Highly Scalable and Robust Rule Learner: Performance Evaluation and Comparison

Lukasz A. Kurgan, *Member, IEEE*, Krzysztof J. Cios, *Senior Member, IEEE*, and Scott Dick, *Member, IEEE*

***Abstract*—Business intelligence and bioinformatics applications increasingly require the mining of datasets consisting of millions of data points, or crafting real-time enterprise-level decision support systems for large corporations and drug companies. In all cases, there needs to be an underlying data mining system, and this mining system must be highly scalable. To this end, we describe a new rule learner called DataSqueezer. The learner belongs to the family of inductive supervised rule extraction algorithms. DataSqueezer is a simple, greedy, rule builder that generates a set of production rules from labeled input data. In spite of its relative simplicity, DataSqueezer is a very effective learner. The rules generated by the algorithm are compact, comprehensible, and have accuracy comparable to rules generated by other state-of-the-art rule extraction algorithms. The main advantages of DataSqueezer are very high efficiency, and missing data resistance. DataSqueezer exhibits log-linear asymptotic complexity with the number of training examples, and it is faster than other state-of-the-art rule learners. The learner is also robust to large quantities of missing data, as verified by extensive experimental comparison with the other learners. DataSqueezer is thus well suited to modern data mining and business intelligence tasks, which commonly involve huge datasets with a large fraction of missing data.**

***Index Terms*—Complexity, data mining, DataSqueezer, machine learning, missing data, rule induction, rule learner.**

I. Introduction

MACHINE Learning (ML) is one of the most popular tools used in the knowledge discovery process. If we define knowledge discovery as a nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns from large collections of data [34], then ML is one of the key tools used to perform data mining tasks in this process [6], [34], [17], [18].

ML is often defined as the ability of a computer program to improve its own performance at some task based on past experience, and as such is a very attractive vehicle for automated classification and generation of patterns [18], [44], [54], [57]. Over last few years ML attracted considerable attention due to the demand for reliable and useful data mining techniques in the information technology, medical, decision making, and gaming industries, to name but a few. ML is most frequently used to solve classification problems, perform intelligent data analysis, and to develop diagnostic, decision support, and expert systems [53], [59].

Our current work focuses on a class of ML algorithms called rule induction systems (or rule learners). A rule induction system takes as input a set of training examples, and produces a set of production rules or IF-THEN rules. Rule induction is distinct from decision trees; while it is trivial to produce a set of production rules given a decision tree (by creating rules that each represent a path from the root to a leaf on the tree), this is an issue of extracting rules from an existing, convenient data structure created by a different ML algorithm. Decision trees have drawn significant attention over the last several years, but the rule learners also exhibit a number of very desirable properties.

- They generate rules that are relatively easy for people to understand [8], which recently gained importance as a very desirable property since production rules appear to be more human-comprehensible than decision trees [14], [70]. People often learn from hypotheses generated by a rule learner, provided the hypothesis is in a human-comprehensible form. In this case, experts can even participate in the learning process by critiquing or improving the learned hypotheses.
- On some problems, rule learners were found to outperform decision trees [58], [69], [81], which shows that decision trees cannot be perceived as superior to rule learners
- The output of a rule leaner can easily be translated to a first-order logic representation, or embedded within knowledge-based or expert systems [25], [50]
- Certain types of prior knowledge were found to be easily communicated to rule learners [26], [60]
- Lastly, the rules can be modified and analyzed because of their modularity, i.e., a single rule can be understood without reference to other rules [40], which is very important when a decision maker needs to understand and validate the generated rules, as in medicine [50], [51].

In the light of the recent explosion of low-cost storage space and the growing interest of research and industrial communities in automated analysis of large quantities of data, the scalability of ML algorithms becomes one of the most important considerations. Applications such as intrusion detection or market basket analysis generate millions of data points daily. Other areas, such as genomics and proteomics are also approaching these data sizes. Recently published results of a 2003 survey on the largest and most heavily used commercial databases report that that the

Manuscript received June 4, 2004; revised November 21, 2004 and February 15, 2005. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada under Grants G121210953 and G121210906. This paper was recommended by Associate Editor D. J. Cook.

L. A. Kurgan and S. Dick are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 2VF, Canada.

K. J. Cios is with the Department of Computer Science and Engineering, University of Colorado at Denver and Health Sciences Center, Denver, CO 80217 USA, the Department of Computer Science, University of Colorado, Boulder, CO 80309 USA, and also with t 4cData, LLC, Golden, CO 80401 USA.

Digital Object Identifier 10.1109/TSMCB.2005.852983

biggest decision support system is now at 29.2 terabytes, which is about 300% larger than the largest system of 2001 [82]. Significant growth in the number of data points is also documented: the average size of Unix databases experienced a six-fold, and Windows databases a fourteen-fold, increase compared to year 2001; large commercial databases now average 10 billion data points [82]. An individual rule generation tasks is usually applied to results of a query posed to such database, which may return between several thousands and several millions of data points. Considering these very large databases, the primary concern is to be able to generate rules in a "reasonable" amount of time. For many applications that means to be able to generate the results in several minutes, or at most hours, because of the rapid rate of deterioration in the usefulness of the generated rules [71]. If the time to generate the rules is too long, then they may become obsolete before the user will have any chance of employing them. This issue is even more important in case of development of real-time decision support system based on ML learners. In this case, it becomes absolutely critical to use highly scalable learners that can provide the results within the predefined amount of time [29]. Many researchers have already recognized the importance of proposing rule learners that are characterized by scalability with the number of training examples [23], [25], [29], [50]. Over the last several years, the ML research community devoted significant efforts to reducing the complexity of the rule learners from being $O(s^4)$ [24], where s is the number of input examples, to the state-of-the-art learners that are nearly linear, and not worse than $O(s \log s)$ [23], [25], [29]. This paper proposes a new rule learner that is log-linear, i.e., $O(s \log s)$, with the size of input data, and at the same time it is significantly faster than other existing learners. Although the performance difference between the proposed and other considered learners are insignificant for small datasets that include several thousand examples, we show that they become very substantial at several million examples.

A number of alternative approaches to improve learner's scalability can be used. Two methods, which include applying data subsampling [63], and using parallel or distributed approaches to handle all the data are dominant. In the latter case a variety of methods exists, such as parallelization of the learner, feature selection, distributed learning on data subsets, use of relational representation, and meta learning [10], [12], [35], [38], [45], [61], [63]. Summary of relevant methods can be found in a survey by Provost and Kolluri [63].

Another very common problem is incompleteness within the datasets that are used to generate rules. Many real world datasets are characterized by an unavoidable problem of missing values. A variety of different reasons, such as manual data entry procedures, incorrect measurements, equipment errors, etc., result in incompleteness in the data. We also note that in some domains, such as medicine, it is not uncommon to encounter datasets that are missing more than 50% of their entries. For example, a set of medical data describing patients with cystic fibrosis missing over 60% of its entries was successfully used to find useful relationships about the disease [51]. Thus, a very desirable property of any ML algorithm is robustness to missing data. Specifically, a rule learner should be able to generate accurate rules in the presence of large quantities of missing data, while maintaining its scalability properties. The rule learner proposed in this paper has been extensively tested and compared to other state-of-the-art learners, to show its robustness to missing data from both accuracy and scalability perspectives.

The remainder of this paper is organized as follows. First, existing state-of-the-art rule learners and decision tree learners are described in Section II; these are the primary competitors for our algorithm. We then describe DataSqueezer, our proposed rule induction system, in Section III, and show that the asymptotic complexity of this algorithm is log-linear in the worst case. In Section IV, we present the results of an experimental comparison of the accuracy of our proposed learner. The results demonstrate that DataSqueezer is competitive with best-in-class competitors in terms of accuracy. In Section V, we present experimental timing results showing that the empirical complexity of DataSqueezer matches the theoretical results, and that DataSqueezer is much faster than other best-in-class rule learners. We also demonstrate DataSqueezer's robustness to missing data in Section V. Finally, we offer a summary and discussion of future work in Section VI.

II. State-of-the-Art Rule Induction and Decision Trees

We can divide competitors for DataSqueezer into rule learners, decision trees, and their hybrids. All of these algorithms follow the supervised ML paradigm, meaning that each training sample must be labeled with its correct classification. Some decision trees learners are CART [7], ID3 [67], C4.5 [65], T1 [41], and C5.0 [73]. Example rule learners are the AQ family of algorithms [42], [56], INDUCE [30], FOIL [68], REP [24], C4.5 rules [65], IREP [37], RISE [31], RIPPER [25], [27], DiVS [75], BEXA [77], DLG [80], SLIPPER [23], LAD [5], LERILS [14], and IREP++ [29]. Hybrid learners are represented by the CN2 [21], [22], and CLIP family of algorithms [16], [19]. A survey of relevant inductive ML learners can be found in [36]. Other inductive ML algorithms, which do not generate rules but some other data models, are for example probabilistic algorithms like Naïve Bayes [32], [54], and statistical algorithms like Support Vector Machines [28], [78]. While rule extraction can be applied to a number of different ML algorithms, these alternatives are beyond the scope of the current work.

The above rule and decision tree learners were intensively studied and cross-compared by the ML community over the last decade. Several major trends in development of new learners can be identified. The newer rule learners are usually characterized by a higher accuracy of the generated rules, often together with a lower complexity of the rules, when compared with the older learners. Another major trend was concerned with lowering computational complexity of the proposed learners. Some of the older learners, such as REP, have $O(s^4)$ complexity, where s is the number of examples in the training dataset. Plainly, such algorithms cannot scale to datasets of millions of examples. Over the last several years the subsequent learners were characterized by significant improvements in lowering the complexity. For example, the C4.5 system has $O(s^3)$ complexity, LERILS learner has $O(s^2)$ complexity, and RIPPER, SLIPPER, and IREP++ have $O(s \log s)$ complexity.

Table I presents a comparison of all low complexity rule and decision tree learners. However, in the current age of datasets including millions of examples, the complexity and speed of the most scalable rule learners becomes insufficient. To this end, the current paper proposes a new rule learner that has log-linear, i.e., $O(s \log s)$, complexity, and is significantly faster than other log-linear learners, which makes it appropriate for use in these very large datasets. The proposed rule learner is carefully evaluated with respect to accuracy and complexity of the generated rules, and is shown to be competitive in these respects to other state-of-the-art, low-complexity learners.

We will first review some of the major competitors for DataSqueezer, and identify a comparison group for our experiments. Since the proposed learner is designed to be scalable, the primary selection criterion for the comparison group is low complexity. Secondary criteria are the ability to handle multi-class problems, and robustness to missing values. The latter two characteristics are critical for most real-world applications of inductive learning systems.

Among all decision tree learners, the most scalable, and at the same time most accurate in terms of the generated rules and trees is the C5.0 learner [23]. C5.0 is a proprietary, commercial, unpublished descendant of C4.5 learner. It is based on the classical theory of decision tree induction [65], [66], and incorporates various advanced speed, memory, and pruning optimization techniques [73]. Thus this learner was selected as the best representative among decision tree learners for our experimental comparison.

A number of different algorithms were considered among the rule learners. First, the learners characterized by substantial complexity were disregarded. Although complexity results for the INDUCE, AQ, and FOIL learners (which use a first order logic based induction engine and Horn clause rule representation), have never been published, they are known to have very substantial scalability problems. These algorithms exhibit unacceptable performance even on very small datasets in the range of several thousand examples. The REP and C4.5 rules learners have $O(s^3)$ complexity, which is also very substantial, and thus are also disregarded.

The IREP learner was developed to reduce the high complexity of the REP learner [37]. The IREP learner was shown to be competitive with REP in terms of accuracy of generated rules, while at the same time was characterized by significantly lower complexity [37]. IREP was further improved, resulting in the RIPPER learner. The RIPPER learner was again shown to have very competitive accuracy, and better complexity when compared with IREP [25]. RIPPER uses a divide-and-conquer approach combined with a greedy set-covering based search procedure during the rule induction process [25]. After a rule is generated, it is immediately pruned in a greedy manner. Based on the above discussion, the RIPPER learner was selected as a representative of rule learners for the experimental comparison. Additionally, RIPPER learner can be used without rule optimization step that results in a faster induction process, which potentially leads to rules of lower accuracy. The nonoptimized version of the RIPPER was included in the experimental comparison set to contrast its scalability with the proposed learner.

The RISE learner has a much worse complexity, i.e., $O(s^2)$, than RIPPER learner, was never experimentally shown to be competitive in terms of accuracy and rule complexity compared with more recent learners, and is thus eliminated [31]. The complexity of the LAD learner was never published, and the paper that introduced the learner presented a statistically inconclusive comparison with other modern learners [5]. This learner has a very complex design, and the complexity appears to be significantly worse than $O(s \log s)$. LAD is thus eliminated. The same applies to BEXA learner, which complexity was not published, but appears significantly worse than complexity of CN2 and C4.5 learners [77]. The complexity of the DLG learner was also not published, but based on the algorithm's pseudocode, it appears to be worse than $O(s \log s)$, which is achieved by modern learners including RIPPER [80]. Therefore the learner is eliminated. Both LERILS and DiVS learners have a much worse complexity, i.e., $O(s^2)$, than the RIPPER learner, and were shown to match the accuracy of RIPPER. Both of these learners are thus eliminated from the comparison group.

SLIPPER is one of the most advanced rule learners. It is shown to improve upon the accuracy of RIPPER learner by applying a boosting strategy in the induction process [74]. In this approach, each subsequent rule covers examples from the training set that are recognized incorrectly by the rules created so far. The boosting strategy replaces the classical set covering approach used by RIPPER, which results in a learner that is substantially different and complementary to RIPPER [23]. At the same time SLIPPER learner is characterized by low, $O(s \log s)$ complexity, which is asymptotically identical to RIPPER's complexity. Based on the above discussion the SLIPPER was included in the comparison group. The IREP++ learner is the most recent extension of the RIPPER learner. It uses an identical induction procedure, but applies optimization with the respect to running time. Although the IREP++ was shown to be faster than RIPPER, it is characterized by the same asymptotic, i.e., log-linear, complexity. At the same time, its currently published version allows to handle only two-class datasets, while a further extension is necessary to handle multi-class problems [29]. Thus, it was not considered for the experimental comparison.

Finally, among the hybrid learners two learners are considered. Both CN2 and CLIP4 (the most recent learner from the CLIP learner family) have the same quadratic complexity. Since the CN2 generates rules using an entropy-based methodology (which is very similar to induction process applied by the decision tree learners), the CLIP4 learner was selected as a representative for the experimental comparison. The CLIP4 learner uses an integer programming based induction process to generate rules, and was shown to be competitive in terms of accuracy with other modern rule learners [19]. Additionally, it is the only learner that generates production rules that exclusively use inequalities, i.e., $\neq$, in the condition part of the generated rules, instead of the equalities generated by other learners. We note that RIPPER learner can be set up to generate inequalities in the rules, but its design focuses on deriving equality based rules. Also, BEXA learner uses negations to specialize generated rules, but the final generated rules are equality based. As such, CLIP4 is complementary to other learners, which provides

TABLE I
COMPARISON OF LOW COMPLEXITY RULE AND DECISION TREE LEARNERS

learner	Complexity	reference	learner	Complexity	reference
REP	$O(s^4)$	[24]	**RIPPER**	O(s logs), nearly linear complexity, not worse than O(s logs)	[25]
C4.5 rules	$O(s^3)$	[25]			
LERILS	$O(s^2)$	[14]	**SLIPPER**	O(s logs), nearly linear complexity, not worse than O(s logs)	[23]
RISE	$O(s^2)$	[31]			
CN2	$O(s^2)$	[31]	IREP++	O(s logs), nearly linear complexity, not worse than O(s logs)	[29]
CLIP4	$O(s^2)$	[19]			
DiVS	$O(s^2)$	[75]	**C5.0**	O(s logs), nearly linear complexity, not worse than O(s logs)	[23]
IREP	$O(s \log^2 s)$	[37]			

```
Given:   POS, NEG, k (number of attributes), s (number of examples)
Step1.
1.1      G_POS = DataReduction(POS, k);
1.2      G_NEG = DataReduction(NEG,k);

Step2.
2.1      Initialize RULES = []; i=1;                    // where rules_i denotes i^th rule stored in RULES
2.2      create LIST = list of all columns in G_POS
2.3      within every G_POS column that is on LIST, for every non missing value a from selected column j compute sum, s_aj, of
         values of gpos_i[k+1] for every row i, in which a appears and multiply s_aj, by the number of values the attribute j has
2.4      select maximal s_aj, remove j from LIST, add "j = a" selector to rules_i
2.5.1    if rules_i does not describe any rows in G_NEG
2.5.2        then remove all rows described by rules_i from G_POS, i=i+1;
2.5.3          if G_POS is not empty go to 2.2, else terminate
2.5.4      else go to 2.3
Output:  RULES describing POS

         DataReduction (D, k)                           // data reduction procedure for D=POS or D=NEG
DR.1     Initialize G = []; i=1; tmp = d_1; g_i = d_1; g_i[k+1]=1;
DR.2.1   for j=1 to N_D                                 // for positive/negative data; N_D is N_POS or N_NEG
DR.2.2     for kk = 1 to k                              // for all attributes
DR.2.3       if (d_j[kk] ≠ tmp[kk] or d_j[kk] = '*')
DR.2.4         then tmp[kk] = '*';                      // '*' denotes missing" do not care" value
DR.2.5     if (number of non missing values in tmp ≥ 2)
DR.2.6         then g_i = tmp; g_i[k+1]++;
DR.2.7       else i++; g_i = d_j; g_i[k+1]=1; tmp = d_j;
DR.2.8   return G;
```

Fig. 1. Pseudocode of the DataSqueezer learner.

additional motivation to include it in the experimental section. The comparison group selected for our experiments is highlighted in boldface in Table I.

III. THE DATASQUEEZER LEARNER

This section introduces the DataSqueezer learner [52]. First, a formal description and pseudocode of the learner are provided. Next, the complexity of the learner is estimated. Finally, the main characteristics and features of the learner are discussed.

A. DataSqueezer Algorithm

Let us denote the training dataset by D, which consists of s examples and k attributes. The sets of positive examples, $\mathrm{D_P}$, and negative examples, $\mathrm{D_N}$, must satisfy three properties: $\mathrm{D_P} \cup \mathrm{D_N} = \mathrm{D}$, $\mathrm{D_P} \cap \mathrm{D_N} = \emptyset$, $\mathrm{D_N} \neq \emptyset$, and $\mathrm{D_P} \neq \emptyset$.

Fig. 1 presents pseudocode for the DataSqueezer learner. The pseudocode uses vector and matrices (tables) that are denoted by capital letters, while their elements are denoted by the same name using small letters. The table of positive examples is denoted as POS and the number of positive examples by $\mathrm{N_{POS}}$, while the table and the number of negative examples are NEG and $\mathrm{N_{NEG}}$, respectively. The POS and NEG tables are created by inserting all positive and negative examples, respectively, where examples are represented by rows and attributes by columns. Positive examples from the POS table and negative examples from NEG table are denoted in the DataReduction procedure by the $\mathrm{d_i[j]}$ values where $\mathrm{j} = 1, \ldots, \mathrm{k}$, is the column number, and i is the example number (row number in the D table, which is initialized with either POS or NEG table). The DataSqueezer learner also uses tables that store intermediate results ($\mathrm{G_{POS}}$ for POS table, and $\mathrm{G_{NEG}}$ for NEG table), which have k columns. Each cell of the $\mathrm{G_{POS}}$ table is denoted as $\mathrm{gpos_i[j]}$, where i is a row number and j is a column number, and similarly for $\mathrm{G_{NEG}}$ table is denoted by $\mathrm{gneg_i[j]}$. The $\mathrm{G_{POS}}$ table stores a reduced subset of the data from POS, and $\mathrm{G_{NEG}}$ table stores a reduced subset of the data from NEG. The meaning of this reduction is explained later. The $\mathrm{G_{NEG}}$ and $\mathrm{G_{POS}}$ tables have an additional $(\mathrm{k}+1)^{\mathrm{th}}$ column that stores the number of examples from the NEG and POS tables, which a particular row in $\mathrm{G_{NEG}}$ and $\mathrm{G_{POS}}$ describes, respectively. Thus, for example $\mathrm{gpos_2[k+1]}$ stores the number of examples from POS that are described by the 2nd row in $\mathrm{G_{POS}}$ table. More detailed description that includes rationale and important features of both of the learner steps follows.

Step 1 Explanation:

Rationale: The learner performs data reduction to generalize information stored in the original data.

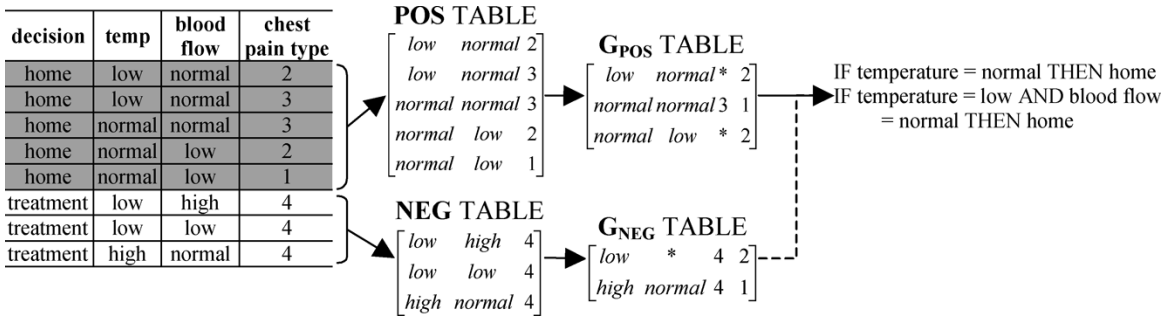

Fig. 2. Example rule generation process of the DataSqueezer learner.

Important Features: Data reduction is performed via use of the prototypical concept learning, which is based on the Find S algorithm of Mitchell [57]. It is performed for both positive and negative data and results in generation of the G_{POS} and G_{NEG} tables. The reduction procedure is also related to the least generalization, as used by the DLG learner [80]. The main difference is that the least generalization is applied multiple times for the entire positive set through a beam search procedure, while DataSqueezer performs it once in a linear fashion by generalizing consecutive examples. Also, the DLG learner does not generalize the negative set.

Step 2 Explanation:

Rationale: The learner generates rules by performing greedy hill-climbing search on the reduced data.

Important Features: A rule is generated by applying the search procedure starting with an empty rule, and adding selectors until the termination criterion fires. The rule, while being generated, consists of selectors generated using G_{POS} dataset, and is checked against the G_{NEG} dataset. If the rule covers any data in the G_{NEG} dataset, a new selector is added to the rules making it more specific, and thus able to better distinguish between positive and negative data. The maximum depth of the search is equal to the number of attributes. Next, the examples covered by the generated rule are removed, and the process is repeated.

An example dataset is used to illustrate working of the learner. It describes patients who are either directed home or undergo treatment based on results of medical tests. Each patient is described by temperature, blood flow, and chest paint type attributes, while the class is the decision made for each patient—see Fig. 2.

For our example, the $\mathrm{POS} = \begin{bmatrix} \text{low} & \text{normal} & 2 \\ \text{low} & \text{normal} & 3 \\ \text{normal} & \text{normal} & 3 \\ \text{normal} & \text{low} & 2 \\ \text{normal} & \text{low} & 1 \end{bmatrix}$,

$\mathrm{NEG} = \begin{bmatrix} \text{low} & \text{high} & 4 \\ \text{low} & \text{low} & 4 \\ \text{high} & \text{normal} & 4 \end{bmatrix}$, $\mathrm{k} = 3$, and $\mathrm{s} = 8$

In step 1, the G_{POS} and G_{NEG} tables are computed:

- First the DataReduction(POS, k) call in line 1.1 is executed. In line DR.1 the variables, $G = []$ and $d_1 = \mathrm{tmp} = [\text{low normal } 2]$, are initialized. After computing DR.2.1 ÷ DR.2.4 for $j = 1$ $\mathrm{tmp} = [\text{low normal} *]$ and after computing DR.2.5 and DR.2.6 $g_1 = [\text{low normal} * 2]$. After executing for loop for $j = 2$ $\mathrm{tmp} = [* \text{ normal } *]$ and thus based on DR.2.7 $g_2 = [\text{normal normal } 3\ 1]$ and $\mathrm{tmp} = [\text{normal normal } 3]$. For $j = 3$ $\mathrm{tmp} = [\text{normal} * *]$ and thus based on DR.2.7 $g_3 = [\text{normal low } 2\ 1]$ and $\mathrm{tmp} = [\text{normal low } 2]$. Finally for $j = 4$ $\mathrm{tmp} = [\text{normal low } *]$ and based on DR.2.6 $g_3 = [\text{normal low} * 2]$. As the result

$$G = G_{POS} = \begin{bmatrix} \text{low} & \text{normal} & * & 2 \\ \text{normal} & \text{normal} & 3 & 1 \\ \text{normal} & \text{low} & * & 2 \end{bmatrix}.$$

- Similarly, when DataReduction(NEG, k) is called in line 1.2 the resulting $G_{NEG} = \begin{bmatrix} \text{low} & * & 4 & 2 \\ \text{high} & \text{normal} & 4 & 1 \end{bmatrix}$. The last column of G_{POS} and G_{NEG} gives the number of rows, from POS and NEG respectively, which a particular row in G_{POS} and G_{NEG} describe. The "*" stands for "do not care" value. For instance, the first row in G_{POS} covers the first two rows in POS.

In the second step, rules are generated using G_{POS} and G_{NEG} tables:

- A rule is generated by incrementally adding selectors using the G_{POS} table. A selector with the highest summed value from the last columns is chosen and added incrementally until a rule will not describe any rows in the G_{NEG} table. Next, the rows described by the generated rule are removed and the process repeats
- The detailed computations follow. First, we initialize RULES = [] and LIST = [1,2,3]. Next in line 2.3 s_{aj} values are computed as $s_{low\ 1} = 6$ (since low has summed value of 2 in the last column in G_{POS} and the total number of values for $j = 1$, which is temperature, is 3), $s_{normal\ 1} = 9$, $s_{normal\ 2} = 9$, $s_{low\ 2} = 6$, $s_{3\ 3} = 4$. In line 2.4 the $s_{normal\ 1}$ is selected, LIST = [2,3], and rules_1 = [temperature = normal]. Next in line 2.5.1 rules_1 is verified not to describe any rows in G_{NEG}. In line 2.5.2 $G_{POS} = [\text{low normal} * 2]$

and since it is not empty (line 2.5.3) the algorithm iterates back to line 2.2. Again, LIST = $[1,2,3]$ and $s_{low\ 1} = 6$ and $s_{normal\ 2} = 6$.
- The $rules_2$ = [temperature = low] and LIST = $[2,3]$ are computed in line 2.4. Since $rules_2$ describes first row in G_{NEG} line 2.3 is executed again to further specialize the rule. The $s_{normal\ 2} = 6$ is computed in 2.3, and $rules_2$ = [temperature = low and blood flow = normal] and LIST = [3] is computed in 2.4. Finally since the $rules_2$ does not describe any rows in G_{POS} and after removing rows in line 2.5.2 G_{POS} becomes empty the algorithm terminates.
- As the result two rules were generated—see **Fig. 2**.

The procedure is similar, in some aspects, to DiVS [75]. The DiVS learner also learns using both positive and negative data, but it uses all examples to generate rules, including the ones covered by already generated rules. For multi-class datasets, the DataSqueezer learner generates rules for every class, each time generating rules that describe the currently chosen (positive) class.

B. Theoretical Complexity

In what follows, the asymptotic complexity of the DataSqueezer learner is determined. Our terminology and assumptions are as follows:

- s is the number of examples, k is the number of attributes, r is the number of generated rules, and c is the number of classes in the problem;
- length of the RULES vector is O(logs) and it is not longer than k;
- size of all POS and NEG matrices is kO(s);
- $s \gg k$;
- r and c are small constants. These constants were used in the analysis to provide general complexity estimation, but the final complexity is a function of s.

To estimate complexity of the entire learner, we break the process into determination of the complexity for particular steps of the learner.

1. Complexity of the initialization (line "Given" from the code in Fig. 1).

   kO(s) to derive POS matrix
   kO(s) to derive NEG matrix

   Thus, the total complexity of the initialization is: kO(s).
2. Complexity of STEP 1 (lines 1.1–1.2 from the code in Fig. 1).

   First, complexity of the DataReduction procedure is estimated.

   Line DR.1: O(1)
   Line DR.2.1: O(s) and applies to lines DR.2.2-DR.2.7
   Line DR.2.2: O(k) and applies to lines DR.2.3, and DR.2.4
   Line DR.2.3: O(1)
   Line DR.2.4: O(1)
   Lines DR.2.5-DR.2.7: O(1)
   Line DR.2.8: O(1)

   Thus, total estimated complexity of DataReduction procedure is

$$O(1)+O(s)\bullet[O(k)\bullet[O(1)+O(1)]+O(1)+O(1)+O(1)]+O(1) = O(1)+O(ks)+O(s)+O(s)+O(s)+O(1) = O(ks).$$

   Since STEP 1 simple calls twice the RataReduction procedure and stores the results in G_{POS} and G_{NEG} tables, which is O(ks), its total complexity equals to O(ks).
3. Complexity of STEP 2 (lines 2.1–2.5.4 from the code in Fig. 1).

   Line 2.1: O(1)
   Line 2.2: O(1)
   Line 2.3: O(ks) one sweep through G_{POS} is sufficient
   Line 2.4: O(k) selection of max s_{aj} is precomputed in 2.3
   Line 2.5.1: O(ks) one sweep through G_{NEG} is required
   Line 2.5.2: O(s)
   Line 2.5.3: O(r) and applies to lines 2.2-2.5.4, since this line will execute "go to 2.2" r times
   Line 2.5.4: O(logs) and applies to lines 2.3-2.5.4, since the longest rules has O(log s) selectors

Thus, the total complexity of STEP 2 is

$$\begin{aligned} &O(1)+O(r)\bullet[O(1)+O(logs)\bullet[O(ks)+O(k)+O(ks)+O(s)]] \\ &= O(1)+O(r)\bullet[O(1)+O(ks\ logs)+O(k\ logs) \\ &\quad +O(ks\ logs)+O(s\ logs)] \\ &= O(1)+O(r)+O(rks\ logs)+O(rk\ logs)+O(rks\ logs) \\ &\quad +O(rs\ logs) = O(rks\ logs). \end{aligned}$$

The complexity of the entire learner is estimated as a sum of complexities for each of the learner's steps as: $kO(s)+O(ks)+O(rks\ logs) = O(rks\ logs)$.

The above estimation concerns generation of rules for one class. The complexity of generation of rules for the problems with c classes is $cO(rks\ logs)$. Since the number of generated rules r and number of classes c are usually small constants, the expected running time of the learner is O(ks logs). This argument shows that the complexity of the DataSqueezer learner is log-linear with the number of examples in the training dataset. We note that for some applications number of attributes k may be large in comparison with the number of examples, i.e., k can be O(logs) or even O(s). In this case, the expected running time of the learner is O(ks logs), i.e., the running time is $O(s^2 logs)$

TABLE II
DESCRIPTION OF THE DATASETS USED FOR THE BENCHMARKING TESTS

#	abbr.	set	size	#class	#attrib.	test data	#	abbr.	set	size	#class	#attrib.	test data
1	**adult**	Adult	48842	2	14	16281	12	**led**	LED display	6000	10	7	4000
2	**bcw**	Wisconsin breast cancer	699	2	9	10CV	13	**pid**	PIMA indian diabetes	768	2	8	10CV
3	**bld**	BUPA liver disorder	345	2	6	10CV	14	**sat**	StatLog satellite image	6435	6	37	2000
4	**bos**	Boston housing	506	3	13	10CV	15	**seg**	image segmentation	2310	7	19	10CV
5	**cid**	census-income	299285	2	40	99762	16	**smo**	attitude smoking restr.	2855	3	13	1000
6	**cmc**	contraceptive method	1473	3	9	10CV	17	**spect**	SPECT heart imaging	267	2	22	187
7	**dna**	StatLog DNA	3190	3	61	1190	18	**tae**	TA evaluation	151	3	5	10CV
8	**forc**	Forest cover	581012	7	54	565892	19	**thy**	thyroid disease	7200	3	21	3428
9	**hea**	StatLog heart disease	270	2	13	10CV	20	**veh**	StatLog vehicle silhouette	846	4	18	10CV
10	**ipum**	IPUMS census	233584	3	61	70076	21	**vot**	congressional voting rec	435	2	16	10CV
11	**kdd**	Intrusion (kdd cup 99)	805050	40	42	311029	22	**wav**	waveform	3600	3	21	3000

when k is $O(s)$. On the other hand, in case when k is a small constant, and since the rule length is limited by k, the expected running time reduces to $O(s)$ showing linear relation with number of examples. We also note that the above estimate holds only for discrete data, while for continuous or mixed-mode data cost of front-end discretization must be added.

We discuss DataSqueezer's general characteristics below, and then proceed with a thorough experimental evaluation in Sections IV and V.

C. General Characteristics

The DataSqueezer learner was first described in [52]. It was further incorporated into a larger data mining system called MetaSqueezer [50], which was successfully applied to analysis of real-life clinical data concerning patients with cystic fibrosis [51]. The current paper is the first to present a detailed description of this learner, together with theoretical and extensive experimental analysis, and comprehensive comparison with other state-of-the-art learners. A very important advantage of the proposed learner is its simplicity. Our C++ implementation of DataSqueezer is less than 400 lines.

DataSqueezer can handle data with missing values. The missing values are processed in steps 1.2.2 and 1.5.2. The learner uses all available information while ignoring missing values, i.e., they are handled as "do not care" values. Experimental analysis, presented later, shows that the learner is robust to a large number of missing values. It can generate correct and compact rules in linear time even from data that have over half of the attribute values missing.

DataSqueezer can handle only discrete-valued, i.e., discrete numerical and nominal, attributes. In case of continuous attributes front-end discretization can be applied. The learner handles nominal data by automatic front-end encoding into numerical values. The generated rules are independent of the encoding scheme since the learner does not calculate distances, nor does it apply any metric during rule induction process.

Some of the inductive learners, such as AQ, CLIP, CN2, also can handle only discrete data, while some others can handle continuous attributes but still perform better with discrete-valued attributes [9], [43]. Several discretization algorithms, including unsupervised algorithms, such as equal width and equal frequency [15], and supervised algorithms, such as Information Entropy Maximization [33], CADD [13], and CAIM [46], [47], [49], can be used. The proposed learner uses CAIM discretization algorithm for smaller datasets, as this algorithm is the most recent and high quality discretization algorithm. For larger datasets, i.e., *cid*, *forc*, *ipums*, and *kdd* (see Table II), it uses equal frequency discretization, as this algorithm scales better with data size. The RIPPER, SLIPPER, and C5.0 learners, which are used in the experimental section, can handle continuous data on their own. The CLIP4 learner also uses CAIM and equal frequency discretization algorithms.

The learner, as it was used in the experimental section, uses two thresholds to reinforce the stopping condition in the second step of the rule induction process. DataSqueezer uses default values for the thresholds, which are set to zero, unless a user specifies alternative values.

- Pruning Threshold is used to prune very specific rules. The rule generation process is terminated if the first selector added to $rules_i$ has s_{aj} value equal or smaller than the threshold's value. The learner induces rules by selecting maximal s_{aj} values (selectors that cover the most positive training examples) and removes examples that are covered by the already generated rules. This has an advantage of leaving small subsets of positive examples, which contain examples different than majority of examples already covered (outliers), and which can be filtered out with the use of this threshold.
- Generalization Threshold is used to allow rules that cover a small amount of negative data. This threshold is used to relax the requirement from line 2.5.1 in Fig. 1, and allow accepting rules that describe a number of examples from negative training set equal or smaller than the threshold's value. Such mechanism is especially valuable in case of data containing overlapping classes, and in case of having inconsistent examples in the training dataset.

Both thresholds are specified as percentage of size of the positive training data set and thus are easily scalable. The thresholds should normally be set to small values.

DataSqueezer generates a separate set of rules for each class in the training dataset. It uses the rules to classify examples from the testing dataset. Due to the rule generation mechanism of the learner and use of the two thresholds, conflicts in the rule-set are possible. In general, two classification outcomes are possible: an example is assigned to a particular class, or is left unclassified. To classify an example and resolve the conflicts, the following procedure is applied.

```
- All the rules that cover (describe) the ex-
ample are found. If no rules cover the example
then it is left unclassified; this may happen
if the example has missing values for all at-
tributes that are used by the rules.
- For every class, the goodness of rules de-
scribing a particular class and covering the
example is summed. The example is assigned to
the class that has the highest summed value.
If there is a tie then the example is un-
classified. The goodness value for each rule
is equal to the percentage of the positive
training examples that it covers.
```

This classification procedure is identical to the procedure used by CLIP4 learner [19]. We note that all unclassified examples are categorized by the proposed learner as incorrect classifications since they are not described by the model. As such, these incorrect classifications contribute toward lowering classification accuracy, which is reflected in test results presented in the next section.

In contrast, other learners, such as RIPPER, SLIPPER, and C5.0 apply a so-called default hypothesis. In this case, if an example is not covered by any rule, it is assigned to the class with the highest frequency (default class) in the training dataset. This means that each example is always classified to one of the classes. Such a mechanism may lead to significant, artificial improvement in the accuracy of the generated rules. In the extreme case, for highly skewed datasets, where one of the classes is in a significant majority, it may lead to generation of the default hypothesis as the only "artificial" rule. Skewness is a common problem in realistic data mining tasks; for instance, fraud detection systems frequently deal with data that is skewed 100:1 against examples of fraud, which are the most important examples to detect. Skewness on the order of 100 000:1 has been reported for other application domains [64]. In its current form, DataSqueezer does not explicitly deal with the problem of skewness, other than by not incorporating a default rule. The two dominant approaches to dealing with skewness in ML are to resample the dataset (oversampling the minority class, undersampling the minority class), which is usually done by preprocessing the dataset as in [11]; or by incorporating misclassification costs into the actual ML algorithm. A study of these options and their impact on DataSqueezer would be a worthwhile extension of our work, but is beyond the scope of the current paper.

IV. Experiments

DataSqueezer was extensively tested to shows two main factors.

- DataSqueezer is competitive with other state-of-the-art rule learners in the accuracy and complexity of the rules it generates.
- DataSqueezer exhibits better scalability than these other state-of-the-art learners. The empirical complexity of DataSqueezer closely matches the calculated log-linear asymptotic complexity, while the running time for DataSqueezer is far shorter than for other learners in the comparison group.

This section focuses on the question of accuracy and rule complexity for a variety of realistic datasets. We will report our scalability studies in Section V. We also demonstrate DataSqueezer's robustness to missing data as a part of our scalability studies in Section V.

The proposed learner was tested on a comprehensive set of 22 datasets. Training datasets ranged between 151 and 500 K examples, the testing datasets between 15 and 500 K examples, the number of attributes between 5 and 61, and the number of classes between 2 and 40. The datasets were obtained from the UCI ML repository [4], the StatLog project datasets repository [79], and from the UCI KDD Archive [39]. A detailed description of the datasets is presented in Table II. We note that for the *ipums* dataset the response years, i.e., 1970, 1980, and 1990 were used as classes, similarly as in [2]. We also note that the CLIP4 learner, due to its very long running time caused by quadratic complexity, was not tested on large datasets.

In addition to accuracy, we also report the number of rules generated, and their complexity as measured by number of selectors. The latter factors are very important from the user point of view, especially in cases when the user needs to understand and evaluate the generated rules. The DataSqueezer learner is directly compared with the learners selected in Section II, i.e., C5.0, RIPPER, RIPPER without optimization, SLIPPER and CLIP4, and also to results reported in [55], which compared 33 other learners, including statistical algorithms, neural networks, decision trees, and rule learners, on the same datasets. The datasets and test procedures were selected to mimic test procedures used in [55] to enable direct comparison between the learners. As in [55], some of the tests were performed using tenfold cross-validation.

Table III reports accuracy for C5.0, RIPPER, SLIPPER, CLIP4, DataSqueezer, and the other 33 learners, for the 22 datasets. The results for the 33 learners report minimum and maximum accuracy, as published in [55]. In addition, we report sensitivity and specificity results for DataSqueezer on each dataset; these are a standard used in medicine where sensitivity and specificity analysis is used to evaluate confidence in the results [20], and are also related to ROC analysis of learner performance [62]. For multi-class problems, the sensitivity and specificity are computed for each class separately (each class being treated as positive class in turn), and the average values are reported. Results for all tenfold cross-validation experiments and mean values include standard deviations.

On average, C5.0 obtains the highest average accuracy, with the RIPPER being second best, followed by DataSqueezer, CLIP4, and SLIPPER. Closer analysis reveals that there is no universally best learner in this comparison group. A summary of the results is given in Fig. 3 and Table IV. In the scatterplot of Fig. 3, each point compares DataSqueezer to some second learner L on a single dataset. The x-axis position of the point is the accuracy of DataSqueezer, and the y-axis position is the accuracy of L. Therefore, points below the $y = x$ line correspond to datasets for which DataSqueezer performs better than some second learner. Visual inspection shows that the

TABLE III
ACCURACY RESULTS FOR ALL LEARNERS; ** EXECUTION STOPPED AFTER STATUS_STACK_OVERFLOW ERROR

set	Reported results			C5.0	RIPPER	RIPPER no optimization	SLIPPER	CLIP4 [19]	DataSqueezer		
	max	min	refs						accuracy	sensitivity	specificity
bcw	97	91	[55]	94 (±2.6)	94 (±2.7)	93 (±2.5)	93 (±3.0)	95 (±2.5)	94 (±2.8)	92 (±3.5)	98 (±3.3)
bld	72	57	[55]	68 (±7.2)	63 (±7.2)	64 (±7.4)	67 (±7.2)	63 (±5.4)	68 (±7.1)	86 (±18.5)	44 (±21.5)
bos	78	69	[55]	75 (±6.1)	75 (±7.0)	73 (±5.9)	72 (±7.5)	71 (±2.7)	70 (±6.4)	70 (±6.1)	88 (±4.3)
cmc	57	40	[55]	53 (±3.4)	54 (±3.7)	49 (±4.0)	53 (±5.9)	47 (±5.1)	44 (±4.3)	40 (±4.2)	73 (±2.0)
dna	95	62	[55]	94	95	93	94	91	92	92	97
hea	86	66	[55]	78 (±7.6)	77 (±6.7)	77 (±9.7)	77 (±9.2)	72 (±10.2)	79 (±6.0)	89 (±8.3)	66 (±13.5)
led	73	18	[55]	74	71	68	48	71	68	68	97
pid	78	69	[55]	75 (±5.0)	74 (±4.6)	73 (±5.4)	74 (±3.6)	71 (±4.5)	76 (±5.6)	83 (±8.5)	61 (±10.3)
sat	90	60	[55]	86	85	83	77	80	80	78	96
seg	98	48	[55]	93 (±1.2)	91 (±3.3)	91 (±2.8)	61 (±10.5)	86 (±1.9)	84 (±2.5)	83 (±2.1)	98 (±0.4)
smo	70	55	[55]	68	68	67	68	68	68	33	67
tae	77	31	[55]	52 (±12.5)	42 (±11.5)	45 (±12.3)	34 (±14.2)	60 (±11.8)	55 (±7.3)	53 (±8.4)	79 (±3.8)
thy	99	11	[55]	99	99	99	99	99	96	95	99
veh	85	51	[55]	75 (±4.4)	66 (±4.1)	63 (±3.5)	58 (±7.2)	56 (±4.5)	61 (±4.2)	61 (±3.2)	88 (±1.6)
vot	96	94	[55]	96 (±3.9)	96 (±2.7)	95 (±3.4)	95 (±3.5)	94 (±2.2)	95 (±2.8)	93 (±3.3)	96 (±5.2)
wav	85	52	[55]	76	78	74	74	75	77	77	89
MEAN stdev	**83.5** (±12.1)	**54.6** (±22.5)	---	**78.5** (±14.4)	**76.8** (±16.2)	**75.4** (±16.1)	**71.5** (±18.2)	**74.9** (±15.0)	**75.4** (±14.9)	74.6 (±19.1)	83.5 (±16.7)
adult	86	79	[4]	85	85	84	85	83	82	94	41
cid	77	95	[39]	95	94	94	94	89	91	94	45
forc	70	58	[3]	65	61	55	39	54	55	56	90
ipums	-	-	-	100	100	100	94	-	84	82	97
kdd	-	-	-	92	92	92	**	-	96	12	91
spect	89	74	[48]	76	70	73	76	86	79	47	81
MEAN all stdev	**82.9** (±11.4)	**59.0** (±22.7)	---	**80.4** (±14.1)	**78.6** (±15.8)	**77.5** (±16.2)	**72.9** (±19.0)	**75.6** (±14.8)	**77.0** (±14.6)	71.7 (±23.0)	80.9 (±19.0)

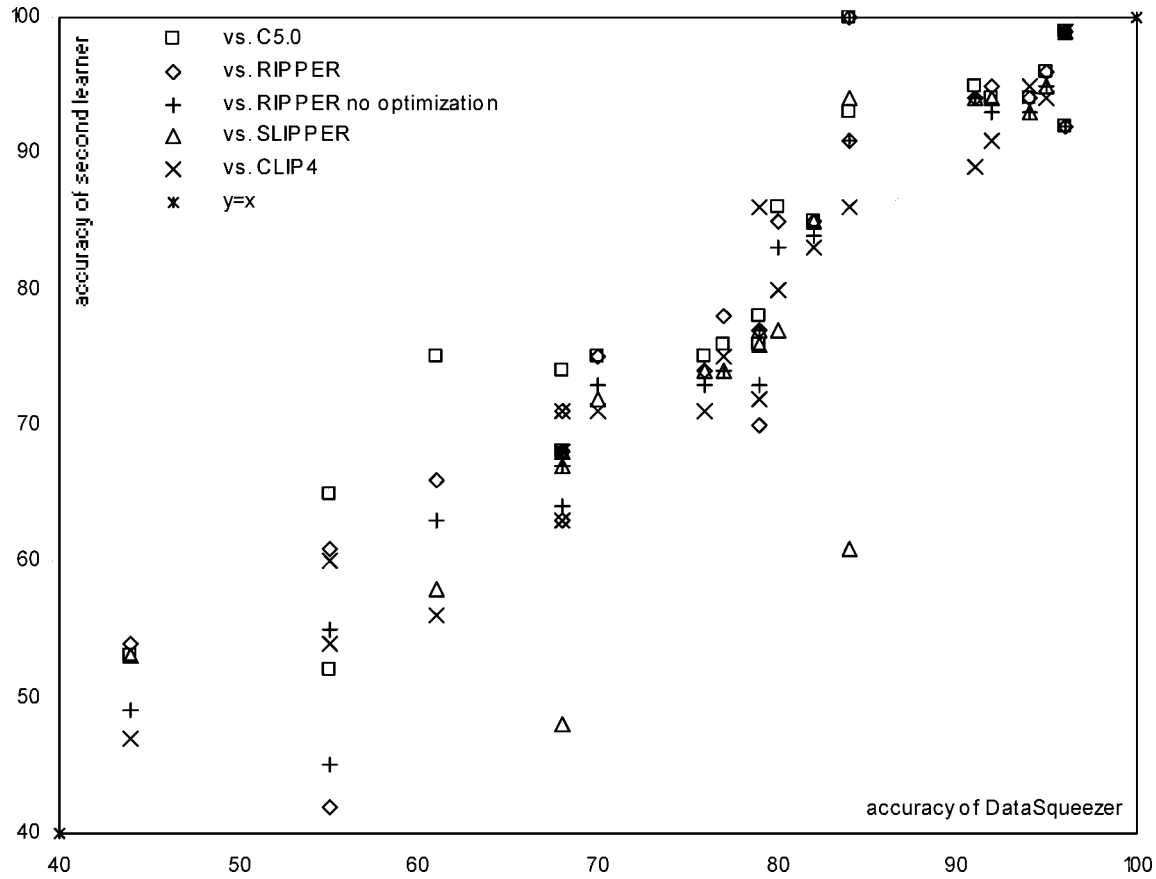

Fig. 3. Summary of accuracy results for the benchmarking test. Points below the $y = x$ line correspond to datasets for which DataSqueezer performs better than some other learner.

TABLE IV
SUMMARY OF ACCURACY RESULTS FOR THE BENCHMARKING TEST. IF L_R AND L_C ARE THE LEARNERS CORRESPONDING TO A ROW AND COLUMN, RESPECTIVELY, THE UPPER TRIANGLE ENTRIES ARE THE AVERAGE OF $\text{accuracy}(L_C)/\text{accuracy}(L_R)$ ON ALL 22 DATASETS. THE LOWER TRIANGLE ENTRIES ARE THE WON-LOSS-TIED RECORD OF LEARNER L_R VERSUS L_C, A "WIN" INDICATING L_R ACHIEVED A HIGHER ACCURACY

	DataSqueezer	C5.0	RIPPER	RIPPER no opt	SLIPPER	CLIP4
DataSqueezer		1.044	1.021	1.006	0.947	0.981
C5.0	13-6-3		0.978	0.964	0.907	0.940
RIPPER	14-6-2	3-11-8		0.986	0.928	0.961
RIPPER no opt	10-9-3	0-19-3	3-13-6		0.941	0.975
SLIPPER	7-12-2	0-15-6	2-13-6	7-8-6		1.036
CLIP4	9-9-2	3-15-2	3-13-4	6-13-1	8-10-2	

TABLE V
RESULTS OF t-TEST BETWEEN THE DATASQUEEZER LEARNER AND A SECOND LEARNER; "++" INDICATES THAT DATASQUEEZER WAS SIGNIFICANTLY BETTER, "+−" INDICATES THAT THERE WAS NO STATISTICALLY SIGNIFICANT DIFFERENCE, AND "−−" INDICATES THAT DATASQUEEZER WAS SIGNIFICANTLY WORSE

set	C5.0	RIPPER	RIPPER no optimization	SLIPPER	CLIP4
bcw	+ –	+ –	+ –	+ –	+ –
bld	+ –	+ –	+ –	+ –	+ –
bos	+ –	+ –	+ –	+ –	+ –
cmc	– –	– –	– –	– –	+ –
hea	+ –	+ –	+ –	+ –	+ –
pid	+ –	+ –	+ –	+ –	+ +
seg	– –	– –	– –	+ +	– –
tae	+ –	+ +	+ +	+ +	+ –
veh	– –	– –	+ –	+ –	+ +
vot	+ –	+ –	+ –	+ –	+ –
# significantly better (+ +)	0	1	1	2	2
# no difference (+ –)	7	6	7	7	7
# significantly worse (– –)	3	3	2	1	1

proposed learner performs on average with accuracy similar to other learners.

In Table IV, let L_R be the learner corresponding to a row in the table, and let the L_C correspond to a column. The upper triangle entries are the average of the quantity $\text{accuracy}(L_C)/\text{accuracy}(L_R)$ across all 22 datasets. For instance, the entries in the last column indicate that CLIP4's accuracy is about 6% lower than C5.0's, 4% lower than RIPPER's, 2% lower than nonoptimized RIPPER's and DataSqueezer's, and about 4% higher than SLIPPER's. The lower triangle entries are the won-loss-tied record of learner L_R versus L_C. "Win" indicates that L_R achieved higher accuracy. For instance, the last entry in the second column indicates that CLIP4 achieves a higher accuracy than DataSqueezer nine times, a lower accuracy nine times, and the same accuracy two times. Close inspection of the entries in the second row and second column, which concern comparison between the proposed learned and the other learners, show that the DataSqueezer generates rules with accuracy similar to accuracy of rules generated by the other learners. The differences in accuracy range between 4% loss to C5.0 learner and 5% win with SLIPPER learner, and are relatively close to each other.

The t-statistics test was used to compare learners for all ten-fold cross validation experiments. The test determines if the difference in accuracy between two learners on a single dataset is statistically significant. The 5% confidence level test was performed. For a given experiment, the two learners are statistically different if the t value is greater then the confidence limit.

Table V shows the test results, where each column describes t-test outcome considering difference in accuracy obtained by DataSqueezer and some second learner given in the top row. The summary row shows that the proposed learner is significantly worse on three datasets and there is no statistically significant difference on seven datasets, when compared with the most accurate C5.0 learner. This shows that DataSqueezer on average is statistically worse than C5.0, although for majority of the datasets they achieve comparable results. Comparison with other learners shows no significant statistical difference in accuracy.

In Table VI, we present our experimental measurements of rule complexity. The proposed learner is compared with the results reported in [55], and the results achieved by the C5.0, RIPPER, RIPPER without optimization, SLIPPER, and CLIP4 learners. The smallest average number of rules is generated by the SLIPPER learner, with the RIPPER being close and the second best. Next is nonoptimized RIPPER, followed by CLIP4, DataSqueezer, and finally C5.0. The number of rules generated by the DataSqueezer is similar to the average number of rules reported by [55]. We note that RIPPER and SLIPPER generate rule sets that are on average half the size of any of the other learners.

Although both number of selectors and number of selectors per rule are reported, the latter measure gives better indication of the complexity of generated rules. The number of selectors per rules is very small and comparable for C5.0, RIPPER, SLIPPER, and DataSqueezer learners. This shows that all these learners generate very compact, and thus easily comprehensible, rules. The CLIP4 learner generates on average longer rules, which is caused by their inequality based format. Most of the datasets considered have a very compact equality based under-

TABLE VI
NUMBER OF RULES AND RULE COMPLEXITY RESULTS FOR THE BENCHMARKING TEST; ** EXECUTION STOPPED AFTER STATUS_STACK_OVERFLOW ERROR

set	Reported median # leaves/ rules [55]	C5.0			RIPPER			RIPPER no optim			SLIPPER			CLIP4 [19]			DataSqueezer		
		mean # rules	mean # select	# select / rule	mean # rules	mean # select	# select / rule	mean # rules	mean # select	# select / rule	mean # rules	mean # select	# select / rule	mean # rules	mean # select	# select / rule	mean # rules	mean # select	# select / rule
bcw	7	16	16	1.0	14	34	2.4	15	36	2.4	5	10	2.0	4	122	30.5	4	13	3.3
bld	10	14	42	3.0	7	29	4.1	7	29	4.1	4	11	2.8	10	272	27.2	3	14	4.7
bos	11	18	68	3.8	7	22	3.1	18	72	4.0	8	24	3.0	10	133	13.3	20	107	5.4
cmc	15	48	184	3.8	8	44	5.5	11	82	7.4	6	27	4.5	8	61	7.6	20	70	3.5
dna	13	40	107	2.7	7	32	4.6	23	121	5.3	8	28	3.5	8	90	11.3	39	97	2.5
hea	6	10	21	2.1	5	14	2.8	5	12	2.4	4	12	3.0	12	192	16.0	5	17	3.4
led	24	20	79	4.0	16	80	5.0	20	115	5.8	25	97	3.9	41	189	4.6	51	194	3.8
pid	7	10	22	2.2	2	6	3.0	2	8	4.0	5	24	4.8	4	64	16.0	2	8	4.0
sat	63	96	498	5.2	39	195	5.0	31	162	5.2	23	105	4.6	61	3199	52.4	57	257	4.5
seg	39	42	181	4.3	28	121	4.3	37	158	4.3	27	101	3.7	39	1170	30.0	57	219	3.8
smo	2	0	0	0	0	0	0.0	5	32	6.4	0	0	0.0	18	242	13.4	6	12	2.0
tae	20	12	33	2.8	4	11	2.8	8	23	2.9	4	9	2.3	9	273	30.3	21	57	2.7
thy	12	7	15	2.1	2	8	4.0	3	11	3.7	5	19	3.8	4	119	29.8	7	28	4.0
veh	38	37	142	3.8	15	55	3.7	24	92	3.8	16	62	3.9	21	381	18.1	24	80	3.3
vot	2	4	6	1.5	1	2	2.0	5	16	3.2	2	7	3.5	10	52	5.2	1	2	2.0
wav	16	30	119	4.0	12	47	3.9	16	68	4.3	8	25	3.1	9	85	9.4	22	65	3.0
MEAN stdev	**17.8** (±16.3)	**25.3** (±23.9)	**95.8** (±123.5)	**2.9** (±1.4)	**10.4** (±10.5)	**43.8** (±51.3)	**3.5** (±1.4)	**14.4** (±10.5)	**64.8** (±52.4)	**4.3** (±1.4)	**9.4** (±8.5)	**35.1** (±35.5)	**3.3** (±1.2)	**16.8** (±16.3)	**415.3** (±789.1)	**18.9** (±12.7)	**21.2** (±19.8)	**77.5** (±80.3)	**3.4** (±0.9)
adult	---	54	181	3.3	20	102	5.1	47	320	6.8	6	17	2.8	72	7561	105.0	61	395	6.5
cid	---	146	412	2.8	6	24	4.0	36	291	8.1	30	217	7.2	19	1895	99.7	15	95	6.3
forc	---	432	1731	4.0	204	1050	5.1	245	1387	5.7	30	202	6.7	63	2438	38.7	59	2105	35.7
Ipums	---	75	197	2.6	85	436	5.1	83	453	5.5	13	44	3.4	-	-	-	108	1492	13.8
kdd	---	108	354	3.3	67	248	3.7	74	291	3.9	**	**	**	-	-	-	26	409	15.7
spect	---	4	6	1.5	1	3	3.0	3	6	2.0	3	12	4.0	1	9	9.0	1	9	9.0
MEAN all stdev	---	**55.6** (±92.3)	**200.6** (±368.6)	**2.9** (±1.2)	**25.0** (±45.5)	**116.5** (±232.7)	**3.7** (±1.3)	**32.6** (±52.3)	**172.0** (±297.4)	**4.6** (±1.6)	**11.0** (±9.9)	**50.1** (±61.7)	**3.6** (±1.5)	**21.2** (±21.8)	**927.4** (±1800.6)	**28.4** (±28.2)	**27.7** (±27.6)	**261.1** (±520.2)	**6.5** (±7.4)

TABLE VII
SUMMARY OF NUMBER OF RULES RESULTS FOR THE BENCHMARKING TEST

	DataSqueezer	C5.0	RIPPER	RIPPER no opt	SLIPPER	CLIP4
DataSqueezer		2.01	0.90	1.18	0.40	0.76
C5.0	7-14-1		0.45	0.59	0.20	0.38
RIPPER	14-4-4	20-1-1		1.30	0.44	0.85
RIPPER no opt	12-7-3	17-3-2	2-17-3		0.34	0.65
SLIPPER	15-6-0	19-1-1	10-9-2	19-3-1		1.91
CLIP4	9-9-2	15-5-0	3-15-2	9-11-0	5-14-1	

TABLE VIII
SUMMARY OF NUMBER OF SELECTORS PER RULE RESULTS FOR THE BENCHMARKING TEST

	DataSqueezer	C5.0	RIPPER	RIPPER no opt	SLIPPER	CLIP4
DataSqueezer		0.47	0.57	0.71	0.56	4.37
C5.0	14-8-0		1.29	1.59	1.26	9.78
RIPPER	12-8-2	4-15-3		1.23	0.97	7.59
RIPPER no opt	9-12-1	0-19-3	3-16-3		0.79	6.17
SLIPPER	12-9-0	8-12-1	13-7-1	14-7-0		7.79
CLIP4	0-19-1	0-20-0	1-19-0	1-19-0	0-20-0	

lying rule base, and thus the rules generated by CLIP4, which represent a negation of the underlying rule base, are long.

Similarly as for the accuracy results, a summary of the results is given in Tables VII and VIII. Table VII shows won-loss-tied record with respect to the number of generated rules for the considered learners, where "win" indicates that L_R generated smaller number of rules. Table VIII shows won-loss-tied record with respect to the number of generated selectors per generated rule for the considered learners, where "win" indicates that L_R generated smaller number of selectors per rule. The tables, in the upper triangle, also present the average across all 22 datasets, of the quantity $\#\text{rules}(L_C)/\#\text{rules}(L_R)$ in case of Table VII, and $\#\text{selector per rule}(L_C)/\#\text{selectors per rule}(L_R)$ in case of Table VIII.

Close inspection of the entries in the second row in Table VII, which concern comparison between the proposed learned and the other learners, show that the DataSqueezer generates about 60% more rules than SLIPPER, about 25% more than CLIP4, 10% more than RIPPER, about 20% less than nonoptimized RIPPER, and over 50% less than C5.0. The same inspection applied to Table VIII shows that DataSqueezer generates 50% more selectors per rule than C5.0, RIPPER and SLIPPER, about 30% more than nonoptimized RIPPER, and significantly less than CLIP4. Results show that C5.0 generates relatively large

TABLE IX
COMPARISON BETWEEN THE RESULTS ACHIEVED BY DATASQUEEZER LEARNER FOR THE EXPERIMENTS INVOLVING CROSS VALIDATION AND FIXED SIZE TESTING DATASETS

training datasets	# datasets	accuracy	# rules	# selectors per rule
all 22 datasets	22	77.0 (±14.6)	27.7 (±27.6)	6.5 (±7.4)
only 10CV datasets	10	72.6 (±16.4)	15.7 (±17.2)	3.6 (±1.0)
only fixed size datasets	12	80.7 (±12.5)	37.7 (±31.2)	8.9 (9.5±)

TABLE X
DESCRIPTION OF THE SYNTHETIC DATASET USED FOR THE SCALABILITY TEST

abbrev.	size	#classes	#attrib.	notes
synth	1200000	3	26	The dataset is described by an underlying rule base consisting of 13 rules with total of 35 selectors. Class distribution is normal. No missing values. Attributes are nominal and have between 1 and 15 values. 10 attributes are irrelevant with respect to the classes; they are not used by the underlying rule base. The dataset was generated using the dataset generator program from http://datasetgenerator.com

number of short rules. We note that SLIPPER learner on average generates smaller number of shorter rules than DataSqueezer. On the other hand, although the difference between the proposed learner and RIPPER learner in terms of number of generated selectors per rule is significant, the number of generated rules is comparable between the learners. Finally, we note that although SLIPPER on average generates the smallest number of compact rules, there is no universally best learner in this category, which can be observed based on the values in entries in the lower triangle in Tables VII and VIII.

An important factor, when discussing difference in accuracy between the selected learners, is that RIPPER, SLIPPER, and C5.0 generate and use default hypotheses, while both CLIP4 and DataSqueezer do not. The default hypothesis is used in cases where no rule covers an example, and it simply classifies such example to the majority class from the training dataset. In case of the latter two learners such an example is left unclassified, which translates into incorrect classification and lowers the accuracy. These two different scoring approaches are used for all 22 datasets considered in the benchmarking test since they are built into the learners. Using default hypothesis may artificially increase accuracy of the generated rules, especially for highly skewed datasets. In some cases, using default hypothesis may even lead to generation of the default hypothesis as the only "artificial" rule. While such a solution generates accurate classification, it has no practical use for the user since no data model (rules) for any of the classes is generated. For example, for *smo* dataset, the CLIP4 and DataSqueezer are the only two learners that generate rules, while the RIPPER, SLIPPER and C5.0 generate only the default hypothesis. Although all these learners achieve the same accuracy for these datasets (see Table III), Table V shows that no rules were generated by the latter three learners. We note that using a default hypothesis may also lower the average number and complexity of the generated rules.

We note that for ten out of 22 considered datasets, the tenfold cross validation procedure for testing was applied. In this case the training dataset is different and reordered for each of the ten folds. The proposed learner performs a linear scan through the training dataset in the first induction step, which is sensitive to a specific order of the input data, while in the second step a greedy search is applied that results in overcoming this sensitivity problem. Comparison between the results achieved for the experiments involving cross validation and fixed size training datasets, which experimentally shows insensitivity to the order of training examples, is presented in Table IX.

The difference in the number of generated rules results from the fact that these numbers are not normalized, and the largest datasets that have the biggest underlying rule based belong to the set of fixed size datasets. Both accuracy and number of selectors per rule are normalized, and show that similar performance is achieved in terms of accuracy, and number of selectors per rule is better for cross validation experiments. These results show that the order of data in the training set does not have impact on the accuracy, and complexity of the generated rules.

To summarize, the DataSqueezer learner is characterized by accuracy that is comparable with accuracy of other state-of-the-art learners. The proposed learner is statistically worse only in comparison with C5.0, although for majority of datasets they are characterized by results of similar quality. It generates rule sets comparable in size to those generated by most other advanced learners, albeit two times larger than rule sets generated by the SLIPPER learner.

V. PERFORMANCE BENCHMARKING

This set of benchmarking tests evaluates and compares the proposed learner to other state-of-the-art learners in terms of scalability and robustness to missing data. All simulations were performed on a TOSHIBA TECTRA laptop equipped with 1.6-GHz Intel Centrino processor, and 512-MB RAM.

We note that the below tests were performed for discrete datasets, while in case of continuous or mixed mode datasets discretization cost should be included. Depending on the selected discretization algorithm, the cost may worsen the asymptotic complexity of the rule generation process, i.e., simplest unsupervised discretization algorithms are linear, while the fastest supervised discretization algorithms are $O(s \log s)$ [46]. We also note that the scalability is evaluated with respect to the number of input data examples. The results do not represent a situation where number of attributes k is large in comparison with number of examples s, i.e., k is $O(s)$ or $O(\log s)$. In the first case the proposed learner is characterized by the is $O(s^2 \log s)$ asymptotic scalability.

A. Scalability Tests

The scalability tests use two large datasets, the *cid* dataset described in Table II, and a synthetic dataset that is described in Table X. The *cid* dataset is a large (around 200 K examples), complex, high dimensional, noisy, natural dataset that includes missing values, while the synthetic dataset is larger (over 1 million examples), has a large number of irrelevant attributes, a

TABLE XI
DETAILED RESULTS ACHIEVED BY DATASQUEEZER, C5.0, CLIP4, RIPPER, AND SLIPPER LEARNERS ON THE *CID* DATASET. ZERO VALUES IN THE # RULES AND # SELECTORS ENTRIES MEAN THAT ONLY THE DEFAULT HYPOTHESIS WAS INDUCED

learner	data size	1000	2000	4000	8000	16000	32000	64000	128000	199523
DataSqueezer	time [ms]	4	7	13	26	62	155	391	955	1752
	time ratio	---	1.8	1.9	2.0	2.4	2.5	2.5	2.4	1.8
	accuracy	87.9	89.9	88.1	87.7	87.9	89.3	89.4	89.1	90.5
	# rules	11	13	10	12	12	13	14	13	15
	# selects	74	80	57	77	74	81	88	83	95
C5.0	time [ms]	10	20	20	50	140	300	830	4230	6701
	time ratio	---	2.0	1.0	2.5	2.8	2.1	2.8	5.1	1.6
	accuracy	93.8	93.8	94.4	94.4	94.4	94.6	94.5	94.8	95.1
	# rules	0	0	4	9	15	31	66	174	146
	# selects	0	0	4	11	26	45	104	489	412
CLIP4	time [ms]	234	1028	5255	32117	82440	306672	1565306	4161503	9725212
	time ratio	---	4.4	5.1	6.1	2.6	3.7	5.1	2.7	2.3
	accuracy	89.8	90	89.3	88.9	89.2	89	88.8	88.8	88.9
	# rules	2	3	6	10	15	17	19	19	19
	# selects	73	172	528	1089	1759	2151	2218	2088	1895
RIPPER	time [ms]	15	10	67	418	1180	3715	8647	28183	69299
	time ratio	---	0.7	6.7	6.2	2.8	3.1	2.3	3.2	2.5
	accuracy	93.7	93.8	94	94.3	94.4	94.3	94.4	94.3	94.3
	# rules	3	0	4	5	9	15	9	19	30
	# selects	10	0	20	28	56	96	53	143	217
RIPPER no optimization	time [ms]	3	2	4	116	378	1050	3448	7656	17726
	time ratio	---	0.6	2	29	3.3	2.8	3.3	2.2	2.3
	accuracy	92.7	93.8	93.8	94.1	94.4	94.5	94.3	94.4	94.4
	# rules	5	0	0	10	12	15	24	25	36
	# selects	23	0	0	61	86	114	192	214	291
SLIPPER	time [ms]	11	42	121	376	1057	2337	4636	11121	20130
	time ratio	---	3.8	2.9	3.1	2.8	2.2	2.0	2.4	1.8
	accuracy	93.8	93.8	94.1	93.8	93.8	94	94.4	94.3	94.4
	# rules	0	6	5	4	6	6	6	6	6
	# selects	0	20	14	20	18	19	21	25	24

relatively simple underlying rule base, and no missing values. These two datasets cover the main characteristics of modern classification problems.

The goal of these tests is to experimentally show that the DataSqueezer learner has log-linear complexity, and that it is faster than other scalable, modern learners; these are the members of the comparison group identified in Section II. For both datasets, we incrementally double the size of the training datasets, using the first n examples from the original training dataset. We note that in case of a linear complexity, the running time should be doubled, while in case of the proposed learner complexity should follow the log-linear increase. The empirical complexity estimate can be derived by plotting the achieved running times and comparing them with corresponding reference complexity curves. Table XI shows detailed results achieved by the selected five learners on the *cid* dataset.

Fig. 4 visualizes the results as a graph of running time versus the size of training data. It uses logarithmic scale on both axes to help visualize data points with low values. Both linear and log-linear curves were plotted on the figure for the reader's convenience. Asymptotic complexity of a learner is evaluated based on the slope of the resulting curves, in comparison with the reference, i.e., linear and log-linear curves, while vertical shifting of the observed curves denotes difference by a constant factor. Both, RIPPER and RIPPER without optimization exhibit the same asymptotic complexity, i.e., their curves are parallel and therefore are analyzed together. Fig. 4 also compares the time ratios for consecutive datasets of increasing size for the considered learners.

The plots show that the fastest learner for this dataset is DataSqueezer, followed by C5.0, SLIPPER and RIPPER. The results for CLIP4 are not shown since they were not as good and would distort analysis of the results. They were included in Table XI to show the difference between $O(s^2)$ and $O(s \log s)$ learners. Closer analysis of the slope of the plots reveals that for this high dimensional and complex dataset DataSqueezer learner exhibits one of the best scalability properties. The time ratios show that DataSqueezer and C5.0 have, on average, log-linear experimental complexity, and DataSqueezer is by far the most stable with the increasing size of training data. Its maximum time ratio is 2.5, which is significantly lower than the maximal time ratios for other learners. At the same time, the proposed learner is at least an order of magnitude faster than both SLIPPER and RIPPER, and four times faster than C5.0 learner.

Fig. 5 show the relation between the running time and the size of training data in the linear-log scale, together with linear approximations for the C5.0, SLIPPER, RIPPER and DataSqueezer learners. Extrapolating the linear approximations, which favors more complex learners, suggests that for the datasets consisting of 20 million examples, which are currently becoming quite common, it would require about 27 min for DataSqueezer, 1 h and 45 min for C5.0, 4 h and 18 min for RIPPER without optimization, 5 h and 15 min for SLIPPER, and finally 16 h 20 min for RIPPER to perform induction using the same hardware.

In addition to running time, the results also report the accuracy, number of generated rules, and number of generated se-

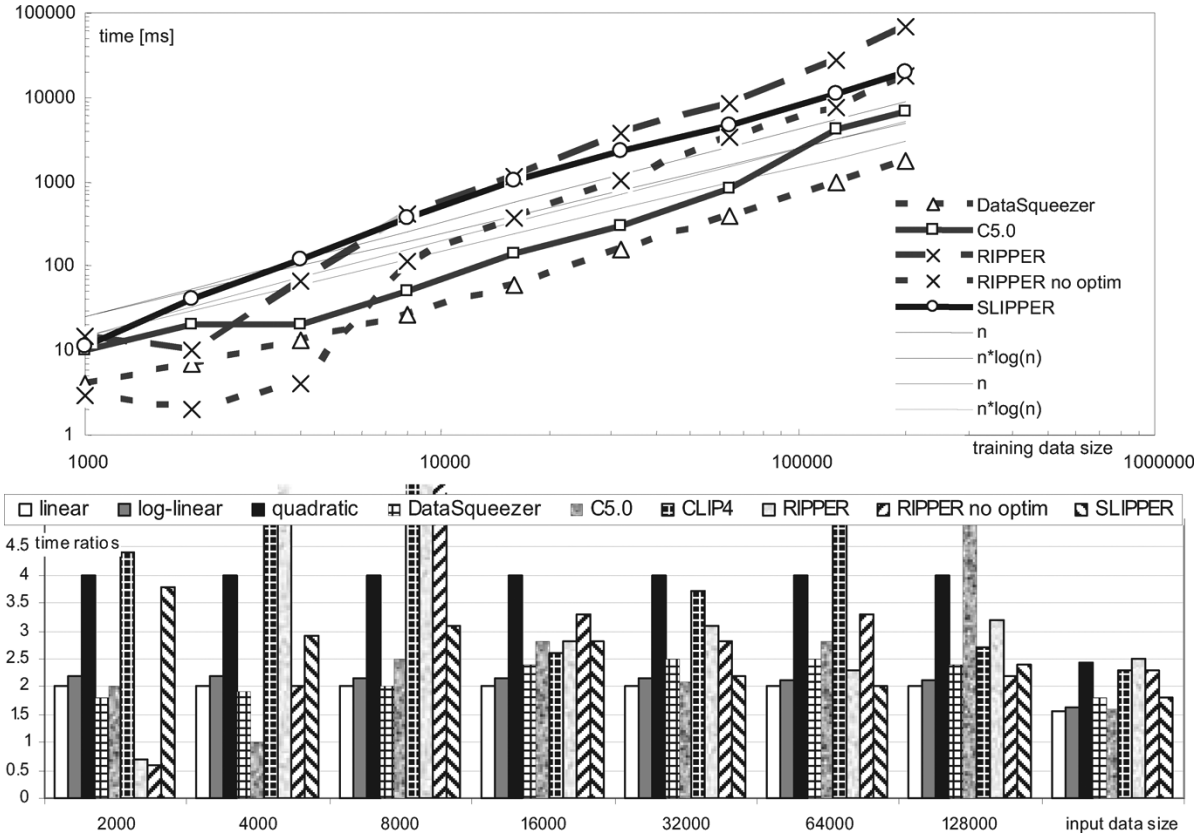

Fig. 4. Top plot shows running time in the log-log scale achieved by DataSqueezer (DS), C5.0, CLIP4, RIPPER, and SLIPPER learners on the *cid* dataset. The bottom bar chart shows time ratios for consecutive input data sized for the considered learners together with reference values for the linear, log-linear and quadratic ratios.

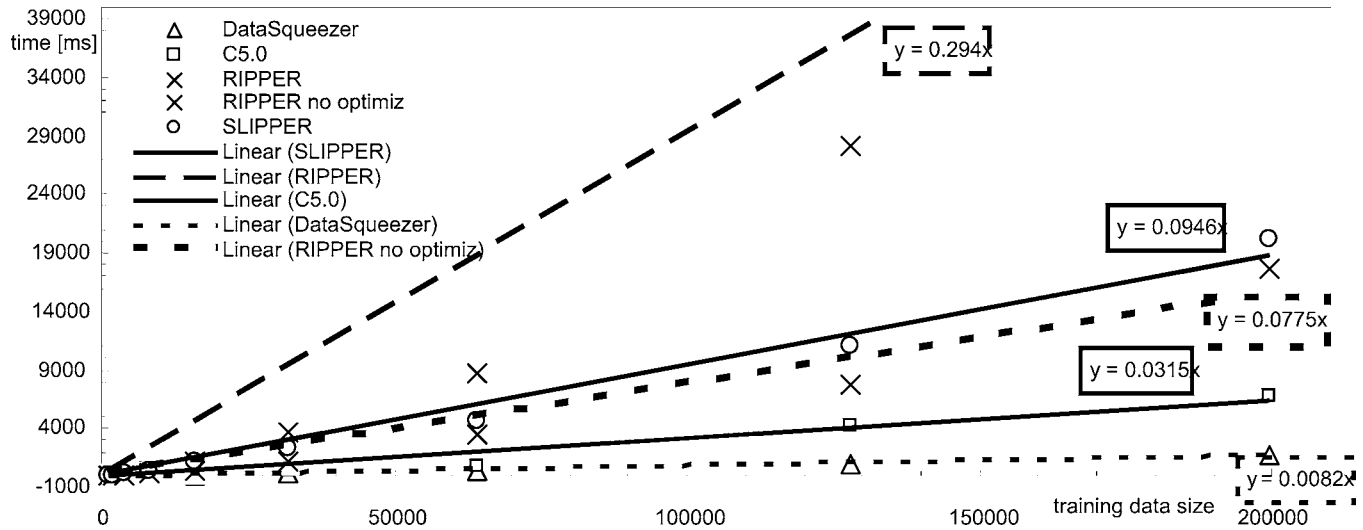

Fig. 5. Running time in the linear-log scale achieved by DataSqueezer (DS), C5.0, CLIP4, RIPPER, and SLIPPER learners on the *cid* dataset.

lectors. Fig. 6 summarizes the results for the *cid* dataset. The plots show that all learners are characterized by stable accuracy with the increasing amount of training data. We note that the default hypothesis for this dataset is around 94% accurate, and thus RIPPER, SLIPPER, and C5.0 have higher accuracy. The same stability is characteristic to DataSqueezer and SLIPPER when analyzing the number of generated rules and selectors. On the other hand, the C5.0 and CLIP4 learners are characterized by a rapid increase of the number of generated rules and selectors with the increasing amount of training data. This shows that the generated rules for the larger training sets are unnecessarily complex and may be overfitted.

Table XII shows detailed results achieved by the selected five learners on the *synth* dataset. Similarly as for the *cid* dataset, the results are visualized in Figs. 7 and 8. The CLIP4 learner was not used, since it is quadratic and would take too much time to run.

Fig. 7 shows that the fastest learner for this dataset is always DataSqueezer, followed by C5.0, RIPPER and SLIPPER. Closer analysis of the slopes of complexity curves and time ra-

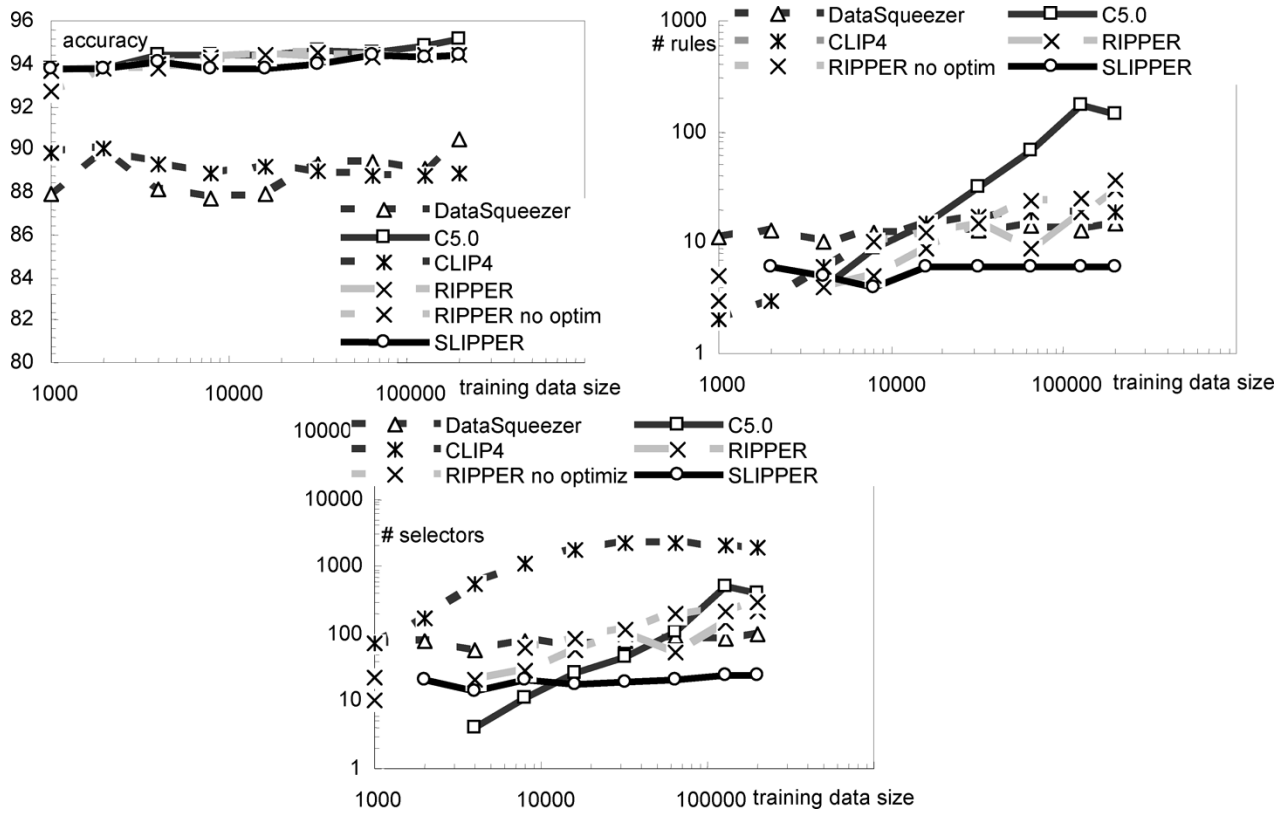

Fig. 6. Accuracy, number of rules and selectors generated by DataSqueezer (DS), C5.0, CLIP4, RIPPER, and SLIPPER learners on the *cid* dataset.

TABLE XII
DETAILED RESULTS ACHIEVED BY DATASQUEEZER, C5.0, RIPPER, AND SLIPPER LEARNERS ON THE *SYNTH* DATASET. ZERO VALUES IN THE # RULES AND # SELECTORS ENTRIES MEAN THAT ONLY THE DEFAULT HYPOTHESIS WAS INDUCED; * EXECUTION STOPPED AFTER 2 H; ** EXECUTION STOPPED AFTER STATUS_STACK_ OVERFLOW ERROR

learner	data size	1000	2000	4000	8000	16000	32000	64000	128000	256000	512000	1024000
DataSqueezer	time [ms]	2	4	8	15	30	63	133	279	593	1213	2471
	time ratio	---	2	2	1.9	2	2.1	2.1	2.1	2.1	2.0	2.0
	accuracy	92.6	94.1	95.7	95.8	95.8	95.8	95.8	95.8	95.8	95.8	95.8
	# rules	13	10	10	10	10	10	10	10	10	10	10
	# selects	31	21	20	22	21	21	21	22	22	22	22
C5.0	time [ms]	5	10	20	20	40	80	150	290	620	1260	2550
	time ratio	---	2	2	1	2	2	1.9	1.9	2.1	2	2
	accuracy	100	100	100	100	100	100	100	100	100	100	100
	# rules	13	13	13	13	13	13	13	13	13	13	13
	# selects	18	18	18	18	18	18	18	18	18	18	18
RIPPER	time [ms]	7	15	33	104	305	652	1337	2729	5657	12950	*
	time ratio	---	2.1	2.2	3.1	2.9	2.1	2.1	2.0	2.1	2.3	*
	accuracy	100	100	100	100	100	100	100	100	100	100	*
	# rules	8	8	8	8	8	8	8	8	8	8	*
	# selects	19	19	19	19	19	19	19	19	19	19	*
RIPPER no optimization	time [ms]	2	5	9	35	110	224	505	1003	2068	4455	*
	time ratio	---	2.5	1.8	3.9	3.1	2.0	2.3	2.0	2.1	2.2	*
	accuracy	99.6	99.8	99.8	100	100	100	100	100	100	100	*
	# rules	8	8	8	8	8	8	8	8	8	8	*
	# selects	18	18	19	19	19	19	19	19	19	19	*
SLIPPER	time [ms]	23	49	100	322	1048	2278	4598	9965	19476	**	**
	time ratio	---	2.1	2	3.2	3.3	2.2	2	2.2	2	**	**
	accuracy	100	100	100	100	100	98.0	100	100	100	**	**
	# rules	9	9	9	9	9	8	9	9	9	**	**
	# selects	21	21	22	20	22	18	22	20	20	**	**

tios chart shows that both DataSqueezer and C5.0 are linear for this dataset, while both RIPPER and SLIPPER are significantly slower and log-linear. We also note that the proposed learner was always the fastest, and the two most stable learners with increasing size of the training data were C5.0 and DataSqueezer. Their maximum time ratio is 2.1, which is significantly lower than the maximal time ratios for other rule learners, and they are at least an order of magnitude faster than both SLIPPER

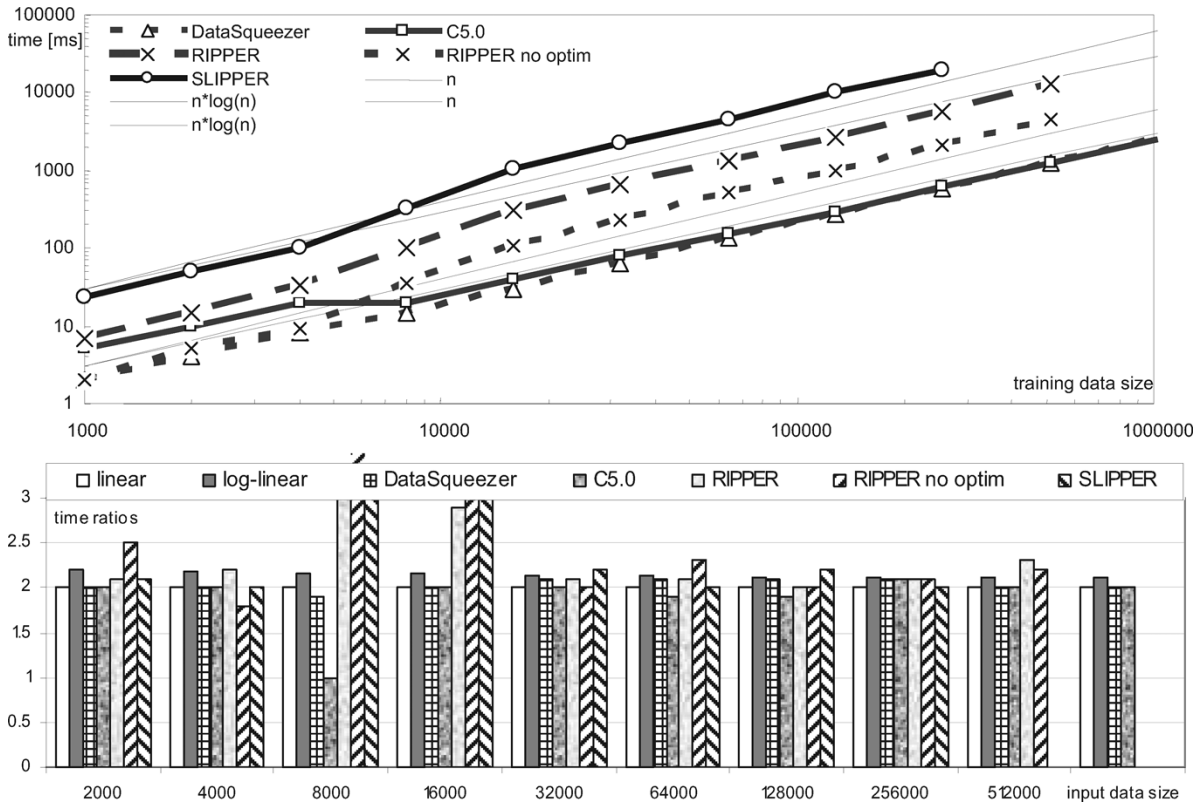

Fig. 7. Top plot shows running time in the log-log scale achieved by DataSqueezer (DS), C5.0, RIPPER, and SLIPPER learners on the *synth* dataset. The bottom bar chart shows time ratios for consecutive input data sized for the considered learners together with reference values for the linear, log-linear and quadratic ratios.

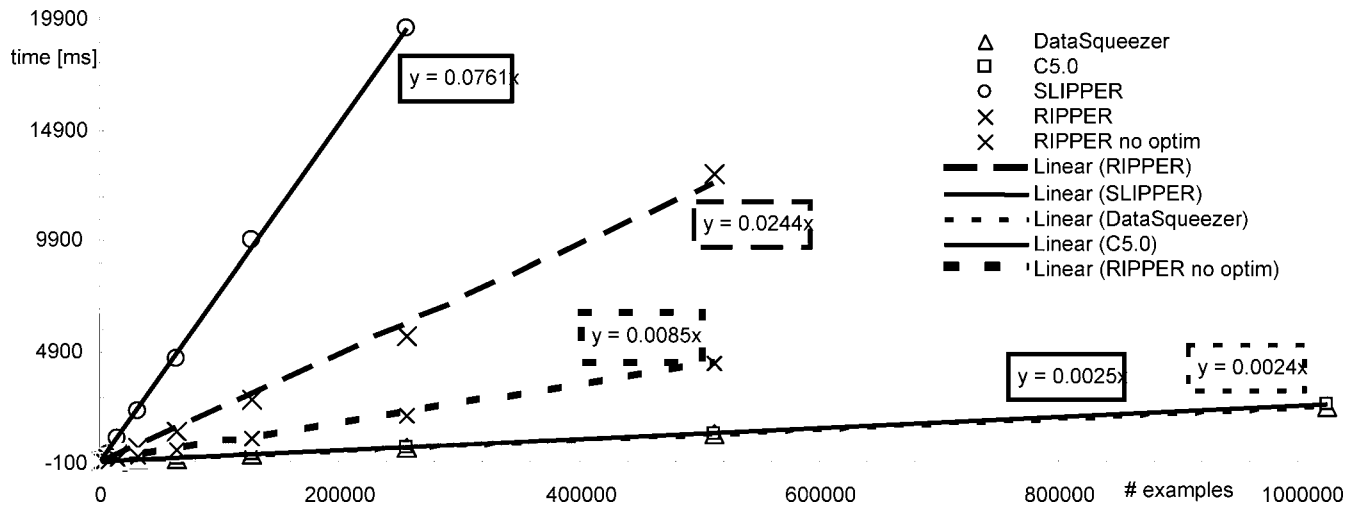

Fig. 8. Running time in the linear-log scale achieved by DataSqueezer (DS), C5.0, RIPPER, and SLIPPER learners on the *synth* dataset.

and RIPPER. We note that this agrees with the theoretical complexity estimate, which reduces to linear in case when number of attributes and rule length are small constants.

Based on the linear approximations for the C5.0, SLIPPER, RIPPER and DataSqueezer learners shown in the Fig. 8, an extrapolation of the running time for a larger (say 100 million examples) dataset can be computed. In this case, it would require about 40 min for DataSqueezer, 42 min for C5.0, 2 h and 22 min for RIPPER without optimization, 6 h 47 min for RIPPER, and finally 21 h 8 min for SLIPPER to perform induction using the same hardware.

Fig. 9 reports accuracy, number of generated rules, and number of generated selectors for the *synth* dataset. The plots show that all learners are very stable for all three characteristics with the increasing amount of training data.

To summarize, C5.0 and DataSqueezer are experimentally verified to be linear for datasets of relatively compact underlying rule base where number of attributes and rule length can be treated as small constants, and for complex datasets are log-linear. On the other hand, RIPPER and SLIPPER experimentally are log-linear in case of "easy" datasets, while they are worse than log-linear for complex datasets. C5.0 overfitted

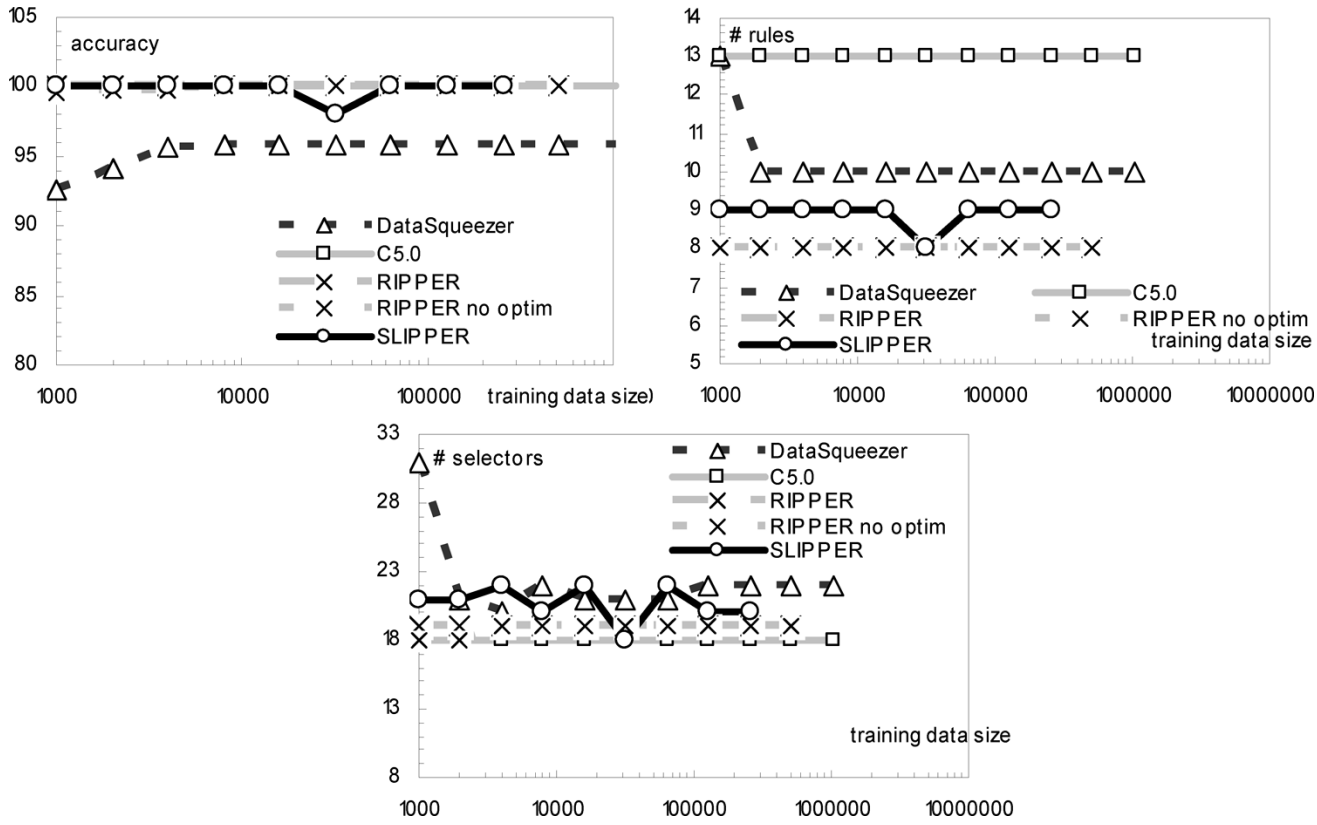

Fig. 9. Accuracy, number of rules and selectors generated by DataSqueezer (DS), C5.0, RIPPER, and SLIPPER learners on the *synth* dataset.

rules for the large complex dataset, which may limit its usability. The DataSqueezer learner was consistently the fastest one, with at least an order of magnitude improvement compared with RIPPER and SLIPPER learners. We also note that the performed analysis does not show that any of the considered learners is asymptotically faster than the others, although it shows which learners are faster. The superiority, in terms of computational speed, of the DataSqueezer learner can be explained by the sequential character of its first step combined with a fast, greedy, hill-climbing search in the second step of the algorithm. This very efficient rule generation procedure puts DataSqueezer in a better position when compared to more complex modern learners.

B. Robustness to Missing Data

These experiments use the two datasets from the scalability test, and two datasets, *sat* and *vaw*, which were used in Section IV. The datasets were selected to cover a wide range of problems. The goal is to evaluate robustness of the learners, in terms of accuracy, number and complexity of generated rules, as well as running time with increasing amounts of missing values in the training dataset. Missing data was introduced into the four training datasets by random deletion in the increasing amounts of 5%, 10%, 20%, 30%, 40%, 50%, and 60%. The nonoptimized RIPPER is not included since the above experimental results show that it is, on average, worse that RIPPER.

Fig. 10 shows the relation between the accuracy of the generated rules and amount of missing data for DataSqueezer, C5.0, RIPPER, and SLIPPER learners for all four datasets. The results show that there is no universally stable learner. C5.0 has problems on the *synth* and *wav* datasets; RIPPER has problems on *sat*, and *wav*; SLIPPER has problems on *wav*, and *cid*; DataSqueezer has problems on *synth*. We note that DataSqueezer is stable and does not worsen accuracy of the rules with increasing amount of missing values. This improvement can be observed for *cid*, and *sat* datasets. On average, we note that DataSqueezer is the most robust among the four learners, with SLIPPER being the second best.

Fig. 11 shows the relation between the number of generated rules and amount of missing data for the four learners. The results show that SLIPPER is almost always the most stable and generates the smallest number of rules. RIPPER is very unstable for *synth*, and overfits the rules for *sat*. C5.0 is the only learner that was able to reduce number of generated rules with increasing amount of missing values. This improvement can be observed for *cid*, *wav*, and *sat* datasets. On the other hand, on average C5.0 generates a significantly higher number of rules than other learners. DataSqueezer overfits the rules on *wav*, and is stable for other datasets. On average, we note that SLIPPER is the most robust, with DataSqueezer being the second best.

Fig. 12 shows the relation between the number of generated selectors and amount of missing data for the four learners. The results again show superiority of SLIPPER learner. RIPPER is again very unstable for *synth*, but reduces number of selectors with increasing amount of missing values for the remaining datasets. C5.0 is the second learner that reduces the number of generated selectors with increasing amount of missing values for *cid*, *wav*, and *sat* datasets, but again on average it generates a significantly higher number of selectors than other learners.

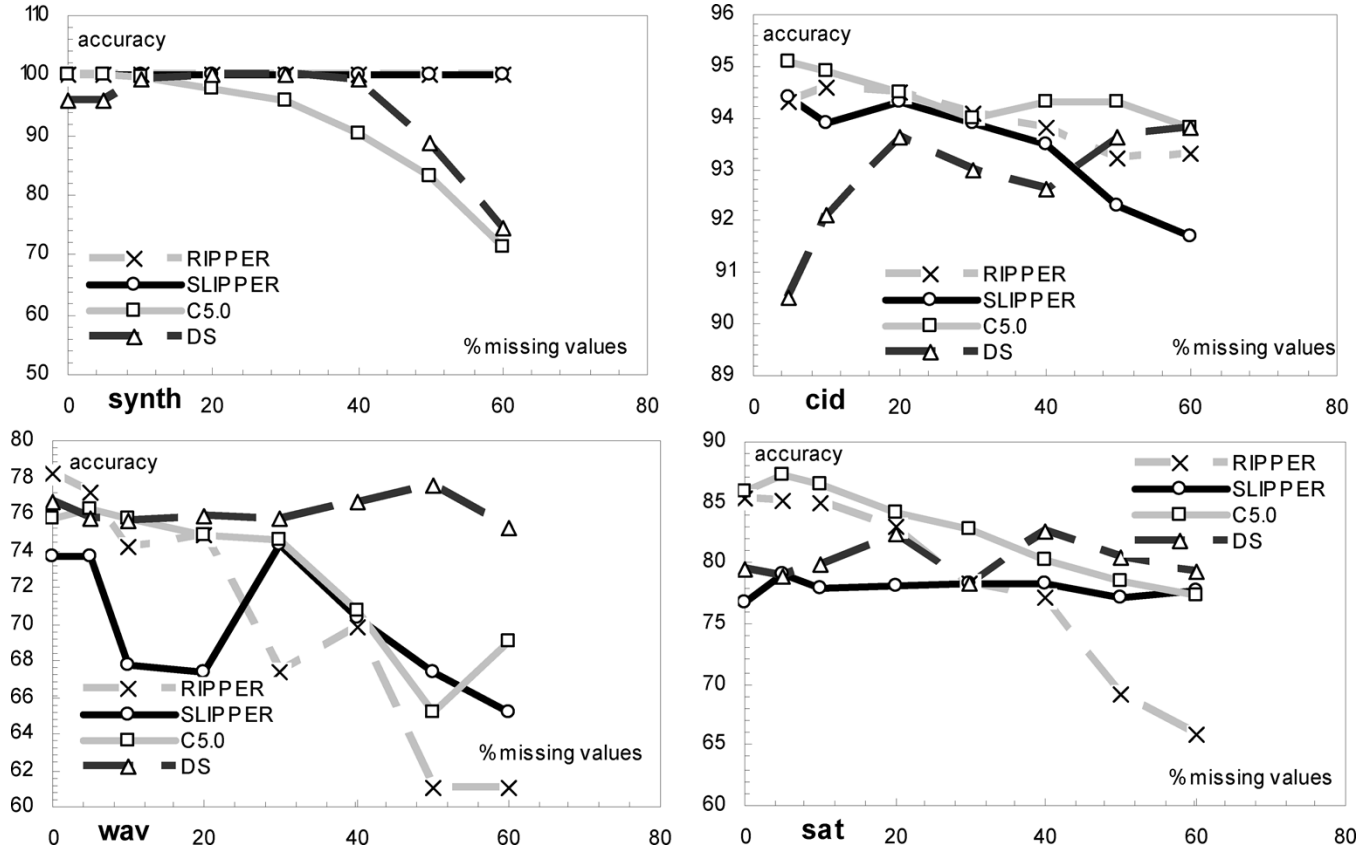

Fig. 10. Accuracy of rules generated by DataSqueezer (DS), C5.0, RIPPER, and SLIPPER learners for the robustness test.

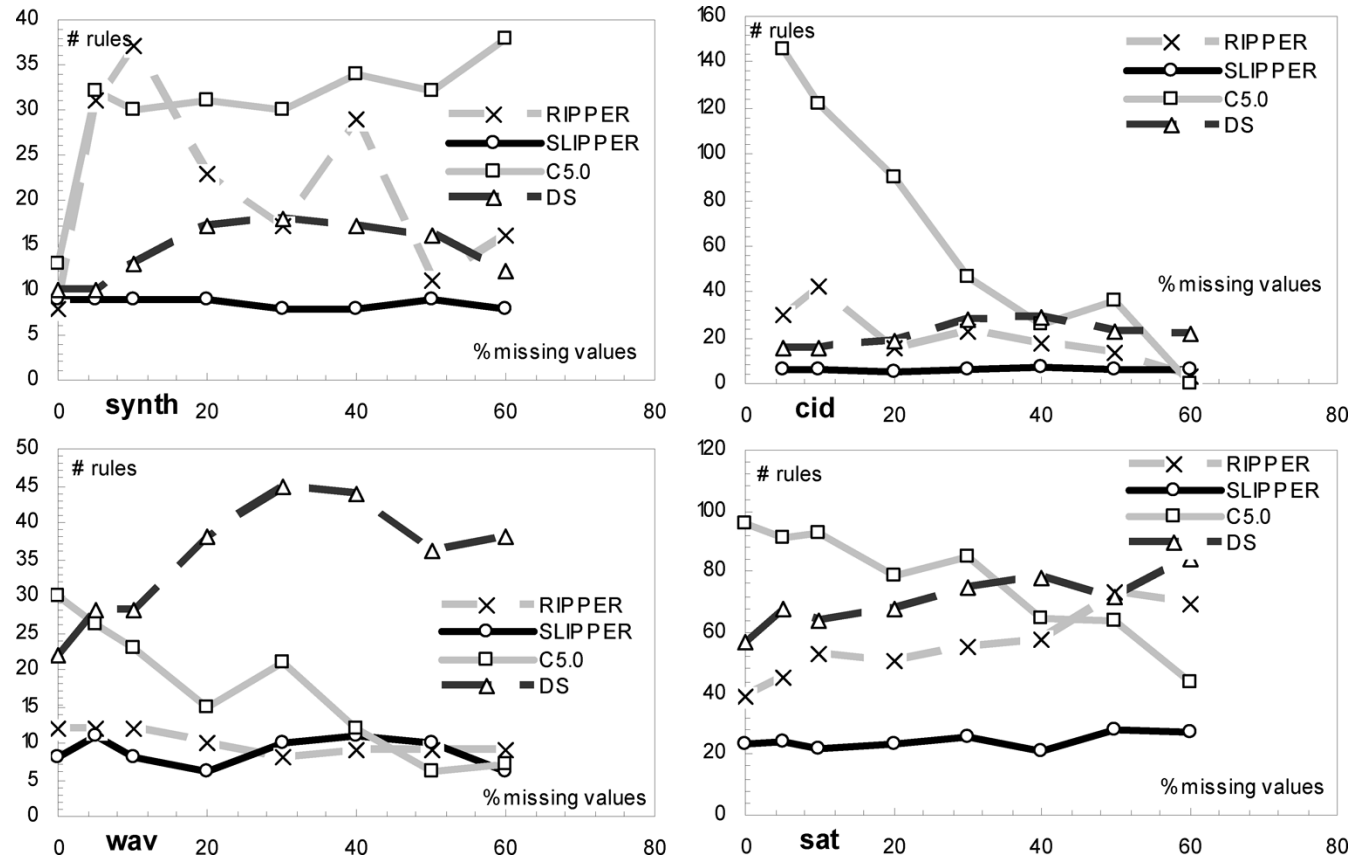

Fig. 11. Number of rules generated by DataSqueezer (DS), C5.0, RIPPER, and SLIPPER learners for the robustness test.

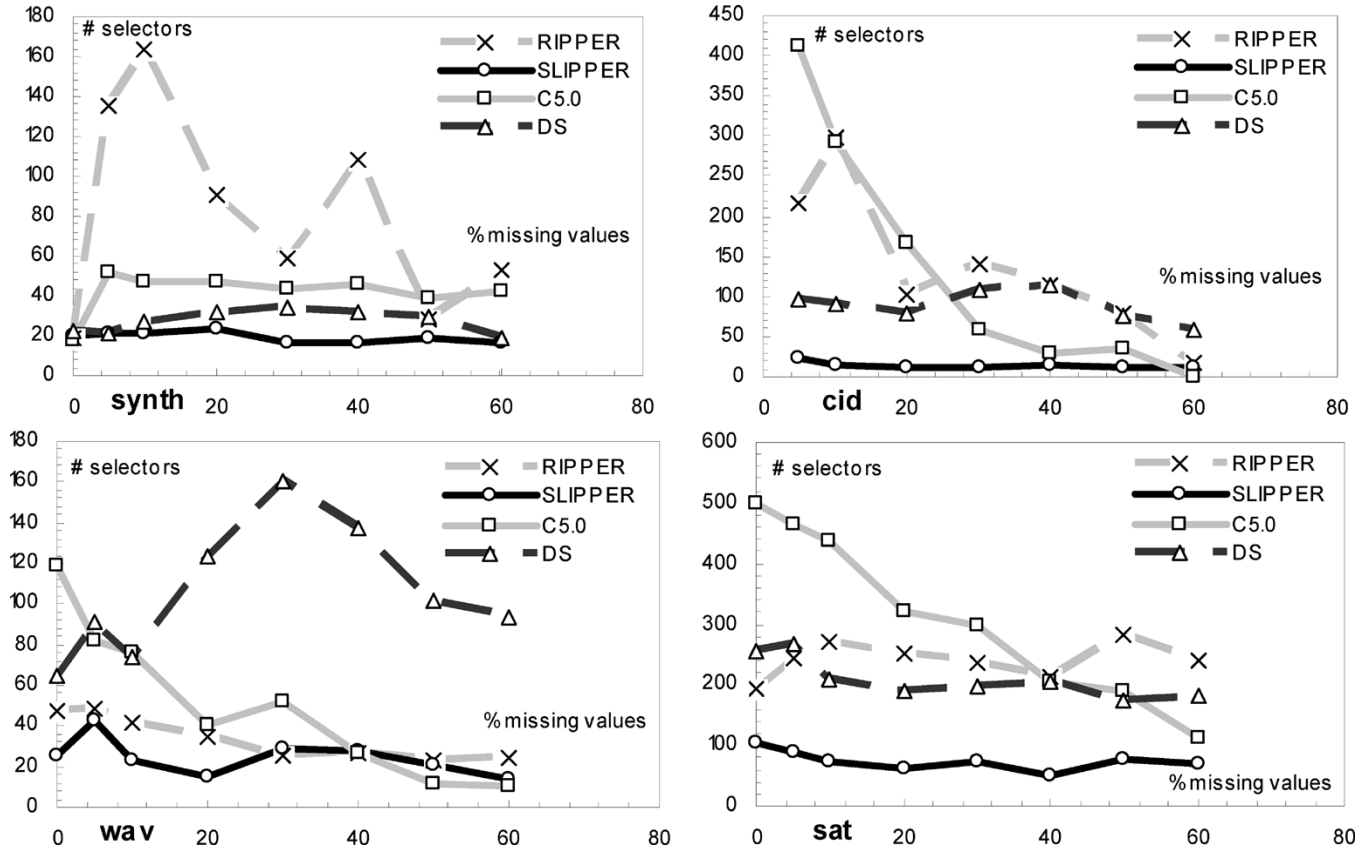

Fig. 12. Number of selectors generated by DataSqueezer (DS), C5.0, RIPPER, and SLIPPER learners for the robustness test.

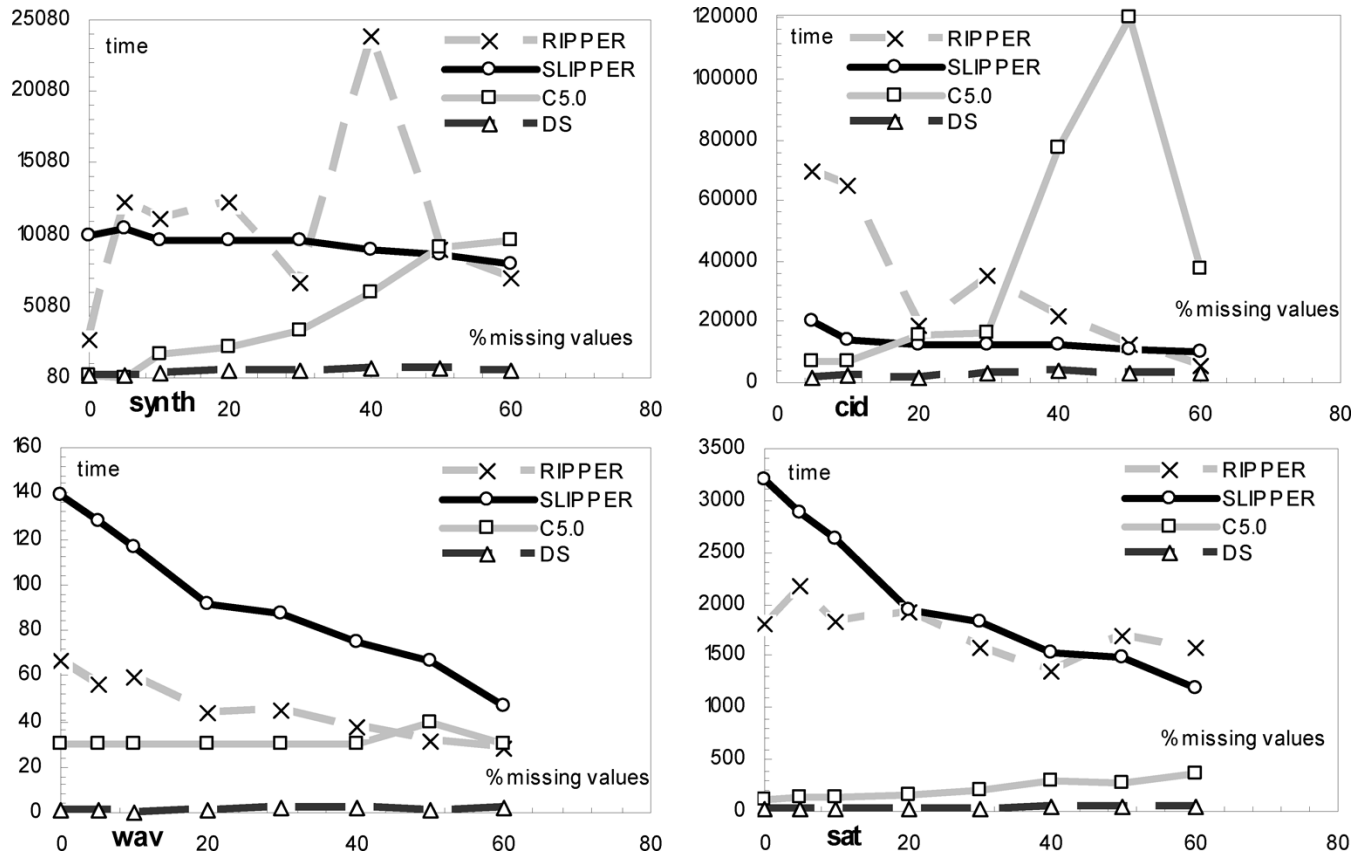

Fig. 13. Running time of DataSqueezer (DS), C5.0, RIPPER, and SLIPPER learners for the robustness test.

DataSqueezer is stable, with the exception of the *wav*. On average, we note that SLIPPER is the most robust, with RIPPER and DataSqueezer being the second best.

Fig. 13 shows the relation between the running time and amount of missing data for the four learners. The results show the superiority of DataSqueezer, which not only is always the fastest, but also its running time is constant with an increasing amount of missing values. The SLIPPER is also very stable with increasing amount of missing values. While for some datasets, such as *synth*, *wav*, and *sat*, its running time even decreased with the increasing amount of missing data, overall SLIPPER was significantly slower than other learners. RIPPER has numerous running time instability problems, which can be observed for *synth* and *cid*. C5.0 achieved very good running time results on *wav* and *sat*, but at the same time shows major instability for *cid*, and a strong increasing trend for *synth*. On average, we note that DataSqueezer is substantially more robust in this category than the other learners.

To summarize, the DataSqueezer shows superior robustness to missing values in terms of running time and a stable level of accuracy. It also shows fairly good robustness in terms of number and complexity of generated rules. The other robust learner is SLIPPER. It shows superior properties in terms of the number and complexity of generated rules with the increasing amount of missing values, and very good properties in terms of very high and fairly stable accuracy of generated rules.

VI. Concluding Remarks

The paper describes a novel inductive rule learner called DataSqueezer. It induces a set of production rules from a supervised training dataset. It is characterized by being relatively simple to implement, and having fast induction process. The learner was extensively tested and compared with several state-of-the-art representative decision tree and rule learners. The tests show that the learner generates rules with accuracy and complexity comparable to the rules generated by other learners. The main two advantages of DataSqueezer are its log-linear complexity combined with speed that is better when compared with other scalable, modern learners, and robustness to missing values. Excellent scalability of the proposed learner was shown both theoretically and experimentally. On all tests, the learner was on average the fastest. At the same time, the accuracy and complexity of the rules DataSqueezer generates are not affected by presence of large amounts of missing values in the training dataset. DataSqueezer is thus well suited to modern data mining and business intelligence tasks, which commonly involve huge datasets with a large fraction of missing data.

A. Future Work

One of open issues is robustness of the proposed learner to noise and inconsistency in input data. Future work will aim to address this issue by performing benchmarking on datasets that include noise. Future work will also include applying the DataSqueezer learner within the Meta Mining architecture. Meta Mining is a generic framework for higher order learning where rules are induced from rule sets generated by ML or data mining algorithms. Its main characteristic is generation of data models, called meta-models, from the already generated data models (referred to as meta-data) [72], [76]. Researchers argue that such meta-models are more suitable for describing knowledge that can be considered interesting, and that reduced complexity of generated rule sets can be achieved [1], [50], [72], [76]. Finally, parallelization for cluster systems will also be a topic of future research.

Acknowledgment

The authors would like to acknowledge and thank reviewers for providing constructive comments.

References

[1] T. Abraham and J. F. Roddick, "Incremental meta-mining from large temporal data sets, advances in database technologies," in *Proc. 1st Int. Workshop on Data Warehousing and Data Mining (DWDM'98)*, 1999, pp. 41–54.

[2] S. D. Bay and M. J. Pazzani, "Detecting group differences: Mining contrast sets," *Data Mining Knowl. Disc.*, vol. 5, no. 3, pp. 213–246, 2001.

[3] J. A. Blackard, "Comparison of Neural Networks and Discriminant Analysis in Predicting Forest Cover Types," Ph.D. dissertation, Dept. Forest Sciences, Colorado State Univ., Fort Collins, 1998.

[4] C. L. Blake and C. J. Merz. (1998) UCI Repository of ML Databases. Univ. of California, Department of Information and Computer Science, Irvine, CA. [Online]. Available: http://www.ics.uci.edu/~mlearn/ML-Repository.html

[5] E. Boros, P. L. Hammer, T. Ibaraki, A. Kogan, E. Mayoraz, and I. Muchnik, "An implementation of logical analysis of data," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 2, pp. 292–306, Feb. 2000.

[6] R. Brachman and T. Anand, "The process of knowledge discovery in databases: A human-centered approach," in *Advances in Knowledge Discovery and Data Mining*, U. M. Fayyad, G. Piatesky-Shapiro, P. Smyth, and R. Uthurusamy, Eds. Cambridge, MA: AAAI/MIT Press, 1996.

[7] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. London, U.K.: Chapman & Hall, 1984.

[8] J. Catlett, "Megainduction: A Test flight," in *Proc. 8th Int. Workshop on Machine Learning*, Ithaca, NY, 1991, pp. 596–599.

[9] ——, "On changing continuous attributes into ordered discrete attributes," in *Proc. Eur. Working Session on Learning*, 1991, pp. 164–178.

[10] P. Chan and S. Stolfo, "On the accuracy of meta-learning for scalable data mining," *J. Intell. Inform. Syst.*, vol. 8, pp. 5–28, 1997.

[11] N. V. Chawla, K. W. Bowyer, L. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, 2002.

[12] N. Chawla, L. Hall, K. Bowyer, and W. Kegelmeyer, "Learning ensembles from bites: A scalable and accurate approach," *J. Mach. Learn. Res.*, vol. 5, pp. 421–451, 2004.

[13] J. Y. Ching, A. K. C. Wong, and K. C. C. Chan, "Class-dependent discretization for inductive learning from continuous and mixed mode data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, no. 7, pp. 641–651, Jul. 1995.

[14] M. Chisholm and P. Tadepalli., "Learning decision rules by randomized iterative local search," in *Proc. Int. Conf. Machine Learning*, 2002, pp. 75–82.

[15] D. Chiu, A. Wong, and B. Cheung, "Information discovery through hierarchical maximum entropy discretization and synthesis," in *Knowledge Discovery in Databases*, G. Piatesky-Shapiro and W. J. Frowley, Eds. Cambridge, MA: MIT Press, 1991.

[16] K. J. Cios and L. A. Kurgan, "Hybrid inductive machine learning: An overview of CLIP algorithms," in *New Learning Paradigms in Soft Computing*, L. C. Jain and J. Kacprzyk, Eds. Berlin, Germany: Physica-Verlag (Springer), 2002, pp. 276–322.

[17] ——, "Trends in data mining and knowledge discovery," in *Advanced Techniques in Knowledge Discovery and Data Mining*, N. R. Pal, L. C. Jain, and N. Teoderesku, Eds. Berlin, Germany: Physica-Verlag (Springer), 2005, pp. 1–26.

[18] K. J. Cios, W. Pedrycz, and R. Swiniarski, *Data Mining Methods for Knowledge Discovery*. Norwell, MA: Kluwer, 1998.

[19] K. J. Cios and L. A. Kurgan, "CLIP4: Hybrid inductive machine learning algorithm that generates inequality rules," in *Inform. Sci.*, S. K. Pal and A. Ghosh, Eds., 2004, vol. 163, Special Issue on Soft Computing Data Mining, pp. 37–83.

[20] K. J. Cios and G. Moore, "Uniqueness of medical data mining," *Artif. Intell. Med.*, vol. 26, no. 1-2, pp. 1–24, 2002.

[21] P. Clark and R. Boswell, "Rule induction with CN2: Some recent improvements," in *Proc. 5th Eur. Machine Learning Conf. (EWSL-91)*, Berlin, Germany, 1991, pp. 151–163.

[22] P. Clark and T. Niblett, "The CN2 algorithm," *Mach. Learn.*, vol. 3, pp. 261–283, 1989.

[23] W. Cohen and Y. Singer, "A simple, fast and effective rule learner," in *Proc. 16th Nat. Conf. Artificial Intelligence*, 1999, pp. 335–342.

[24] W. Cohen, "Efficient pruning methods for separate-and-conquer rule learning systems," in *Proc. 13th Int. Joint Conf. Artificial Intelligence*, Chambery, France, 1993, pp. 988–994.

[25] ——, "Fast effective rule induction," in *Proc. 12th Int. Conf. Machine Learning*, Lake Tahoe, CA, 1995, pp. 115–123.

[26] ——, "Grammatically biased learning: Learning logic programs using an explicit antecedent description language," *Artif. Intell.*, vol. 68, pp. 303–366, 1994.

[27] ——, "Learning trees and rules with set-valued features," in *Proc. 13th Nat. Conf. Artificial Intelligence*, Portland, OR, 1996, pp. 709–716.

[28] C. Cortes and V. N. Vapnik, "Support vector networks," *Mach. Learn.*, vol. 20, pp. 273–279, 1995.

[29] O. Dain, R. Cunningham, and S. Boyer, "IREP++ a faster rule learning algorithm," in *Proc. 4th SIAM Int. Conf. Data Mining*, Lake Buena Vista, FL, 2004, pp. 138–146.

[30] T. G. Dietterich and R. S. Michalski, "Inductive learning of structural descriptions: Evaluation criteria and comparative review of selected methods," *Artif. Intell.*, vol. 16, no. 3, pp. 257–294, 1981.

[31] P. Domingos, "The RISE system: Conquering without separating," in *Proc. 6th IEEE Int. Conf. Tools with Artificial Intelligence*, New Orleans, LA, 1994, pp. 704–707.

[32] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.

[33] U. M. Fayyad and K. B. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," in *Proc. 13th Int. Joint Conf. Artificial Intelligence*, San Francisco, CA, 1993, pp. 1022–1027.

[34] U. M. Fayyad, G. Piatesky-Shapiro, P. Smyth, and R. Uthurusamy, *Advances in Knowledge Discovery and Data Mining*: AAAI/MIT Press, 1996.

[35] A. Freitas, "A survey of parallel data mining," in *Proc. 2nd Int. Conf. Practical Applications of Knowledge Discovery and Data Mining*, London, U.K., 1998, pp. 287–300.

[36] J. Furnkranz, "Separate-and-conquer rule learning," *Artif. Intell. Rev.*, vol. 13, no. 1, pp. 3–54, 1999.

[37] J. Furnkranz and G. Widmer, "Incremental reduced error pruning," in *Machine Learning: Proc. 11th Annu. Conf.*, New Brunswick, NJ, 1994, pp. 70–77.

[38] L. Hall, N. Chawla, K. Bowyer, and W. Kegelmeyer, "Learning rules from distributed data," in *Proc. Workshop on Large-Scale Parallel KDD Systems, KDD'99*, 1999, pp. 77–83.

[39] S. Hettich and S. D. Bay. (1999) The UCI KDD Archive. University of California, Dept. Inform. Comput. Sci., Irvine, CA. [Online]. Available: http://kdd.ics.uci.edu

[40] M. Holsheimer and A. P. Siebes, "Data Mining: The Search for Knowledge in Databases,", Tech. Rep. CS-R9406, 1994.

[41] R. C. Holte, "Very simple classification rules perform well on most commonly used datasets," *Mach. Learn.*, vol. 11, pp. 63–90, 1993.

[42] K. A. Kaufman and R. S. Michalski, "Learning from inconsistent and noisy data: The AQ18 approach," in *Proc. 11th Int. Symp. Methodologies for Intelligent Systems*, Warsaw, Poland, 1999, pp. 411–419.

[43] R. Kerber, "ChiMerge: Discretization of numeric attributes," in *Proc. 9th Int. Conf. Artificial Intelligence (AAAI-91)*, 1992, pp. 123–128.

[44] Y. Kodratoff, *Introduction to Machine Learning*. New York: Morgan-Kaufmann, 1988.

[45] R. Kufrin, "Generating C4.5 production rules in parallel," in *Proc. 14th Nat. Conf. Artificial Intelligence (AAAI-97)*, 1997, pp. 565–670.

[46] L. A. Kurgan and K. J. Cios, "CAIM discretization algorithm," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 2, pp. 145–153, Feb. 2004.

[47] ——, "Discretization algorithm that uses class-attribute interdependence maximization," in *Proc. 2001 Int. Conf. Artificial Intelligence (IC-AI 2001)*, Las Vegas, NV, 2001, pp. 980–987.

[48] ——, "Ensemble of classifiers to improve accuracy of the CLIP4 machine learning algorithm," in *Proc. SPIE's Int. Conf. Sensor Fusion: Architectures, Algorithms, and Applications VI*, AeroSence 2002, Orlando, FL, 2002, pp. 22–31.

[49] ——, "Fast class-attribute interdependence maximization (CAIM) discretization algorithm," in *Proc. 2003 Int. Conf. Machine Learning and Applications (ICMLA'03)*, Los Angeles, CA, 2003, pp. 30–36.

[50] ——, "Meta mining architecture for supervised learning," in *Proc. 7th Int. Workshop on High Performance and Distributed Mining, in conjunction with the 4th Int. SIAM Conf. Data Mining*, Lake Buena Vista, FL, 2004, pp. 18–26.

[51] L. A. Kurgan, K. J. Cios, M. Sontag, and F. J. Accurso, "Mining the cystic fibrosis data," in *Next Generation of Data-Mining Applications*, J. Zurada and M. Kantardzic, Eds. New York: IEEE Press, 2005, pp. 415–444.

[52] L. A. Kurgan, "Meta Mining System for Supervised Learning," Ph.D, Dept. Comput. Sci., Univ. Colorado at Boulder, 2003.

[53] P. Langley and H. Simon, "Applications of machine learning and rule induction," *Commun. ACM*, vol. 38, no. 11, pp. 55–64, 1995.

[54] P. Langley, *Elements of Machine Learning*. New York: Morgan-Kaufmann, 1996.

[55] T.-S. Lim, W.-Y. Loh, and Y.-S. Shih, "A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms," *Mach. Learn.*, vol. 40, pp. 203–228, 2000.

[56] R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac, "The multipurpose incremental learning system AQ15 and its testing application to three medical domains," in *Proc. 5th Nat. Conf. Artificial Intelligence*, 1986, pp. 1041–1045.

[57] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[58] G. Pagallo and D. Haussler, "Boolean feature discovery in empirical learning," *Mach. Learn.*, vol. 5, no. 1, pp. 71–99, 1990.

[59] *Machine Learning and its Applications*, vol. 2049, Lecture Notes in Computer Science, G. Paliouras, V. Karkaletsis, and C. Spyropoulos, Eds., Springer, Berlin, Germany, 2001.

[60] M. Pazzani and D. Kibler, "The utility of knowledge in inductive learning," *Mach. Learn.*, vol. 9, no. 1, pp. 57–94, 1992.

[61] F. Provost and D. Hennessy, "Scaling up: Distributed machine learning with cooperation," in *Proc. 13th Nat. Conf. Artificial Intelligence*, 1996, pp. 74–79.

[62] F. Provost and T. Fawcett, "Analysis and visualization of classifier performance: Comparison under imprecise class and cost distribution," in *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining*, 1997, pp. 43–48.

[63] F. Provost and V. Kolluri, "A survey of methods for scaling up inductive algorithms," *Data Mining Knowl. Disc.*, vol. 3, no. 2, pp. 131–169, 1999.

[64] F. Provost and T. Fawcett, "Robust classification for imprecise environments," *Mach. Learn.*, vol. 42, no. 3, pp. 203–231, 2001.

[65] J. R. Quinlan, *C4.5 Programs for Machine Learning*. New York: Morgan-Kaufmann, 1993.

[66] ——, "Improved use of continuous attributes in C4.5," *J. Artif. Intell. Res.*, vol. 7, pp. 77–90, 1996.

[67] ——, "Induction of decision trees," *Mach. Learn.*, vol. 1, pp. 81–106, 1986.

[68] ——, "Learning logical definitions from relations," *Mach. Learn.*, vol. 5, pp. 239–266, 1990.

[69] ——, "Simplifying decision trees," *Int. J. Man-Mach. Stud.*, vol. 27, no. 3, pp. 221–248, 1987.

[70] ——, "Some elements of machine learning," in *Proc. 16th Int. Conf. Machine Learning*, 1999, pp. 523–525.

[71] S. Ranka, "Real-time data mining on grids," in *(Invited Presentation) 7th Int. Workshop on High Performance and Distributed Mining, in Conjunction with the 4th Int. SIAM Conf. Data Mining*, Lake Buena Vista, FL, 2004.

[72] J. F. Roddick and M. Spiliopoulou, "A survey of temporal knowledge discovery paradigms and methods," *IEEE Trans. Knowl. Data Eng.*, vol. 14, no. 4, pp. 750–767, Apr. 2002.

[73] RuleQuest Research, C5.0 (2003). [Online]. Available: http://www.rulequest.com/see5-info.html

[74] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," in *Proc. 11th Annu. Conf. Computational Learning Theory*, 1998, pp. 80–91.

[75] M. Sebag, "Delaying the choice of bias: A disjunctive version space approach," in *Proc. 13th Int. Conf. Machine Learning*, 1996, pp. 444–452.

[76] M. Spiliopoulou and J. F. Roddick, "Higher order mining: Modeling and mining the results of knowledge discovery," in *Data Mining I: Proc. 2nd Int. Conf. Data Mining Methods and Databases*, 2000, pp. 309–320.

[77] H. Theron and I. Cloete, "BEXA: A covering algorithm for learning propositional concept descriptions," *Mach. Learn.*, vol. 24, pp. 5–40, 1996.

[78] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer, 1995.

[79] P. Vlachos. (2000) StatLib Project Repository. [Online]. Available: http://lib.stat.cmu.edu

[80] G. I. Webb. and J. Agar, "Inducing diagnostic rules for glomerular disease with the DLG machine learning algorithm," *Artif. Intell. Med.*, vol. 4, pp. 3–14, 1992.
[81] S. Weiss and N. Indurkhya, "Reduced complexity rule induction," in *Proc. 12th Int. Joint Conf. Artificial Intelligence*, Sydney, Australia, 1991, pp. 678–684.
[82] R. Winter and K. Auerbach, "Contents under pressure," *Intell. Enterprise*, May 2004.

Lukasz A. Kurgan (M'02) received the M.Sc. (Hons.) degree (recognized by an Outstanding Student Award) in automation and robotics from the AGH University of Science and Technology, Krakow, Poland, in 1999, and the Ph.D. degree in computer science from the University of Colorado at Boulder in 2003.

He is currently an Assistant Professor in the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada. His research interests include data mining and knowledge discovery, machine learning, computational biology and bioinformatics. He authored and co-authored several inductive machine learning and data mining algorithms. He has published about 30 journal and conference articles, including several IEEE Transactions journals.

Dr. Kurgan is a member of a steering committee of the International Conference on Machine Learning and Applications, being also actively involved in organization of numerous special sessions related to computational biology and bioinformatics. He is a member of ACM.

Krzysztof J. Cios (SM'90) received the M.S. and Ph.D. degrees from the AGH University of Science and Technology, Krakow, Poland, the MBA from the University of Toledo, Toledo, OH, and the D.Sc. degree from the Polish Academy of Sciences.

He is currently a Professor at the University of Colorado at Denver and Health Sciences Center, and Associate Director of both, and a Professor at the University of Colorado Bioenergetics Institute and the University of Colorado Center for Computational Biology. He also directs the Data Mining and Bioinformatics Laboratory. He is a well-known Researcher in the areas of machine learning, biomedical informatics and data mining. His research has been funded by NASA, the NSF, American Heart Association, Ohio Aerospace Institute, NATO, Colorado Institute of Technology, U.S. Air Force, NIH, and Microsoft. He published two books, over 50 journal articles, 12 book chapters, and over 60 peer-reviewed conference papers.

Dr. Cios serves on the Editorial Boards of *Neurocomputing, IEEE Engineering in Medicine and Biology Magazine*, *International Journal of Computational Intelligence*, and *Integrative Neuroscience*. He has been the recipient of the Norbert Wiener Outstanding Paper Award, the *Neurocomputing* Best Paper Award, the University of Toledo Outstanding Faculty Research Award, and the Fulbright Senior Scholar Award. He is a Senior Member of ACM, AAAI, Sigma Xi, and PIASA. He serves as Chair of the Computational Intelligence Society Chapter, in the IEEE Denver section. In 2004, he was elected a Foreign Member of the Polish Academy of Arts and Sciences.

Scott Dick (M'03) received the B.Sc. degree in 1997, the M.Sc. degree in 1999, and the Ph.D. degree in 2002, all from the University of South Florida, Tampa.

He has been an Assistant Professor in the Department of Electrical and Computer Engineering at the University of Alberta, Edmonton, AB, Canada, since July 2002, and has published over a dozen scientific articles in journals and conferences.

Dr. Dick is a member of the ACM, ASEE, and Sigma Xi. His Ph.D. dissertation received the USF Outstanding Dissertation Prize in 2003.