Avoiding the Hay for the Needle in the Stack: Online Rule Pruning in Rare Events Detection

Ioannis T. Christou

Big Data Mining Group, Athens Information Technology, Monumental Plaza, Bld. C, 44 Kifisias Ave., Marousi 15125, Greece,

ichr@ait.edu.gr

Abstract—In certain machine learning application domains such as those that arise in Industry 4.0 settings that involve rare events detection and prediction, classifiers based on Rule Mining often have an advantage over other algorithms due to their design that allows them to deal with highly skewed class distributions. We describe R4RE, an algorithm that mines *all* quantitative association rules from large datasets that meet certain user-defined criteria of interestingness that is not limited to the classical support/confidence framework. When dealing with very small classes, the possibility arises that rules with very small support may be valid in the training set that do not hold in the test set. To avoid generating such rules, we implement the approach found in the IREP and other systems where rules may be pruned immediately after discovery. Results from real world factory data show the efficiency of our approach for predictive maintenance purposes.

Index Terms—I.2.6.g-Machine Learning, G.2.1.a-Combinatorial Algorithms

I INTRODUCTION

Quantitative Association Rule Mining (QARM) is a very active research in Data Mining and in the Association Rule Mining in particular. It has been shown [1] that mining for interesting rules in datasets related to health prognosis and maintenance of machines and tools in smart factory shop-floors is an approach that often outperforms other more traditional approaches including Neural Networks (shallow or deep ones). The QARMA algorithm described in [1] is a highly parallel/distributed algorithm that implements an efficient Breadth-First-Search method with optimized short-cuts that can produce all interesting (widest and/or non-dominated) rules that can be extracted from any given multi-dimensional dataset. In the context of Predictive Maintenance (PdM) and Remaining Useful Life (RUL) prediction, the dataset consists of a set of sensor readings, including possibly process data such as raw material characteristics etc. together with an estimate of the RUL quantity computed at each measurement. The rules produced will always have as their consequent item the RUL quantity -quantified either from the left, or from the right, resulting in a right-open or left-open interval as the RUL estimate of the rule when it fires. Such rule-sets can then form the basis of a predictor in a straight-forward manner, in that given a set of sensor readings, each extracted rule is tested to see if it fires, and if so, its estimate is recorded; a (possibly weighted) average of the recorded estimates then forms the final regression estimate of the rule-set.

However, experiments on real-world datasets show that due to the fact that the minimum required support threshold for the rules to be produced has to be set to sufficiently low levels for interesting rules to be valid for discovery in the training set, the output ruleset often contains rules that never trigger or, even worse, have far worse performance on the test set.

We present "Rules 4 Rare Events" (R4RE), an algorithm that solves two serious short-comings of the QARMA algorithm on which it is based on: (a) It allows quantifications of the consequent item in closed intervals to allow better estimates of the target RUL feature, and (b) it addresses the high error rates in the test set problem by incorporating online pruning of the generated rules within its search process, in the spirit of algorithms such as IREP [2] and RIPPER [3].

The results from a real-world Use-Case show the competitiveness of the approach on field data.

A. Related Work

Piatetsky–Shapiro [4] presented the first algorithm for QARM in 1991, even though the exposition was for single antecedent and single consequent attributes. Srikant & Agrawal [5] showed how to overcome the problem of finding summaries for all combinations of attributes (which is exponentially large) required for a straight–forward extension of Piatetsky–Shapiro's algorithm to multiple antecedent and/or consequent attributes in the rules by a decomposition/partitioning of the quantitative attributes' intervals followed by possible merging and pruning the search space of candidate itemsets. Salleb–Aouissi et al. [6] developed QuantMiner, a Genetic Algorithm (GA) for mining interesting Quantitative Association Rules (QARs) via an optimization process; the idea of optimizing QARs originally appeared in Fukuda et al. [7] where however the algorithms presented aim to find intervals for quantitative attributes that maximize support and/or confidence of the produced rules. The work of Ruckert et al. [8] is also related to our work; in this paper the authors proposed looking for half-space conditions on the quantitative attributes instead of hyper-rectangular representations. Very recently, Map/Reduce-based approaches to Association Rule Mining were proposed in [9], and in [10] the authors present a framework for executing any QARM algorithm implementation on Apache Spark as a Map/Reduce job.

II THE R4RE SYSTEM

A. Data Model

The model for the *input* data of the system is shown in fig. 1 below. As can be seen from the figure, *our input data model is significantly more complicated than in standard tabular format data repositories*: the database consists of a set of user histories that "consume" items at certain points in time; an item can be "consumed" multiple times (at different points in time) by the same user, allowing for example multiple sensor readings of the same machine at different times before the machine breaks down and requires maintenance. In the context of PdM, a user history can be the history of a machine/component/tool configuration, running until failure. Items correspond to sensors (piezo, acoustic, vibration sensors etc.) attached to the machine, and they are consumed each time a measurement is taken and recorded for them.

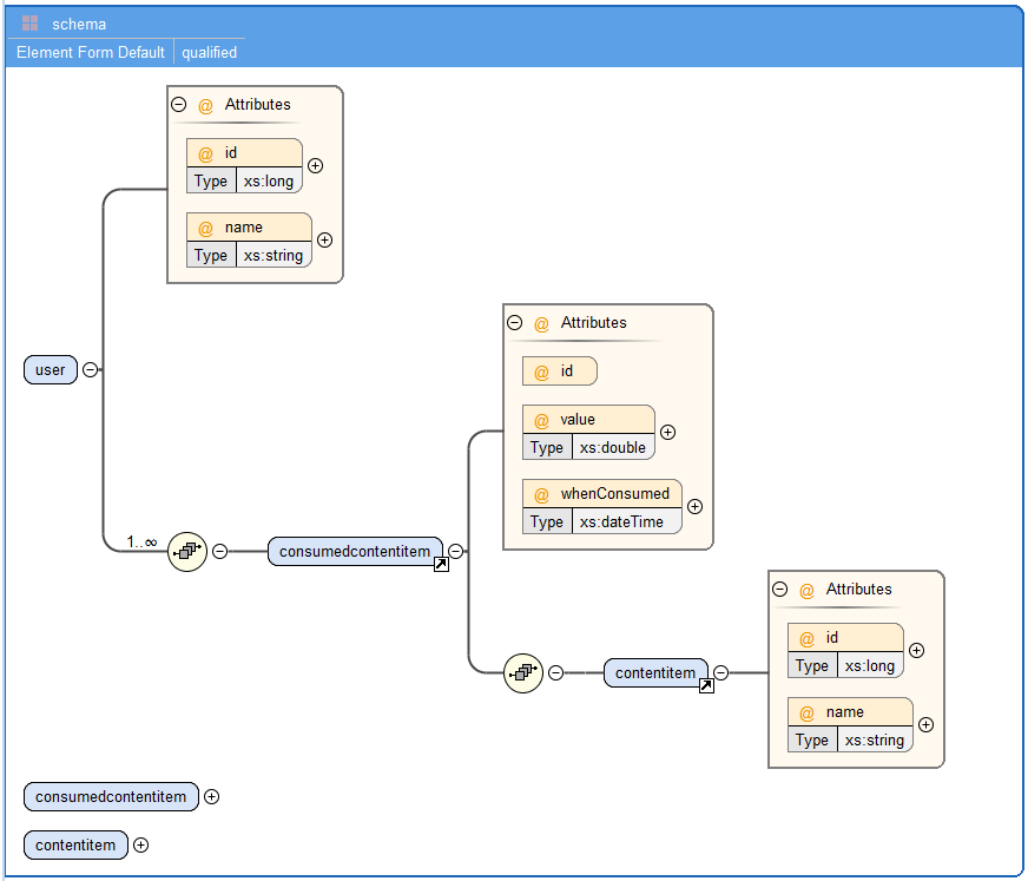


Figure 1: Input Data Model for R4RE

B. Further Innovations

As mentioned above, the first innovation of R4RE is the fact that it deals directly with multiple appearances of items in instances ("user histories"), an approach allowing for the straight-forward modeling of the input data. Furthermore, to our knowledge R4RE is the only system for mining QARs that incorporates both online pruning of rules, "stealing" ideas from classical rule induction approaches, and allows for lower-bounding as well as upper-bounding the consequent item's value in every rule produced; both the latter features are missing from the original QARMA framework; R4RE nevertheless inherits from QARMA its fully parallel/distributed nature, and guarantees that all produced rules meet all interestingness criteria set by the user, and that there are no "useless" rules in the result (notion discussed below, after Definition 1, and in much more detail in [1]).

C. Definitions

In the following, we assume that S denotes the set of items in our database of user histories D, and that Q denotes a finite set of tuples of the form (i, v) where $i \in S, \quad v \in \mathbb{R}$. The item i forms a key of that set so that no two tuples with the same item can exist in Q. We denote a quantitative association rule of the form as $r = (B \to I \in [l, h] | Q)$ where $B \subset S, I \in S$, with the following interpretation: with sufficiently high support and interestingness, the existence of all items in B in a user history t such that for each item $i \in B$ that appears in any pair $(i, v) \in Q$ there exists a transaction in t for that item with value $v_a \geq v$ imply the existence of I in that user's set of historical transactions, and that the value of item I is in the interval $[l, h]$. For a given dataset D we denote the set of all possible quantitative rules in that set by R^D (not to be confused with the d-dimensional real space $\mathbb{R}^d$.)

Definition 1. (Rule Confidence) A rule $r = (B \to I \in [l, h] | Q)$ has **confidence** c denoted by $CONFIDENCE(r)$ that equals the number of user histories that contain each of the items in B with values at least equal to the value specified for that item–value pair in Q **and** the value of item I is in the interval $[l, h]$, **divided** by the number of user histories in which each of the items in B appears with value at least equal to the value specified for that item–value pair in Q. If an item does not appear in any of the pairs in Q, the condition is reduced to the mere existence of the item in one or more transactions of the user history, regardless of value. We define the notions of rule support, conviction, and lift in an analogous manner; we also define the notion of rule dominance and widening in the same manner as in [1], but for the special case of items having a single attribute only. The notion of dominance is introduced so as to *discard valid* (satisfying minimum support and interestingness) *but useless rules*: the dominated rule provides no new information, in that, whenever the dominated rule fires and signals a value interval for the consequent item I, the dominant rule also fires, and signals an interval enclosed in the dominated rule's interval (providing equal or more information), with equal or higher support and interestingness values. The *widening* relation between rules is the same as dominance, but without the requirement that the dominant rule must have same or better support and interestingness metrics.

Definition 2. (Rule Coverage) The covering set of a quantitative rule $r = (B \rightarrow I \in [l,h]|Q)$ is defined as the subset of user-histories in the dataset for which the rule holds: $COVERAGE(r) = \{t \in D: (\forall (i \in B, v) \in Q): (\exists (i, v' \geq v) \in t) \wedge \exists (I, v') \in t: v' \in [l,h]\}$. The covering set of a set of rules $RS \subseteq R^D$ is defined as the union of the covering sets of each member of that rule–set: $COVERAGE(RS) = \cup_{r \in RS} COVERAGE(r)$. From the definition of support, we have $SUPPORT(r) = |COVERAGE(r)|/|D|$.

D. Algorithm Design

The definition of what constitutes the input database in section 2.1 allows for *multiple* "consumptions" of the same item by the same user, which is very common in most cases R4RE is designed for application. As an example, in patient medical records, the same patient over a period of time, has the same medical test (e.g. blood pressure measurement) multiple times, and so on. Items are assumed to be in a transitive order relationship, given by a map $ord: S \rightarrow \mathbb{N}$ (that can be the sequence id by which items are stored in the db.) The algorithm is a highly distributed Breadth-First algorithm that works in phases as follows: first, the "users" in the database are divided randomly into two subsets: a **GROWTH** set, and a **PRUNE** set in a 2:1 ratio (as in [2,3]). Then, all the frequent itemsets in the **GROWTH** database that contain the particular item whose value we want to predict (the consequent) are identified using any algorithm for frequent itemset generation such as FP-Growth [11], and a result-set R is initialized to the empty set. In the final and main phase, for each set of frequent itemsets of length k=2,3,4… R4RE considers every itemset in it in parallel (in any machine available for computation), setting as consequent the target item, and starts quantifying the resulting association rule in a Breadth-First Search as follows: it first quantifies the consequent item's interval, with values searched in the order shown in figure 2. This search order for the consequent item's value is such that if a constructed rule r meets its support and interestingness metrics (i.e. is a valid rule) and at the time of construction no other previously found rule is wider than r, it is guaranteed that no valid rule exists that is wider than r. Immediately after quantifying the consequent item's interval, the partially quantified rule is checked for its support level: if it exceeds the minimum support threshold required on the **GROWTH** set, it enters a FIFO-queue T, otherwise the same rule's consequent item is quantified with the next interval in the search order. Then, while the queue T is non-empty, the first rule in the queue is popped from it, and its items in the antecedents are quantified, one at a time, with the values they take on in the database, examined in ascending order. If the resulting support meets threshold requirements on the **GROWTH** set, it enters T; in addition, if the rule exceeds the interestingness thresholds *on the* **PRUNE** *set*, and no other rule in set R is wider than the current one, the current rule enters R and we stop any further quantifications of this rule. Finally, the set R is returned. The exact details of the R4RE algorithm can be found in Appendix 1.

Notice that the algorithm as described is especially well–suited to both a *shared–memory multi–processing model which is commonly found in today's multi/many–core computers* as well as to a fully distributed computational environment. The distributed nature of R4RE is shown in the deployment diagram in fig. 3.

III. COMPUTATIONAL RESULTS

Within the context of the PROPHESY project, we were given access to a database of sensor readings of factory machines/tools configurations for a period between October and December 2018. From this initial database, we were able to extract a dataset that included measurements of four basic quantities taken at 5700 different times during the 3-month period mentioned above. We associated each measurement with a single target attribute, namely the time (in hours) remaining to failure ("breakage") of the machine/tool being monitored, referred to as RUL-time. The dataset was augmented by another 10 attributes corresponding to a window of 7 time-points before for one of the attributes, and a window of the previous time-point measurement for each of the remaining 3 attributes, resulting in a total of 14 basic attributes to be used for predicting the target RUL-time attribute. We split this dataset into a training dataset with 5000 rows, and a test dataset of the remaining 700 rows. The R4RE algorithm, applied to this augmented training dataset, produced about 4500 rules. For each time-stamp on the test

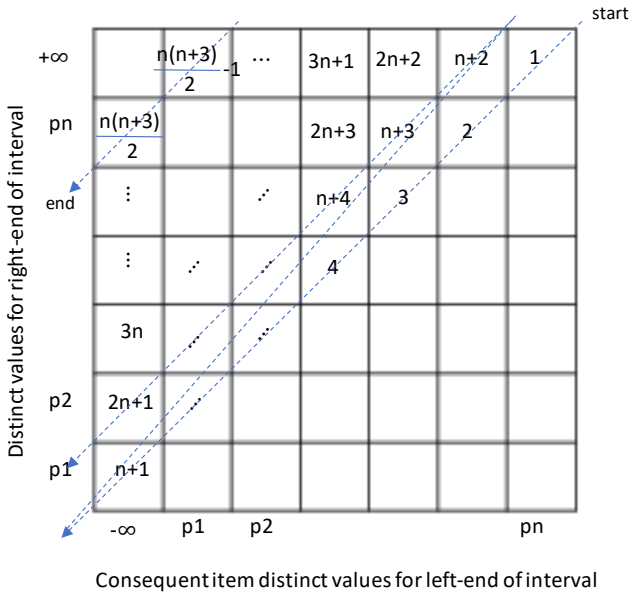


Figure 2: Consequent Item Search Order (order shown inside each cell). The first interval to be examined for the consequent item is $[p_n, +\infty)$, then $[p_{n-1}, p_n]$ ... then $(-\infty, p_1]$, then $[p_{n-1}, +\infty)$, $[p_{n-2}, p_n]$, $[p_{n-3}, p_{n-1}]$… and finally $(-\infty, p_n]$. Cell (p_i, p_j) represents the interval $[p_i, p_j]$

set, we produce a RUL-time estimate as a weighted average of the minimum, the average, and the maximum estimate of RUL-time from any rule that fires on the row for that timestamp.

We have implemented R4RE in Java 8, using the QARMA codebase [1], as well as the popt4jlib Open Source library [12]. In Table I we show the performance of our algorithm on the held-out test set, measured in terms of root mean square error, mean absolute error, and mean absolute percentage error and we compare against another 8 well-known Machine Learning algorithms for Regression found in the well-known WEKA toolkit [13]: M5-Rules (M5Rules), SVM for regression (SMOReg), Linear Regression (LR), Radial Basis Function Network (RBFNet), Multi-Layer Perceptron (ANN), Additive Regression (AddReg), Bagging (Bag), and Decision Tables (Dec.Table). As can be seen, R4RE compares well with all other methods, on all metrics measured.

Furthermore, using an expanded database from the same manufacturer, we were able to extract a dataset that included measurements of eight basic quantities taken at 5632 different times during the 3-month period mentioned above, but this time we could associate each measurement with the parts (in units) remaining to failure ("breakage") of the machine/tool being monitored, referred to as RUL-parts. The RUL measured as parts is a much more robust quantity as it does not account for the time when the machine is idle or off (e.g. during weekends). The dataset was augmented as in the 1st test-case to a total of 17 attributes (items). We split the entire dataset in the same way as before, keeping 4800 instancs for training and the rest for testing. In Table II, we compare the results of R4RE against the same set of 8 ML algorithms as before, and the results are again favorable for the use of R4RE on PdM applications (R4RE is best in terms of MAPE%, beats all with good mar-

gin except M5Rules and Bagging in all measures); a graphical depiction of the quality of the produced rules (selecting a rule from the list of all rules produced, showing in the bottom table all instances on which the rule triggers) is shown in Fig. 4.

Table I: Regression Results for **RUL-time** measured in seconds remaining on Unseen Testset1 Data

Metric\Method	R4RE	M5Rules	SMOReg	LR	RBFNet	ANN	AddReg	Bag	Dec.Table
RMS	**34.2**	101.4	173.8	144	67.7	125.1	120.8	93	90.4
MAE	**28.7**	79	167	137.3	65.35	102.1	112	77.1	72.8
MAPE%	**20.1**	201.2	426.6	350.5	166.9	260.7	286	196.7	185.9

Table II: Regression Results for **RUL-parts** measured in parts remaining on Unseen Testset2 Data

Metric\Method	R4RE	M5Rules	SMOReg	LR	RBFNet	ANN	AddReg	Bag	Dec.Table
RMS	668.7	371.8	1150.5	988.2	1123.5	509.8	717.5	**365.6**	582.1
MAE	120.8	**57.8**	445.9	577.6	979.3	220	383.1	58.8	136.5
MAPE%	**3.76**	3.91	30.16	39.06	66.2	14.8	25.9	3.98	9.23

IV. Conclusions & Future Directions

We presented R4RE, a novel highly parallel distributed algorithm for deriving all quantitative association rules in datasets containing many quantitative attributes. The algorithm incorporates online learning techniques to prune rules that may pass support and interestingness thresholds by chance, and produces only non-dominated such rules that maximize dataset coverage! We have shown that R4RE seems to be well-suited to Predictive Maintenance applications, and its performance on a real-world dataset compares well to that of all other algorithms we have tested it against.

In the near future, we plan to test the R4RE algorithm on more real-world datasets as they become available, and compare its performance to that of Recurrent Neural Networks, Support Vector Machines, Linear Regression, and other techniques. Further incorporating online learning (also known as continuous learning) techniques into R4RE so as to allow incremental processing of the dataset as more user histories are added into the data, or as more data are added to existing user histories, without the need to process from scratch the entire dataset, is another very important and

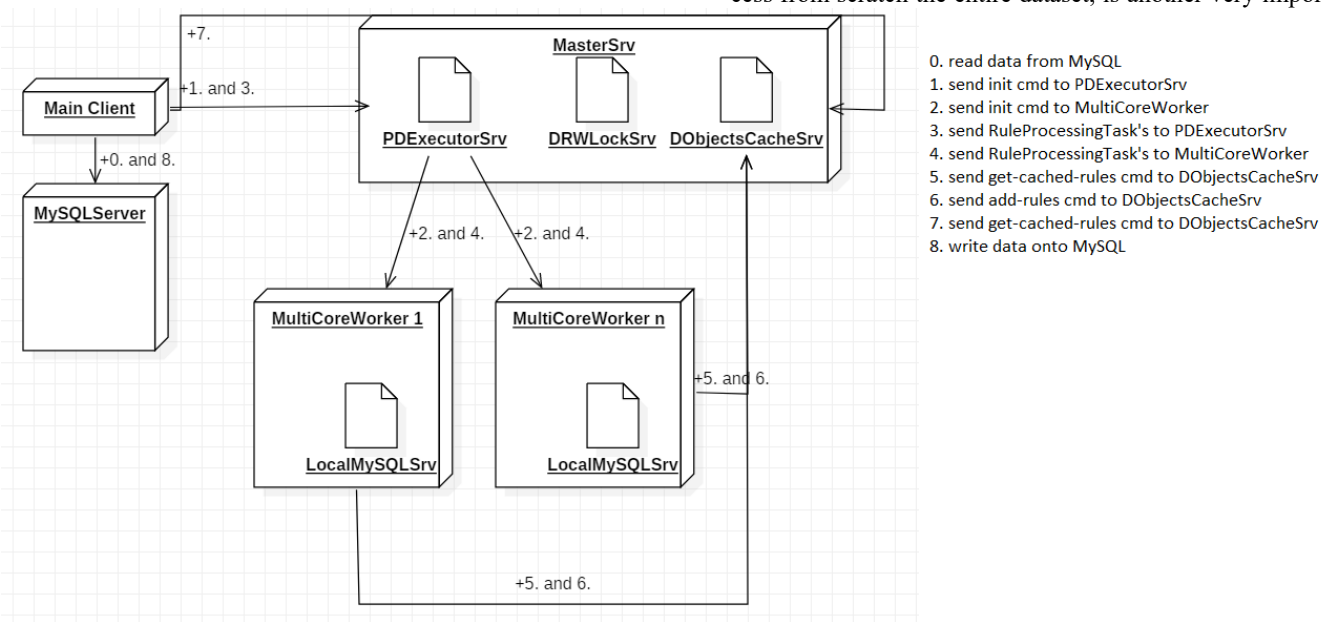
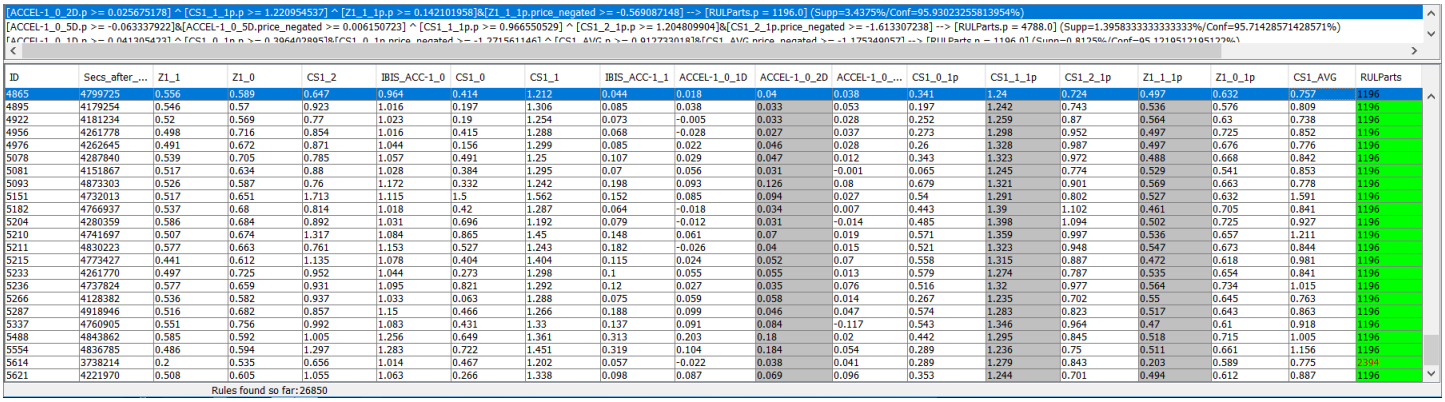


Figure 3: R4RE Distributed Deployment Architecture



ID	Secs_after_...	Z1_1	Z1_0	CS1_2	IBIS_ACC-1_0	CS1_0	CS1_1	IBIS_ACC-1_1	ACCEL-1_0_1D	ACCEL-1_0_2D	ACCEL-1_0_...	CS1_0_1p	CS1_1_1p	CS1_2_1p	Z1_1_1p	Z1_0_1p	CS1_AVG	RULParts
4865	4799725	0.556	0.589	0.647	0.964	0.414	1.212	0.044	0.018	0.04	0.038	0.341	1.24	0.724	0.497	0.632	0.757	1196
4895	4179254	0.546	0.57	0.923	1.016	0.197	1.306	0.085	0.038	0.033	0.053	0.197	1.242	0.743	0.536	0.576	0.809	1196
4922	4181234	0.52	0.569	0.77	1.023	0.19	1.254	0.073	-0.005	0.033	0.028	0.252	1.259	0.87	0.564	0.63	0.738	1196
4956	4261778	0.498	0.716	0.854	1.016	0.415	1.288	0.068	-0.028	0.027	0.037	0.273	1.298	0.952	0.497	0.725	0.852	1196
4976	4262645	0.491	0.672	0.871	1.044	0.156	1.299	0.085	0.022	0.046	0.028	0.26	1.328	0.987	0.497	0.676	0.776	1196
5078	4287840	0.539	0.705	0.785	1.057	0.491	1.25	0.107	0.029	0.047	0.012	0.343	1.323	0.972	0.488	0.668	0.842	1196
5081	4151867	0.517	0.634	0.88	1.028	0.384	1.295	0.07	0.056	0.031	-0.001	0.065	1.245	0.774	0.529	0.541	0.853	1196
5093	4873303	0.526	0.587	0.76	1.172	0.332	1.242	0.198	0.093	0.126	0.08	0.679	1.321	0.901	0.569	0.663	0.778	1196
5151	4732013	0.517	0.651	1.713	1.115	1.5	1.562	0.152	0.085	0.094	0.027	0.54	1.291	0.802	0.527	0.632	1.591	1196
5182	4766937	0.537	0.68	0.814	1.018	0.42	1.287	0.064	-0.018	0.034	0.007	0.443	1.39	1.102	0.461	0.705	0.841	1196
5204	4280359	0.586	0.684	0.892	1.031	0.696	1.192	0.079	-0.012	0.031	-0.014	0.485	1.398	1.094	0.502	0.725	0.927	1196
5210	4741697	0.507	0.674	1.317	1.084	0.865	1.45	0.148	0.061	0.07	0.019	0.571	1.359	0.997	0.536	0.657	1.211	1196
5211	4830223	0.577	0.663	0.761	1.153	0.527	1.243	0.182	-0.026	0.04	0.015	0.521	1.323	0.948	0.547	0.673	0.844	1196
5215	4773427	0.441	0.612	1.135	1.078	0.404	1.404	0.115	0.024	0.052	0.07	0.558	1.315	0.887	0.472	0.618	0.981	1196
5233	4261770	0.497	0.725	0.952	1.044	0.273	1.298	0.1	0.055	0.055	0.013	0.579	1.274	0.787	0.535	0.654	0.841	1196
5236	4737824	0.577	0.659	0.931	1.095	0.821	1.292	0.12	0.027	0.035	0.076	0.516	1.32	0.977	0.564	0.734	1.015	1196
5266	4128382	0.536	0.582	0.937	1.033	0.063	1.288	0.075	0.059	0.058	0.014	0.267	1.235	0.702	0.55	0.645	0.763	1196
5287	4918946	0.516	0.682	0.857	1.15	0.466	1.266	0.188	0.099	0.046	0.047	0.574	1.283	0.823	0.517	0.643	0.863	1196
5337	4760905	0.551	0.756	0.992	1.083	0.431	1.33	0.137	0.091	0.084	-0.117	0.543	1.346	0.964	0.47	0.61	0.918	1196
5488	4843862	0.585	0.592	1.005	1.256	0.649	1.361	0.313	0.203	0.18	0.02	0.442	1.295	0.845	0.518	0.715	1.005	1196
5554	4836785	0.486	0.594	1.297	1.283	0.722	1.451	0.319	0.104	0.184	0.054	0.289	1.236	0.75	0.511	0.661	1.156	1196
5614	3738214	0.2	0.535	0.656	1.014	0.467	1.202	0.057	-0.022	0.038	0.041	0.289	1.279	0.843	0.203	0.589	0.775	2394
5621	4221970	0.508	0.605	1.055	1.063	0.266	1.338	0.098	0.087	0.069	0.096	0.353	1.244	0.701	0.494	0.612	0.887	1196

Figure 4: Visualization of R4RE Rule Results: The selected rule's antecedents are greyed in the bottom table, and the consequent (the RUL-parts attribute, last column) is showing in green; incorrect predictions marked with red pencil. Only triggering instances shown in bottom table.

exciting future line of work.

ACKNOWLEDGEMENTS

Part of this work has been carried out in the scope of the H2020 PROPHESY project (contract number 766994), which is co-funded by the European Commission in the scope of its H2020 program. The author acknowledges valuable help and contributions from all partners of the project. The work is also partially supported by CHIST-ERA (call 2017 on "Big Data for Industry 4.0") via the FIREMAN consortium, funded by national foundations, including the Hellenic General Secretariat of Research and Technology.

REFERENCES

[1] I.T. Christou, E. Amolochitis, Z.-H. Tan, "A parallel/distributed algorithmic framework for mining all quantitative association rules", arXiv preprint arXiv:1804.06764, 2018.
[2] J. Furnkranz, G. Widmer, "Incremental reduced error pruning", In: Proc. Intl. Conf. on Mach. Learn., New Brunswick, NJ. 1994.
[3] W. W. Cohen, "Fast effective rule induction", In: Proc. Intl. Conf. on Mach. Learn., 1995.
[4] G. Piatetsky-Shapiro, "Discovery, analysis, and presentation of strong rules", In: Piatetsky-Shapiro G., and Frawley, W.J. (eds), *Knowledge Discovery in Databases*, AAAI/MIT Press, Menlo Park, CA, 1991.
[5] R. Srikant and R. Agrawal, "Mining quantitative association rules in large relational tables", In: *Proc. ACM SIGMOD Conf. on Management of Data*, pp. 1-12, 1996.
[6] A. Saleb-Aouissi, C. Vrain, C. Nortet, "QuantMiner: a genetic algorithm for mining quantitative association rules", In: *Proc. International Joint Conference on Artificial Intelligence (IJCAI '07)*, 2007.
[7] T. Fukuda, Y. Murimoto, S. Morishita, T. Tokuyama, "Mining optimized association rules for numeric attributes", In: *Proc. 15th ACM Symposium on Principles of Database Systems (PODS '96)*, pp. 182-191, 1996.
[8] U. Ruckert, L. Richter, S. Kramer, "Quantitative association rules based on half-spaces: an optimization approach", In: *Proc. 4th Intl. Conf. on Data Mining (ICDM '04)*, pp. 507-510, 2004.
[9] F. Padillo, J.M. Luna, F. Herrera, S. Ventura, "Mining association rules on Big Data through MapReduce genetic programming", Integrated Computer-Aided Engineering 25:1, pp. 31-48, 2018.
[10] D. Martín, M. Martínez-Ballesteros, D. García-Gil, J. Alcalá-Fdez, J.C. Riquelme-Santos, F. Herrera, "MRQAR: a generic MapReduce framework to discover Quantitative Association Rules in Big Data problems", Knowledge-Based Systems 153, pp. 176-192, 2018.
[11] J. Han, J. Pei, Y. Yin, "Mining frequent patterns without candidate generation", In: *Proc. ACM SIGMOD Conf. on Management of Data*, pp. 1-12, 2000.
[12] https://github.com/ioannischristou/popt4jlib: Open Source library for parallel / distributed optimization in Java.
[13] I. H. Witten, E. Frank, M.A. Hall, "Data Mining: Practical Machine Learning Tools & Techniques", 3rd Ed. Morgan-Kaufmann, Burlington, MA. 2011.

APPENDIX 1: DESCRIPTION OF THE R4RE ALGORITHM

In the following, the procedure INIT() produces two results: (a) the set of all frequent itemsets in the **GROWTH** database, and (b) a map $P()$ mapping each item in the entire database D to a vector whose components are all the distinct values the item takes on D in increasing order. Finally, the predicate function Exists-wider() returns true when there is no rule in the rule-set passed as its second argument that is *wider* than the rule passed as its first argument.

R4RE Algorithm

INPUTS: Database D, target item I, minimum required support s, set of interestingness metrics $M \subset \{F: R^D \to \mathbb{R}\}$, minimum required interestingness thresholds $c(m)\ \forall m \in M$, ordering map $ord: S \to \mathbb{N}$.

OUTPUTS: All non-dominated QARs of the above form with support, interestingness above the minimum specified thresholds.

Begin
0. Randomly split the database D into *GROWTH* and *PRUNE* sets.
1. run procedure INIT($GROWTH, D, s, M, c(.)$) to produce set of all frequent itemsets F_s in *GROWTH* containing I and map P mapping each item J to $P(J)$: a vector whose components are all the distinct values item J takes on D in increasing order.
2. let $R = \emptyset$.
3. foreach $k = 2 \ldots max_{\ell \in F_s} \|\ell\|$ do:
  foreach unit of execution do:
    Atomically get global read-lock.
    set R_{local} = copy of R local to current unit of execution.
    Atomically release global read-lock.
  endfor.
  parallel foreach frequent k-itemset $\ell_k \in F_s$ do:
    set rule $r = (B = \ell_k \setminus \{I\} \to I)$.
    let $T = \emptyset$.
    // T is a FIFO queue of partially quatified rules
    foreach $[l, h]$ in the order shown in Figure 2 do:
      let $Q = \{\ \}, r = (B \to I \in [l, h] | Q)$.
      // set the values interval of I in descending order.
      if $SUPPORT(B \to I \in [l, h] | Q, GROWTH) < s$
        continue.
      Insert r onto T.
      while $T \neq \emptyset$ do:
        Remove the first $r = (B \to I \in [l, h] | Q)$ from T.
        let cont = true.
        foreach $J \in B$ do:
          if ¬cont break.
          if $ord(J) < max_{L \in B}\{ord(L): \exists v: (L, v) \in Q\}$
            continue. // avoid duplicate set creation
          endif
          foreach $j = 1 \ldots n_J \left(= dim(P(J))\right)$ do:
            let $Q' = Q \cup \{(J, P(J)_j)\}$.
            if $SUPPORT(B \to I \in [l, h] | Q', GROWTH) \geq s$
              Insert Q' onto T.
              if $\forall m \in M: m((B \to I \in [l, h] | Q'), PRUNE) \geq c(m)$
                if ¬(Exists-wider($(B \to I \in [l, h] | Q'), R_{local}$))
                  Add $(B \to I \in [l, h] | Q')$ to R_{local}.
                  set cont = false.
                  break.
                endif // no dominating rule exists
              endif // interestingness ok on the PRUNE set
            else break. // support not ok on GROWTH set
            endif. // support
          endfor. // j
        endfor. // J
      endwhile. // T
    endfor. // $[l, h]$ in sequence order of fig. 2
  endfor parallel. // loop over ℓ_k
  foreach unit of execution do:
    Atomically get global write-lock.
    set $R = R \cup R_{local}$.
    Atomically release global write-lock.
  endfor.
4. endfor. // k
5. return R.
End