the theory in previous sections or the experimental results in the next section. Instead, we refer the reader who is interested in details on all individual steps to [13].

6 Experiment

We experiment on the *Mammals* dataset [14], which combines information from three domains: ① the location of grid cells covering Europe (latitude, longitude); ② the climate within these grid cells (monthly temperatures, precipitation, annual trends as captured by the BIOCLIM scheme [15]); ③ the presence or absence of species of mammals in the grid cells. The data from these three domains were pre-processed into one coherent flat-table dataset by Heikinheimo et al. [14]. This version of the dataset is the one we also use in our experiments. We feed the 19 BIOCLIM features from the second domain and all features from the third domain to the RealKrimp algorithm, to see its performance on a mixture



(a) Whole dataset (N=2221) (b) Hyperinterval (N=916) (c) Sub-hyperinterval (N=367)

Fig. 2. Climate data for the hyperintervals of Figure 1a. The solid lines represent the mean monthly temperature quartiles; the dashed lines represent the precipitation quartiles. Temperatures range from −5 to 20°C; precipitation ranges from 0 to 175mm.

of numeric and binary features. We withhold the location information and the other 48 climate features from the algorithm for evaluation purposes.

For the sake of the distance computation in line 2 of the RealKrimp algorithm, we need to define a way to handle the binary features from the third domain. Given two sets of present species, we assign a distance determined by the species in the symmetric difference between those sets. For each species in this symmetric difference, we add an amount to the distance equal to the binary entropy of that species in the dataset; if a species in the symmetric difference occurs k times in the dataset, it adds $-\frac{k}{N}\log\frac{k}{N} - \frac{N-k}{N}\log\frac{N-k}{N}$ to the distance. We consider this distance computation a parameter of the algorithm; treating binary attributes by computing its entropy is a convenient domain-agnostic way, but given particular domain information one might prefer other solutions.

A RealKrimp run resulted in many interesting hyperintervals, including the one depicted by the red dots in Figure 1a. Inspecting the boundaries of all 120 features that define the hyperinterval is infeasible; instead we make some observations that stand out. One bioclimatic variable is left unbounded: the mean diurnal range. The species that are necessarily present in the hyperinterval are the *Vulpes vulpes* (Red Fox), the *Capreolus capreolus* (European roe deer), and the *Lepus europaeus* (European hare). Applying the RealKrimp algorithm recursively, we find the sub-hyperinterval depicted by the dark red dots in Figure 1a. We inspect the relation between the whole dataset, the hyperinterval, and the sub-hyperinterval, by aggregating information from the 48 climate variables withheld from RealKrimp to draw up the *climographs* of Figure 2. These climographs can be used to illustrate the differences between groups in the Köppen climate classification [16].

Comparing the hyperinterval with the whole dataset in Figure 1a and considering the corresponding areas on the Köppen classification chart of Figure 1b, we observe that the hyperinterval removes the subarctic (Dfc and Dfd, teal), semi-arid (BSh, sand-colored), and Mediterranean (Csa, Csb, yellow) climate types from the dataset. Comparing the hyperinterval climograph (Figure 2b) with the one for the whole dataset (Figure 2a), we see that particularly the mean temperature inter-quartile range in summer and autumn is reduced.

Reducing the hyperinterval to the sub-hyperinterval, Figures 1a and 1b show that we remove the humid continental (Dfb, blue) climate type. Comparing the sub-hyperinterval climograph (Figure 2c) with both others, we see an increase in precipitation and temperature in the winter. This is consistent with the temperate oceanic (Cfb, green) climate type dominant in the sub-hyperinterval.

Due to space constraints we decided to not present more experimental results in this paper; doing so would be detrimental to either the development of the theory or to the relatively extensive discussion of the experimental results we do discuss in the paper, and we are willing to pay neither of these prices. However, more experiments were performed, and can be accessed elsewhere. More experimental results on artificial data can be found in the technical report [13]. Part of these experiments were performed on benevolent artificial data, whose underlying structure comes in an ideal form to be represented by hyperintervals, and part were performed on antagonistic artificial data, whose underlying structure is particularly problematic for RealKrimp. These artificial experiments illustrate what can be expected from RealKrimp when presented with a variety of patterns to discover. More experimental results on the Mammals dataset can be inspected online, at `http://www.math.leidenuniv.nl/~jwitteve/worldclim/`. The main page displays the map corresponding to found hyperintervals. Clicking on such a hyperinterval will display the results of a RealKrimp run mining for sub-hyperintervals.

7 Conclusions

We introduce RealKrimp: an algorithm that finds well-compressing hyperintervals in a real-valued dataset, based on the Minimum Description Length Principle. The hyperintervals are bounded hyperrectangles, with edges parallel to coordinate axes, and the interesting ones are those with a relatively high density of records in the dataset. In order to allow RealKrimp to search for compressing hyperintervals, the formal relation between codes and probabilities on Euclidean spaces is expressed by the *lengthiness*, a codelength-like function. We then discuss the MDL Principle for hypothesis selection and its application within RealKrimp, and describe a two-part MDL code for hyperintervals. The RealKrimp algorithm employs this code to heuristically mine for well-compressing hyperintervals. Hence, RealKrimp can be seen as a real-valued cousin of the well-known Krimp algorithm.

On the Mammals data, RealKrimp finds hyperintervals defined on BIOCLIM and zoogeographical attributes. Evaluation of the hyperintervals on withheld attributes shows that the found regions are spatially coherent, that they

correspond to climate types on the Köppen classification chart, and that they display meteorological behavior that is to be expected with these climate types. These observations provide evidence that RealKrimp finds hyperintervals representing real-life phenomena on real-life data from a real-valued domain.

Acknowledgments. This research is supported in part by the Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis", project C1. Tony Mitchell-Jones and the Societas Europaea Mammalogica kindly provided the European mammals data [17].

References

1. Vreeken, J., van Leeuwen, M., Siebes, A.: KRIMP: Mining Itemsets that Compress. Data Mining and Knowledge Discovery 23, 169–214 (2011)
2. Rissanen, J.: Modeling by Shortest Data Descriptions. Automatica 14(1), 465–471 (1978)
3. Grünwald, P.D.: The Minimum Description Length Principle. MIT Press, Cambridge (2007)
4. Li, M., Vitányi, P.: An Introduction to Kolmogorov Complexity and its Applications. Springer, New York (1993)
5. Faloutsos, C., Megalooikonomou, V.: On Data Mining, Compression and Kolmogorov Complexity. Data Mining and Knowledge Discovery 15(1), 3–20 (2007)
6. Pfahringer, B.: Compression-Based Feature Subset Selection. In: Proc. IJCAI Workshop on Data Engineering for Inductive Learning, pp. 109–119 (1995)
7. Chakrabarti, S., Sarawagi, S., Dom, B.: Mining Surprising Patterns Using Temporal Description Length. In: Proc. VLDB, pp. 606–617 (1998)
8. Keogh, E., Lonardi, S., Ratanamahatana, C.A.: Towards Parameter-Free Data Mining. In: Proc. KDD, pp. 206–215 (2004)
9. Geerts, F., Goethals, B., Mielikäinen, T.: Tiling Databases. In: Proc. DS, pp. 278–289 (2004)
10. Tatti, N., Vreeken, J.: Finding Good Itemsets by Packing Data. In: Proc. ICDM, pp. 588–597 (2008)
11. Heikinheimo, H., Vreeken, J., Siebes, A., Mannila, H.: Low-Entropy Set Selection. In: Proc. SDM, pp. 569–579 (2009)
12. Cover, T.M., Thomas, J.A.: Elements of Information Theory. Wiley (2006)
13. Witteveen, J.: Mining Hyperintervals – Getting to Grips With Real-Valued Data, Bachelor's thesis, Leiden University (2012)
14. Heikinheimo, H., Fortelius, M., Eronen, J., Manilla, H.: Biogeography of European land mammals shows environmentally distinct and spatially coherent clusters. Journal of Biogeography 34(6), 1053–1064 (2007)
15. Nix, H.A.: BIOCLIM — a Bioclimatic Analysis and Prediction System, research report, CSIRO Division of Water and Land Resources, pp. 59–60 (1986)
16. Peel, M.C., Finlayson, B.L., McMahon, T.A.: Updated World Map of the Köppen-Geiger Climate Classification. Hydrology and Earth System Sciences 11, 1633–1644 (2007)
17. Mitchell-Jones, T., et al.: The Atlas of European Mammals, Poyser natural history (1999)