Informative Summarization of Numeric Data

Michael Vollmer
Karlsruhe Institute of Technology
michael.vollmer@kit.edu

Lukasz Golab
University of Waterloo
lgolab@uwaterloo.ca

Klemens Böhm
Karlsruhe Institute of Technology
klemens.boehm@kit.edu

Divesh Srivastava
AT&T Labs – Research
divesh@research.att.com

ABSTRACT

We consider the following data summarization problem. We are given a dataset including ordinal or numeric explanatory attributes and an outcome attribute. We want to produce a summary of how the explanatory attributes affect the outcome attribute. The summary must be human-interpretable, concise, and informative in the sense that it can accurately approximate the distribution of the outcome attribute. We propose a solution that addresses the fundamental challenge of this problem–handling large numeric domains–and we experimentally show the effectiveness and efficiency of our approach on real datasets.

CCS CONCEPTS

• **Information systems** → **Summarization**; **Data mining**; **Multidimensional range search**; • **Human-centered computing** → Information visualization.

ACM Reference Format:
Michael Vollmer, Lukasz Golab, Klemens Böhm, and Divesh Srivastava. 2019. Informative Summarization of Numeric Data. In *31st International Conference on Scientific and Statistical Database Management (SSDBM '19), July 23–25, 2019, Santa Cruz, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3335783.3335797

1 INTRODUCTION

A common data analysis task is to determine how a set of explanatory attributes affects a class or an outcome attribute. However, before building complex models and making data-driven decisions, data scientists are often interested in exploring and summarizing the data. This motivates the problem of summarizing how the explanatory attributes affect an outcome attribute in a given dataset.

Prior work proposed the concept of *explanation tables* [7–9] to solve this problem in a concise, interpretable and informative way, with informativeness defined as the ability to capture the distribution of the outcome attribute. However, only discrete explanatory attributes were supported. In this paper, we argue that data summarization is particularly compelling and timely in the presence of ordinal and numeric explanatory attributes, representing, e.g., time, measured quantities or locations. Such attributes are common in machine-generated data produced by the Internet of Things and by smart infrastructure. To fill this gap, we propose a method for Lightweight Extraction of Numeric Summaries (LENS). Numeric domains can be very large, which makes summarization techniques critical, but also technically challenging. Addressing these challenges with LENS is the main contribution of this paper.

Example: Consider the `Occupancy` dataset [4], an excerpt of which is shown in Table 1. The dataset includes sensor measurements from a smart building (time, temperature, humidity, light level, carbon dioxide level) as well as an outcome attribute denoting whether a given room was occupied at the time. Suppose a data scientist wants to understand how the sensor measurements inform occupancy status. Table 2 shows the corresponding explanation table [7]. Each row is a pattern that represents a subset of the data matching the given values of the explanatory attributes, with "*" matching all values. Each pattern also includes the count of matching records and the fraction of records within this subset having a true outcome. The first row indicates that 23 percent of all records correspond to occupied rooms. The second row, chosen to provide the most additional information about the distribution of the outcome attribute, suggests that rooms are not occupied when the light level is zero. The next two rows similarly suggest that rooms are not occupied on Saturdays and Sundays. In contrast, LENS produces the summary shown in Table 3. The use of value ranges allows LENS summaries to capture patterns such as weekdays ("Mon-Fri") or evening hours ("15-23"), which would have to be pieced together from multiple rows of an explanation table. For example, the second row indicates that 4911 records, i.e., 23 percent of the data, correspond to occupied rooms on weekdays when the light level is high, which would require five separate rows without ranges. Subsequent rows identify additional subsets whose outcome distribution diverges from the expectation such as occupied rooms during evening hours with lights on (which, again, would require a separate pattern for each hour without ranges). The summary also contains a column specifying the outcome for each row to enable non-binary nominal outcomes, e.g., distinguishing between different persons occupying a room.

Challenges: Informative summarization for numeric and ordinal domains faces a technical challenge: the number of possible patterns is larger compared to nominal domains. In addition to constants and stars, as in Table 2, there are quadratically many possible ranges (with respect to the domain size) over each ordered attribute, as in Table 3. This problem is made worse by the large domain sizes that numeric attributes often have, and unfortunately, discretizing or binning the domains beforehand can miss semantically meaningful ranges such as Monday-Friday. Effectively addressing this challenge is the main contribution of this paper.

Table 1: A fragment of a building Occupancy dataset

Day	Hour	Temp.	Humid.	Light	CO_2	Occupied
Mon	14	23.7	26.27	585.2	749.2	True
Mon	18	22.39	25	0	805.5	False
Wed	10	23.39	25.6	738	1042	True
Thu	15	22.5	27	469	1063	True
Fri	02	21	25	0	440	False
Sun	13	20.5	28.7	265	426.7	False
Tue	11	22.2	27.6	535.7	1137	False
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 2: An explanation table for the Occupancy dataset

Day	Hour	Light	CO_2	Count	Occupied
*	*	*	*	20560	23.1%
*	*	0	*	12772	0.01%
Sun	*	*	*	2880	0.00%
Sat	*	*	*	2880	0.00%
Thu	13	*	*	122	13.85%
Thu	14	*	*	118	41.48%

Table 3: An informative summary for the Occupancy dataset created by LENS

Day	Hour	Light	Count	Occupied	Correct
*	*	*	20560	False	76.9%
Mon–Fri	*	356–1697	4911	True	96.27%
Mon–Tue	*	428–536	2025	True	99.41%
*	15–23	429–576	1522	False	0.33%
Thu–Fri	9	*	240	True	99.58%

After discussing related work in Section 2, we present the following contributions.

Conceptual Formalization. We present a framework for informative summaries in Section 3. It extends information theoretic foundations of explanation tables to ordered explanatory attributes and arbitrary (not only binary) nominal outcomes.

LENS Framework. Section 4 describes our method for pruning the pattern space. We leverage existing methods to identify informative patterns with only stars and constants, and then we "grow" the constants into intervals of exponentially increasing lengths. This allows LENS to scale well without discretizing numeric domains beforehand, as many interval mining techniques do [11, 16, 23].

Speeding up LENS. Measuring informativeness is computationally costly, even for the moderate numbers of candidate patterns considered by LENS. Section 4.3 presents a *Sparse Cumulative Cube* that stores a small set of useful aggregates for computing the information content of the patterns considered by LENS.

Experimental Evaluation. In Section 5, we use several real-world datasets to compare LENS against methods from related fields in terms of informativeness and runtime. We also explore the parameter sensitivity of LENS and scalability with the number of rows and columns. Our results show that LENS outperforms competing approaches and its informativeness has little dependence on parameter choices.

Table 4: Subgroups in the Occupancy dataset

Day	Hour	Temp.	Humid.	Light	CO_2
Mon–Fri	*	*	*	389–1697	*
Mon–Fri	*	*	*	390–1697	*
Mon–Sat	*	*	*	362–1697	450–2076
Mon–Sat	*	*	*	389–1697	*
Mon–Sat	*	*	*	362–1697	*

Finally, Section 6 concludes the paper with directions for future work.

2 RELATED WORK

In this section, we discuss related work that studied the influence of explanatory attributes on an outcome attribute with a focus on human interpretability. We defer a discussion of explanation tables to the next section as part of our formalization.

Subgroup Discovery. These methods find subsets of data whose distribution of outcomes diverges from the overall distribution [1, 12, 13, 24]. This is related to our summarization problem since interesting subgroups may correspond to interesting relationships between the explanatory attributes and the outcome attribute. However, each subgroup is individually assigned an interestingness score without considering other subgroups. This leads to subgroups such as those in Table 4, which were produced by the Diverse Subgroup Set Discover algorithm [23] for the Occupancy data. These subgroups are interesting on their own but redundant (they all mainly describe weekdays with high illumination) and therefore not informative as a whole. In Section 5, we compare LENS to two state-of-the-art subgroup discovery techniques that support ordinal and numeric explanatory attributes.

The first approach, *MergeSD* [11], introduces pruning techniques to reduce the number of candidate subgroups. The interestingness measure used for subgroups is the difference in likelihood of a true outcome between the overall data and the subgroup, weighted by subgroup size. They show an upper bound for this measure under "refinement" of subgroups. That is, for each subgroup, characterized as a pattern, they determine the maximum interestingness for any pattern matching a subset of this subgroup. This upper bound is then used in a depth-first search to prune the pattern space. Unfortunately, such a bound is not useful for informativeness because small groups can still be informative if the distribution of their outcomes is very different from the overall distribution. As a result, an analogous bound for informativeness would be very loose, even for small groups, and not suitable for pruning.

The second method is *Diverse Subgroup Set Discovery* (DSSD) [23], which aims to find non-redundant subgroups. DSSD uses a three phase approach to ensure subgroup diversity. First, it uses a greedy best-first search that maintains a fixed number of candidates to find a large set of relevant subsets. Next, DSSD diversifies the candidate set by pruning dominated subgroups. Here, a subgroup dominates another subgroup if it is formed by a subset of the conditions and has higher interestingness. Finally, the best subgroups from this

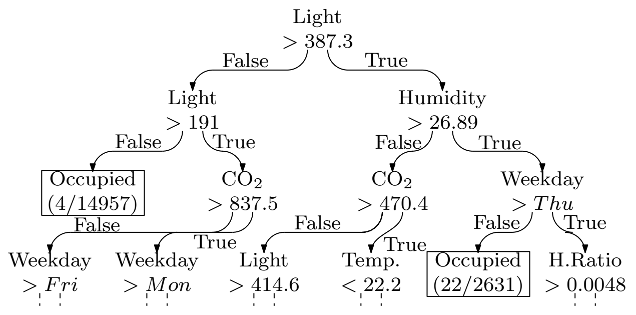


Figure 1: Fragment of a decision tree for the Occupancy dataset

diversified set are selected. However, this approach may still select similar subgroups, as seen in Table 4, if they are not strictly dominated by others, e.g., through slightly different conditions or higher interestingness.

Classification. Instead of summarizing the relationship between a class attribute and the remaining attributes, classification uses these relationships to predict the class. As a result, classifiers are judged by their accuracy on data outside the initial "training" set instead of their ability to summarize existing data. Additionally, human interpretability is often sacrificed for accuracy through feature engineering, ensembles or neural networks. Rule based classifiers such as decision trees [20] are an exception as human-interpretable classification methods that partition the data into non-overlapping sets. Interpretable Decision Sets [17] are another exception, which allow a minimal amount of overlap to reduce the number of decision boundaries. However, these approaches are not concise, even after applying optimizations such as tree or rule pruning. For example, Figure 1 shows an excerpt of a decision tree created by the YaDT implementation [21] with default parameters for the Occupancy dataset. It is not concise: there are over 50 leaves, making it difficult to observe any patterns. In Section 5, we will experimentally compare LENS with YaDT.

3 FUNDAMENTALS AND FORMALIZATION

We now formalize informative summaries as a generalization of explanation tables [7–9] to ordered attributes, patterns with intervals, and multi-valued outcomes. We then present our problem statement and a greedy framework for building informative summaries. Table 5 summarizes the notation used in this paper.

3.1 Informative Summaries

Let $A_1, \ldots, A_d$ be a set of d explanatory attributes with data space $\mathcal{A} = A_1 \times \cdots \times A_d$ and let O be an *outcome* attribute. For each attribute domain $A_i = \{a_i^1, \ldots, a_i^{|A_i|}\}$, if A_i is ordered, we assume that a_i^j is before a_i^k in the ordering if and only if $j < k$. We assume finite domains, i.e., we restrict the domains of real values to the values present in the data. A dataset is a multiset of tuples over $\mathcal{A} \times O$. We represent it as a function $f : \mathcal{A} \times O \longrightarrow \mathbb{N}$ that counts the number of tuples for specific combinations of attribute and outcome values. The total number of tuples in the dataset is $n = \sum_{t \in \mathcal{A}} \sum_{o \in O} f(t, o)$.

Table 5: Notation

Symbol	Meaning
A_i	Explanatory attribute
d	Number of explanatory attributes
$\mathcal{A}$	Data space formed by $A_1 \times \cdots \times A_d$
t	A tuple of explanatory attributes ($t \in \mathcal{A}$)
O	Outcome attribute
$o \in O$	An outcome value
$f(t, o)$	A dataset represented by tuple counts
$f(o)$	Count of tuples with outcome o
$f(t)$	Count of tuples t disregarding outcome
n	Total number of tuples
p	A pattern
$supp(p)$	Number of tuples matching p
$cnt(p, o)$	Number of tuples matching p with outcome o
$r(p, o)$	A rule formed by a pattern p and outcome o
S	An informative summary as set of rules
$P(o\|t)$	Probability for a tuple t to have outcome o
$P_S(o\|t)$	Maximum entropy estimate of $P(o\|t)$ by S
$g_S(p)$	Maximal gain by including any rule with p

For brevity, we overload f to sum over the omitted parameter, that is, $f(o) = \sum_{t \in \mathcal{A}} f(t, o)$ and $f(t) = \sum_{o \in O} f(t, o)$.

Patterns are compact specifications of subsets of $\mathcal{A}$. For each attribute A_i, a pattern p specifies a closed interval $[p_i, q_i]$ with $p_i, q_i \in \{1, \ldots, |A_i|\}$. Note that for attribute A_i without ordering, the only meaningful intervals are single values, $[p_i, p_i]$, and all values, $[1, |A_i|]$. A tuple $t = (t_1, \ldots, t_d) \in \mathcal{A}$ *matches* a pattern $p = ([p_1, q_1], \ldots, [p_d, q_d])$ if $a_i^{p_i} \leq t_i \leq a_i^{q_i}$ for all $i \in \{1, \ldots, d\}$. We also write $t \asymp p$ for "t matches p". Next, the *support* of p is the count of matching tuples regardless of their outcome: $supp(p) = \sum_{t \in \mathcal{A} \asymp p} f(t)$. Additionally, for each outcome value $o \in O$, $cnt(p, o) = \sum_{t \in \mathcal{A} \asymp p} f(t, o)$ is the count of matching tuples with this outcome. A rule $r(p, o)$ is a combination of a pattern p, outcome o, pattern support $supp(p)$ and outcome percentage $cnt(p, o)/supp(p)$. An informative summary S is a set of such rules.

We use the following notation for patterns. Intervals $[p_i, q_i]$ that cover all values, i.e., $p_i = 1$ and $q_i = |A_i|$, are called *wildcards* and are represented by "$*$". If an interval covers exactly one value, i.e. $p_i = q_i$, it is a *constant*, represented by $a_i^{p_i}$. All other intervals include both endpoints: $[a_i^{p_i}, a_i^{q_i}]$. Additionally, we say a pattern is *simple* if it consists of only constants or wildcards.

Let $P(o|t)$ be the conditional distribution of outcome value o for a given combination of explanatory attribute values t. For a given dataset, we can calculate $P(o|t)$ emprically from f. Formally, for all $o \in O$ and $t \in \mathcal{A}$, $P(o|t) = \frac{f(t,o)}{f(t)}$ if $f(t) > 0$ and zero otherwise. While the set of all these probability distributions is informative, it is too large for human interpretation, i.e., it is not concise. We consider informative summaries as compact representations of these conditional probabilities.

Following previous work on data summarization and exploration [7, 9, 18, 22], we use information theoretic methods to quantify informativeness. Let $\mathcal{P}_S = \{P_S(o|t) : o \in O, t \in \mathcal{A}\}$ be a model of the conditional probabilities based on the information contained in a summary S. By the *maximum-entropy principle*, a preferred distribution is one with the highest entropy, i.e., the most uniform distribution that agrees with the numbers reported in S without making any other assumptions. Formally, this is $\mathcal{P}_S$ that maximizes

$$H(\mathcal{P}_S) = -\sum_{t\in\mathcal{A}}\sum_{o\in O}\frac{f(t)}{n}P_S(o|t)\log\left(P_S(o|t)\right) \tag{1}$$

with the constraints that $0 \le P_S(o|t) \le 1$, and for all rules $r(p, o) \in S$, that $cnt(p, o) = \sum_{t\asymp p} f(t)\cdot P_S(o|t)$. Computing $\mathcal{P}_S$ is non-trivial as it has no closed form and requires numeric methods such as *iterative scaling* [2]. We omit the technical details of iterative scaling as these are orthogonal to our contributions, and instead give a brief example of calculating $\mathcal{P}_S$ below.

Example 3.1. Consider the Occupancy dataset from Table 1 and an informative summary S containing the first two patterns from Table 2, call them p_1 and p_2. Since p_1 matches all tuples, the maximum-entropy model for $\mathcal{P}_S$ distinguishes only between tuples that match p_2 and those that do not. Thus, we have to find $x_1 = P_S(\text{true}|t)$ for $t \asymp p_2$ and $x_2 = P_S(\text{true}|t')$ for $t' \not\asymp p_2$. Since the outcome is binary, the complementary probabilities are $P_S(\text{false}|t) = 1 - P_S(\text{true}|t)$ and $P_S(\text{false}|t') = 1 - P_S(\text{true}|t')$. Based on S, the restrictions are $20560 \cdot 0.231 = 12772 \cdot x_1 + 7788 \cdot x_2$ and $12772 \cdot 0.01 = 12772 \cdot x_1$, which yields $x_1 = 0.01$ and $x_2 = 0.59$. Naturally, this becomes harder when there are more probabilities to determine than constraints, as induced by overlapping patterns [9].

We use the Kullback-Leibler (KL) Divergence between $P(o|t)$ and $P_S(o|t)$ to measure the accuracy of S in estimating $P(o|t)$. For each tuple t, this divergence is defined as

$$KL_t\left(P\middle\|P_S\right) = \sum_{o\in O} P(o|t)\log\left(\frac{P(o|t)}{P_S(o|t)}\right). \tag{2}$$

Since this yields one divergence value per tuple, we aggregate the divergences and weigh them by tuple frequency to obtain the total error of a summary S:

$$div\left(P\middle\|P_S\right) = \sum_{t\in\mathcal{A}} f(t)KL_t\left(P\middle\|P_S\right). \tag{3}$$

The informativeness of a summary is measured as the reduction in error compared to only knowing the overall outcome distribution in the entire dataset. That is, we compare the divergence of the maximum-entropy estimate to a baseline model that uses only the total outcome distribution. For all $o \in O$ and $t \in \mathcal{A}$, the baseline is $P_B(o|t) = \frac{f(o)}{n}$. The information gain based on an informative summary S is then

$$gain(S) = div\left(P\middle\|P_B\right) - div\left(P\middle\|P_S\right) \tag{4}$$

Our formal problem statement is as follows.

Informative Summarization Problem. Given a dataset represented as function f counting entries for each attribute and outcome combination and a desired number of rules s, compute an informative summary S with $|S| = s$ that maximizes $gain(S)$.

3.2 Constructing Informative Summaries

Since informative summaries generalize explanation tables, the NP-hardness to construct optimal explanation tables [7] extends to optimal informative summaries. As a result, exact solutions are infeasible assuming $P \neq NP$. Instead, a common approach [7, 9, 18] for this and similar problems is a greedy approach, as shown in Algorithm 1. Starting with an empty summary S, the rules are selected one at a time until $|S| = s$[1]. For each selection, iterative scaling is performed to obtain the current model $\mathcal{P}_S$. Then, a set of candidate rules R is considered and a rule $r(p, o) \in R$ that maximizes $gain(S \cup \{r(p, o)\})$ is selected.

Algorithm 1: Greedy Construction of Informative Summaries

```
1 $S \leftarrow \emptyset$
2 while $|S| < s$ do
3     $\mathcal{P}_S \leftarrow iterative\_scaling(S)$
4     $R \leftarrow$ candidate rule set
5     $r(p, o) \leftarrow \arg\max_{q\in R} gain(S \cup \{q\})$
6     $S \leftarrow S \cup \{r(p, o)\}$
7 return $S$
```

Since R can be very large, even if only simple patterns are considered, one way to speed up Algorithm 1 is to *prune R*. For example, the Flashlight [7] and SIRUM [9] techniques perform sample-based pruning as follows. In each iteration of the while loop, a random sample is drawn from the dataset, and R is populated with only those simple patterns that match at least one sampled tuple. The intuition is that informative patterns are likely to have high support and therefore should match at least one sampled tuple.

Even after pruning the candidate rule set, Algorithm 1 may be too expensive because line 5 requires gain calculations for each candidate. For efficiency, prior work uses the following approximation for binary outcomes [7, 18, 22]. Let $r(p, o)$ be a candidate rule and $S^+ = S \cup \{r(p, o)\}$. It is assumed that $P_{S^+}(o|t) \approx \frac{cnt(p,o)}{supp(p)}$ for $t \asymp p$ and $P_{S^+}(o|t) \approx P_S(o|t)$ otherwise. That is, when computing gain, it is assumed that adding $r(p, o)$ only refines the outcome distribution estimates for tuples matching p, without recomputing the estimates for any other tuples (which would require iterative scaling). Additionally, in the current model based on S without the new candidate, all tuples matching p are given the same probability $P_S(o|t) \approx \frac{\sum_{t\asymp p} f(t)\cdot P_S(o|t)}{supp(p)}$. With this approximation, the improvement in gain by adding rule $r(p, o)$ is [7]:

$$\begin{aligned} &gain(S^+) - gain(S) \qquad (5)\\ &\approx cnt(p,o)\cdot\log\left(\frac{cnt(p,o)}{\sum_{t\asymp p} f(t)\cdot P_S(o|t)}\right)\\ &+ (supp(p) - cnt(p,o))\cdot\log\left(\frac{supp(p) - cnt(p,o)}{\sum_{t\asymp p} f(t)\cdot(1 - P_S(o|t))}\right). \end{aligned}$$

Call the two possible outcome values o and o'. Note that the second term above accounts for the gain due to refining the probability estimates for the complementary outcome value o' because $supp(p) - cnt(p, o) = cnt(p, o')$ and $1 - P_S(o|t) = P_S(o'|t)$.

We now extend the above approximation to non-binary outcomes. A rule $r(p, o)$ describes the frequency of one particular outcome value o within its matching tuples. With binary outcomes, we can immediately infer the frequency of the other value o'. With

[1]In the unlikely case that no rule yields additional gain, we stop the algorithm early.

non-binary outcomes, the question is how to adjust the probability estimates due to $r(p, o)$ for *all* the other outcome values besides o. Formally, Equation 5 becomes

$$gain(S^+) - gain(S) \approx cnt(p, o) \log\left(\frac{cnt(p, o)}{\sum_{t \asymp p} f(t) \cdot P_S(o|t)}\right) + \sum_{o' \in O \setminus \{o\}} cnt(p, o') \log\left(\frac{\sum_{t \asymp p} f(t) \cdot P_{S^+}(o'|t)}{\sum_{t \asymp p} f(t) \cdot P_S(o'|t)}\right). \quad (6)$$

To approximate P_{S^+} without iterative scaling, we again appeal to the maximum-entropy principle. When computing gain due to $r(p, o)$, we adjust the probabilities $P(o'|t)$ for each $o' \neq o$ proportionally to the adjustment for $P(o|t)$. This approximates the necessary adjustments to P_{S^+} without introducing any assumptions beyond $r(p, o)$. Formally this is

$$P_{S^+}(o'|t) \approx P_S(o'|t)\frac{1 - P_{S^+}(o|t)}{1 - P_S(o|t)} \approx P_S(o'|t)\frac{supp(p) - cnt(p, o)}{supp(p) - \sum_{t' \asymp p} f(t') \cdot P_S(o|t')}. \quad (7)$$

With this equation, we can now substitute P_{S^+} in Equation 6. Cancelling $\sum_{t \asymp p} f(t) \cdot P_S(o'|t)$ from the fraction within the logarithm of the sum yields

$$cnt(p, o) \log\left(\frac{cnt(p, o)}{\sum_{t \asymp p} f(t) \cdot P_S(o|t)}\right) + \sum_{o' \in O \setminus \{o\}} cnt(p, o') \log\left(\frac{supp(p) - cnt(p, o)}{supp(p) - \sum_{t' \asymp p} f(t') \cdot P_S(o|t')}\right) \quad (8)$$

as an approximation of $gain(S^+) - gain(S)$.

Note that with non-binary outcomes, we need to make two decisions when seeking the next rule with the highest gain: its pattern p and its outcome value o. We use $g_S(p)$ to denote the highest gain of any rule with pattern p over all the possible outcome values, i.e., $g_S(p) = \max_{o \in O} gain(S \cup \{r(p, o)\}) - gain(S)$.

Finally, we note that some existing techniques that use similar information-theoretic methods [9, 22] are compatible with numeric outcomes. The idea is to scale each outcome value by the sum of all the outcomes, which means that the scaled outcomes add up to one and can be thought of as a probability distribution. We omit the details and remark that this transformation is compatible with our summarization method for numeric explanatory attributes.

4 LENS APPROACH

We now present the **L**ightweight **E**xtraction of **N**umeric **S**ummaries (LENS) approach for informative summaries with numeric attributes. We motivate our approach and present an overview (Section 4.1), followed by a discussion of candidate rule generation (Section 4.2) and how we speed up the gain computation of the selected candidates (Section 4.3). We end with a discussion of computational complexity (Section 4.4).

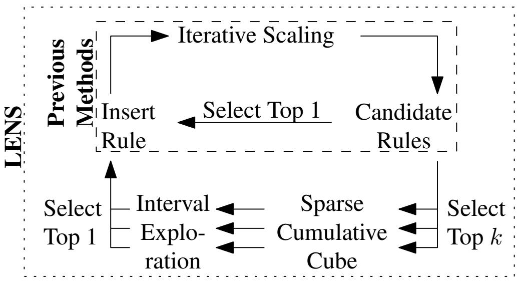


Figure 2: Summary of LENS

4.1 Motivation and Overview

Numeric and ordinal explanatory attributes produce a much larger pattern space that cannot be handled effectively by existing methods. For example, a straightforward extension of the sample-based pruning method used by Flashlight and SIRUM is to consider all patterns with all possible intervals that match at least one sampled tuple. However, when we tested the extension on the Occupancy dataset, this modified version of Flashlight did not produce a single rule within 3 hours. This means that the candidate rule space is too large even after sample-based pruning.

Another straightforward optimization that is used in many interval pattern mining techniques is to discretize or bin numeric domains. However, this leads to information loss, and minimizing the impact of this loss requires careful selection of the discretizing technique depending on both the application and the data [3, 10]. Alternatively, one can discretize the data manually. However, this requires domain knowledge for meaningful intervals, and even then, it may not be clear which intervals are informative. For example, in the Occupancy dataset, one would have to know that days of the week can be binned into weekdays and weekends, but even then, we would lose interesting patterns if, e.g., the building was occupied differently at the beginning of a workweek than at the end. Since informative summaries are meant to provide insights for users who may not be familiar with the data, requiring domain knowledge beforehand defeats their purpose. This means that we need a solution that is data driven and does not perform any discretization apriori.

Figure 2 summarizes the ideas behind LENS. Instead of abandoning existing methods completely, we leverage their strengths (finding simple informative patterns) while avoiding their weaknesses (inability to scale to large ordered domains). In each iteration, LENS first uses an existing method (shown at the top of the figure) to find top k *simple* informative patterns. Notably, LENS is compatible with any greedy method for finding simple informative patterns such as Flashlight, SIRUM or SURPRISE [22]. LENS then "grows" the (presumably informative) constants in the ordered attributes of these patterns in a principled way to form informative intervals. Furthermore, the intervals considered by LENS are deterministic for each underlying simple pattern, and therefore benefit from precomputations. To do this, we present a data structure called *Sparse Cumulative Cube* (SCC) that stores intermediate sums required by Equation 8 to estimate gain.

Algorithm 2 provides the details. Starting with an empty summary (Line 1), LENS performs iterative scaling (Line 3) and selects the next best rule (Lines 3–14) s times (Line 2). For each rule, LENS

Algorithm 2: LENS

```
Input: Data $f$, summary size $s$, number of simple patterns $k$
Output: Informative summary $S$
1  $S \leftarrow \emptyset$
2  while $|S| < s$ do
3      $\mathcal{P}_S \leftarrow iterative\_scaling(S)$
4      $maxGain \leftarrow 0$
5      $Candidates \leftarrow$ top $k$ simple patterns
6      foreach $p = ([p_1, q_1], \ldots, [p_d, q_d]) \in$ Candidates do
7          if $|\{i \in \{1, \ldots, d\} : p_i = q_i\}| < 5$ then
8              foreach $o \in O$ do
9                  Build $C_o$ and $C_o^S$                          c.f. Section 4.3
10             $q \leftarrow$ IntervalExploration$(p)$               c.f. Algorithm 3
11             if $g_S(q) >$ maxGain then
12                 $r_{best} \leftarrow \arg\max_{r \in \{r(q,o):o \in O\}} gain(S \cup \{r\})$
13                 $maxGain \leftarrow gain(S \cup \{r_{best}\})$
14     $S \leftarrow S \cup \{r_{best}\}$
15 return $S$
```

Algorithm 3: IntervalExploration

```
Input: Simple pattern $([p_1, q_1], \ldots, [p_d, q_d])$
1  $List \leftarrow \{(([p_1, q_1], \ldots, [p_d, q_d]), 0)\}$
2  $bestGain \leftarrow 0$
3  while List $\neq \emptyset$ do
4      $NextList \leftarrow \emptyset$
5      foreach $(([p'_1, q'_1], \ldots, [p'_d, q'_d]), \text{prevGain}) \in$ List do
6          $thisGain \leftarrow g_S(([p'_1, q'_1], \ldots, [p'_d, q'_d]))$
7          if thisGain > bestGain then
8              $bestGain \leftarrow thisGain$
9              $bestPattern \leftarrow ([p'_1, q'_1], \ldots, [p'_d, q'_d])$
10         if thisGain > prevGain then
11             forall $1 \leq i \leq d$ do
12                 if $p'_i > 1$ then
13                     $nextLim \leftarrow \max(1, p'_i - (2 \cdot |p_i - p'_i|) + 1)$
14                     $nextPat \leftarrow ([p'_1, q'_1], \ldots, [nextLim, q'_i], \ldots, [p'_d, q'_d])$
15                     $NextList.insert(nextPat, thisGain)$
16                 if $q'_i < |A_i|$ then
17                     $nextLim \leftarrow \min(|A_i|, q'_i + (2 \cdot |q_i - q'_i| + 1))$
18                     $nextPat \leftarrow ([p'_1, q'_1], \ldots, [p'_i, nextLim], \ldots, [p'_d, q'_d])$
19                     $NextList.insert(nextPat, thisGain)$
20     $List \leftarrow NextList$
21 return bestPattern
```

first obtains k simple patterns using some existing method (Line 5). For each of these patterns (Line 6), LENS explores patterns with intervals over ordered attributes (Line 10), as will be described in Section 4.2, with q being the best pattern in terms of gain. To speed up the gain computation, we build sparse cumulative cubes (Lines 8-9), as will be described in Section 4.3. While q is the most informative pattern based on a particular simple pattern p, r_{best} stores the rule with highest gain across all patterns (Lines 11-13), which is added to the summary (Line 14). We restrict the search to simple patterns with fewer than five constants (Line 7) for interpretability and efficiency. Patterns with many constants or intervals are harder to interpret [17], and, as we discuss in Section 4.3, the size of the sparse cumulative cube grows exponentially with the number of constants.

4.2 Interval Exploration

This section describes how LENS creates intervals based on a simple pattern p. Intervals are created only over ordered attributes that have a constant in p; unordered attributes or ordered attributes with a wildcard in p are not considered. However, the number of possible intervals over the considered attributes can still be prohibitively large. Take an ordered attribute A_i and a constant $a_i \in A_i$. There are up to $\frac{|A_i|}{2}$ values before and after a^i in the ordering as valid interval endpoints. This gives up to $\left(\frac{|A_i|}{2}\right)^2$ intervals on A_i containing a_i. If a simple pattern has more than one constant, the set of potential patterns is the Cartesian product of the intervals for each attribute.

To reduce the set of candidate rules, we consider exponentially increasing intervals. For an attribute A_i with ordered values $a_i^1, \ldots, a_i^{|A_i|}$ and a constant p_i occurring in a simple pattern, we consider starting points $p_i, p_i - 1, p_i - 3, p_i - 7, \ldots$ and ending points $p_i, p_i + 1, p_i + 3, p_i + 7, \ldots$, bounded by 1 and $|A_i|$, respectively. Note that these exponential steps are based on the rank order of attribute values and not the values themselves, meaning that all pairs of consecutive values are treated as equidistant. Also, note that the considered intervals are more fine grained close to p_i. This is desirable because p_i was chosen to be an informative constant in the simple pattern, so values close to p_i could also be informative. Additionally, for any interval $[p'_i, q'_i]$ on A_i containing a_i, we consider starting and ending points that are similar to p'_i and q'_i in distance to a_i. This exponential strategy reduces the number of intervals over an attribute A_i to at most $\left(\log_2\left(\frac{|A_i|}{2}\right)\right)^2$. Overall, there are up to $\Pi_{i=1}^{d}\left(\log_2\left(\frac{|A_i|}{2}\right)\right)^2$ patterns with intervals for each simple pattern. Note that our approach for interval selection is adaptive to the data and to the rules included into the summary so far as it is centered around constants from a pattern that is informative in this context. This is different than other interval selection schemes such as those using quantiles, which are similar to apriori discretization.

To further reduce the candidate search space, we use a greedy breadth-first search (BFS) that prunes some intervals along the way. We say that a pattern p is *incremented* if it is extended by moving its starting or ending point by one position in the allowed set of positions listed above. For any pattern p' resulting from an increment of p, we further explore the increments of p' only if p' has higher gain than p. Starting with p, the BFS considers patterns with the same number of increments from p during each iteration. As a result, duplicate patterns during the search could only occur within one iteration of the BFS. Keeping track of the candidates in each iteration via a hash table therefore prevents redundant computations during the interval exploration of a simple pattern p without explicitly listing all the previously considered patterns.

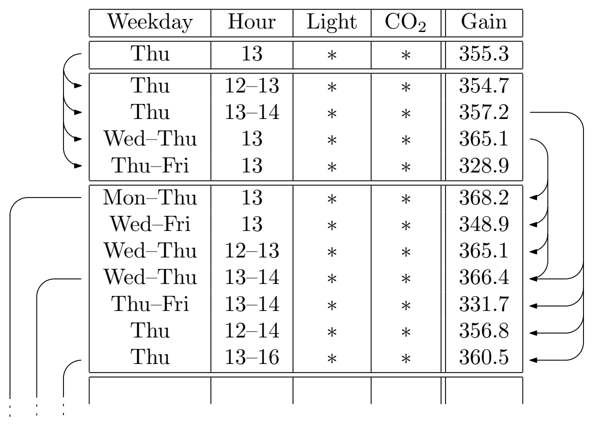
Algorithm 3 shows the interval exploration for one simple pattern, assuming for simplicity that all d attributes are ordered. Unordered attributes do not change; that is, they keep the same constant or wildcard as in the original simple pattern. For multiple simple patterns, the algorithm runs separately for each one, as seen in Algorithm 2. *List* is the set of patterns considered during the current iteration of the BFS, together with the gain thresholds used to test whether the increment producing the pattern increases gain. Starting with the simple pattern itself and zero as the threshold (Line 1), the algorithm considers all patterns in *List* (Line 5) and evaluates their gain (Line 6). If it is the highest gaining pattern so far, we store it as *bestPattern* (Lines 7-9). If the gain is higher than the accompanying threshold in *List*, the increment producing this pattern improves gain, and we include further increments in the next BFS iteration (Lines 10-19). Specifically, for each attribute (Line 11), we determine the next smaller starting point (Line 13) if it is not the smallest value, i.e. 1 (Line 12). The new pattern obtained by replacing this starting point (Line 14) is then inserted with the gain of the current pattern into *nextList*. *nextList* collects patterns for the next BFS iteration (Line 20). Analogously, a pattern with the next ending point, if possible, is inserted (Lines 16-19). Note that attributes with wildcards in the original simple pattern are never altered because their intervals already span from 1 to $|A_i|$. Finally, the algorithm returns the best pattern (Line 21) when there is no pattern left for another BFS iteration (Line 3). While this termination can happen early if no increment has resulted in a higher gain, there are at most $\sum_{i=1}^{d} 2 \cdot \log_2\left(\frac{|A_i|}{2}\right)$ increments to any pattern before it consists entirely of wildcards. Similarly, since no pattern is considered twice, the inner loop (Lines 6-19) is executed at most $\Pi_{i=1}^{d}\left(\log_2\left(\frac{|A_i|}{2}\right)\right)^2$ times.

We illustrate our interval search with an example and the accompanying Figure 3. Note that the gain values depend on the current model $\mathcal{P}_S$ and are just used for illustrative purposes. The figure shows each iteration of the BFS as one block of patterns. Incrementing a pattern for the next BFS iteration is displayed as an arrow from the original pattern towards the incremented one. For brevity, we refer to increments as $p'_i \rightarrow X$. This means that the pattern is the same except for the starting or an ending point p'_i of one of the attributes being replaced by X.

Example 4.1. Consider the Occupancy dataset in Table 1 and the simple pattern $p = (Thu, 13, *, *)$, which, in our notation, is $([p_1, q_1], [p_2, q_2], [p_3, q_3], [p_4, q_4]) = ([4, 4], [13, 13], [1, |\text{Light}|], [1, |\text{CO}_2|])$, with gain $g_S(p) = 355.3$. Since this pattern already covers the full range for Light and CO$_2$, the available increments are $p'_1 \rightarrow p_1 - 1$, $q'_1 \rightarrow q_1 + 1$, $p'_2 \rightarrow p_2 - 1$ and $q'_2 \rightarrow q_2 + 1$. As shown in Figure 3, only $q'_2 \rightarrow 14$ and $p'_1 \rightarrow 3$ result in higher gain, of 357.2 and 365.1 respectively. Next, we consider the patterns obtained through increments of these two patterns. We then check which of these patterns increase the gain over their predecessor. This process is repeated until no improvements are found or all the intervals become wildcards.

4.3 Efficient Gain Estimation

Even though we have reduced the number of patterns to consider, it is inefficient to evaluate the gain of each candidate (Equation 8) using a separate linear scan of the data. We introduce a data structure to quickly provide $cnt(p, o)$ and $\sum_{t \asymp p} f(t) \cdot P_S(o|t)$ for all outcomes $o \in O$. With $f : \mathcal{A} \times O \rightarrow \mathbb{N}$ for ordered attributes being integers in a $(d+1)$–dimensional space, $cnt(p, o)$ becomes a range-sum query. Similarly, the expected number of tuples $\sum_{t \asymp p} f(t) \cdot P_S(o|t)$ based



Weekday	Hour	Light	CO$_2$	Gain
Thu	13	$*$	$*$	355.3
Thu	12–13	$*$	$*$	354.7
Thu	13–14	$*$	$*$	357.2
Wed–Thu	13	$*$	$*$	365.1
Thu–Fri	13	$*$	$*$	328.9
Mon–Thu	13	$*$	$*$	368.2
Wed–Fri	13	$*$	$*$	348.9
Wed–Thu	12–13	$*$	$*$	365.1
Wed–Thu	13–14	$*$	$*$	366.4
Thu–Fri	13–14	$*$	$*$	331.7
Thu	12–14	$*$	$*$	356.8
Thu	13–16	$*$	$*$	360.5

Figure 3: BFS Pattern Exploration

on the current summary S is a range-sum query on similar data with $f_S(t, o) = f(t) \cdot P_S(o|t)$.

For a simple example, consider an ordered attribute $A = (a^1, \ldots, a^{15})$. One way to count the number of records matching pattern $[a^3, a^8]$ is to sequentially scan the dataset. However, if we precompute an array C where $C[i]$ stores the count of records with $A \leq a^i$, then the number of records matching our pattern is simply $C[8] - C[2]$.

There exists a data structure for such range-sum queries called *Prefix-Sum Array* [14]. It precomputes cumulative sums across all data dimensions and answers range-sum queries in time $O(2^{d+1})$, i.e., independently of n and of attribute-domain sizes. However, these cumulative sums are stored for all attribute combinations, leading to excessive memory consumption. In our case, this data structure has $|O| \cdot \Pi_{1 \leq i \leq d}|A_i|$ sums, and we need two data structures, one for f and one for f_S. Since numeric attributes can have very large domains, this memory footprint is too large, even with few attributes. Even for the smallest real-world dataset that we use in our experiments, it is $\Pi_{1 \leq i \leq d}|A_i| \geq 10^{19}$, i.e., at least 10^{10} gigabytes of memory.

Our solution is to create compact versions of this data structure for each simple pattern tailored to our pattern exploration scheme. That is, they are built for each simple pattern LENS explores and used only to speed up gain evaluation of increments from that simple pattern, which is Line 6 in Algorithm 3. We use two properties of our exploration scheme, namely that wildcards are never reduced to smaller intervals and that intervals grow exponentially. The first property allows us to ignore attributes with a wildcard in the simple pattern in the sense that we only store the sum of all values of these attributes. Second, the predefined interval limits indicate which subset sums will be required. For instance, if a simple pattern p has a constant value p_i for an attribute A_i, our exploration does not consider any pattern with an ending point of $p_i + 2$. Since prefix-sum arrays store cumulative sums, this aggregation is implicit and only the following upper bounds of aggregatable intervals are relevant

$$\begin{aligned} B_i(p_i) = (&p_i - 2^{\lfloor \log(p_i - 1) \rfloor}, \ldots, p_i - 8, p_i - 4, p_i - 2, p_i - 1, \\ &p_i, p_i + 1, p_i + 3, p_i + 7, \ldots, |A_i|) \end{aligned} \tag{9}$$

$$\begin{array}{lcccccccccccccc}
A_1 = (& a_1^1, & a_1^2, & a_1^3, & a_1^4, & a_1^5, & a_1^6, & a_1^7, & a_1^8, & a_1^9, & a_1^{10}, & a_1^{11}, & a_1^{12}, & a_1^{13}) \\
 & [& & & [& & [& [&] &] & &] & &] \\
B_1(8) = (& & & & 4, & & 6, & 7, & 8, & 9, & & 11, & & 13)
\end{array}$$

Figure 4: Aggregation thresholds for attribute $A_1 = (a_1^1, \ldots, a_1^{13})$ and constant $p_1 = 8$. The brackets represent starting and ending points considered by Algorithm 3 during interval exploration.

as illustrated in Figure 4. Note that these upper bounds are equal to the interval ending points we consider during pattern exploration, while they are shifted by one for the starting points. This is because range-sum queries on these cumulative sums subtract the sum *up to* the lower bound from the sum at the upper bound. As a result, we reduce both the dimensionality of our data structure and the domain size per dimension.

Additionally, note that we use this data structure to compute $cnt(p, o)$ and $\sum_{t \preceq p} f(t) \cdot P_S(o|t)$, respectively, which means that each query covers exactly one outcome value. Therefore, we can build the data structure separately for each outcome value instead of treating the outcome value as an additional dimension. In contrast to the previous optimizations, this does not reduce memory consumption as we need the same number of cells, which are simply spread across multiple instances. However, since the time complexity of range-sum queries is exponential in the dimensionality of the data structure, this optimization halves the query time, i.e., $O(2^d)$ instead of $O(2^{d+1})$.

To improve readability, and without loss of generality, suppose that the attributes are ordered so that the x attributes where a simple pattern $p = ([p_1, q_1], \ldots, [p_d, q_d])$ uses a constant are at the front. Thus, there exists an integer $1 \leq x \leq d$ with $p_i = q_i$ for $i \leq x$ and $p_i \neq q_i$ for $i > x$. Our *sparse cumulative cube* (SCC) is an x-dimensional array, where each cell with $i_1, \ldots, i_d$ in $B_1(p_1), \ldots, B_d(p_d)$, respectively, is defined as

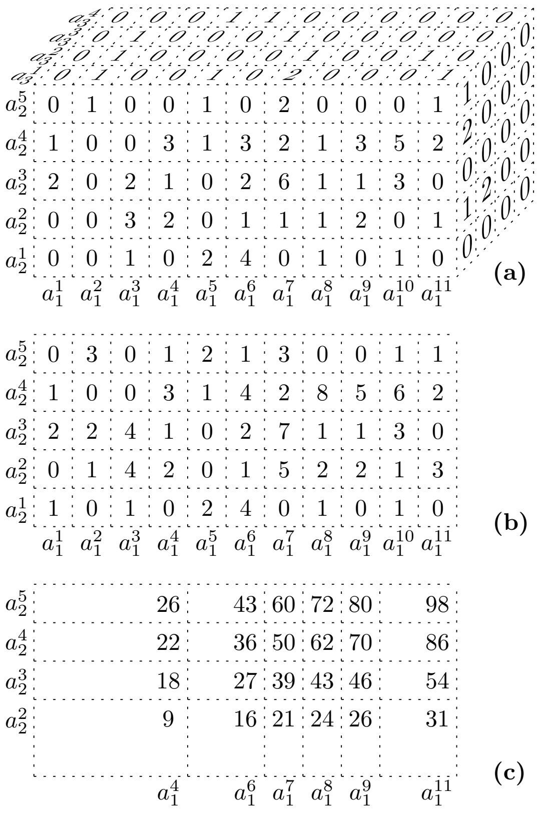
$$C_o[i_1] \cdots [i_x] = \sum_{j_1=1}^{i_1} \cdots \sum_{j_x=1}^{i_x} \sum_{j_{x+1}=1}^{|A_{x+1}|} \cdots \sum_{j_d=1}^{|A_d|} f((a_1^{j_1}, \ldots, a_d^{j_d}), o) \quad (10)$$

Analogously to [14], a range-sum query, in our case $cnt(p, o)$, is

$$cnt(p, o) = \sum_{i_1 \in \{p_1 - 1, q_1\}} \cdots \sum_{i_x \in \{p_x - 1, q_x\}} C_o[i_1] \cdots [i_x] \cdot (-1)^{\sum_{j=1}^{x} \mathbb{I}(i_j, p_j - 1)} \quad (11)$$

where $\mathbb{I}(i_j, p_j - 1)$ is the identity function, i.e., it is 1 if $i_j = p_j - 1$ and 0 otherwise. Note that C_o only includes range sums for the first x attributes (with constants in the simple pattern) and implicitly aggregates all values of the remaining attributes (with wildcards in the simple pattern). Similarly, we build a SCC C_o^S by substituting $f(t, o)$ with $f_S(t, o) = f(t) \cdot P_S(o|t)$ in Equation 10. This allows us to compute $\sum_{t \preceq p} f(t) \cdot P_S(o|t)$. Note that both of our SCCs can be constructed in a single pass over the data for each simple pattern considered for interval exploration: the interval endpoints are known beforehand because they depend on the constants in the simple pattern.

Example 4.2. Consider a dataset with attributes $A_1 = (a_1^1, \ldots, a_1^{11})$, $A_2 = (a_2^1, \ldots, a_2^5)$ and $A_3 = (a_3^1, \ldots, a_3^4)$, and two outcomes o and o'. Figure 5(a) shows the tuple counts f for one fixed outcome o as a cube. Further, let $p = ([8, 8], [4, 4], [1, 4])$



(a)

	a_1^1	a_1^2	a_1^3	a_1^4	a_1^5	a_1^6	a_1^7	a_1^8	a_1^9	a_1^{10}	a_1^{11}
a_3^4	0	0	0	1	1	0	0	0	0	0	0
a_3^3	0	1	0	0	0	1	0	0	0	0	0
a_3^2	0	1	0	0	0	0	1	0	0	1	0
a_3^1	0	1	0	0	1	0	2	0	0	0	1
a_2^5	0	1	0	0	1	0	2	0	0	0	1
a_2^4	1	0	0	3	1	3	2	1	3	5	2
a_2^3	2	0	2	1	0	2	6	1	1	3	0
a_2^2	0	0	3	2	0	1	1	1	2	0	1
a_2^1	0	0	1	0	2	4	0	1	0	1	0

(b)

	a_1^1	a_1^2	a_1^3	a_1^4	a_1^5	a_1^6	a_1^7	a_1^8	a_1^9	a_1^{10}	a_1^{11}
a_2^5	0	3	0	1	2	1	3	0	0	1	1
a_2^4	1	0	0	3	1	4	2	8	5	6	2
a_2^3	2	2	4	1	0	2	7	1	1	3	0
a_2^2	0	1	4	2	0	1	5	2	2	1	3
a_2^1	1	0	1	0	2	4	0	1	0	1	0

(c)

	a_1^4	a_1^6	a_1^7	a_1^8	a_1^9	a_1^{11}
a_2^5	26	43	60	72	80	98
a_2^4	22	36	50	62	70	86
a_2^3	18	27	39	43	46	54
a_2^2	9	16	21	24	26	31

Figure 5: (a) Illustration of f as cube displaying the count of tuples with outcome o. (b) Projection of A_3, i.e. summation, to a 2-dimensional matrix. (c) Sparse cumulative cube C_o.

be a simple pattern, i.e., $(8, 4, *)$. All patterns considered in `IntervalExploration`(p) therefore cover the full range of A_3. This means that the SCC is two dimensional to accommodate queries with different ranges for A_1 and A_2. To provide all the relevant numbers for this example, Figure 5(b) shows the counts of items with outcome o summed up over different values for A_3, which may not be visible in (a). As 8 and 4 are the constants of p, the relevant aggregation thresholds for our data structures are $B_1(8) = (4, 6, 7, 8, 9, 10)$ and $B_2(4) = (2, 3, 4, 5)$. The SCC C_o is thus a 6×4–matrix with accumulated item counts up to the thresholds $B_1(8)$ and $B_2(4)$ as shown in Figure 5(c).

One pattern that may be considered is $p' = ([5, 11], [3, 4], [1, 4])$. Without C_o, evaluating $cnt(p', o)$ would require a linear scan of all entries of f, or at least a summation of 40 cells if f is stored with random access such as the cube in Figure 5 (a). With C_o, we can determine $cnt(p', o)$ with 4 cells by Equation 11:

$$\begin{aligned}
cnt(p, o) &= \sum_{a_i \in \{4, 11\}} \sum_{a_2 \in \{2, 4\}} C_o[a_1][a_2] \cdot (-1)^{\mathbb{I}(a_1, a_1^4) + \mathbb{I}(a_2, a_2^2)} \\
&= C_o[4][2] \cdot (-1)^2 + C_o[11][2] \cdot (-1)^1 \\
&\quad + C_o[4][4] \cdot (-1)^1 + C_o[11][4] \cdot (-1)^0 \\
&= 9 - 31 - 22 + 86 = 42.
\end{aligned}$$

4.4 Time Complexity

We now study the worst-case time complexity of LENS. Since LENS can use any method for simple pattern generation and since iterative scaling runs until convergence without bounded complexity, our analysis focuses on the complexity of our contribution, which is Lines 6-14 in Algorithm 2.

In Lines 8-9, we build two SCCs for each outcome, which repeat for each of the k simple patterns and s rules generated. The size of an SCC depends on the domain size of the attributes, which is generally only bounded by n since every record could have a different value. As LENS limits the maximum number of constants of simple patterns to five in Line 7, each SCC has a maximal size of $(2 \cdot \log(\frac{n}{2}))^5$. Since a single scan of the data suffices to build an SCC, each construction takes time $O(n + \log^5(n)) = O(n)$.

Next, the worst case for our interval exploration scheme is that there is no pruning. As a result, for each simple pattern, all patterns obtainable through our increments would be considered once. For each attribute A_i, we would consider up to $2 \cdot \log_2(|A_i|)$ intervals. Since the attribute domain sizes can still only be bounded by n, up to $((\log_2(\frac{n}{2}))^2)^5 \leq \log_2^{10}(n)$ patterns with intervals would be considered for each simple pattern. Note that the only operation in Algorithm 3 that is not an elementary assignment or set insertion is the computation of gain in Line 6. By Equation 8, gain can be computed for a pattern p with $cnt(p, o)$ and $\sum_{t \asymp p} f(t) \cdot P_S(o|t)$ for each $o \in O$. These intermediate values can be computed each using 2^5 cells of our SSCs by Equation 11. In total, this step can be performed in time $O(1)$, and thus Line 10 in Algorithm 2 takes time $O(\log_2^{10}(n))$ to build an informative summary with s rules.

The remaining lines in Algorithm 2, except for the simple pattern generation and iterative scaling, consist of assignments and gain computations, which take constant time with our SCCs. Let $\mathcal{T}_{IS}$ and $\mathcal{T}_{SP}$ be the time complexity of iterative scaling and the method used to select simple patterns. The total complexity of LENS is then $O(s \cdot (\mathcal{T}_{IS} + \mathcal{T}_{SP} + k \cdot n))$. Since our pattern exploration scales linearly, it is unlikely to dominate the total complexity of LENS.

5 EXPERIMENTS

In this section, we experimentally evaluate the performance of LENS on real datasets. We analyze the informativeness and runtime of LENS against related methods, parameter choices and scalability.

5.1 Setup

We implemented LENS in C++ and compiled it using the Microsoft® C/C++ Optimizing Compiler Version 19.00. We use Flashlight [7] as the simple pattern generation subroutine. All experiments are conducted on Windows 10 using a single core of an Intel® Core™ i5-6300U processor clocked at 2.4 GHz and 20GB RAM. Unless otherwise noted, all presented results are arithmetic means over 100 runs. We use the following datasets.

`Occupancy` measures room temperature, humidity, lighting and CO_2 concentration, and includes a binary outcome indicating whether the room was occupied. This dataset is described in [4] and is available in the UCI repository[2]. Splitting the available timestamps into "Day of the week" and "Hour of the day", this dataset has 7 attributes and 20560 entries.

`Wine` described in [6] and available in the UCI repository[3], measures chemical wine properties and includes a quality score as an outcome. Merging the red wine and white wine parts of the data and using colour as an additional attribute yields 6497 entries with 12 attributes. Although there are 10 scores as outcome, we discretize them into "good" (scores 6 and higher) and "bad" to enable a comparison with subgroup discovery approaches.

`Electricity` presents data from the Australian New South Wales energy market. Each of the 45312 entries has the rising or falling price trend as outcome and 7 attributes such as energy price and demand in new South Wales and the neighboring Victoria region, excluding dates. The data is in the openML repository[4].

`Gas` measures the effect of two stimuli, bananas and wine, on an array of 8 gas sensors. The data is provided by Huerta et al. [15] and is available in the UCI repository[5]. There are 928991 entries with 10 attributes after adding temperature and humidity to the gas sensors, and one of three outcomes (no stimulus, wine and banana).

`Appliance` measures power consumption of household appliances with temperature and humidity per room as well as local weather measurements. The data was described in [5] and is available via Github[6]. The outcome in our experiments is the total power consumption, discretized into two values of equal frequency. Each entry has 25 other measurement attributes.

Table 6: Dataset characteristics

Name	Rows	Columns	Outcomes	Data space size
`Occupancy`	20560	7	2	$1.9 \cdot 10^{19}$
`Wine`	6497	12	2	$1.1 \cdot 10^{25}$
`Electricity`	45312	7	2	$1.4 \cdot 10^{20}$
`Gas`	928991	10	3	$8.3 \cdot 10^{53}$
`Appliance`	19735	25	2	$5.7 \cdot 10^{79}$

Table 6 contains statistics for these data sets. Most of our experiments use `Occupancy`, `Electricity` and `Wine` because they have similar size and all competing approaches produce summaries in reasonable time. These datasets represent tall (`Electricity`), wide (`Wine`) and balanced (`Occupancy`) data. To evaluate scalability, we use `Gas` and `Appliance` as these datasets have many rows and columns, respectively.

5.2 Parameter Sensitivity

Before comparing LENS to its competitors, we explore parameter choices and sensitivity. Our approach has two parameters: the sample size *FLS* for Flashlight's sample-based pruning, and the number k of simple patterns for exploration by LENS. To establish sensitivity, we evaluate the impact on gain and runtime for informative

[2]https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+

[3]http://archive.ics.uci.edu/ml/datasets/Wine+Quality

[4]https://www.openml.org/d/151

[5]http://archive.ics.uci.edu/ml/datasets/Gas+sensors+for+home+activity+monitoring

[6]https://github.com/LuisM78/Appliances-energy-prediction-data/

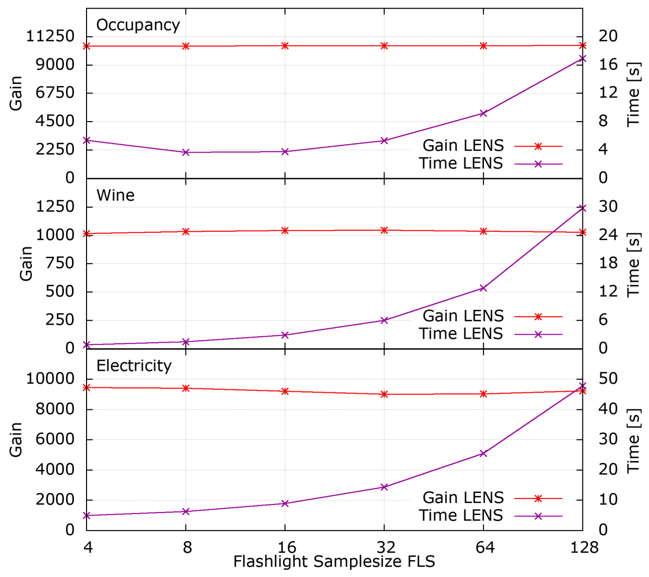


Figure 6: Impact of *FLS* on runtime of LENS and informativeness for `Occupancy` (top), `Wine` (middle) and `Electricity` (bottom)

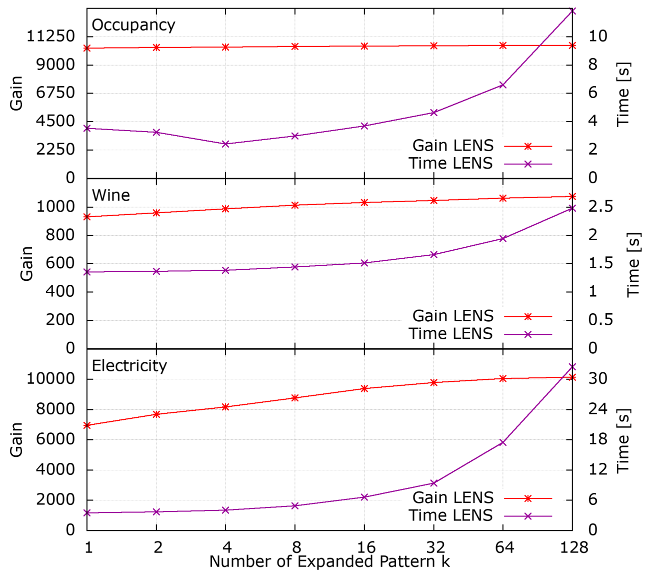


Figure 7: Impact of k on runtime of LENS and informativeness for the `Occupancy` (top), `Wine` (middle) and `Electricity` (bottom) datasets

summaries of size 20 for our datasets, with the other parameter being fixed to $FLS = 8$ or $k = 16$, respectively.

The impact of the sample size *FLS* is graphed in Figure 6 with a logarithmic x-axis. These graphs show that larger samples increase the runtime without a clear benefit in terms of informativeness. For consistent results and to avoid outliers, we omit extremely small sample sizes. These results are in line with the conclusion of the authors of Flashlight that small samples suffice to find the highest gaining (simple) patterns [7].

For the number k of explored simple patterns for each rule, the impact is shown in Figure 7 with a logarithmic x-axis. As one might expect, k represents a classic time-quality tradeoff, as it directly controls how many (non-simple) patterns are considered. Based on these experiments, runtime increases significantly for large k while the gain stagnates.

Our conclusion regarding parameter sensitivity is that moderate choices for k and *FLS* suffice, as larger values increase runtime without a significant gain improvement. Specifically, we use $FLS = 8$ and $k = 16$ in all further experiments.

5.3 Competing Approaches

Since neither Subgroup Discovery nor Decision Trees are intended for informative summarization, we now describe how we adapted these methods to produce informative summaries comparable to those produced by LENS.

Subgroup discovery algorithms produce subgroups without a specified outcome value or model for the outcome distribution. To compare their informativeness with LENS, we use datasets with a binary outcome. For each subgroup, a comparable summary contains one rule with the subgroup as the pattern and an arbitrary outcome value (recall that for binary outcomes, we can easily infer the distribution of the other outcome value). We then use iterative scaling, as in LENS, to compute the maximum-entropy estimates. However, the time for iterative scaling is not included in the reported runtime of subgroup discovery methods as it is only necessary for our calculation of gain for comparison with LENS.

As mentioned in Section 2, we use DSSD [23] and mergeSD [11] due to their focus on subgroup diversity and numeric data, respectively. While we use the official C++ implementation of DSSD[7], we reimplemented mergeSD as no C++ implementation was available. Due to the abundance of parameters for DSSD, we use mostly default settings, e.g., *beamWidth*=100, *qualMeasure*=WKL, *beamStrategy*=cover and *minCoverage*=10. The parameters *maxDepth* (maximum number of non-wildcard intervals of a pattern), *floatNumSplits* (number of bins for discretization) and *beamVarWidth* (dynamic adaption of beam width) all appear to show time-quality-tradeoffs. MergeSD also uses the parameters *maxDepth* and *floatNumSplits* but no other parameters. We found that the parameter choices mostly affect runtime but not gain in our experiments. As such, we use the following parameters with low runtime. For DSSD we use *maxDepth=1* and *floatNumSplits=10* and *beamVarWidth=false*, and for MergeSD we use *maxDepth=1* and *floatNumSplits=5*. We suspect that gain does not increase because these methods focus on different quality criteria, not informativeness.

For decision trees, each leaf is equivalent to a rule. For each leaf, we represent the collection of decisions leading to this leaf as a pattern and the dominant class of the leaf as the outcome. Different than production rules from decision trees [19], we also consider the precision of the dominant class for the data represented by the leaf. Since these leaves form a partition of the data

[7]Available at http://www.patternsthatmatter.org/software.php#dssd

and the outcome is binary, the decision tree therefore provides a well-defined model for $P(o|t)$. This enables a comparison of informativeness, where the summary size is the number of leaves in the decision tree. For this comparison, we use the C++ implementation[8] of "Yet another Decision Tree builder" (YaDT) [21], version 2.2.0, with several optimizations to reduce runtime and memory consumption. However, a problem with decision tree algorithms, including YaDT, is that we cannot build a tree of a particular size. For instance, using the default parameter values, the decision trees have 51, 477 and 1427 leaves for the Occupancy, Wine and Electricity data, respectively. This means that we have to perform a parameter search for every requested summary size and dataset in order to find an appropriate tree. As a result, we perform a grid search on the parameters $confidence \in \{0.01, 0.04, 0.16, 0.64\}$, $minCasesToSplit \in \{2, 8, 32, \ldots, 2048\}$ and $maxDepth \in \{1, 2, 4, 8\}$. As in previous work [7], the reported time is the total time for the grid search.

In contrast, explanation tables are informative summaries by design. We used our C++ implementation of Flashlight [7] that is also used as part of LENS. Note that this is the original version of Flashlight using only patterns without intervals. For fairness, we also use $FLS = 8$ when evaluating stand-alone Flashlight.

Finally, note that some approaches, such as mergeSD, assume that the global distribution of outcomes is known and only present subgroups whose distributions are significantly different. Others, like Flashlight, explicitly report the global distribution through an all-wildcard first rule (recall Table 2). To level the playing field, each informative summary and subgroup collection is allowed to contain an all-wildcard rule or subgroup without counting it towards the summary size.

5.4 Conciseness and Efficiency

We use a single plot that shows runtime and gain of summaries of various sizes for each method. In Figure 8, each datapoint represents the average gain and runtime for summaries of a certain size, noted by a small text label, for each method. Note that YaDT cannot produce decision trees of size one, so the corresponding datapoints are missing. Overall, the figure shows that LENS is the best overall choice in terms of gain per unit time.

Next, we zoom in on the time required to build informative summaries of certain sizes. For up to ten rules, LENS is a close second to unmodified Flashlight, whose explanation tables contain no intervals. Since LENS uses Flashlight as a subroutine, LENS cannot be faster if both are using the same sample size FLS. Beyond ten rules, DSSD is usually faster, as its runtime does not depend on the summary size. In contrast, MergeSD and YaDT are significantly slower.

Figure 8 also shows that LENS provides the highest gain per summary size with the exception of very small summaries on the Electricity dataset. Subgroup discovery algorithms are very inconsistent across datasets: while there is sometimes significant gain between summaries of size 1 and 2 and no gain from rules 11–20, it may also be the other way around. On the Occupancy and Wine datasets, this phenomenon manifests itself differently for MergeSD and DSSD. The reason for this erratic behavior is most likely the

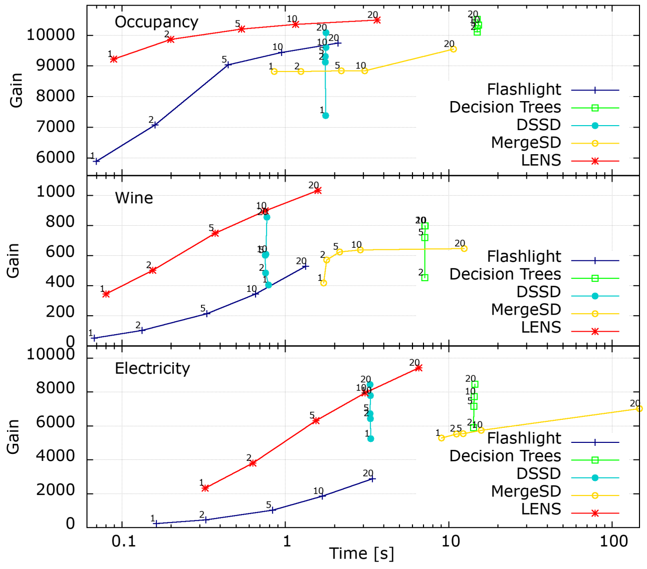


Figure 8: Runtime and gain for all methods and various summary sizes on Occupancy (top), Wine (middle) and Electricity (bottom)

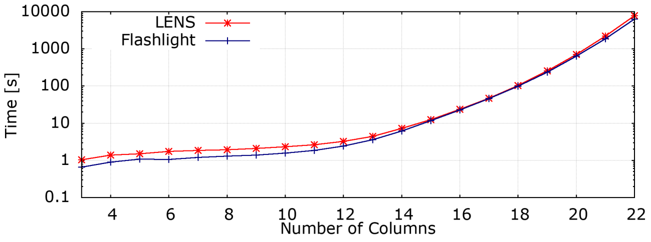


Figure 9: Scalability of LENS and Flashlight with the number of attributes using Appliance

different optimization goals of subgroup discovery which may or may not coincide with information gain. As a final note, LENS shows highest consistency of improvement of gain as the number of rules increases.

5.5 Scalability

To evaluate how well LENS scales with the number of columns, we use the Appliance dataset. For any number d of attributes, we use the first d columns in the dataset. Figure 9 shows the runtime to build informative summaries of size 20 on a logarithmic scale depending on the number of columns, averaged over 10 runs. Runtime increases superlinearly with the number of columns for both Flashlight and LENS. Given that the runtime of Flashlight gets closer to the runtime of LENS as the number of columns increases, we conclude that Flashlight dominates the runtime of LENS. This means that any speedup to the process of selecting simple patterns would directly speed up LENS, be it an improvement to Flashlight like SIRUM [9], or an entirely different method. Given this result, we defer the speedup of simple pattern generation to future work.

[8] Available at http://pages.di.unipi.it/ruggieri/software.html

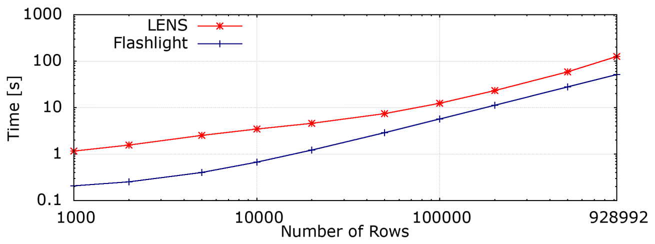


Figure 10: Scalability of LENS and Flashlight depending on the number of rows using Gas

For the Gas dataset, Figure 10 shows the row scalability of LENS and Flashlight as a double log plot. Here, we create smaller datasets by randomly sampling from Gas, to obtain unbiased subsets of the original attribute domains. LENS and Flashlight scale roughly linearly with the number of rows. While LENS requires nearly an order of magnitude more time than Flashlight for a smaller number of rows, it requires only twice as much time as Flashlight on the full dataset. This observation is also consistent with Figure 8. One explanation for the relatively high runtimes of LENS on small data is that small samples contained a number of columns with very few duplicates. As a result, many simple patterns matched only one tuple and therefore many patterns were pruned away during sample-based pruning. In contrast, LENS considered patterns with various intervals that matched at least one sampled tuple.

5.6 Summary of Results

Our main experimental findings are that LENS generally produces the most informative summaries of a given desired size and excels at informativeness per unit of runtime. In terms of scalability, LENS scales linearly with the number of rows and exponentially with the number of columns in the data. However, this exponential scaling is due to using Flashlight as subroutine and could be improved with existing parallel techniques for Flashlight [9] or with different algorithms for simple pattern selection. Finally, we find that the choices for the parameters sample size FLS and number of candidates k influence the runtime significantly, while FLS does not impact gain and k influences gain only moderately. As a result, small values, i.e., $FLS = 8$ and $k = 16$, lead to low runtimes without sacrificing information gain.

6 CONCLUSIONS

In this paper, we studied a generalized version of the informative summarization problem, whose goal is to summarize the effects of explanatory attributes (which may be ordered or numeric) on an outcome attribute (which may be binary, non-binary or numeric). We focused on handling numeric and ordered explanatory attributes, which greatly increase the search space of possible summaries. Our solution, LENS, leverages the strengths of existing techniques for unordered domains and adds new optimizations for ordered domains. Experiments on real datasets showed the efficiency and effectiveness of our solution compared to existing approaches that can be adapted to solve our problem. A possible direction for future work is to consider more than one outcome per row, e.g., responses of the same patient to different types of drugs in a pharmaceutical dataset.

ACKNOWLEDGMENTS

This work was partially supported by the DFG Research Training Group 2153: "Energy Status Data – Informatics Methods for its Collection, Analysis and Exploitation".

REFERENCES

[1] Martin Atzmueller. 2015. Subgroup discovery. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 5, 1 (2015), 35–49.

[2] Adam L Berger, Vincent J Della Pietra, and Stephen A Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational linguistics* 22, 1 (1996), 39–71.

[3] Ivan Bruha, Pavel Kralik, and Petr Berka. 2000. Genetic learner: Discretization and fuzzification of numerical attributes. *Intelligent Data Analysis* 4, 5 (2000), 445–460.

[4] Luis M Candanedo and Véronique Feldheim. 2016. Accurate occupancy detection of an office room from light, temperature, humidity and CO2 measurements using statistical learning models. *Energy and Buildings* 112 (2016), 28–39.

[5] Luis M Candanedo, Véronique Feldheim, and Dominique Deramaix. 2017. Data driven prediction models of energy use of appliances in a low-energy house. *Energy and buildings* 140 (2017), 81–97.

[6] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. 2009. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems* 47, 4 (2009), 547–553.

[7] Kareem El Gebaly, Parag Agrawal, Lukasz Golab, Flip Korn, and Divesh Srivastava. 2014. Interpretable and informative explanations of outcomes. *Proceedings of the VLDB Endowment* 8, 1 (2014), 61–72.

[8] Kareem El Gebaly, Guoyao Feng, Lukasz Golab, Flip Korn, and Divesh Srivastava. 2018. Explanation Tables. *IEEE Data Eng. Bull.* 41, 3 (2018), 43–51.

[9] Guoyao Feng, Lukasz Golab, and Divesh Srivastava. 2017. Scalable informative rule mining. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*. IEEE, 437–448.

[10] Salvador Garcia, Julian Luengo, José Antonio Sáez, Victoria Lopez, and Francisco Herrera. 2013. A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *IEEE Transactions on Knowledge and Data Engineering* 25, 4 (2013), 734–750.

[11] Henrik Grosskreutz and Stefan Rüping. 2009. On subgroup discovery in numerical domains. *Data mining and knowledge discovery* 19, 2 (2009), 210–226.

[12] Sumyea Helal. 2016. Subgroup discovery algorithms: a survey and empirical evaluation. *Journal of Computer Science and Technology* 31, 3 (2016), 561–576.

[13] Francisco Herrera, Cristóbal José Carmona, Pedro González, and María José Del Jesus. 2011. An overview on subgroup discovery: foundations and applications. *Knowledge and information systems* 29, 3 (2011), 495–525.

[14] Ching-Tien Ho, Rakesh Agrawal, Nimrod Megiddo, and Ramakrishnan Srikant. 1997. Range Queries in OLAP Data Cubes. *SIGMOD Rec.* 26, 2 (1997), 73–88.

[15] Ramon Huerta, Thiago Mosqueiro, Jordi Fonollosa, Nikolai F Rulkov, and Irene Rodriguez-Lujan. 2016. Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring. *Chemometrics and Intelligent Laboratory Systems* 157 (2016), 169–176.

[16] Yiping Ke, James Cheng, and Wilfred Ng. 2008. An information-theoretic approach to quantitative association rule mining. *Knowledge and Information Systems* 16, 2 (2008), 213–244.

[17] Himabindu Lakkaraju, Stephen H Bach, and Jure Leskovec. 2016. Interpretable decision sets: A joint framework for description and prediction. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1675–1684.

[18] Michael Mampaey, Nikolaj Tatti, and Jilles Vreeken. 2011. Tell me what i need to know: succinctly summarizing data with itemsets. In *ACM SIGKDD international conference on Knowledge discovery and data mining*. 573–581.

[19] J. Ross Quinlan. 1987. Simplifying decision trees. *International journal of man-machine studies* 27, 3 (1987), 221–234.

[20] J. Ross Quinlan. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc.

[21] Salvatore Ruggieri. 2004. Yadt: Yet another decision tree builder. In *Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on*. 260–265.

[22] Sunita Sarawagi. 2001. User-cognizant multidimensional analysis. *The VLDB Journal* 10, 2-3 (2001), 224–239.

[23] Matthijs van Leeuwen and Arno Knobbe. 2012. Diverse subgroup set discovery. *Data Mining and Knowledge Discovery* 25, 2 (2012), 208–242.

[24] Sebastián Ventura and José María Luna. 2018. Subgroup Discovery. In *Supervised Descriptive Pattern Mining*. Springer, 71–98.