Meta Mining Architecture for Supervised Learning

Lukasz A. Kurgan[1] Krzysztof J. Cios[2]

Abstract

Compactness of a generated data model and the scalability of data mining algorithms are important in any data mining (DM) undertaking. This paper addresses the problems by introducing a novel DM system called MetaSqueezer. The system is probably the first machine-learning-based system to use a Meta Mining concept to generate data models from already generated meta-data. The main advantages of the system are the compactness of the knowledge models, scalability, and suitability for parallelization. The system generates a set of production rules that describe target concepts from supervised data. The system was extensively benchmarked and is shown to generate data models in a linear time, which together with its parallelizable architecture makes it applicable for handling large amounts of data.

Keywords

Data Mining, Machine Learning, Meta Mining, Parallel Architectures, Production Rules, Supervised Learning, MetaSqueezer

1 Introduction.

Machine Learning (ML) is one of the key and most popular DM tools used in the knowledge discovery (KD) process, defined as a nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns from large collections of data [9].

One of the frequently used ML techniques for generation of data models is induction, which infers generalized information, or knowledge, by searching for regularities among the data. The output of an inductive ML algorithm usually takes the form of production IF… THEN… rules, or decision trees that can be converted into rules. One of the reasons why rules or trees are very popular (e.g. in knowledge-based systems) is that they are simple and easy to interpret and learn from. The rules can be modified because of their modularity; i.e. a single rule can be understood without reference to other rules [15]. This is why ML is very popular DM method in situations where a decision maker needs to understand and validate the generated model, like in medicine.

Below, we describe a novel DM system for analysis of supervised data, which has three key features:

1. It generates compact models, in the form of production rules. It generates small number of rules that are very compact, in terms of the number of selectors they use, which makes them easy to evaluate and understand.
2. It has linear complexity with respect to the number of examples in the input data, which enables using it on very large datasets.
3. The system's novel architecture supports parallelization, which results in possibility of further performance improvements. The system is able to naturally and profitable adapt to distributed environments.

The architecture of the system is based on a Meta Mining concept, explained later, which is largely responsible for achieving the above three characteristics.

The system is an alternative to already existing parallel implementation of many DM systems. In general two standard parallel computation models exist: message passing model that uses distributed memory, and shared memory model that uses physically shared memory. The current approach in parallelization of DM algorithms uses both models. The message passing solutions concern a wide range of algorithms including association rules [2] [13] [37], clustering [7], and decision trees [12] [34]. More recently, limited number of shared memory parallelization solutions were developed including association rules [26] [16], sequence mining [38], and decision trees [39]. The system can be successfully implemented using both paradigms, and at the same time is linear, and provides very compact results.

[1] University of Alberta, Edmonton, AB, Canada
email: lkurgan@ece.ualberta.ca
[2] University of Colorado at Denver, Denver, CO, U.S.A.
University of Colorado at Boulder, Boulder, CO, U.S.A.
University of Colorado Health Sciences Center, Denver, CO, U.S.A.
4cData, Golden, CO, U.S.A
email: Krys.Cios@cudenver.edu

2 Overview of the System.

The system uses supervised inductive ML methods and a Meta Mining (MM) concept. MM is a generic framework for higher order mining. Its main characteristic is generation of data models, called meta-models (often meta-rules), from the already generated data models (usually rules, called meta-data) [33]. The system has three steps. First, it divides the input data into subsets. Next, it generates a data model for each subset. Then it takes the generated data models and generates the meta-model from them. There are several advantages to using MM:

- Generation of compact data models. Since a MM system generates results from already mined data models, they capture patterns present in meta-data. The results generated by the MM system are different than results of mining from the entire data. Researchers argue that meta results better describe knowledge that can be considered interesting [1] [30] [33].
- Scalability. Although most of ML tools are not scalable there are some that scale well with the size of input data [32] [11]. The recent approach to dealing with scalability of ML algorithms is to use the MM concept [19] [21]. The MM system analyses many small datasets (i.e. subsets of original data, and the data models) instead of one large input dataset. This results in reduction of computational time, especially for systems that use non-linear algorithms for generation of data models, and most of all in ability to implement it in parallel or distributed fashion.

The MM model usually applies the same base-learner (algorithm) on the data to produce a hypothesis, but performs it in two steps where the outcome is generated from results of the first step. In contrast, the meta-learning aims to discover the best learning strategy through continuing adaptation of the algorithms at different levels of abstraction, like for example through dynamic selection of bias [35].

The MM concept already has found some applications. It was used to generate association rules, which resulted in their ability to describe changes in the data rather than the data itself [29], and for incremental discovery of meta-rules [1].

3 Description of the System.

The system uses inductive ML techniques to generate meta-rules from supervised data in these steps:

- Preprocessing.
    - the data is validated by repairing or removing incorrect records, and marking unknown values,
    - continuous attribute are discretized by a supervised discretization algorithm CAIM [18] [20] [23].
    - the data is divided into subsets. Selection of the proper number of subsets depends on the input data size. The number of subsets usually should be relatively small, so that the size of each input subset would allow generation of quality rule sets for each of the classes. In case of a large number of subsets the system may generate inaccurate rules since the amount of examples for each subset would be too small to generate correct meta-data. The simplest way to divide input data is to perform random splitting. It is also possible to divide the input data in a predefined way. For example, the system can be used for analysis of temporal data, which can be divided into subsets corresponding to different time intervals. As another example, the system was already used to analyze large medical dataset, where the data was divided into subsets corresponding to different stages of a disease [21] [22].
- Data Mining
    - production rules are generated from data for each of the defined subsets by the DataSqueezer algorithm [19] [21].
    - a rule table, which stores rules in a format that is identical to the format of the original input data, is created from the generated rules and separately for each of the subsets. Each table stores meta-data about one of the data subsets.
- Meta Mining
    - MM generates meta-rules from rule tables. First, all rule tables are concatenated into a single table. Next, the meta-rules are generated by the DataSqueezer algorithm. The meta-rules describe the most important patterns associated with the target concept over the entire input dataset.

3.1 The DataSqueezer Algorithm.

It is the core algorithm used in the system. The DataSqueezer algorithm is used to generate meta-data during the DM step, and the meta-rules in the MM step. A survey of relevant inductive ML algorithms can be found in [10]. DataSqueezer is an inductive ML algorithm that generates rules by finding regularities in the data [19] [21].

```
Given:   POS, NEG, K (number of attributes), S (number of examples)
Step1.
1.1      Initialize $G_{POS}$ = []; i=1; j=1; k=1; tmp = $pos_1$;
1.2.1    for k = 1 to K                                        // for all attributes
1.2.2      if ($pos_j[k] \neq tmp[k]$ or $pos_j[k]$ = '∗')
1.2.3        then tmp[k] = '∗';                                // '∗' denotes missing value
1.2.4    if (number of non missing values in tmp $\geq 2$)
1.2.5        then $gpos_i$ = tmp; $gpos_i[K+1]$ ++;
1.2.6      else i ++; tmp = $pos_j$;
1.3      set j++; and until $j \leq N_{POS}$ go to 1.2.1
1.4      Initialize $G_{NEG}$ = []; i=1; j=1; k=1; tmp = $neg_1$;
1.5.1    for k = 1 to K                                        // for all attributes
1.5.2      if ($neg_j[k] \neq tmp[k]$ or $neg_j[k]$ = '∗')
1.5.3        then tmp[k] = '∗';                                // '∗' denotes missing value
1.5.4    if (number of non missing values in tmp $\geq 2$)
1.5.5        then $gneg_i$ = tmp; $gneg_i[K+1]$ ++;
1.5.6      else i ++; tmp = $neg_j$;
1.6      set j++; and until $j \leq N_{NEG}$ go to 1.5.1
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 column of $G_{POS}$ that is on LIST, for every non missing value a from selected column k compute sum, $s_{ak}$, of values of
         $gpos_i[K+1]$ for every row i, in which a appears (multiply every $s_{ak}$, by the number of values the attribute k has)
2.4      select maximal $s_{ak}$, remove k from LIST, add "k = 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
```

Figure 1. Pseudo-code of the DataSqueezer algorithm.

Let us denote the input dataset by D, which consists of S examples. The sets of positive examples, D_P, and negative examples, D_N, must satisfy three properties: $D_P \cup D_N = D$, $D_P \cap D_N = \emptyset$, $D_N \neq \emptyset$, and $D_P \neq \emptyset$. The positive examples are those describing the class for which we currently generate rules, while negative examples are the remaining examples. Examples are described by a set of K attribute-value pairs: $e = \wedge_{j=1}^{K}[a_j \# v_j]$, where a_j denotes jth attribute with value $v_j \in d_j$ (domain of values of jth attribute), # is a relation (=, <, ≈, ≤, etc.), and K is the number of attributes. In the DataSqueezer algorithm the relation we use is equality. An example, e, consists of set of selectors $s_j = [a_j = v_j]$. The DataSqueezer algorithm generates production rules in the form of: IF (s_1 and … and s_m) THEN class = $class_i$, where $s_i = [a_j = v_j]$ is a single selector, and m is the number of selectors in the rule. Figure 1 shows pseudo-code for the DataSqueezer algorithm.

D_P and D_N are tables whose rows represent examples and columns correspond to attributes. Table of positive examples is denoted as POS and the number of positive examples by N_{POS}, while the table and the number of negative examples as NEG and 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 are described by the set of values: $pos_i[j]$ where j=1,…,K, is the column number, and i is the example number (row number in the POS table). The negative examples are described similarly by a set of $neg_i[j]$ values. The DataSqueezer algorithm also uses tables that store intermediate results (G_{POS} for POS table, and G_{NEG} for NEG table), which have K columns. Each cell of the G_{POS} table is denoted as $gpos_i[j]$, where i is a row number and j is a column number, and similarly for G_{NEG} table is denoted by $gneg_i[j]$. The G_{POS} table stores reduced subset of the data from POS, and G_{NEG} table stores reduced subset of the data from NEG. The meaning of this reduction is explained later. The G_{NEG} and G_{POS} tables have an additional (K+1)th column that stores number of examples from the NEG and POS, which a particular row in G_{NEG} and G_{POS} describes, respectively. Thus, for example $gpos_2[K+1]$ stores number of examples from POS, which are described by the 2nd row in G_{POS} table.

The rule generation mechanism used by the DataSqueezer algorithm is based on the inductive learning hypothesis [25]. It states that any hypothesis found to approximate the target function (target concept defined by a class attribute) well, over a sufficiently large set of training examples, will also well approximate the target function over other unobserved examples. Based on this assumption, the algorithm first performs data reduction via use of the prototypical concept learning, which is very similar to the Find S algorithm of Mitchell [25]. It performs data reduction to generalize information stored in the original data. Data reduction is done for both positive and negative data. Next, the algorithm generates rules by performing greedy hill-climbing search on the reduced data. A rule is generated by applying the search procedure starting with an empty rule, and adding selectors until the termination criterion fires. The max 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. Another closely related algorithm is the Disjunctive Version Spaces (DiVS) algorithm [31]. The DiVS algorithm 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 problems the DataSqueezer algorithm generates rules for every class, each time generating rules that describe the currently chosen (positive) class.

The algorithm has the following features:

- it generates production rules that involve no more than one selector per attribute. This property allows for storing of the rules in a table that has identical structure as the original data table. For example, for data described by attributes *A*, *B*, and *C*, and describing two classes *white*, and black, the following rule can be generated: *IF A=1 and C=1 THEN black*. The rule can be written as (1, ?, 1, black), following the format of the input data, which defines values of attributes A,B, and C, and adds class attribute. This data can be used as an input to the same algorithm that was used to generate the rules.
- it generates rules that are very compact in terms of the number of selectors (shown experimentally later).
- it can handle data with large number of missing values. DataSqueezer algorithm can cope with data that has large number of missing values. It uses only complete data, including all examples from the original data, even those that contain missing values. In other words it uses all available information while ignoring missing values, i.e. they are handled as "do not care" values.
- it has linear complexity, in respect to the number of examples in the datasets [21].

3.2 The MetaSqueezer System.

The architecture of the MetaSqueezer systems is shown in Figure 2. The raw data are first preprocessed and discretized using the CAIM algorithm, since the DataSqueezer algorithm receives only discrete numerical or nominal data as its input. Next, data is divided into subsets that are fed into the DM step. DM generates meta-data from the data subsets using the DataSqueezer algorithm. In the MM step the meta-data generated for each of the subsets is concatenated and fed again into DataSqueezer to generate meta-rules. The idea of splitting up the dataset, learning a rule set on each split, and combining the results has been previously introduced [8], but the MetaSqueezer system is the first that combines the rule sets in a separate, second learning phase.

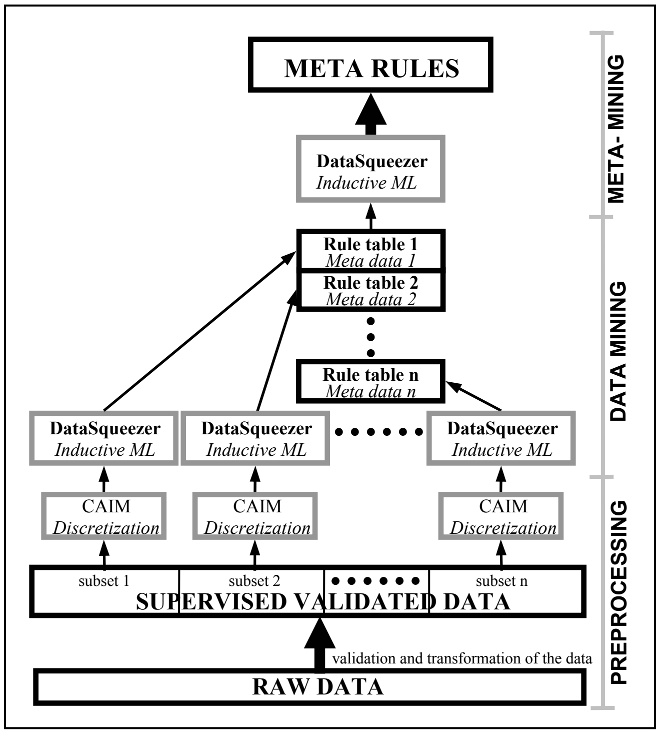

Figure 2. Architecture of the MetaSqueezer system.

3.3 The MetaSqueezer System.

The architecture of the MetaSqueezer systems is shown in Figure 2. The raw data are first preprocessed and discretized using the CAIM algorithm, since the DataSqueezer algorithm receives only discrete numerical or nominal data as its input. Next, data is divided into subsets that are fed into the DM step. DM generates meta-data from the data subsets using the DataSqueezer algorithm. In the MM step the meta-data generated for each of the subsets is concatenated and fed again into DataSqueezer to generate meta-rules. The idea of splitting up the dataset, learning a rule set on each split, and combining the results has been previously introduced [8], but the MetaSqueezer system is the first that combines the rule sets in a separate, second learning phase.

The complexity of the MetaSqueezer system is determined by complexity of the DataSqueezer algorithm. Assuming that S is the number of examples in the original dataset, the MetaSqueezer system divides the data into n subsets of the same size, which is equal to S/n. In the DM step the MetaSqueezer system uses DataSqueezer algorithm n times, which gives total complexity of nO(S) = O(S). In the MM step the DataSqueezer algorithm is run once with the data of size O(S). Thus, complexity of the MetaSqueezer system is O(S) + O(S) = O(S). This result is also verified experimentally in section 4.2.

Table 1. Description of datasets used for benchmarking.

set	size	# classes	# attrib.	test data	# subsets	set	size	# classes	# attrib.	test data	# subsets
Wisconsin breast cancer (bcw)	699	2	9	10CV	4	**image segmentation (seg)**	2310	7	19	10CV	6
BUPA liver disorder (bld)	345	2	6	10CV	3	**attitude smoking restr. (smo)**	2855	3	13	1000	4
Boston housing (bos)	506	3	13	10CV	3	**thyroid disease (thy)**	7200	3	21	3428	6
contraceptive method choice (cmc)	1473	3	9	10CV	7	**StatLog vehicle silhouette (veh)**	846	4	18	10CV	4
StatLog DNA (dna)	3190	3	61	1190	8	**congressional voting rec (vot)**	435	2	16	10CV	3
StatLog heart disease (hea)	270	2	13	10CV	3	**waveform (wav)**	3600	3	21	3000	3
LED display (led)	6000	10	7	4000	10	**TA evaluation (tae)**	151	3	5	10CV	2
PIMA indian diabetes (pid)	768	2	8	10CV	4	**census-income (cid)**	299285	2	40	99762	10
StatLog satellite image (sat)	6435	6	37	2000	10						

Table 2. Accuracy results for the MetaSqueezer, DataSqueezer, CLIP4, and the other 33 ML algorithms.

set	Reported results [24]		CLIP4 [6]	**DataSqueezer**			**MetaSqueezer**		
	max	min	accuracy	**mean accuracy**	mean sensitivity	mean specificity	**mean accuracy**	mean sensitivity	mean specificity
bcw	97	91	95	94	92	98	94	97	85
bld	72	57	63	68	86	44	70	93	38
bos	78	69	71	70	70	88	71	70	86
cmc	57	40	47	44	40	73	47	43	72
dna	95	62	91	92	92	97	90	89	95
hea	86	66	72	79	89	66	79	87	70
led	73	18	71	68	68	97	69	69	97
pid	78	69	71	76	83	61	75	83	59
sat	90	60	80	80	78	96	74	73	95
seg	98	48	86	84	83	98	81	81	97
smo	70	55	68	68	33	67	67	33	69
thy	99	11	99	96	95	99	96	86	99
veh	85	51	56	61	61	88	60	59	87
vot	96	94	94	95	93	96	94	92	99
wav	85	52	75	77	77	89	77	76	89
tae	77	31	60	55	53	79	52	51	76
MEAN	**83.5**	**54.6**	**74.9**	**75.4**	74.6	83.5	**74.8**	74.0	82.1
set	algorithm (accuracy) [reference]			**mean accuracy**	mean sensitivity	mean specificity	**mean accuracy**	mean sensitivity	mean specificity
cid	C4.5 (95.2), C5.0 (95.3), C5.0 rules (95.3), C5.0 boosted (95.4), Naïve-Bayes (76.8) [14]			91	94	45	90	93	49

Table 3. Number of rules and selectors for the MetaSqueezer, DataSqueezer, CLIP4, and the other 33 algorithms.

set	Reported median # of leaves/rules [24]	CLIP4 [6]			**DataSqueezer**			**MetaSqueezer**		
		mean # rules	mean # selectors	# selectors per rule	mean # rules	mean # selectors	# selectors per rule	mean # rules	mean # selectors	# selectors per rule
bcw	7	4.2	121.6	29.0	4.5	12.8	2.8	6.3	12.3	1.9
bld	10	9.7	272.4	28.1	3.4	14.0	4.1	2.6	7.7	3.0
bos	11	10.5	133.5	12.7	19.8	107	5.4	17.9	56.3	3.1
cmc	15	8	60.7	7.6	20.2	70.5	3.5	17.4	42.1	2.4
dna	13	8	90	11.3	39.0	97.0	2.5	34.0	53.0	1.6
hea	6	11.6	192.3	16.6	4.7	17.1	3.6	1.9	3.7	1.9
led	24	41	189	4.6	51	194	3.8	51	141	2.8
pid	7	4	64.1	16.0	1.8	8.0	4.4	2.1	9.3	4.4
sat	63	61	3199	52.4	57	257	4.5	55	104	1.9
seg	39	39.2	1169.9	29.8	57.3	219	3.8	50.7	89.3	1.8
smo	2	18	242	13.4	6	12	2.0	3	10	3.7
thy	12	4	119	29.8	7	28	4.0	6	6	1.0
veh	38	21.3	380.7	17.9	23.7	80.2	3.4	22.4	41.4	1.8
vot	2	9.7	51.7	5.3	1.4	1.6	1.1	1	1	1.0
wav	16	9	85	9.4	22	65	2.9	17	18	1.0
tae	20	9.3	273.2	29.4	21.2	57.2	2.7	14.7	27.8	1.9
MEAN	**17.8**	**16.8**	**415.3**	**19.6**	**21.3**	**77.5**	**3.4**	**18.9**	**38.9**	**2.2**
cid		---	---	---	15	95	6.3	6	34	5.7

The architecture of the system allows natural implementation in a parallel or distributed manner. Generation of each of the data models from a subset of the original data, which is performed in the DM step, can be handled by a distinct serial program. The generated data models can be easily combined and fed back to compute the MM step. Although the experimental results shown in this paper are based on a non-parallel implementation, it is important to note that this architecture has a potential to be implemented applying grid computing resources, in contrast to most of the inductive ML algorithms.

4 Experiments.

The Meta Squeezer system was tested on 17 datasets characterized by the size of training datasets (between 151 and 200K examples), the size of testing datasets (between 15 and 100K examples), the number of attributes: (between 5 and 61), and the number of classes (between 2 and 10).
The datasets were obtained from the UCI ML repository [3], and from StatLog project datasets repository [36]. The datasets were randomly divided into a number of equal size subsets, depending on the size of the data, to be used as an input to the DM step. The description of the datasets and number of subsets is shown in Table 1.

We compare the benchmarking results in terms of accuracy of the rules, number of rules and selectors used, and the execution time. The system was compared to CLIP4 algorithm [5] [6], DataSqueezer algorithm, and 33 other ML algorithms, for which the results are available [24]. The first 16 datasets were used to perform the comparison. The last, cid, dataset was primarily used to experimentally analyze complexity of the system.

The test procedure of the DataSqueezer algorithm was the same as test procedures used in [24] to enable direct and fair comparison between the algorithms. As in [24] some of the tests were performed using k-fold cross validation.

4.1 Discussion of the Results.

The benchmarking results, shown in Table 2, compare accuracy of the MetaSqueezer system with other ML algorithms. For the 33 algorithms used, we show maximum and minimum accuracy. Only the MetaSqueezer algorithm uses the MM concept while the other algorithms use the entire datasets (not divided into subset) as their input.
By far the most popular in the ML field for comparison of the results is the accuracy test, however, more precise verification test results are reported for the MetaSqueezer and DataSqueezer. The verification test is a standard used in medicine where sensitivity and specificity analysis is used to evaluate confidence in the results [4], and is related to ROC analysis of classifier performance [28]. 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.

The mean accuracy of the MetaSqueezer system for the 16 datasets is 74.8%. For comparison, the POLYCLASS algorithm [17] achieved the highest mean accuracy of 80.5%. The [24] calculated statistical significance of error rates. It shows that a difference between the mean accuracies of two algorithms is statistically significant at the 10% level if they differ by more than 5.9%. They reported that 26 out of 33 algorithms were not statistically significantly different from POLYCLASS. The difference in error rates between the MetaSqueezer and POLYCLASS also fells within the same category. The same conclusion can be drawn for the CLIP4 [6], and the DataSqueezer algorithms.

Table 3 shows the number of rules, and the number of selectors. The MetaSqueezer system is compared with the results of the 33 algorithms reported in [24], and with results of the CLIP4 [6], and DataSqueezer algorithms. For the 33 algorithms the median number of rules (the authors reported the number of tree leaves for 21 decision tree algorithms) is reported. Additionally, for the MetaSqueezer, DataSqueezer, and CLIP4 algorithms we report number of selectors per rule. The last measure enables direct comparison of complexity of the generated rules. The mean number of rules generated by the MetaSqueezer system is 18.9. In [24] the median number of tree leaves, equivalent to the number of rules, for the 21 tested decision tree algorithms was 17.8. The number of rules generated by the CLIP4 algorithm was 16.8, and by the DataSqueezer algorithm 21.3. We also note that the DataSqueezer generates more rules than the MetaSqueezer system.

An important advantage of the MetaSqueezer system can be seen by analyzing the number of selectors and the number of selectors per rule; for the 16 datasets it is 2.2. That means that on average each rule generated by the system involves only 2.2 attribute-values pairs in the condition part of the rule. This is significantly less than the number of selectors per rule achieved by the DataSqueezer algorithm (35% less), and the CLIP4 algorithm (almost 90% less). This is primarily due to using the MM concept where the meta-rules are generated from the meta-data. All other algorithms generate rules directly from input data.

We note that the execution times of the 33 algorithms are either higher or at best similar to execution time of the MetaSqueezer system. For example, the POLYCLASS algorithm had a mean execution time of 3.2 hours, while both DataSqueezer and MetaSqueezer, and best algorithms among the 33 had the execution time of about 5 seconds when comparing the results on a common hardware platform [24].

To summarize, two main advantages of the MetaSqueezer system are the compactness of the generated rules and low computational cost. The experimental results show that the system is fast and generates very low number of selectors per rule. The difference in accuracy between the system and the best among all tested ML algorithms is not statistically significant.

4.2 Scalability Analysis.

The MetaSqueezer system and the DataSqueezer algorithm are tested to experimentally show their linear complexity. The tests are performed using the cid dataset. The dataset has 40 attributes and 300K samples. The training part of the original dataset was used to derive training sets for the complexity analysis. The training datasets were derived by selecting a number of first records from the original training dataset. Standard procedure of using training datasets of doubled size to verify if the execution time is also doubled was followed. Figure 3 visualizes the results as a graph of relation between execution time and the size of data. It uses logarithmic scale on both axes to help visualize data points with low values. The results show linear relationship between the execution time and the size of the input data for both algorithms that agrees with theoretical complexity analysis. The time ratio is always close to the data size ratio. The graph also shows that the MetaSqueezer algorithm has additional overhead caused by division of data into subsets, but it becomes insignificant with the growing size of input data. This property is exhibited by initially lower execution time of the DataSqueezer algorithm, while the difference in execution time between the two algorithms shrinks with growth of data size.

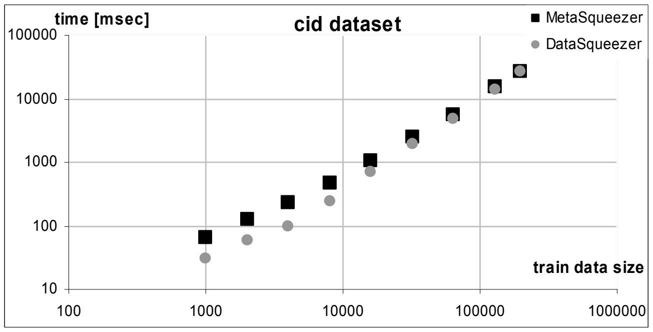

Figure 3. Relation between execution time and input data size for the MetaSqueezer system and the DataSqueezer algorithm.

5 Summary and Conclusions.

Often data mining and, in particular, machine learning tools lack scalability for generation of potentially new and useful knowledge. Most of the ML algorithms do not scale well with the size of the input data and may generate very complex rules. In this work we introduced a novel Machine Learning system, called MetaSqueezer, which was designed using the Meta Mining (MM) concept. The system generates a set of meta-rules from supervised data and can be used for data analysis tasks in various domains. The main advantages of the system are high compactness of the generated meta-rules, low computational cost, and ability to implement it in a parallel or distributed fashion. The extensive tests showed that the system is fast and generates the least complex data models, as compared with other systems, while at the same time generating very good results.

The main reason for compactness of the results is the use of the MM concept. While many other Machine Learning algorithms generate rules directly from input data, the MetaSqueezer generates meta-rules from previously generated meta-data. Our results, as well the results of other researchers [1], indicate that MM concept helps to generate rules more focused on the target concept. The main benefit of the reduced complexity of the models is increased comprehension of the generated knowledge.

The use of the MM concept results in ability to implement the MetaSqueezer's in parallel or distributed fashion. The algorithm works by merging partial knowledge extracted from subsets of data, which can be computed in parallel on (possibly disjoint) subsets of training data [27]. The biggest advantage of the described system is that the application of the distributed or parallel approach for rule generation does not result in lower accuracy of the results. These features make it applicable in many domains that use grid computing resources for working on very large datasets.

Low computational cost of the MetaSqueezer system stems from the fact that the execution time grows linearly with linear growth of the size of input data thus making it applicable for very large datasets. Good application domains for the system are those that involve temporal or ordered data. In that case the rules are generated for particular time intervals first and only then the meta-knowledge is generated.

In the future we plan to develop a parallel implementation of the algorithm in the message passing environment. The goal is to develop an algorithm suitable to efficiently analyze massive datasets.

Acknowledgments

The authors thank anonymous reviewers for their comments.

Reference

[1] Abraham, T., & Roddick, J. F., Incremental Meta-mining from Large Temporal Data Sets, Advances in Database Technologies, *Proceedings of the 1st International Workshop on Data Warehousing and Data Mining* (DWDM'98), pp.41-54, 1999

[2] Agrawal, R., & Shafer, J., Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering*, 8:6, pp.962-969, 1996

[3] Blake, C.L. & Merz, C.J., *UCI Repository of ML Databases*, http://www.ics.uci.edu/~mlearn/MLRep ository.html, Irvine, CA: University of California, Department of Information and Computer Science, 1998

[4] Cios, K.J., & Moore, G., Uniqueness of Medical Data Mining, *Artificial Intelligence in Medicine*, 26:1-2, pp.1-24, 2002

[5] Cios K. J. & Kurgan L., Hybrid Inductive Machine Learning: An Overview of CLIP Algorithms, In: Jain L.C., & Kacprzyk, J., (Eds.) *New Learning Paradigms in Soft Computing*, Physica-Verlag (Springer), pp. 276-322, 2001

[6] Cios, K.J., & Kurgan, L., CLIP4: Hybrid Inductive Machine Learning Algorithm that Generates Inequality Rules, *Information Sciences*, special issue on *Soft Computing Data Mining*, in print, 2004

[7] Dhillon, I., & Modha, D., A Data-Clustering Algorithm on Distributed Memory Multiprocessors, *Proceedings of Workshop on Large-Scale Parallel Knowledge Discovery in Databases Systems*, with *ACM SIGKDD-99*, pp.47–56, 1999

[8] Domingos, P., Using Partitioning to Speed up Specific-to-General Rule Induction, *Proceedings of the AAAI-96 Workshop on Integrating Multiple Learned Models for Improving and Scaling Machine Learning Algorithms*, Portland, OR, 1996

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

[10] Furnkranz, J., Separate-and-conquer Rule Learning, *Artificial Intelligence Review*, 13:1, pp. 3-54, 1999

[11] Gehrke, J., Ramakrishnan, R., and Ganti, V., RainForest - a Framework for Fast Decision Tree Construction of Large Datasets, *Proceedings of the 24th International Conference on Very Large Data Bases*, San Francisco, pp. 416-427, 1998

[12] Goil, S., & Choudhary, A., Efficient Parallel Classification Using Dimensional Aggregates, *Proceedings of Workshop on Large-Scalable Parallel Knowledge Discovery in Databases Systems*, with *ACM SIGKDD-99*, 1999

[13] Han, E, Karypis, G, & Kumar, V, Scalable Parallel Data Mining for Association Rules, *IEEE Transactions on Knowledge and Data Engineering*, 12:3, pp.337-352, 2000

[14] Hettich, S. & Bay, S. D., *The UCI KDD Archive*, http://kdd.ics.uci.edu, Irvine, CA: University of California, Department of Information and Computer Science, 1999

[15] Holsheimer, M., & Siebes, A.P., Data Mining: The Search for Knowledge in Databases, *Technical report CS-R9406*, 1994

[16] Jin, R., & Agrawal, G., Shared Memory Parallelization of Data Mining Algorithms: Techniques, Programming Interface, and Performance, *Proceedings of the 2nd SIAM Conference on Data Mining*, 2002

[17] Kooperberg, C., Bose, S. & Stone, C.J., Polychotomous Regression, *Journal of the American Statistical Association*, 92, pp.117-127, 1997

[18] Kurgan, L., & Cios, K.J., Discretization Algorithm that Uses Class-Attribute Interdependence Maximization, *Proceedings of the 2001 Inter. Conf. on Artificial Intelligence* (IC-AI 2001), Las Vegas, Nevada, pp.980-987, 2001

[19] Kurgan, L. & Cios, K.J., DataSqueezer Algorithm that Generates Small Number of Short Rules, *IEE Proceedings: Vision, Image and Signal Processing*, submitted, 2003

[20] Kurgan, L., & Cios, K.J., Fast Class-Attribute Interdependence Maximization (CAIM) Discretization Algorithm, *Proceedings of the 2003 International Conference on Machine Learning and Applications* (ICMLA'03), Los Angeles, pp.30-36, 2003

[21] Kurgan, L., *Meta Mining System for Supervised Learning*, Ph.D dissertation, the University of Colorado at Boulder, Department of Computer Science, 2003

[22] Kurgan, L., Cios, K.J., Sontag, M., & Accurso, F.J., Mining the Cystic Fibrosis Data, In: Zurada, J., & Kantardzic, M. (Eds.), *Novel Applications in Data Mining*, IEEE Press, accepted, 2003

[23] Kurgan, L., & Cios, K.J., CAIM Discretization Algorithm, *IEEE Transactions of Knowledge and Data Engineering*, 16:2, pp.145-153, 2004

[24] Lim, T.-S., Loh, W.-Y. & Y.-S. Shih, A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-three Old and New Classification Algorithms, *Machine Learning*, 40, pp.203-228, 2000

[25] Mitchell T.M., *Machine Learning*, McGraw-Hill, 1997

[26] Parthasarathy, S., Zaki, M., Ogihara, M., & Li, W., Parallel Data Mining for Association Rules on Shared-Memory Systems, *Knowledge and Information Systems*, 3:1, pp.1-29, 2001

[27] Prodromidis, A., Chan, P., & Stolfo, S., Meta-Learning in Distributed Data Mining Systems: Issues and Approaches, In: Kargupta H. & Chan P. (Eds.), *Advances in Distributed and Parallel Knowledge Discovery*, AAAI/MITPress, 2000

[28] Provost, F., & Fawcett, T., Analysis and Visualization of Classifier Performance: Comparison Under Imprecise Class and Cost Distribution, *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pp.43-48, 1997

[29] Rainsford, C.P., & Roddick, J.F., Adding Temporal Semantics to Association Rules, *Proceeding of the 3rd European Conf. on Principles and Practice of Knowledge Discovery in Databases* (PKDD'99), pp.504-509, 1999

[30] Roddick, J.F., & Spiliopoulou, M., A Survey of Temporal Knowledge Discovery Paradigms and Methods, *IEEE Transactions on Knowledge and Data Engineering*, 14:4, pp.750-767, 2002

[31] Sebag, M., Delaying the Choice of Bias: A Disjunctive Version Space Approach, *Proceedings of the 13th International Conference on Machine Learning*, pp.444-452, 1996

[32] Shafer, J., Agrawal, R., and Mehta, M., SPRINT: A Scalable Parallel Classifier for Data Mining, *Proceedings of the 22nd International Conference on Very Large Data Bases*, San Francisco, pp. 544-555, 1996

[33] Spiliopoulou, M., & Roddick, J. F., Higher Order Mining: Modelling and Mining the Results of Knowledge Discovery, *Data Mining I: Proceedings of the Second International Conference on Data Mining Methods and Databases*, pp.309-320, 2000

[34] Srivastava, A., Han, E., Kumar, V., & Singh, V., Parallel Formulations of Decision-Tree Classification Algorithms, *Data Mining and Knowledge Discovery*, 3:3, pp.237-261, 1999

[35] Vilalta, R., & Drissi, Y., A Perspective View and Survey of Meta-Learning, *Artificial Intelligence Review*, 18:2, pp.77-95, 2002

[36] Vlachos, P., *StatLib Project Repository*, http://lib.stat.cmu.edu, 2000

[37] Zaki, M., Parallel and Distributed Association Mining: A Survey, *IEEE Concurrency*, 7:4, pp.14-25, 1999

[38] Zaki, M., Parallel Sequence Mining on Shared-Memory Machines, In Kumar, Ranka, & Singh, (Eds), *Journal of Parallel and Distributed Computing*, special issue on *High Performance Data Mining*, 61, pp.401-426, 2001

[39] Zaki, M., Ho, C., & Agrawal, R., Parallel Classification for Data Mining on Shared Memory Multiprocessors, *IEEE International Conference on Data Engineering*, pp.198–205, 1999