Prototype–based Mining of Numeric Data Streams [*]

Francisco Ferrer–Troyano
Dept. of Computer Science
University of Seville
Av. Reina Mercedes S/N
41012, Seville, Spain

ferrer@lsi.us.es

Jesús S. Aguilar–Ruiz
Dept. of Computer Science
University of Seville
Av. Reina Mercedes S/N
41012, Seville, Spain

aguilar@lsi.us.es

José C. Riquelme
Dept. of Computer Science
University of Seville
Av. Reina Mercedes S/N
41012, Seville, Spain

riquelme@lsi.us.es

ABSTRACT

Great organizations collect open-ended and time-changing data received at a high speed. The possibility of extracting useful knowledge from these potentially infinite databases is a new challenge in Data Mining. In this paper we propose an anytime incremental learning algorithm for mining numeric data streams. Within Supervised Learning, our approach is based on prototypes and hypercubic decision rules, concerning with the simplicity of the model provided and the time complexity as primary goals. Experimental results with synthetic databases of 100 gigabytes show a good performance from streams of data in continuous transformation.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*data mining*; I.2.6 [**Artificial Intelligence**]: Learning—*concept learning*; I.5.2 [**Pattern Recognition**]: Design Methodology—*classifier design and evaluation*

General Terms

Incremental induction, on-line learning, data streams

1. INTRODUCTION

Astronomy, meteorology, satellite environmental assessment, telecommunications, ATM transactions, retail chains, scientific projects, etc. All of them are some examples of different fields for which gigabytes of data are daily generated and stored and where each new record arrives at a rapid rate. Due to these records are usually on permanent traffic, they are oversensitive to noise, missing, and inconsistent values. This situation has brought a defining challenge for KDD research community: designing near-linear[1] time complexity learning-algorithms to mine open-ended, high-speed and time-changing data streams so that requirements in time and memory compel to give an approximate answer which is not adversely affect by the ordering of the arriving records [4, 8].

When distribution of input examples is dynamic in time, algorithms based on data partitioning techniques [11] (instance sampling and feature sampling) are oversensitive to underfitting (important patterns either are passed over or are not weighed up so important) and overfitting (important patterns in some subsets may not be it in the global set). Apart several incremental learners build a decision tree based predictive model concerning with increasing accuracy and not allowing for its understanding on the part of the business analyst. Mining high-speed and time-changing data streams with this approach, updating frequently the model when subtrees become *obsolete* may made it difficult to keep a near-linear time complexity.

In this work we describe and evaluate SCARP (Scalable Classifying Algorithm based on Relevant Prototypes), an algorithm to mine numeric data streams based on hypercubes with an associated representative synthetic vector. These hypercubes are decision rules which SCARP removes when they have become *obsolete* while new examples are still to be read. This reduction of the model does not affect adversely the computational cost but quite the opposite speeds up its subsequent updating. With the usefulness of the knowledge provided (few rules of small disjuncts) as primary goal, we introduce an algorithm that updates the model with each novel case. To do this, SCARP does not process the whole search space but those regions with highest influence, i.e., the most representative patterns. Since time is a principal requirement, we must to limit the maximum number of rules of the model giving an approximate answer. This approach is different from several scalable algorithms, which need to read a block of δ new registers for going forward where δ will change according to the input data. The main shortcomings of SCARP are two: it is a distance-based algorithm and it can not process nominal values. Our approach bases the built model on the Euclidean distance among the new record and the rules obtained. Since attributes with large ranges outweigh attributes with smaller ranges, normalization is necessary to make equal the influence of all attributes. Therefore, SCARP needs to know the ranges of every feature before starting, so that each example is firstly normalized and then taken to update the model. In the next sections we explain the algorithm and its performance on six synthetic databases with different degree of complexity.

---

[1]At the most in the order of $O(e^{3/2})$ where e is the number of examples processed [7].

Procedure buildModel(Stream sequence, int α
double β, int γ, int δ, int θ, double κ, double μ)
    <i, ruleSets>:=<0, EMPTY>
    while(sequence.hasMoreElements())
      <e, i>:=<sequence.nextElement(), i+1>
      updateModel(normalize(e), α, β, ruleSets)
    if(refineFrequency(i, δ))
      refineModel(γ, i-θ, κ, ruleSets)
    assembleModel(μ, ruleSets)
  return(ruleSets)

Figure 1: The SCARP algorithm.

2. THE SCARP ALGORITHM

2.1 Building the model

Figure 1 shows in pseudo-code the SCARP's main procedure which takes six user parameters $(\alpha, \beta, \gamma, \delta, \theta, \kappa)$. The algorithm builds an output model made up of several sets of rules, one set for each label. The maximum size (number of rules) for every set is given by α. Each rule R, for its part, comprises six elements: $R=\{I,C,N,F,s,e,u\}$. $I=\{I_1,\ldots,I_m\}$ is a set of m closed intervals, one for each relevant attribute. The lower and upper bounds (I_{il}, I_{iu}) of each interval I_i are the vertexes of an axis-parallel hypercube that define the influence region of R. C is a prototype in R^m called centroid and generated as weight average of the same label examples covered by R. N and F are the vectors of two of those read examples covered by R until the current time: the nearest and the most distant example to C, respectively. s is the number of same label examples covered by R (the support) and is used as weight to classify a new query. u stores the number of the last example that updated to R. e is the number of different label examples covered by R whose maximum value is given by β (the confidence) so that: $\frac{s}{s+e} \geq \beta$. The rules with a support smaller than γ per cent of the number of read examples are removed from each set. This pruning is run every δ read examples by the procedure *refineModel* which also removes those rules that had not undergone before the last θ read examples. Only rules belonging to the same set can make non-empty intersections among them until the last example arrives. When the last example is read, the procedure *assembleModel* simplifies the model allowing non-empty intersections among different label rules whose maximum volume is given by μ.

Every time a new example $x=(q,l)$ is read and q is normalized (q is a vector in R^m and l is a nominal value), the procedure *updateModel* looks for rules that cover it. If there is any rule that covers x, the updating varies according to the label l. If x is covered by rules for the same label l, each one is undergone with two simple operations: the increase in one unit of the associated support and the calculation of the new centroid.

When q is covered by rules generated for a different label $l'\neq l$, the rule R_i with the nearest centroid to q is founded. If the confidence of R_i is still greater or equal than the minimum given by the user $(e_i+1\leq s_i(1/\beta\text{-}1))$ then e_i is increased in one unit; else R_i is split into two new rules R_{i1} and R_{i2} in which C_{i1} and C_{i2} are N_i and F_i, respectively. Both rules have volume 0 so that I_{ij}, N_{ij} and F_{ij} are null ($j\in\{1,2\}$). The support s_{ij} associated to each new rule is inversely proportional to the Euclidean distance from N_i and F_i to C_i, respectively. It is possible that both new rules can be joined without causing a non-empty intersection with another label rule. In this case R_i is reduced according to N_i and F_i. It is possible too that either one or both new rules R_{ij} are covered by previous rules generated for the same label l'. In this second case each new covered rule R_{ij} is not added in the model and only the covering rule R_k whose centroid is the nearest to such covered rule is updated as the preceding case (its new centroid C_k is calculated from C_{ij} and its support s_k is increased with the value of s_{ij}). Finally, if the number of rules generated for the label l is smaller than α, a new *point-rule* R_{i3} for the new *splitter* example is generated (with $s_{i3} = 1$).

Figure 2 shows the situation described above. Let's take, for example, R_1 and R_2 as two rules for the label A with $\frac{s}{s+e} = \beta$ and which have covered 1000 and 2000 examples until the current time, respectively. A new example $x = (q,B)$ arrives and it is located inside the region shared by both rules. As the rule with the nearest centroid to q is R_2, it may be split. Let N_2 and F_2 be distant 2 and 9 units from C_2, respectively. Then three new *point-rules* are initially generated: one rule R_{21} for N_2, another rule R_{22} for F_2, and a third rule R_{23} for q. The support s_{23} is equal to 1 and the supports s_{21} and s_{22} are $\lceil\frac{2000/2}{1/2+1/9}\rceil$ and $\lceil\frac{2000/9}{1/2+1/9}\rceil$, respectively. But R_{21} is covered by R_1, so this latter rule is updated with R_{21} which is not added in the model.

Although the new example may be covered by several rules generated for a different label, only the rule with the nearest centroid is split. We have decided on this criterion under the next hypothesis: *"if the new example belongs to a pattern, then a near example with the same label will be read and wrong rules will be split"*. When noise is read, dividing each rule that covers it may involve an unnecessary computational effort. In addition, errors due to overfitting may be made. Our primary goal is to extract a reduced set of decision rules with a good accuracy rate. In this sense, if the support of a rule is proportionally small in regards to the total number of examples already read, SCARP will remove it because it does not contribute as reliable or significant pattern. Subsequently the rules affected by that noisy example can be generalized in one rule. With a similar criterion, we could update only the nearest rule that covers a new example instead of each rule for the same label. Nevertheless we decided on the second option due to two reasons. The first is that the computational cost is not harmed. The second is that the centroid associated to each rule will tend to the centroid of an equal solid hypercube with the same space location and with uniform mass density.

When there is not any rule that covers the new example read, the nearest expandable rule for the same label to such example is looked for. An hypercube can extend to cover an example with equal label if it does not make a non-empty intersection with other rule generated for a different label. If this rule is found, it is updated moving its centroid, extending its edges, and increasing one unit its support. But, what is the used distance between an example and a rule?. We have applied as such distance the increase of volume of a rule R when such rule extends (R') to cover a new example $x=(q,l)$ so that the rule finally expanded will be that whose *growth* is the minor according to the next formula:

$$\mathsf{REDist}(\mathsf{R},\mathsf{q}) = \mathsf{Vol}(\mathsf{R}') - \mathsf{Vol}(\mathsf{R}); \quad \mathsf{R}' = \mathsf{R} \cup \mathsf{q};$$

$$\mathsf{Vol}(\mathsf{R}) = \prod_{i=1}^{m}(10^{\phi}(\mathsf{I}_{iu} - \mathsf{I}_{il})) \mid \mathsf{I}_{iu} > \mathsf{I}_{il}, \quad \phi \in \mathcal{N};$$
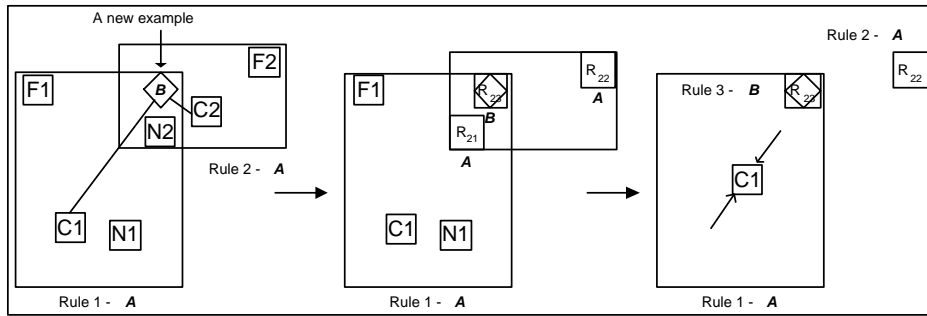
Figure 2: A new unseen example splits a rule generated for a different label.

Let's take, for example, $\phi = 1$, $x=(\{4,5,6\},A)$ as an example in R^3, and R_j as a rule in R^3 for the label A with intervals $I_j =\{[2,4],[1,3],[6,6]\}$ (volume 4). Then R'_j, would have the intervals $I'_j =\{[2,4],[1,5],[6,6]\}$ (volume 8). Therefore the *growth* of R_j to cover the example x would be of 4 units. As the volume of a rule consider only those dimensions where the lower bound is strictly smallest than the upper bound, the term 10^ϕ is used to do that rules without expansion in a certain dimension k ($I_{kl} = I_{ku}$) have smaller volume than rules with expansion in such dimension k and the same intervals in the rest of dimensions $m \neq k$. Since SCARP normalizes each example before process it, the range for an interval is at the most 1. Without the term 10^ϕ, a rule without expansion in a dimension k would have higher volume than a rule with expansion in k and the same intervals in the rest. We have used $\phi = 3$ in our experiments.

In a first version of SCARP we used the Euclidean distance between an example and the centroid of a rule to decide which the most suitable rule for a new example was. With this approach we benefit apparently to keep in the model the smaller number of rules as far as possible because these rules try to cover greater regions. And this would be a good decision knowing that noise is not present in data. However, this situation seems unlikely for data streams, where permanent traffic, the high numerosity and dimensionality of data, and many times a high cardinality in the values make errors appear easy. With the introduced distance, the model will have a greater number of rules and these rules will have smaller volume. Therefore, if our initial aim was to generate a set of rules as reduced as possible, are not we going in the opposite way?. Since data streams present a high sensitivity to noise and we don't know when noisy values will be read, we must try to avoid splitting valid rules. Generating reduced hypercubes, the likelihood that noisy examples are located inside rules for a different label will be smaller than if we try to generalize rules as large as possible. So, after reading the last example, the number of rules will be rather greater than the optimal. To reduce the model size and to make easy the classification task, the procedure *assembleModel* is called as final stage.

When it is not possible to generalize any rule, a new *point–rule* for the given example is generated provided the number of rules in the model for the label of the example is smaller than α. It happens when all expansions cause a non-empty intersection between two rules for different labels. When all the examples are read, through the procedure *assembleModel* intersections between rules for different labels are allowed. But only when such intersections cover an insignificant percentage of examples with respect to the support of each involved rule.

2.2 Pruning rules

Every δ examples the pruning method *refineModel* is called to remove irrelevant attributes and invalid rules that stem for noise. When original attributes are dropped, the rules gain simplicity and thereby the algorithm gains speed for the subsequent updates. First *refineModel* removes the rules don't updated with the last θ read examples and those one whose support is smaller than γ per cent of the total number of read examples at that time. If noise is present in data, those *noisy* rules must have a low support and a low updating rate. Later SCARP tries to reject irrelevant attributes as long as there is at least one rule per label. If all the rules at least range μ per cent of an attribute's domain, it is signed as irrelevant and is dropped. In this manner, if it has not been read any example for a certain label yet, all the attributes are kept. Herein it is important to point out two questions:

- Can a dropped attribute be relevant again?.

- If a particular region's examples arrive with a frequency lower than the model refining frequency, will the algorithm be able to model this region?, i.e., will the algorithm pass out important patterns?.

With regard to the first question, for a future work we are evaluating an extended version of SCARP that takes into account for each dropped attribute the interval of new values read after its dropping. If one of these intervals has size smaller than $\kappa\%$ then the associated attribute can be relevant again. In relation to the second question, reducing the update frequency can solve the problem.

When the the last example is read, the procedure *assembleModel* is called to try joining rules generated for the same label. When it is not possible any union the procedure ends. In every iteration the two nearest rules for each label whose union is possible are looked for. The two nearest rules are those whose union has the smallest volume in relation to the volume of the rest of possible unions. The union between two rules R_i and R_j is possible if either the intersection with any different label rule R_k does not cover a number of examples greater than $\mu\%$ of the support of R_k or the space shared by R_i and R_j is greater than $(100-\mu)\%$ of the space of each rule.

```
Procedure refineModel (int γ, int ϑ, double κ,
ArraySet ruleSets)
    for each label L in labelSet
        for each rule R in ruleSets[L]
            if(R.s< γ OR R.u< ϑ)
                ruleSets[L].remove(R)
        removeAttributes(κ,ruleSets)
    end procedure
```

Figure 3: Pseudo-code of the procedure to refine the model.

2.3 Classifying new queries

To classify a new query q SCARP looks for the hypercubes that cover it. If only one hypercube covers q then its label associated is assigned to q. When several rules cover the new query, that rule with higher influence on the region where q is located decides the label to be assigned. The influence of a rule R_i on a new query is given by the formula:

$$IF(R_i, q) = \frac{10}{\sqrt{m}} \cdot EucliDist(C_i, q)^{-1} + \frac{S_i}{T}; \quad T = \sum_{j=1}^{n} S_j;$$

With the above heuristic measure we try to take into account, in the next priority order, the distance to the centroid C_i, and the percentage of examples covered by the rule R_i with respect to the total number T of examples covered by the rules finally retained. If some rules were pruned in the previous phase then T may be different to the total number of read examples. The term $\frac{10}{\sqrt{m}}$ normalizes the distance in the interval $[0,10]$ so that m is the number of relevant attributes (the dimensions used in the model).

When no rule cover the new query, in a similar form to the anterior situation, the aim is to take into account, in the next priority order, the volume increase after cover the new query, the distance to the centroid, and the support of each rule. Only the rules that does not make a non-empty intersection with another label rules are used in order to label the new query. Further, we think that when the number of examples to be read tends to infinity then the centroid associated to each rule tends to be that of an equal hypercube with uniform mass. Therefore, the rule with the nearest centroid will be that with the smallest *growth*. In this sense to use either the distance to the centroid or the increase of volume must result equal. Thus, the query is labelled with the label associated to that rule whose *InfF* is the maximum.

3. EMPIRICAL EVALUATION

We ran all our experiments on a AMD x86/1.4Ghz and 256Mb DDR RAM PC running Windows XP. SCARP was tested with five synthetic databases whose static distribution was determined a priori (Figure 4) and one database whose distribution was changing in time every 10 millions of examples according to the five previous domains. In every database the attribute values were generated with a simple uniform number generator as a stream of pseudo–random numbers in the real interval $[0,1]$. Using a 48-bit (sixteen-figure decimal number) seed with the actual system time as initial value, each new value was obtained according to the linear congruential formula defined by Lehmer in 1951:

Table 1: Evaluation of SCARP with synthetic databases (500 millions of examples and 25 real attributes).

SDB	T	NR	NS	NA	NC	PA	CT
S1	462	9	240	16	2.2	99.9	39
S2	660	14	217	16	3.1	87.3	35.5
S3	650	33	2343	18	3.3	91.5	83.5
S4	637	30	408	11	2.7	84.9	70
S5	988	17	602	13	2.4	77.7	32.5
S6	923	12	1722	20	3.8	64.1	31

$X_{i+1} = A \cdot X_i (mod\ M)$. For every distribution only the two first attributes discriminate the label of an example. When the example was located in plane, the label was assigned as a function of its position. We carried out all tests without ever writing the training examples to disk (i.e., generating them on the fly and passing them directly to SCARP). We defined the regions for each label with similar area due to it is mathematically valid to assume that the example labels will be uniformly distributed when, in the course of time, millions of examples have been processed. For the six databases we evaluated three issues: the model size, the computational cost, and the prediction accuracy. These aspects were measured for 500 millions of examples with 25 attributes (using 8 bytes to store a double value it involves a database of 100 gigabytes), and 5% of class noise in data (i.e., 25 millions of examples with a *wrong* label). For each database, 50 millions of examples were used for testing.

The used values for the parameters of the algorithm were: $\alpha = 50$ (a maximum of 50 rules for each label), $\beta = 0.9$ (a maximum of 10% of examples with different label covered by each rule until the *assembleModel* procedure is called), $\gamma = 0.1$ (for each rule, every δ read examples it was called for a minimum support of 1% regarding to the total number of read examples at that time), $\delta = 10^6$, $\theta = 10^5$ (every 10^6 examples, SCARP removed those rules that were not updated with some of the last 10^5 examples), $\kappa = 0.96$, and $\mu = 0.05$. SCARP was implemented in JAVA so CPU time taken to build the model is not a very precise measure.

Table 1 shows the results obtained. Column T shows the CPU time in minutes. The final number of rules retained and the total times the rules were split are showed in Columns NR and NS, respectively. Column NA shows the number of attributes retained and Column NC gives the average number of conjunctions for all the rules. The prediction accuracy is showed in Column PA and the time taken to classify the 50 millions of test examples is given in minutes in Column CT. It's significant the performance obtained by SCARP with the databases S1 and S3 (with parallel and non-parallel axes regions for each label) where the average complexity of a rule and the average size of the model are 2.5 and 17, respectively.

4. RELATED WORK

Approaches based on subsampling methods for mining large databases are proposed by Catlett in [3]. Gehrke et al. obtain in [5] an approximate tree through a subsample of fixed size. In contrast, SLIQ [10] and SPRINT [13] do not learn with data load in memory but they are disk-based learners that use all the examples and focus on optimizing
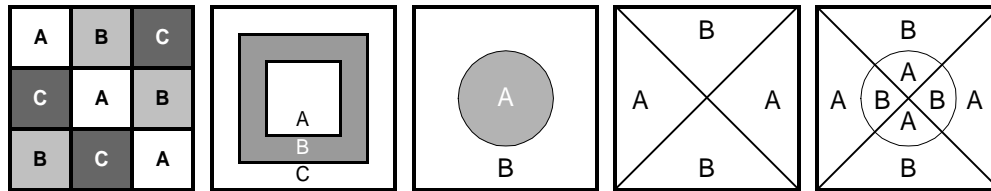
Figure 4: Synthetic databases.

sequential access to disk. Model pruning and updating algorithms are in [2, 12]. There is also a large literature on incremental clustering algorithms [6, 14], association rules [1], and classification techniques built through the union of several disciplines [9] with the accuracy as the main goal. However, not many of them are in particular addressed to usefulness and reduced complexity of the model generated.

5. CONCLUSIONS AND FUTURE WORK

An incremental learner based on hypercubes and prototypes has been introduced in this paper. With the simplicity of the model and its usefulness for the business analyst as primary goal, we have developed a classifier for mining numeric data streams with each novel case that is different from the most of approaches in literature. With an intermediate pruning method as part of the algorithm, SCARP is able to remove *obsolete* rules and wrong rules caused by noise without additional computational effort. To achieve it, SCARP does not mine all the search space but only the most representative patterns.

For a future work, we are studying several heuristics to recover dropped attributes turned relevant again and to improve classification accuracy and the capacity for mining nominal attributes in order to compare SCARP with another scalable classifiers as CVFDT [8] and SPRINT [13].

6. ADDITIONAL AUTHORS

Daniel Mateos García, Department of Computer Science, University of Seville, mateos@lsi.us.es

7. REFERENCES

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.

[2] Necip Fazil Ayan, Abdullah Uz Tansel, and M. Erol Arkun. An efficient algorithm to update large itemsets with early pruning. In *Knowledge Discovery and Data Mining*, pages 287–291, 1999.

[3] J. Cattlet. *Megainduction: machine learning on very large databases*. PhD thesis, Basser Department of Computer Science, University of Sydney, Australia, 1991.

[4] P. Domingos and G. Hulten. Mining high-speed data streams. In *Knowledge Discovery and Data Mining*, pages 71–80, 2000.

[5] J. Gehrke, V. Ganti, R. Ramakrishnan, and W.Y. Loh. BOAT – optimistic decision tree construction. In

1999 ACM SIGMOD Conference, pages 169–180, Philadelphia, Pennsylvania, 1999.

[6] Sudipto Guha, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams. In *IEEE Symposium on Foundations of Computer Science*, pages 359–366, 2000.

[7] P. Huber. From large to huge: A statistician's reaction to kdd and dm. In *Third International Conference on Knowledge Discovery and Data Mining*, pages 304–308, Menlo Park, CA, 1997. AAAI Press.

[8] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106, San Francisco, CA, 2001. ACM Press.

[9] Joshi, Karypis, and Kumar. ScalparC: A new scalable and efficient parallel classification algorithm for mining large datasets. In *IPPS: 11th International Parallel Processing Symposium*. IEEE Computer Society Press, 1998.

[10] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining. In *Extending Database Technology*, pages 18–32, 1996.

[11] F. Provost and V. Kolluri. A survey of methods for scaling up inductive algorithms. *Data Mining and Knowledge Discovery*, 3(2):131–169, 1999.

[12] Rajeev Rastogi and Kyuseok Shim. PUBLIC: A decision tree classifier that integrates building and pruning. *Data Mining and Knowledge Discovery*, 4(4):315–344, 2000.

[13] J.C. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proc. 22nd Int. Conf. Very Large Databases, VLDB*, pages 544–555, 1996.

[14] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: an efficient data clustering method for very large databases. In *ACM SIGMOD International Conference on Management of Data*, pages 103–114, Montreal, Canada, June 1996.