Discovering Overlapping Quantitative Associations by Density-Based Mining of Relevant Attributes

Thomas Van Brussel[1], Emmanuel Müller[1,2], and Bart Goethals[1]

[1] University of Antwerp, Belgium
[2] Hasso-Plattner-Institute, Germany

Abstract. Association rule mining is an often used method to find relationships in the data and has been extensively studied in the literature. Unfortunately, most of these methods do not work well for numerical attributes. State-of-the-art quantitative association rule mining algorithms follow a common routine: (1) discretize the data and (2) mine for association rules. Unfortunately, this two-step approach can be rather inaccurate as discretization partitions the data space. This misses rules that are present in overlapping intervals.
In this paper, we explore the data for quantitative association rules hidden in overlapping regions of numeric data. Our method works without the need for a discretization step, and thus, prevents information loss in partitioning numeric attributes prior to the mining step. It exploits a statistical test for selecting relevant attributes, detects relationships of dense intervals in these attributes, and finally combines them into quantitative association rules. We evaluate our method on synthetic and real data to show its efficiency and quality improvement compared to state-of-the-art methods.

1 Introduction

Ever since its introduction [1], association rule mining has been a popular method for discovering interesting patterns in databases. However, the methods used to find these rules were originally only intended for boolean or categorical data, that is, rules of the form BUY[TOOTHBRUSH] $\rightarrow$ BUY[TOOTHPASTE]. However, since more and more of the data we gather is numerical, an interesting extension to classical association rule mining is quantitative association rule mining [21, 19, 25, 3]. In their purest form, quantitative association rules (QAR) denote relations in numeric data but can be extended to cover categorical items as well. Two examples of such association rules including numeric data are the following:

Example 1.

$$\text{AGE}[16, 90] \wedge \text{US-CITIZEN}[\text{YES}] \rightarrow \text{DRIVERS-LICENSE}[\text{YES}]$$

and

$$\text{AGE}[0, 21] \wedge \text{US-CITIZEN}[\text{YES}] \rightarrow \text{ALCOHOL-ALLOWED}[\text{NO}]$$

In a real-life database containing such information, such rules would have high support and confidence [1]: (1) US citizens are allowed to get a drivers license from the age of 16 and above, and (2) US citizens are not allowed to drink alcohol under the age of 21. The overlap in the numeric age attribute is a very natural and desired one. In contrast to these natural rules, discretization of the data in traditional methods would miss one of these two rules due to pre-selected intervals (e.g. by equal-width discretization) and their disjoint partitioning before mining the quantitative association rules.

The first attempts to mine data for quantitative association rules fall prey to this limitation. The first publication by Srikant et al. [21] transforms the original (numeric) data such that it could be handled by traditional association rule mining algorithms. This is done by a pre-processing step, in which each attribute is partitioned into an equal number of disjoint bins. This is still the most common procedure for quantitative association rules [23, 11]. However, the disjoint partitioning remains a major unresolved challenge for all of these techniques. Two regions that overlap each other in a certain attribute are not detectable after discretizing the data (cf. Example 1). As shown in our example, two rules might be naturally present in overlapping intervals. The association rule mining algorithm should select these relevant attributes, and further, detect the most appropriate interval for each rule individually.

Many techniques [3, 7, 19] try to bypass this problem and can generally be seen as optimization strategies. These strategies attempt to find optimal rules with respect to specific rule templates. Hence, they are limited to certain criteria, such as limited dimensionality, specific shapes, etc. Furthermore, choosing the optimal boundaries for this partitioning has been shown to be an NP-complete and intractable problem [26].

In our work, we focus on overlapping quantitative association rules. We propose a method that works without the need for a discretization step, and thus, prevents information loss in partitioning numeric attributes prior to the mining step. It exploits a statistical test for selecting relevant attributes for association rule mining. We select attributes that show a dependency with an interval from a different dimension. We then detect relationships of dense intervals in these attributes, and finally combine them into quantitative association rules.

In summary, our approach called DRule has the following contributions, (i) it does not require a partitioning of the data as pre-processing step; and detects overlapping rules. (ii) It does not impose any restriction on the type of rule that can be found; and hence provides a more flexible framework for quantitative association rules. (iii) It selects statistically relevant attributes and dense regions in each of these attributes while mining association rules. (iv) It provides an efficient computation of overlapping quantitative association rules for large databases.

2 Formal Problem Statement

Database For a database $\mathcal{DB}$ of mixed attribute types (binary, categorical, and numerical) we consider each object $o \in \mathcal{DB}$ to be represented by a vector

$$\boldsymbol{o} \in \{\mathbb{R} \cup \{\mathbf{c_1}, \ldots, \mathbf{c_k}\} \cup \{\mathbf{false}, \mathbf{true}\}\}^m .$$

We denote $o(A_i)$ the value of object o for attribute A_i and call the set of all attributes $\mathcal{A}$. We use the notation $n = |\mathcal{DB}|$ as the number of objects in the database and $m = |\mathcal{A}|$ the dimensionality of the database.

Quantitative Predicate Let $\mathcal{A} = \{A_1, \ldots, A_m\}$ be the set of all attributes in the database. A *quantitative predicate* is then defined as follows:

Definition 1 (Quantitative Predicate).
Given one attribute $A_i \in \mathcal{A}$ and lower and upper bounds $(l, u) \in \mathrm{dom}^2(A_i)$,

$$A[l, u] \text{ defines a quantitative predicate}$$

with $l \leq u$ in case of numeric attribute A_i and with a constant c equal to both bounds $l = u = c$ in the case of categorical or binary attributes.

This general definition covers all attribute types. We distinguish between two notations: $A[l, u]$ for numeric attributes and $A[c]$ for binary and categorical ones. Further, we say that an object $o \in \mathcal{DB}$ is covered by a quantitative predicate $A_i[l, u]$ *iff* $l \leq o(A_i) \leq u$ and $A_i[c]$ *iff* $c = o(A_i)$.

Set of Predicates and Quantitative Association Rules Based on the above definition, a *set of predicates* is simply defined as a conjunction of predicates. An object (or transaction) is covered by a set of predicates *iff* it is covered by each predicate in the set. For a predicate set $\mathcal{X}$ we denote its attributes by $attr(\mathcal{X}) \subseteq \mathcal{A}$.

Furthermore, the objects that are covered by such a set of predicates are defined as $I(\mathcal{X})$, the function that returns all $o \in \mathcal{DB}$ that are covered by all predicates given in $\mathcal{X}$:

$$I(\mathcal{X}) = |\{o \in \mathcal{DB} \mid o \text{ is covered by } \mathcal{X}\}|$$

We define the *support* and frequency of $\mathcal{X}$ as

$$\mathrm{supp}(\mathcal{X}) = |I(\mathcal{X})| \qquad \mathrm{freq}(\mathcal{X}) = \frac{|I(\mathcal{X})|}{|\mathcal{DB}|}$$

Definition 2 (Quantitative Association Rule). *A quantitative association rule (QAR) R is defined as*

$$\mathcal{P} \rightarrow \mathcal{Q}$$

with $\mathcal{P}$ and $\mathcal{Q}$ predicate sets, $\mathcal{Q} \neq \emptyset$, and $\mathcal{P} \cap \mathcal{Q} = \emptyset$.

The support of a rule $R : \mathcal{P} \rightarrow \mathcal{Q}$ is defined as $\text{supp}(R) = \text{supp}(\mathcal{P} \cup \mathcal{Q})$, the number of objects that satisfy the predicates in both $\mathcal{P}$ and $\mathcal{Q}$. The confidence is defined as $\text{conf}(R) = \frac{\text{supp}(R)}{\text{supp}(\mathcal{P})}$. A rule has to fulfill $\text{supp}(R) \geq minSupport$ and $\text{conf}(R) \geq minConfidence$ parameters as in traditional association rule mining [1].

Note that support and confidence are traditional measures for assessing association rules, but they are not sufficient for detecting interesting intervals in numeric data. Both support and confidence can be naively optimized by selecting the entire domain of a numeric attribute. For a more reasonable detection of interesting intervals, we introduce *density* in numeric data. We will further introduce this in Section 3. We have chosen for density as a measure as it is well understood and easy to interpret and inspiration can be drawn from existing efficient algorithms. This also meshes well with our approach to search space reduction introduced in the next section.

Generalized Problem Statement Mining association rules according to Definition 2 is a general problem statement for quantitative association rule mining: (1) it is a flexible definition in terms of choice of attributes, (2) It allows different intervals for each individual rule, (3) and it allows overlapping intervals for the numeric attributes. Let us discuss these three properties formally and contrast them to the more restricted definitions found in the literature:

Property 1 (Free choice of attributes).
The attribute set $\mathcal{P} \cup \mathcal{Q}$ of a rule can be any subset of all given attributes $\mathcal{A}$. We do not impose any restrictions on the number of attributes in a rule nor the configuration in which they appear.

In contrast to this free choice, other methods are not as flexible: *pre-selection of the attributes* by the user is a widely used restriction [19]. It is limited in both LHS and RHS of the rule, which are selected by the user beforehand. Such restriction does not allow for rule discovery outside this template of pre-selected LHS and RHS. Other methods perform restrictions on the size of a rule. For example, the restriction to mine 2D rules only [25, 3]. Such methods are designed for a specific goal and can not be easily generalised.

Property 2 (Individual intervals for each rule). For a numeric attribute $A_i \in \mathcal{P} \cup \mathcal{Q}$ of a rule, its corresponding interval can be any $[l, u] \in dom^2(A_i)$, i.e. we do not have any restriction on the intervals and they can be determined for each rule individually.

While we allow for arbitrary intervals, related algorithms already mentioned before choose their interval bounds from a set of predetermined disjoint intervals [15, 21]. Formally, $[l, u] \in \{[l_1, u_1), [l_2, u_2), \ldots, [l_k, u_k]\}$ is fixed for each attribute A_i. These interval bounds create a true partioning of the data space. That is, the possible intervals are fixed for each attribute A_i for all rules. In contrast to such a limited choice of intervals, we aim at an on-the-fly selection of intervals within the rule mining process. That is, for each rule, we aim at density-based intervals

directly on the numeric data, which allows for individual and better intervals per rule. In addition, a third key property automatically follows from the previous one. In addition to finding individual intervals for each rule, it also allows overlap between these intervals as well. Techniques that rely on a pre-discretization of numeric attributes [15, 21] do not allow for such overlap.

3 DRule

A straightforward solution for the problem described in the previous section would be exhaustive search. Such an approach would detect interesting intervals by checking each combination of dimensions and even checking every possible pair of interval bounds. This is obviously an *intractable* solution. However, due to its exhaustive nature it provides us all the rules contained in the data.

With DRule we propose an efficient method that allows us to find the same results as exhaustive search would, i.e. without missing any rules, while avoiding the tremendous runtime complexity. In order to find all rules, we start from each dimension, just like exhaustive search does. However, we quickly prune parts of the search space to prevent uninteresting rules or regions from slowing down our algorithm. To this end, while going through the search space, we demand that dimensions that are considered together are not uniformly related to each other. As such, uninteresting rules can be pruned.

In the following sections we will describe DRule in more detail. Our algorithm achieves the detection of overlapping QARs by using an approach that can be split into three parts, namely detecting interesting regions through *search space reduction*, *mining for individual intervals*, and *generating quantitative association rules*. In each of these three parts we ensure flexibility in both the selection of relevant attributes and the detection of individual (possibly overlapping) intervals for each rule. These three parts will be further explored in the following sections.

3.1 Mining for intervals

To successfully create rules, we need meaningful interval bounds. As mentioned before, fixing these interval bounds beforehand severely limits the rules we can find, while considering all possible intervals is infeasible. To solve this problem, we use a method that allows us to efficiently generate interval bounds on-the-fly for each rule.

Algorithm Overview The general idea of DRule is to mine all possible predicate sets in a depth-first search. For efficient mining of intervals $A_i[l, u]$, we first detect interesting (i.e. dense) regions. These regions are hyper-rectangles in the multi-dimensional space and approximate the individual intervals of one rule. These hyper-rectangles, which we shall refer to as *interesting regions*, describe areas in the data that show promise for rule generation from a density-based viewpoint. When generating interesting regions, we perform an analysis to determine which

combinations of attributes can provide us with interesting results. That is, we already filter out those combinations of attributes that leave us with predictable and uninteresting results (cf. Section 3.2). Given one of these interesting regions (i.e. one hyper-rectangle), DRule refines this hyper-rectangle by identifying the dense regions in each dimension and providing a list of dense intervals. These can be used as a quantitative predicate; however, we still need to visit the other dimensions in order to produce meaningful rules. We proceed the search in a depth-first manner for each of the dense regions detected using the dense region as a dataset for the next step. After each recursive step, we generate rules for the intervals detected so far. An overview of DRule is given in Algorithm 1. Note that Algorithm 1 is called from each dimension in a region R.

Algorithm 1 DRule (Interesting region R, dimension d)

```
1: Find all dense regions in R in dimension d
2: for all Dense regions r do
3:    Generate_Rules(r, minimum confidence)
4:    for all Dimensions d' interesting to dimension d in R do
5:       if d' has not yet been visited then
6:          DRule (r, d')
7:       end if
8:    end for
9: end for
```

Comparison to Existing Processing Schemes There are three important sides to the algorithm that have not been explicitly stated. One, the algorithm does not guarantee that the hyper-rectangle forms a dense region in all dimensions. That is, it only guarantees that each interval discovered by the algorithm is dense. This requirement is less restrictive as it allows us to find a larger set of possibly interesting rules. Secondly, the order in which we visit dimensions is important. This is not considered by other algorithms (e.g. for quantitative frequent itemsets [24]): first visiting dimension 1, followed by dimension 2 can lead to different results compared to visiting them in reverse order. Failure to do so can result in many missed patterns.

The previous remark is one of the key aspects in allowing us to find overlapping regions and allows us to distinguish ourselves from our competitors, such as QFIMiner [24]. Working on a per dimension basis and not taking the ordering of dimensions into account, can result in regions not being split properly. Taking this into account, our strategy allows us to find overlapping regions (conform Property 3).

3.2 Search Space Reduction

In order to reduce the infinite search space for intervals in a numeric data space, we propose to mine for interesting regions. These interesting regions should have

high density (similar to the notion of high frequency for traditional itemset mining). Please note that this step might be considered as a sort of discretization as we have to fix intervals at some point. However, it is by far more flexible than pre-discretization as it allows for on-the-fly generation of density-based regions within the rule mining. Unfortunately, such a density computation in arbitrary attribute combinations is known to be computationally very expensive [18]. Thus, we will restrict to one-dimensional density computation using DBSCAN [6, 12], similar to colossal pattern mining for frequency assessment [27]. This heuristic is well-known for approximate subspace clustering [12, 17, 16] and we will exploit its efficient search for finding overlapping QARs. Our extension to this reduction scheme can handle both numeric and categorical attributes, and reduces the search even further by assessing statistical dependence between pairs of attributes. Our detection method is flexible in using any subset of the given attributes (conform Property 1) and any intervals in the respective attribute domain (conform Property 2). Overall, this allows us to reduce the potential interval boundaries based on the data distribution observed in any attribute combination. For more information about this approach and more detailed descriptions of the choices, we refer to the original paper on this kind of search space reduction [12].

One-Dimensional Clustering Our search space reduction works as follows. Start by clustering each dimension separately according to some clustering algorithm. In our case we have chosen to use the well-known DBSCAN algorithm [6] for density-based clustering. Since DRule only needs dense regions, we can substitute DBSCAN for any algorithm that finds these kinds of regions or algorithms such as spectral clustering.

We support categorical items by performing a frequency analysis on the categorical dimensions and treat them as clusters if they are sufficiently frequent. We will also refer to these 1D clusters as *base-clusters* and they will be denoted as C^1. These base-clusters form the foundation from which we will decide what points and dimensions are relevant.

Merge of Candidates Since we are not interested in clusters that exist in only one dimension, we search for approximations of interesting regions by merging the base-clusters found in the previous step. This way, we can quickly determine i) which points form a cluster and ii) in which dimensions these points form a cluster. To find these approximations, we cannot simply merge base-clusters and hope for the best as this would be exponential in nature. Therefore, for our merge of cluster candidates, we employ a greedy merge of the most similar base-clusters. Let us first provide the formal definitions of this step before we continue.

Definition 3 (The Most Similar Cluster).
Given a 1D cluster $c \in C^1$, the most similar cluster (MSC) $\hat{c} \neq c$ is defined by:

$$\forall c_i \in C^1 \setminus \{c\} : sim(c, \hat{c}) \geq sim(c, c_i)$$

with similarity instantiated by $sim(c_i, c_j) = |c_i \cap c_j|$.

If the most similar cluster $\hat{c}$ shares a large number of objects with the base cluster c, this is usually indicative of the presence of a multi-dimensional cluster, i.e. a combination of these two 1D clusters.

In order to gain a set of merge candidates (i.e. 1D clusters that can be merged), we need a merge procedure to identify multiple similar clusters. We consider merging the base cluster with its most similar 1D clusters if they share many objects.

Definition 4 (k-Most-Similar Clusters).
Given a 1D cluster $c \in C^1$, we call $MSC_k(c) \subseteq C^1$ the k-most-similar clusters iff it is the smallest subset of C^1 that contains at least k base-clusters, where the following condition holds: $\forall c_i \in MSC_k(c), \forall c_j \in C^1 \setminus MSC_k(c) : sim(c, c_i) > sim(c, c_j)$.

In order to select which of these clusters should be merged, we compute the best-merge candidates (BMC) as follows.

Definition 5 (Best-Merge Candidates).
Given a 1D cluster $c \in C^1$ and $k, \mu \in \mathbb{N}^+ (\mu \leq k)$. The best-merge candidates are defined as:

$$BMC(c) := \{x \in C^1 \mid |MSC_k(c) \cap MSC_k(x)| \geq \mu\}$$

Merging all best-merge candidates filters out merge candidates that do not add a lot of value. However, due to the parameter k, this method might be too restrictive, causing a loss of cluster information. Since we want maximal-dimensional regions, we bypass this problem by merging best-merge candidate sets. That is, we decide to merge two base clusters if they share at least one base-cluster which fulfils the properties of a best-merge cluster.

Definition 6 (Best-Merge Cluster).
Given a 1D cluster $c \in C^1$ and $minClu \in \mathbb{N}^+$. Then c is called a best-merge cluster if $|BMC(c)| \geq minClu$.

Generation of Interesting Regions Now that we have found which clusters can be merged, we can start our search space reduction. We start by generating our approximative regions. These are the regions that will ultimately be refined into rules if possible. First, generate all best-merge clusters. This will give us a list of base clusters that are most suited to be merged. Then, find all pairs of best-merge clusters c_A and c_B where $c_A \in BMC(c_b)$ and $c_B \in BMC(c_A)$. As explained above, simply merging c_A and c_B is not enough, we therefore add all best-merge candidates of c_A and c_B to this group. This ensures that we find maximal-dimensional approximations of the regions we are interested in. The resulting set of base-clusters form our region. Formally, a region is then defined as follows.

Definition 7 (Region). *A region R is a set of 1D clusters (base-clusters),*

$$R = \{c \mid c \in (BMC(c_A) \cup BMC(c_B))\} \cup \{c_A, c_B\},$$

where $c_A, c_B \in R$, $c_A \in BMC(c_B) \wedge c_B \in BMC(c_A)$.

The points of a region are found by taking the union of all points in the base-clusters. We take the union as to not be too restrictive on the points that are part of the region.

Intuitively, a region defines a hyper-rectangle in the original data that can be seen as a smaller version of the original database. This hyper-rectangle approximates a quantitative predicate set (cf. Definition 1). That is, given a detected region, we can consider the intervals of this region to define a quantitative predicate. Of special interest is that these regions do not have to be disjoint. That is, we allow two regions R_1 and R_2 with $R_1 \cap R_2 \neq \emptyset$ (conform Property 3).

Selecting relevant attributes Now that we have found our *possibly* interesting regions, we can continue with generating our rules. Before we do this, however, we first introduce an intermediary step that will help us filter out uninteresting rules. That is, given a region, we will verify that each dimension of this region is interesting w.r.t. all of its other dimensions. For example, suppose that we have found a dense region denoting all people between a certain age. It would be uninteresting to report to the user that approximately half of these people are female. To avoid considering combinations of mutually uninteresting dimensions, we propose to use an additional measure.

We extend the previous approach by introducing a pairwise independence test between all attributes using the χ^2-test for uniformity. That is, given a region R, we decide, for each pair of dimensions of this region, whether the points in this region are uniformly spread in the second dimension when viewed from the first, and vice versa.

Optimizations Recall that to find overlapping rules, we consider the order in which we visit dimensions. However, looking at the data from every possible angle will not always result in different rules. To stop the algorithm from performing a lot of unnecessary computations, we can opt for a *short-circuiting* strategy. That is, if we find that visiting attribute A_1 from attribute A_2 yields the same result as visiting them in reverse order, we can choose to terminate our computation as we will not be able to find new results.

Another optimization that can be done is when visiting a certain set of attributes $\mathcal{X}$ yields largely the same results as visiting a set of attributes $\mathcal{Y}$. All attributes we visit starting from $\mathcal{X}$ will yield the same result as when visiting them from $\mathcal{Y}$. Therefore, we can stop our computation for one of these sets and create a link between the results, appending the results to that set in the end. Please note that this optimization, while reducing the output size of the algorithm, increases the memory consumption of the algorithm as we have to keep track of intermediary results. This also does not remove any results from the rules we would have discovered had we not run this step. We would simply halt the computation of one branch when the results would be same from that point on. The end results can then be added to the branch for which computation was halted.

3.3 Generating QARs

The final step of the algorithm is to take these interval bounds and transform them into rules. This step is the simplest of the three and boils down to a simple counting strategy. That is, given a set of intervals, for each interval compute the points of the dataset that are contained within that interval. Then, try placing each dimension of the region on the right hand side of the rule and compute the remaining left hand side. Intersecting the points of the intervals on the left hand side and comparing them with the points on the right hand side allows us to quickly compute the confidence of the rule.

Note that this step does not have to separate from the previous step as rules can be generated at each level to provide more immediate feedback.

4 Experimental Results

We ran our experiments on a machine running Mac OSX 10.9.1 with a 2.3 GHz Intel Core i7 and 8 GB of memory. The algorithm was written in Python, making heavy use of NumPy which provide bindings to efficient linear algebra libraries. We compare our method to state of the art methods such as QFIMiner [24] and QuantMiner [19] and also to an equi-width discretization technique. The source code for QuantMiner can be found online, the implementations for the other algorithms were created by us. We test the algorithms on synthetic data to provide an objective comparison, and on real data to show that our results can be used in real world situations. The synthetic data does not contain any categorical attributes as QuantMiner is not capable of handling them.

Synthetic data

The synthetic data we use in this section was generated by creating dense regions in subsets of all attributes and using overlapping intervals in the individual attribute domains. Unless otherwise specified, we set the minimum confidence level to 60%. That is, only rules that are true at least 60% of the time are reported.

Efficiency To test the efficiency and scalability of our algorithm, we compared our runtime against that of our three competitors. We generated a dataset of 50000 objects and increased its dimensionality.

Note that there are several caveats with respect to this comparison. First, to compare to QFIMiner, we have included our own routine to generate quantitative association rules. Since QFIMiner only mines the data for quantitative frequent itemsets (*QFIs*), we still need to transform this data to *QARs*. Second, while QuantMiner is included in the comparison, it is very hard to directly compare it to QFIMiner or DRule as it works in a different way. QuantMiner takes as input a dataset and a rule template. A rule template tells the algorithm which attributes to use in the LHS of the rule and which to use in the RHS of the

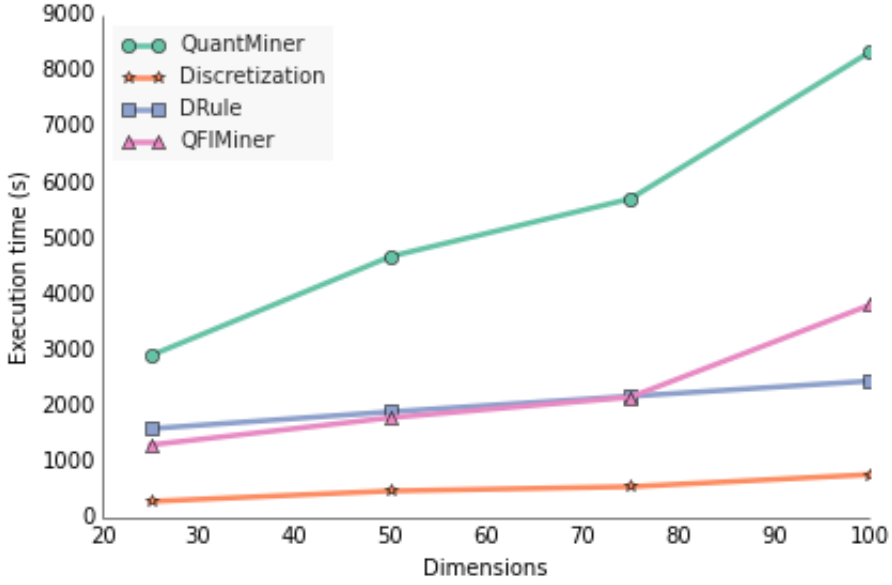

Fig. 1. Comparison of execution time between DRule, QFIMiner, QuantMiner, and discretization. The dataset in all instances contains 50000 transactions.

rule. QuantMiner then looks for optimal rules with respect to this template. To generate all rules of a certain size, the user has to loop through all possible combinations of dimensions manually. To provide a comparison, we only use QuantMiner to mine for rules in the dimensions in which the other algorithms found rules.

From Figure 1 we can conclude that DRule performs well when compared to its competitors. The only competitor consistently outperforming our method is the method using discretization. For reasons mentioned before, this method does, however, not provide us with satisfactory results. When comparing to QFIMiner, we can see that our method performs equally well. However, QFIMiner misses some of the hidden rules, which in contrast are correctly detected by our method. Specifically, rules that can only be found when visiting dimensions in a certain order are not found by QFIMiner. An easy example is when looking at a dataset in the shape of a face with the mouth overlapping with both eyes in the x dimensions. If an algorithm first splits on the x dimension and then on the y dimension, the eyes are not separated, while this does happen when first visiting the y dimension and then x dimension.DRule is capable of generating more rules than QFIMiner while keeping up in terms of runtime. We can see that our method consistently outperforms QuantMiner, but it should be noted that QuantMiner was designed for rules using user-templates and not for an unsupervised detection of rules with a free choice of attributes.

It should be noted that it can easily be shown that we find all rules that QFIMiner finds as both algorithms use a similar density-based approach to find intervals.

Rules output For this experiment, we generated datasets of varying dimensionality and compared the amount of rules that were generated with and without checkout for uniformity between dimensions and using the short-circuiting strategy discussed in Section 3.

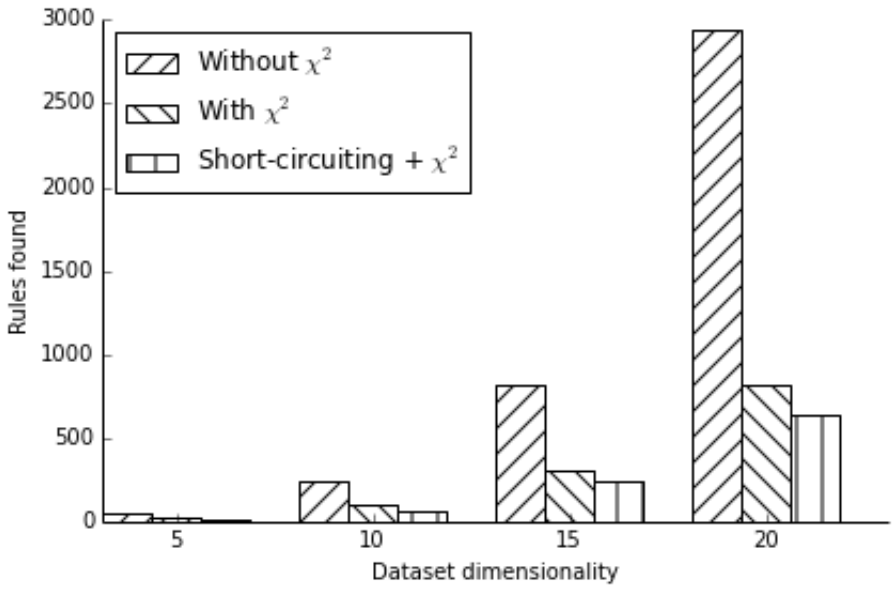

Fig. 2. The amount of rules the algorithm outputs with uniformity checks and without and using the short-circuiting strategy. We reject uniformity if $p \leq 0.05$.

In Figure 2 we demonstrate how much our checks for uniformity reduce the amount of work the algorithm has to perform. We can see that checking for uniformity drastically reduces the amount of rules we output and, therefore, the amount of computations our algorithm has to perform. Using the short-circuiting strategy further reduces the rules that are generated. We should note, however, that this strategy more than doubled the memory usage of the algorithm.

Overlap An important characteristic of our algorithm is its capability of finding overlapping regions. To test whether our algorithm is really capable of finding them, we designed a dataset that serves as a demonstration. It consists of several interesting regions that overlap in one or more dimensions. The order in which

the algorithms visit the attributes is of great importance for the successful completion of this test. The dataset consists of three sets of records, each of size 10000, along with an additional 10000 items that were added to introduce noise. These record sets are dense in all dimensions they exist in. A summary of this data is shown in Table 1.

Table 1. Summary of the dataset generated to demonstrate overlap. The boundaries the sets in each of the dimensions is shown. Using an approach that visits each dimension separately requires that dimension z is visited first.

Pattern ID	x	y	z
1	[5, 8]	[10, 15]	[10, 15]
2	[10, 18]	[8, 12]	[9, 16]
3	[3, 15]	[5, 15]	[3, 7]

Using this dataset, we came to the following conclusions:

- Discretization does not correctly handle this dataset. That is, while dimension z can be properly discretized, the other dimensions are cut up into chunks. When using confidence as a measure, some of these chunks are discarded as no confident rules can be created from them, leading to information loss.
- QFIMiner is only partially capable of correctly finding all QFIs. QFIMiner uses an Apriori-like build-up of the dimensions that contain dense regions, but does not take into account that the order in which these dimensions are visited can be important. Since it first visits dimension x, then dimension y, and only then dimension z in the last phase, it is only able to detect two regions. QFIMiner's density threshold also causes problems in certain situations. That is, it requires that regions are more dense than the average density of the dataset in the dimensions in which the regions exist. In the case of fairly clean data or data where one region is far more dense than the others, this can lead to regions that go unnoticed.
- DRule is capable of detecting a superset of the rules that QFIMiner finds. That is, since it also visits the dimensions in the same order that QFIMiner does, it finds the same rules. But since it also takes other orderings into account, it is able to detect all three regions and provide rules for them. This ensures that DRule provides more accurate rules compared to competing approaches.

Noise In Figure 3 we show how resistant our algorithm is to noise and compare it to the other algorithms (excluding QuantMiner, for reasons mentioned above). Thanks to the first step in our algorithm, the detection of interesting regions, we can quickly filter out dimensions that are not interesting to us. The only work that has to be done on an uninteresting region, is the preliminary 1D clustering, which, depending on the implementation of the algorithm, can be done

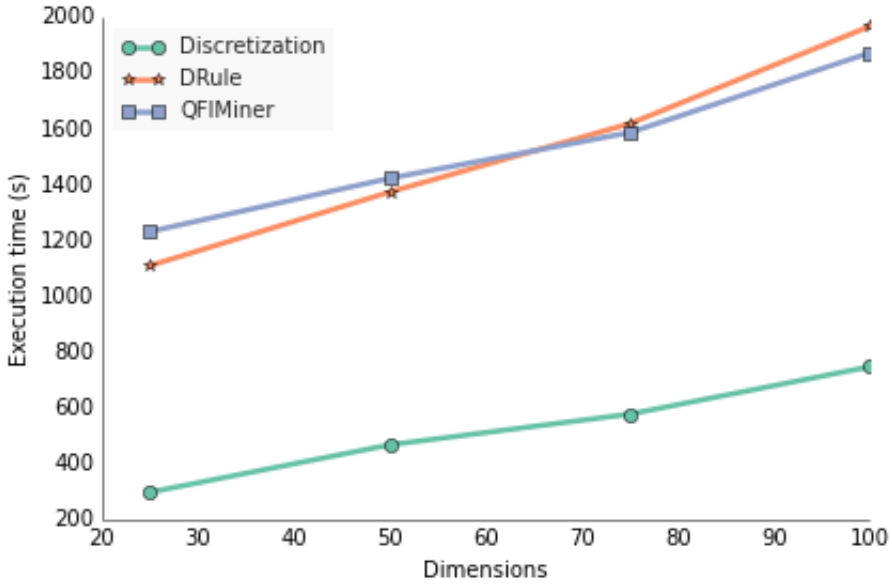

Fig. 3. The algorithm's susceptibility to noise. The dataset contains 10000 items and contains one interesting region.

rather quickly. After this step, uninteresting regions are no longer considered. For this test, we kept the size and amount of interesting regions constant, while introducing additional noisy dimensions with a uniform distribution.

We can see that our algorithm slightly outperforms QFIMiner in terms of efficiency for a lower amount of dimensions, while being slightly outperformed in higher-dimensional data. We outperform QFIMiner in lower-dimensional datasets due to the apriori-like build-up of the dimensions it considers. This approach wastes some time on trying to combine irrelevant dimensions. Due to our reduction step, which efficiently groups dimensions worth exploring together, we can avoid this problem. In higher-dimensional datasets we become less efficient as the amount of patterns we generate increases. Again, as mentioned in the previous test, QFIMiner misses a lot of these patterns.

Real Data

After testing our method on synthetic data to benchmark its efficiency, noise sensitivity, amount of rules it output, and its ability to detect overlap, we move on to real data. We subjectively evaluate the result of our algorithm to illustrate that our algorithm is able to find good rules in the sense that they are very interpretable by the user. To perform this evaluation, we used well-known datasets from the Bilkent University Function Approximation Repository [2], namely the

basketball and *quake* datasets, and also the *iris* dataset. In Table 2 we present some of the rules found by DRule .

Examining these results, we can immediately see that they are very easy to interpret. Consider for example, the three rules found for the *basketball* dataset. We can immediately conclude that shorter players pass the ball around much more than taller players. Also notable is that the intervals of Assists per minute actually overlap. While a seemingly small difference, it can prove useful for maximizing your bets.

Our algorithm ran for 1.21, 0.89, and 0.57 seconds for each of these datasets respectively while QFIMiner ran for 1.02, 0.78, and 0.56 seconds respectively.

Table 2. Table listing example rules found for real datasets.

Examples (confidence (%))
Iris
Sepal width$[2.3, 4.2]$, Petal Length$[1.2, 1.6] \rightarrow$ Sepal length$[4.4, 5.5]$ (98)
Class[Iris-versicolor] $\rightarrow$ Petal Length$[3, 5.1]$ (100)
Petal length$[4.8, 6.9] \rightarrow$ Petal width$[1.4, 2.5]$ (100)
Quake
Latitude$[49.83, 50.05]$, Longitude$[78.69, 79.1] \rightarrow$ Focal depth$[0, 9]$ (100)
Longitude$[78.69, 79.1]$, Richter$[5.9, 6.1] \rightarrow$ Latitude$[49.83, 50.01]$(95.1)
Focal Depth$[0, 24]$, Longitude$[78.69, 79.1] \rightarrow$ Richter$[5.9, 6.1]$ (71.9)
Basketball
Height$[196, 198] \rightarrow$ Assists per min$[0.09, 0.14]$ (72)
Height$[183, 185] \rightarrow$ Assists per min$[0.12, 0.29]$ (95.5)
Height$[191, 193]$, Points per min$[0.39, 0.49] \rightarrow$ Assists per min$[0.07, 0.23]$ (100)

5 Related Work

Association rule mining was first introduced by Agrawal et al. [1]. Since its introduction, different methods have been extensively studied, as can be demonstrated by the numerous optimizations that have been developed for the Apriori algorithm. However, since these algorithms were developed for discrete values, the so called supermarket basket analysis, they fall short when dealing with numeric attributes. Over the years, many different approaches have been developed to deal with these kinds of attributes. In the following paragraphs, we will give an overview of these approaches and classify them according to how they relate to our approach.

Discretization/Partitioning The first and classical method of dealing with numeric attributes is to perform a preprocessing step, namely partitioning the data (discretization, binning). This method groups similar items into the same bin. When the data has been partitioned, frequent itemset mining techniques can freely be applied [15, 21].

Unfortunately, most of these methods fall short as they perform a univariate discretization, i.e., every dimension is partitioned separately without taking correlations into account. Since quantitative association rules (QAR) generally encompass more than one dimension, these interactions between dimensions remain important. To solve the problem of univariate discretization we can use a multivariate discretization, see for example Bay [4]. These methods attempt to preserve interactions between dimensions. While they improve upon the traditional methods, they still show some shortcomings, such as the assigning each point to a unique bin. This can result in many missed patterns.

A more promising approach to discretization is through the use of fuzzy rules. These allow for some overlap between rules but still have the downside of having to define boundaries beforehand [20].

Clustering-based approaches Another noteworthy attempt at partitioning the data is through the use of clustering algorithms [8]. This approach clusters the data and uses the resulting clusters as boundaries for the bins. Traditional clustering algorithms were designed to handle all attributes simultaneously, which can become a problem as the dimensionality of the data increases. A workaround for the increasing dimensionality of the data has been created through the development of subspace clustering algorithms, e.g. SUBCLU [13] and the FIRES algorithm [12]. A drawback of these techniques is, however, that they are computationally expensive and that they were not designed for rule extraction.

An approach based on the SUBCLU algorithm, which is capable of finding quantitative frequent itemsets, was introduced by Washio et al. [24]. This techniques first generates a partition of the data in each dimension and then uses the anti-monotonicity property of density to combine dimensions. Drawbacks of this technique include: restriction to non-overlapping quantitative predicates, fixed order of processing attributes and thus loss of some rule, and a limitation to quantitative frequent itemsets with no extraction of QARs.

Optimization of Intervals Other than partitioning approaches, several techniques that allow for non-discretized numeric attributes also exist. These algorithms typically attempt to optimize some criteria in order to interesting rules. The first approach, developed by Fukuda et al. [7] uses techniques from image segmentation to find a region that produces an optimized association rule. This technique was further expanded by Brin et al. [5] to allow for an extra dimension in the rules. A shortcoming is that these techniques are limited in the number of dimensions a rule can contain and that expanding them to higher-dimensional rules is non-trivial. Most recently, Tatti [22] proposes a binarization for real-valued data by selecting sets of thresholds, treating them as random variables and computing the average support.

A wildly different approach was first introduced by Mata et al. [14] and further expanded upon by Salleb-Aouissi et al.[19]. They use a genetic algorithm to find optimal rules according to a rule template. This template has to be passed to the algorithm, so the user has to know which types of rules to look for beforehand.

A last type of techniques operate in a framework of Formal Concept Analysis, and find all possible non-equivalent (or so called closed) vectors of intervals covering a minimal number of points in the data [10, 9]. Unfortunately, such techniques are not feasible for large databases.

6 Conclusion

In the present paper, we have introduced a novel method to tackle the problem of finding overlapping quantitative association rules. The rules we find allow for both numeric and categoric attributes and our algorithm is thus more general than many of its competitors. We have demonstrated the algorithm's efficiency and scalability and have shown its resistance to noise. To (partially) solve the problem of increasing rule output, we have also introduced two strategies that decrease the amount of rules the algorithm outputs. Finally, we have demonstrated that DRule is capable of finding association rules showing overlap where other algorithms fail to do so, or only find an incomplete set. Testing DRule on real data leads us to conclude that it has its merit in the field of quantitative association rule mining.

7 Future Work

In the future, we aim to explore different quality measures and different heuristics, other than density, for our rules evaluation and discovery to allow for a more efficient and a more accurate algorithm.

Other than these optimizations, we intend to investigate methods to make our method more robust. That is, currently, as is the case with most ARM algorithms, small changes in parameters can lead to drastic increases in runtime. While a lot of pruning is already done by our algorithm, this can still lead to an increase in the ordering of dimensions that have to be visited.

References

1. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. ACM SIGMOD 22(2), 207–216 (1993)
2. Altay Guvenir, H., Uysal, I.: Bilkent university function approximation repository (2000), http://funapp.cs.bilkent.edu.tr
3. Aumann, Y., Lindell, Y.: A statistical theory for quantitative association rules. In: ACM SIGKDD. pp. 261–270 (1999)
4. Bay, S.D.: Multivariate discretization for set mining. Knowl. Inf. Syst. 3(4), 491–512 (2001)
5. Brin, S., Rastogi, R., Shim, K.: Mining optimized gain rules for numeric attributes. IEEE Trans. Knowl. Data Eng. 15(2), 324–338 (2003)
6. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: ACM SIGKDD. pp. 226–231 (1996)

7. Fukuda, T., Morimoto, Y., Morishita, S., Tokuyama, T.: Mining optimized association rules for numeric attributes. J. Comput. Syst. Sci. 58(1), 1–12 (1999)
8. Grzymala-Busse, J.W.: Three strategies to rule induction from data with numerical attributes. Transactions on Rough Sets 3135, 54–62 (2005)
9. Kaytoue, M., Kuznetsov, S.O., Napoli, A.: Revisiting numerical pattern mining with formal concept analysis. International Joint Conference on Artificial Intelligence (IJCAI) arXiv preprint arXiv:1111.5689 (2011)
10. Kaytoue, M., Kuznetsov, S.O., Napoli, A., Duplessis, S.: Mining gene expression data with pattern structures in formal concept analysis. Information Sciences 181(10), 1989–2001 (2011)
11. Ke, Y., Cheng, J., Ng, W.: Mic framework: an information-theoretic approach to quantitative association rule mining. In: IEEE ICDE. pp. 112–112 (2006)
12. Kriegel, H.P., Kröger, P., Renz, M., Wurst, S.H.R.: A generic framework for efficient subspace clustering of high-dimensional data. In: IEEE ICDM. pp. 250–257 (2005)
13. Kröger, P., Kriegel, H.P., Kailing, K.: Density-connected subspace clustering for high-dimensional data. In: SIAM SDM. pp. 246–256 (2004)
14. Mata, J., Alvarez, J.L., Riquelme, J.C.: An evolutionary algorithm to discover numeric association rules. In: ACM SAC. pp. 590–594 (2002)
15. Miller, R.J., Yang, Y.: Association rules over interval data. ACM SIGMOD 26(2), 452–461 (1997)
16. Müller, E., Assent, I., Günnemann, S., Seidl, T.: Scalable density-based subspace clustering. In: ACM CIKM. pp. 1077–1086 (2011)
17. Müller, E., Assent, I., Krieger, R., Günnemann, S., Seidl, T.: DensEst: Density estimation for data mining in high dimensional spaces. In: SIAM SDM. pp. 175–186 (2009)
18. Müller, E., Günnemann, S., Assent, I., Seidl, T.: Evaluating clustering in subspace projections of high dimensional data. PVLDB 2(1), 1270–1281 (2009)
19. Salleb-Aouissi, A., Vrain, C., Nortet, C., Kong, X., Rathod, V., Cassard, D.: Quantminer for mining quantitative association rules. Journal of Machine Learning Research 14(1), 3153–3157 (2013)
20. Serrurier, M., Dubois, D., Prade, H., Sudkamp, T.: Learning fuzzy rules with their implication operators. Data Knowl. Eng. 60(1), 71–89 (2007), http://dx.doi.org/10.1016/j.datak.2006.01.007
21. Srikant, R., Agrawal, R.: Mining quantitative association rules in large relational tables. In: ACM SIGMOD. pp. 1–12 (1996)
22. Tatti, N.: Itemsets for real-valued datasets. In: IEEE ICDM. pp. 717–726 (2013)
23. Vannucci, M., Colla, V.: Meaningful discretization of continuous features for association rules mining by means of a som. In: ESANN. pp. 489–494 (2004)
24. Washio, T., Mitsunaga, Y., Motoda, H.: Mining quantitative frequent itemsets using adaptive density-based subspace clustering. In: IEEE ICDM. pp. 793–796 (2005)
25. Webb, G.I.: Discovering associations with numeric variables. In: ACM SIGKDD. pp. 383–388 (2001)
26. Wijsen, J., Meersman, R.: On the complexity of mining quantitative association rules. Data Min. Knowl. Discov. 2(3), 263–281 (1998)
27. Zhu, F., Yan, X., Han, J., Yu, P.S., Cheng, H.: Mining colossal frequent patterns by core pattern fusion. In: IEEE ICDE. pp. 706–715 (2007)