Mining Low Dimensionality Data Streams of Continuous Attributes

Francisco J. Ferrer-Troyano, Jesús S. Aguilar-Ruiz, and José C. Riquelme

Department of Computer Science, University of Seville
Avenida Reina Mercedes s/n, 41012 Seville, Spain
{ferrer, aguilar, riquelme}@lsi.us.es

Abstract. This paper presents an incremental and scalable learning algorithm in order to mine numeric, low dimensionality, high–cardinality, time–changing data streams. Within the Supervised Learning field, our approach, named SCALLOP, provides a set of decision rules whose size is very near to the number of concepts to be extracted. Experimental results with synthetic databases of different complexity degrees show a good performance from streams of data received at a rapid rate, whose label distribution may not be stationary in time.

Keywords: Classification, decision rules, incremental learning, scalable learning algorithms, data streams.

1 Introduction

In recent years, designing scaling–up and scalable algorithms has consolidated as an important challenge within data mining research community. Memory and time limitations compel make such system to give an approximate answer from few scans (ideally only one) assuring that both result and performance are not adversely affected by the order of the examples. In addition, when the distribution is not stationary (records are collected over months), algorithms based on data partitioning techniques (instance/feature sampling) are oversensitive to both underfitting and overfitting. Many scalable learning algorithms are based on decision trees, modelling the whole search space hierarchically as disjointed hypercubes. The large and complex trees given by these systems cast doubts on its capabilities as suitable knowledge representation due to the user need to explore paths of several dozen of levels to know interesting patterns. In addition, mining time–changing data streams may involve to check an *out–of–date* sub–tree, increasing the computational cost. A common approach in these systems consists in repeatedly applying the learner to a sliding window of w examples. The main goal in these systems is then to find the value for w that optimizes the performance as a function of the input data. So interactive and user–controlled data mining systems are becoming increasingly developed. Such approaches trade accuracy for simplicity providing more meaningful and understandable models.

This paper introduces a scalable classification algorithm named SCALLOP (SCALabLe classification algorithm tO learning Patterns) that provides a model

Fernando Moura Pires, Salvador Abreu (Eds.): EPIA 2003, LNAI 2902, pp. 264–278, 2003.

on demand from several user–defined parameters. Using a window of size 1, our approach obtains hypercubic decision rules sorted in a relevance order according to the regions whose characteristics interest the user. In the next sections we describe our approach and discuss its major drawbacks. Next we present experimental results on synthetic data of different degrees of complexity which show its usefulness and effectiveness to mine numerical, low–dimensionality, high–speed, time–changing data steams.

2 Related Work

There has been many recent works on mining very large databases and data streams. Domingos and Hulten introduce VFDT [7] and CVFDT [14] which build a decision tree using constant time and memory per example. Their approach is based on Hoeffding bounds, which guarantee that the output is asymptotically nearly identical to that given by a batch conventional learner from enough examples. They also apply Hoeffding's inequalities to build a scaling–up method that is applicable to any induction algorithm based on discrete search [15]. Decision tree based classifiers are also Gehrke et al.'s BOAT [10] and Agrawal et al.'s SPRINT [20]. BOAT obtains an approximate tree through a subsample of fixed size. Previous approaches based on subsampling methods are also proposed by Catlett [4]. In contrast, SPRINT [20] is disk–based learner that use all the examples and focus on optimizing sequential access to disk.

Garofalakis et al. [17,9] propose several algorithms for building decision trees under size and accuracy constraints in order to give an meaningful model for the user. Dobra and Gehrke's SECRET [6] is a regression tree based algorithm providing a more easy to interpret output than that given by decision trees. SECRET uses EM clustering algorithm for Gaussian mixtures to generate leaves as linear transformation from found close clusters.

There is also large literature on scaling–up algorithms [19,11,3], model maintenance [8,16,5,2], and scalable clustering algorithms [18,12,13,21].

3 The SCALLOP Algorithm

Classification is generally defined as follows. An input finite data set of training examples is given. Every training example is a pair $e = (x, y)$ where x is a vector of m attribute values (each of which may be numeric or symbolic) and y is a class discrete value named label. The goal is to obtain a model $y = f(x)$ to classify or decide the label for new non–labelled test examples named queries. Henceforth, the next notation is used to describe our approach. Let m be the number of continuous attributes. Let $Y = \{y_1, \ldots, y_z\}$ be the set of class labels. Let $e_i = (x_i, y_i)$ be the i^{th} training example to be read, where x_i is a normalized vector in $\mathcal{R}^m$ and y_i is a discrete value in Y. SCALLOP builds a model formed by z sets of decision rules, one set per label.

In order to achieve a balanced performance between the running time and the classification accuracy, each rule R is a data structure that comprises twelve elements:

- Definition limits I: set of m closed intervals $[I_{jl}, I_{ju}]$, one per numerical attribute ($j \in \{1, \ldots, m\}$). l denotes lower bound and u upper bound.
- Centroid C: vector in $\mathcal{R}^m$ generated as weighted mean from the examples covered by R.
- Central vector N: vector of the nearest example to C from all those examples covered by R.
- Delimiters D: set of the most distant β vector to each other from all examples covered by R. β is an user–defined parameter.
- Markers M: set of the nearest β examples to N that have been covered by R. M, D, and C are used to modify a wrong rule.
- Overlapping rules O: set of links that reference others rules with which R shares a sub–region belonging to the region defined by I. Only rules belonging to the same set can make non-empty intersections among themselves.
- Growth limits B: set of m open intervals (B_{jl}, B_{ju}) (one per numerical attribute) so that: $B_{jl} < I_{jl} \leq I_{ju} < B_{ju}$; $\forall j \in [1, m]$. These limits play an important role since by knowing where a rule can extend to, the algorithm makes stable the model faster. Furthermore, this information is very helpful to classify new queries with few rules by voting.
- s is the support or number of examples covered by R which are associated with the same label.
- d is the number of different labelled examples covered by R. The maximum value of d is computed as a function of s and the user parameter γ, so that: $\frac{s-d}{s} \geq \gamma$.
- f is a boolean value that indicates if R was split ever.
- r is the index of the last example covered by R.
- u is a boolean value that indicates if R was extended with some of the last δ read records. δ is an user parameter.

In addition, the search space is modelled according to five user–defined parameters: the maximum number of rules per label α, the minimum support λ, the minimum confidence γ, the minimum update frequency μ, and the pruning frequency δ. Every δ read examples, if the number of rules for a label is greater than α then SCALLOP attempts to simplify the model by joining similar rules and by removing *uninteresting* rules: those ones with a support smaller than λ per cent of the number of read examples and those ones that were not updated with the latest μ read examples.

The algorithm starts with α rules per label generated from the first α read examples of each label. These rules are not hypercubes but points. We name these rules *point–rules*. When there have been read more than α examples for a certain label y_i, three situations are differentiated with every new example $e_i = (x_i, y_i)$:

- ***Correct covering***: x_i is covered by one or several rules associated with the same label y_i.

Algorithm 1 updateModel

```
Input:e: Example; i, α, β, γ: integer;
Output:covered: boolean
Input/Output:model: Array of (List of DecisionRule)
< x,y >=< normalize(e.vector()),e.label() >
if received[y] < α then
  model[y].add(new DecisionRule(x))
  {a new point–rule is generated, basic case}
else
  covered ⇐ checkCover(γ,i,y,x,model)
  if NOT covered then
    aRule⇐ findExpandibleRule(β,x,y,model)
    if aRule ≠ null then {case 2}
      extendRule(aRule,x,model[y])
    else
      if model[y].size() < α then {basic case}
        model[y].add(new DecisionRule(x))
      end if
    end if
  end if
end if
received[y]++
```

Fig. 1. Three cases are differentiated by updating the model with a new example. The label distribution is stored for using it in the classification phase, if necessary.

- ***Possible expansion***: x_i is not covered by any rule in the model but there is at least one rule associate with the same label that can be extended to cover it without overlapping with a different labelled rule.
- ***Possible split***: x_i is covered by one or several rules associated with a different label $y' \neq y_i$.

Cases 1 and 3 take turns to be firstly checked. If none of them come true, then the actions associated with the case 2 are run. After each pruning (every δ read examples), SCALLOP counts how many times the cases 1 and 3 come true. For the next δ examples SCALLOP will check firstly that case with the highest count according to the preceding δ read examples.

Correct covering: every rule that covers x_i increases its positive support by one unit, updates the index of the last covered example and moves its centroid (procedure *checkCover* in Figure 1). When the first rule R_c that covers the new example is found, the other rules that may also cover it are found thanks to the links of R_c to them ($R_c.O$). All the rules that also cover x_i are updated with the same three operations.

One rule out of R_c and those linked by $R_c.O$, the rule R_n whose centroid $R_n.C$ is the nearest to x_i, may be updated with respect to $R_n.M$ and $R_n.N$.

If x_i is nearer to $R_n.C$ than $R_n.N$, then x_i replaces $R_n.N$, and $R_n.N$ is checked to be in $R_n.M$. If the size of $R_n.M$ is smaller than β, then x_i (or the

previous $R_n.N$ if the replacement was done) is directly added to $R_n.M$. If the size of $R_n.M$ is equal to β but x_i is nearer to $R_n.N$ than the *marker* z most distant from $R_n.N$, then z is replaced by x_i.

Possible expansion. When no rule covers x_i, SCALLOP looks for rules R_e associated with the same label y_i which can extend to cover x_i. A rule R_e can be expanded to seize the point x_i if fulfills two conditions:

1. x_i is not beyond the growth–bounds $R_e.B$, that is:

$$\forall j \in \{[1,\ldots,m\} \cdot \ x_{ij} \in (R_e.B_{jl}, R_e.B_{ju})$$

2. The resulting extended rule does not intersect with any rule associated with a label different to y_i.

Only one rule, out of all the candidate rules that can extend to cover the new example, is able to grow: that one whose *growth* or volume increase is the smallest when covering x_i (see Figure 2).

Definition 1 (Growth of a rule) *Let R be a rule in $\mathcal{R}^m$. Let x be a point in $\mathcal{R}^m$. It is defined the growth G of the rule R in order to cover the point x_i as:*

$$G(R, x_i) = \prod_{j=1,\ g_j>0}^{m} 10^{\phi} g_j \ - \prod_{j=1,\ r_j>0}^{m} 10^{\phi} r_j;$$

$$g_j = u_j - l_j;\ u_j = \max(x_{ij}, R.I_{ju});\ l_j = \min(x_{ij}, R.I_{jl});$$
$$r_j = R.I_{ju} - R.I_{jl};\ \phi \in \mathcal{N}$$

In order to measure only the new region that is taken when a rule R extends to cover a new example x_i, the *growth* takes into account only those dimensions for which there is an expansion.

Since SCALLOP normalizes each attribute value in [0,1] before processing an example, the term 10^{ϕ} is used to avoid that a rule R_a without expansion in a certain dimension k ($R_a.I_{kl} = R_a.I_{ku}$) has greater *growth* than a rule R_b that has expansion in such a dimension and the same intervals in the rest of dimensions ($R_a.I_j = R_b.I_j\ ; \forall\ j \neq k$). Consider two rules in $\mathcal{R}^2$, R_a and R_b, which can be extended to cover a new point $x = \{0.5, 0.5\}$, so that: $R_a.I = \{[0.1, 0.4]; [0.5, 0.5]\}$ (a segment) and $R_b.I = \{[0.1, 0.4]; [0.6, 0.7]\}$ (a surface). Without the term 10^{ϕ}, it results in: $G(R_a, x) = 0.4 - 0.3$; $G(R_b, x) = 0.4 \cdot 0.2 - 0.3 \cdot 0.1$. That is, contrary to expected, R_a grows more than R_b. We have used $\phi = 3$ in our experiments.

A rule can extend to cover a point if it does not overlap with any rule associated with a different label. If this rule R_e is found, it is updated with six operations:

- $R_e.r \leftarrow i$
- $R_e.u \leftarrow$ **true**
- $R_e.s \leftarrow R_e.s + 1$
- $R_e.I_{jb} \leftarrow Min(R_e.I_{jb}, x_{ij})$;

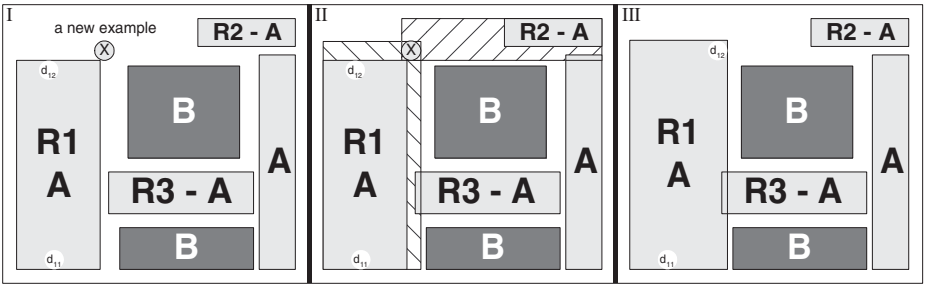

Fig. 2. Case ***Possible expansion***. (1) A new A–example x is not covered by any rule. (2) The rules *R1* and *R2* do not intersect with any rule associated with a different label when are extended to cover x. (3) Because of the *R2's growth* is greater than *R1's*, this latter rule extends to cover x. The set *R1.D* is updated with x, and the rules *R1* and *R3* are linked each other through *R1.O* and *R3.O*, respectively.

- $R_e.I_{ju} \leftarrow Max(R_e.I_{jb}, x_{ij})$;
- $R_e.C_j \leftarrow \frac{R_e.C_j \cdot R_e.s + x_{ij}}{R_e.s+1}$; $\forall j \in \{1, \ldots, m\}$

When a rule R_e is extended, it may overlap with other rules associated with the same label, so that SCALLOP updates the set of links in both directions. In addition, its set of *delimiters* $R_e.D = \{d_{e1}, \ldots, d_{en}\}$ may be updated. If the number of *delimiters* at the current time (n) is smaller than β, then x_i is added to $R_e.D$. When n is equal to β, a delimiter d_{ek} may be replaced by x_i. Let d_{ek} be the nearest delimiter to x_i. Let d_{eq} be the nearest delimiter to d_{ek}. If the Euclidean distance between x_i and d_{eq} is greater than the Euclidean distance between d_{ek} and d_{eq}, then d_{ek} is replaced by x_i (see Figure 2).

In a first prototype of SCALLOP the rule selected to cover a new example x_i was that one with the nearest centroid to x_i. With this approach we searched for a smaller number of rules by attempting to obtain regions greater than the regions covered by the current rules. But this approach was not a good choice when noise is present in data. *Clean* streams are unlikely since the nonstop traffic of high dimensionality and high cardinality in the attribute values gives rise to noise, missing, and inconsistencies frequently.

With the *growth* introduced, the model will have a greater number of rules, and these rules will have smaller volume. Nevertheless, by generating more reduced hypercubes, the likelihood that noisy examples are located inside will be smaller than by trying to generalize rules as large as possible. To compact scarcely crowded and dispersed rules, the procedure *refineModel* is called every δ read examples. This means an extra computational cost only at the beginning of the process, as when the model reaches stability, such a refining is almost unnecessary.

Algorithm 2 checkEnemyCover

```
Input:γ, y: integer; x: Vector
Output:covered: boolean
Input/Output:model: Array of (List of DecisionRule)
{Called by the procedure checkCover (in Figure 1)}
if exists a rule R of a label y' ≠ y that covers to x then
  covered ← true
  R' ⇐ nearestRule(R, R.O, x)
  confidence ← (R.s − R.d − 1)/R.s
  if confidence ≥ γ then
    R.d ← R.d + 1
  else
    reducedRules ⇐ splitRule(R, x)
    removeWrongRule(R, model[y'])
    addNewRules(reducedRules, model[y'])
    model[y].add(new DecisionRule(x))
  end if
else
  covered ⇐ false
end if
```

Fig. 3. A rule R is split when its confidence is smaller than γ. In this case, several derived rules based on $R.N$,$R.D$, and $R.M$ are generated.

Possible split: when x_i is covered by a rule generated for a different label y', the covering rule R' with the nearest centroid to x_i is founded as in the first case.

If the new confidence of R' $((R'.s - R'.d - 1)/R'.s)$ is still greater than or equal to the minimum γ given by the user, then the number of enemies $(R'.d)$ covered by R' is increased by one unit.

If the new confidence is smaller than γ, then a new *point–rule* R_p for x_i is added to the model. Next the support of R' resolves two new subcases. If $R'.s$ is smaller than $\lambda\%$ examples received, then R' is directly removed. On the contrary, if $R'.s$ is greater than or equal to $\lambda\%$ examples read, then R' is replaced by several derived rules that are built using an iterative procedure from $R'.N$, $R'.D$, and $R'.M$ (procedure *splitRule* in Figure 3):

1. $R'.N$, the delimiters of $R'.D$, and the markers of $R'.M$ are sorted in a list LS by the Euclidean distance to x_i in a decreasing order. Let sd and sm be the size of $R'.D$ and $R'.M$, respectively.
2. A new *point–rule* R'' is created for the first element in LS, the most distant vector to x_i with respect to the vectors of LS. Then $R''.f$ is initialized as *true* and such a generating vector is removed from LS. While it is possible, SCALLOP attempts to extend R'' with the first vector in the list. R'' can be extended when the resulting region does not cover x_i. If R'' is extended, the first element in LS is removed again. If the vector that creates or extends R'' is a delimiter, then $R''.s$ is initialized to or increased by one unit. If such

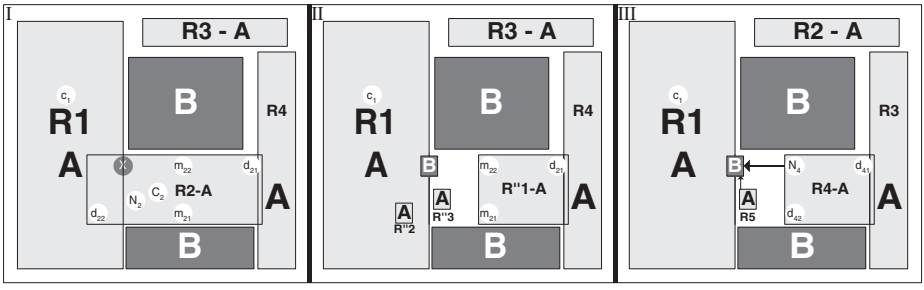

Fig. 4. Case ***Possible split***. (1) A new B–example x is covered by two rules (*R1* and *R2*) associated with the label A. (2) The rule whose centroid is the nearest to such an example (*R2*) results in three new reduced rules (*R"1*, *R"2*, and *R"3*) that are generated according to *R2.N*, *R2.D* and *R2.M*. (3) *R"1* overlaps with a previous rule so that both rules updates their set of links. Moreover, *R"2* is removed since is totally covered by *R1*. Because of *R2* was never split, *R"1 R"3* update their growth bounds by x. *R1* is not changed.

a vector is $R'.N$ or belongs to $R'.M$, then $R''.s$ is initialized to or increased by $(R'.s - sd)/(sm + 1)$.

3. When R'' can not be extended, $R''.B$ is updated with $R'.B$ (see Figure 6). If $R'.f$ is *true*, then $R''.B$ may be updated with x_i: when R'' covers exactly $m-1$ attribute values of x_i, then the value not covered means either an upper growth bound or a lower growth bound (procedure *updateGrowthBounds* in Figure 5). Finally, R'' is added to the list *reducedRules*. If LS is not empty, the procedure repeats the loop (2) again.

A new rule R'' generated by the procedure *splitRule* might be partially or totally covered by a previous rule $R_c \neq R'$, which must is linked through $R'.O$.

If a rule R'' is totally cover by R_c, then R'' is not included in the model (rule *R"2* in Figure 4–II). Moreover, R_c is not updated by R'' because the examples covered by R' and located inside R'' they updated $R_c.r$, $R_c.C$, and $R_c.s$ when they were read (case ***Correct covering***).

If R'' is partially covered by R_c (rule *R"1* in Figure 4–II, *R4* in Figure 4–III), then both rules update to each other the sets of links $R''.O$ and $R_c.O$, respectively.

If a new rule R'' is disjointed with all the rules associated with its same label (rule *R"3* in Figure 4–II, *R5* in Figure 4–III), then is directly added to the model.

Although the new example x_i may be covered by several rules associated with a different label (rules *R1* and *R2* in Figure 4), only the rule whose centroid is the nearest to x_i is split. We have decided on this criterion under the assumption that if the example x_i belongs to a pattern, then near examples associated with the same label y_i must be read shortly later, and wrong rules will be corrected.

Algorithm 3 updateGrowthBounds

Input: x: Vector; $R1$: DecisionRule
Input/Output: $R2$: DecisionRule
if $R1.f$ **then**
  OverlappedDims(x, $R2$,$numOD$,$attNO$)
  {$numOD$ is the number if dimensions in which x and $R2$ overlap}
  {$attOD$ is a dimension in which x and $R2$ do not overlap}
end if
for all $j = 1$ to m **do**
  if $numOD = m - 1$ **AND** $j = attOD$ **then**
    $R2.B_{jl} \Leftarrow \max(x_j, R1.B_{jl})$
    $R2.B_{ju} \Leftarrow \min(x_j, R1.B_{ju})$
  else
    $R2.B_{jl} \Leftarrow R1.B_{jl}$
    $R2.B_{ju} \Leftarrow R1.B_{ju}$
  end if
end for

Fig. 5. Updating the growth bounds of the rule $R2$ based on the growth bounds of the split rule $R1$ and the attribute values of the example x that split $R1$.

Every time a noisy example x is read, dividing only one rule instead of all the rules that cover x avoids an unnecessary computational cost. In addition, the subsequent updating of the model is hardly harmed as the number of new rules scarcely increases (only the rules from a split that have enough support are included in the model). If x_i is noisy or belongs to a minority pattern, then R_p will be removed in the next pruning. Furthermore, the number of overfitting–errors in the final classification phase will be smaller since the probability of covering a new query will be higher. Owing to data streams present a high sensitivity to noise, to make sure a rule is wrong before splitting involves a decisive issue to quickly attain the stability of the model.

With a similar criterion, only the nearest rule that covers a new example x_i could be updated instead of all the rules for the same label. Nevertheless we have decided on the second option because the centroid associated to each rule will tend to the centroid of an equal solid hypercube with the same spatial location and with uniform mass density. It may benefit the splitting of a rule and the computation of its delimiters and markers, in that it will tend to be more accurate. In addition, the computational cost is not adversely affected.

When no rule associated with the label y_i can cover x_i, a new *point–rule* is generated provided the number of rules associated with y_i is smaller than α (*basic case*). This happens when every expansion causes a non–empty intersection among rules associated with different labels.

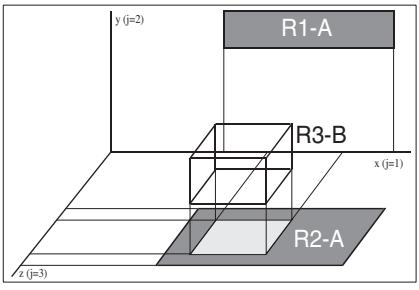

Fig. 6. Updating the growth bounds of a rule in $\mathcal{R}^3$. The rules $R2$ and $R3$ overlap in two dimensions (x, z), whereas $R1$ and $R3$ overlap only in one dimension. $R2.B_{3u}$ is updated with $R3.I_{3l}$ and $R3.B_{3l}$ is updated with $R2.I_{3u}$ $(= R2.I_{3l} = 0)$, respectively. $R1.B$ is not changed.

3.1 Refining the Model

The set of rules is refined every δ new examples in order to achieve a simpler and more accurate model. This periodical attempt of improving consists of three phases.

First, an iterative procedure is run to join rules associated with the same label. When no union is possible the procedure ends. In every iteration, the two nearest rules to each other whose union is possible are analyzed. The two nearest rules are those whose resulting volume is the smallest in relation to the volume of the rest of the possible unions. The union R_u from two rules R_a and R_b of the same label is done if two conditions are fulfilled:

1. R_u does not intersect with any rule associated with a different label.
2. The resulting hypercube is located inside the hypercube obtained from the growth bounds of R_a and R_b. That is, the growth bounds of R_a and R_b allow such a joint, so that:
   $rl_j, ru_j \in (bl_j, bu_j)$;
   $rl_j = \min(R_a.I_{jl}, R_b.I_{jl})$;
   $ru_j = \max(R_a.I_{ju}, R_b.I_{ju})$;
   $bl_j = \max(R_a.B_{jl}, R_b.B_{jl})$;
   $bu_j = \min(R_a.B_{ju}, R_b.B_{ju})$; $\forall j \in \{1, \ldots, m\}$

When two rules R_a and R_b are joined as R_u, R_a and R_b are removed and R_u is added to the model, so that:

- $R_u.u \leftarrow R_a.u$ **OR** $R_b.u$
- $R_u.f \leftarrow R_a.f$ **AND** $R_b.f$
- $R_u.r \leftarrow \max(R_a.r, R_b.r)$
- $R_u.s \leftarrow R_a.s + R_b.s$ - Intersection(R_a, R_b)
- $R_u.C_j \leftarrow \frac{R_a.C_j \cdot R_a.s + R_b.C_j \cdot R_b.s}{R_u.s}$; $\forall j \in \{1, \ldots, m\}$

- $R_u.I_{jl} \leftarrow min(R_a.I_{jl}, R_b.I_{jl})$;
- $R_u.I_{ju} \leftarrow max(R_a.I_{ju}, R_b.I_{ju})$;
- $R_u.B_{jl} \leftarrow max(R_a.B_{jl}, R_b.B_{jl})$;
- $R_u.B_{ju} \leftarrow min(R_a.B_{ju}, R_b.B_{ju})$;
- $R_u.D \leftarrow$ **selectFrom**$(R_a.D, R_b.D)$
- $R_u.M \leftarrow$ **selectFrom**$(R_a.N, R_a.M, R_b.N, R_b.N)$
- $R_u.N \leftarrow$ **selectFrom**$(R_a.N, R_a.M, R_b.N, R_b.N)$

In a second step every rule has to satisfy two conditions to stay in the model: 1) must cover at least one of the last δ read examples; 2) the positive support must be greater than or equal to the minimum given by the user (as percentage of the total number of examples read at that time). δ is another user parameter. If noise is present in data, those *wrong* rules that stem from noise are likely to have a low support and a variable update rate. If after this prune the number n of rules is still greater than α, then they are sorted by both the positive support and the index of the last covered example, in a decreasing order, so that the last $n - \alpha$ rules are directly removed. Every rules that stay in the model reset its number of enemies ($R.d \leftarrow 0$) and its expansion ($R.u \leftarrow$ **false**).

Before removing a rule R_r, some rules associated with a different label may update their growth bounds. If R_r was extended with one of the last δ read examples and was not split ever, then SCALLOP takes it as a valid minority rule (not noise). Therefore, the rules of different label that will remain in the model should not extend across the region given by the definition limits of R_r (Figure 6). To avoid a wrong expansion that may involve a splitting shortly after, SCALLOP updates the growth bounds of every different labelled rule R_s that overlaps with R_r in all dimensions except one j, so that:

$$\textbf{if } R_r.I_{jl} > R_s.I_{ju} \textbf{ then } R_s.B_{ju} \leftarrow \min(R_s.B_{ju}, R_r.I_{jl})$$
$$\textbf{if } R_r.I_{ju} < R_s.I_{jl} \textbf{ then } R_s.B_{jl} \leftarrow \max(R_s.B_{jl}, R_r.I_{ju})$$

3.2 Classifying New Queries by Voting

If a new query Q is covered by a rule R_q, then Q is directly classified as the label associated with R_q. If there is no rule covering the new query, SCALLOP tries to infer which labels are not possible for Q and it is classified by voting. Figure 7 shows this procedure. If the query is beyond the growth bounds of all the rules associated with a certain label l, then l is rejected to classify Q. If Q is beyond the growth bounds of a rule R_y with label y, then the votes against y are increased in one unit. If a rule R_t of label t can be extended to cover Q (it is inside the growth bounds of R_t and the resulting expansion does not intersect with any rule associated with a label different to y), then the votes for t are increased in one unit. Thus, the label assigned is that with the highest number of votes. When two labels have the same number of votes, the label distribution (*received*) decides which is the class value for the new query.

Algorithm 4 classifyQuery

```
Input:Q: Vector in R^m;
Output:label: Discrete;
if there is a rule R_q that covers Q then
  label ← the label associated with R_q
else
  for all label y_k ∈ Y do {Y = {y_1, ..., y_z}}
    validLabel[k] ← false
    for all rule R associated with the label y_k do
      if Q is beyond the growth limits of R then
        votesAgainst[k] ← votesAgainst[k] + 1
      else
        if R can extend to cover Q then
          validLabel[k] ← true
          votesFor[k] ← votesFor[k] + 1
        end if
      end if
    end for
  end for
  label ← decide(validLabel,votesFor,votesAgainst,received)
end if
```

Fig. 7. Algorithm to classify new queries.

4 Empirical Evaluation

We have run all our experiments on an AMD x86/1.4Ghz and 256Mb DDR RAM PC running Windows XP. SCALLOP have been tested for 15 continuous attributes using a method similar to [7]. The concepts to be learned are created by randomly generation of decision trees with 8 levels (128 concepts to be learned). Each leaf is randomly assigned a class label between only two possible values, 0/1. The tree was grown in each internal node with a random pair (attribute,value) that is consistent with the path from the root to such a node. For every example of the training stream, the 15 attribute values are generated with a simple uniform number generator as a stream of pseudo–random numbers in the real interval [0,1]. The class label associated with each training example is then assigned according to the target tree. As in [7], 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 the algorithm).

We have evaluated three aspects of the performance given by SCALLOP: the prediction accuracy, the stabilization speed, and the running time. They have been measured for different sizes of the training stream. We have added a class label noise level of 1% so that every 100 examples, one of them was passed with a random label. For each training set, 10% of examples were used for testing. We have carried out ten evaluations for each training and ten training for each size. The values used for the parameters of the algorithm were: $\alpha = 100$, $\beta = 3$, $\gamma = 0.9$, $\delta = 10^4$, $\lambda = 0.01$, $\mu = 2 \cdot 10^4$, and $\sigma = 10^6$.

Table 1. Performance given by SCALLOP learning 128 concepts of 15 continuous dimensions. NE is the number of examples; CA is the classification accuracy; TC is the percentage of test examples covered by the ruleset; AC is the accuracy obtained by direct covering; NR is the final number of rules; NS is the total number of rules split during the process; and RP is the number of new rules generated before α examples of each label are read.

NE	%CA	%TC	%AC	NR	NS	RP
$5 \cdot 10^4$	66.0 ± 0.70	29.3	28.8	190	780	480
$1 \cdot 10^5$	74.0 ± 0.40	45.5	45.4	185	1400	1000
$2 \cdot 10^5$	84.0 ± 0.17	64.5	64.4	185	2150	2050
$3 \cdot 10^5$	91.5 ± 0.15	73.6	73.5	185	2550	3080
$4 \cdot 10^5$	93.0 ± 0.11	81.8	81.7	185	3150	4180
$5 \cdot 10^5$	94.0 ± 0.11	84.9	84.9	185	3225	5380
$1 \cdot 10^6$	95.3 ± 0.02	90.4	90.4	170	3330	12370
$5 \cdot 10^6$	95.8 ± 0.02	90.3	90.3	137	4000	72200

Table 2. Running time to build the model and classify 10% test examples. NE is the number of examples; TL is the time needed to built the model (in seconds); TC is the time to classify the test examples (in seconds); and %UE is the percentage of examples that are not used to update the model.

NE	TL	TC	%UE
$5 \cdot 10^4$	65	0.4	42
$1 \cdot 10^5$	115	0.6	33
$2 \cdot 10^5$	165	1.0	24
$3 \cdot 10^5$	210	1.3	18
$4 \cdot 10^5$	250	1.6	16
$5 \cdot 10^5$	290	1.6	15
$1 \cdot 10^6$	340	2.2	6
$5 \cdot 10^6$	925	10.8	1

Table 1 shows the results obtained with respect to the accuracy and the stability given by SCALLOP. The last row shows the results for a changing–tree, so that every million example the target tree was replaced making SCALLOP to reject the generated rules. From 5 million examples, the tree was stationary. The accuracy becomes stable from one million examples like the number of covered examples. It is important the high percentage of test examples correctly classified without direct cover for $5 \cdot 10^4$ tests (almost 37% of correctly classified) and for 10^5 (almost 30%). The number of split rules does not increase under a linear trend but from one million examples tend to be asymptotic bounded. From five million examples, the number of final rules is very near to the number of concepts to be learned.

Table 2 shows the results obtained in running time (seconds). Column TL shows that the running time to update the model is proportionally decreasing

as the number of examples increases, so that the system's stability increases as the number of examples. Column UE shows that the quality of the rules is increasingly nearer to the real concepts to be extracted. These results lead us to think that SCALLOP is a good choice to mine major patterns from continuous data streams.

5 Conclusions

A scalable classification learning algorithm based on decision rules and prototypes has been introduced in this paper. Providing a model on demand, which improves its simplicity and helpfulness for the user, we have developed a system for mining numerical, low–dimensionality, high–speed, time–changing data streams that updates the model with each new example. With a refining method as part of the algorithm, SCALLOP is able to remove *out–of–date* rules that have become uninteresting for the user and wrong rules caused by noise. This periodical pruning does not adversely affect the computational cost but rather speeds up its subsequent updating by helping to make the model more stable.

The strong point of our algorithm is that the generated rules *know* where can extend to, what provides few rules to classify new queries without decreasing the accuracy. This approach is different to decision tree based algorithms in that the whole search space is not modelled and the new queries are classified by voting.

Our future research directions are oriented to drop irrelevant dimensions, and recover dropped attributes turned relevant later (there is no much literature on feature selection from data streams). We are also studying to deal with nominal attributes in order to be able to compare SCALLOP with another classification algorithms, as CVFDT [14] and SPRINT [20].

Acknowledgements. The research was supported by the Spanish CICYT under grant TIC2001-1143-C03-02.

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc.* 20^{th} *International Conf. on Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.
2. P.L. Bartlett, S. Ben-David, and S.R. Kulkarni. Learning changing concepts by exploiting the structure of change. In *Computational Learing Theory*, pages 131–139, 1996.
3. P.S. Bradley, U.M. Fayyad, and C. Reina. Scaling clustering algorithms to large database. *Knowledge Discovery and Data Mining*, pages 9–15, 1998.
4. J. Cattlet. *Megainduction: machine learning on very large databases.* PhD thesis, Basser Department of Computer Science, University of Sydney, Australia, 1991.
5. D. Wai-Lok Cheung, J. Han, V. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *ICDE*, pages 106–114, 1996.

6. A. Dobra and J. Gehrke. Secret: A scalable linear regression tree algorithm. In *Proc.* 8^{th} *ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining*, Edmonton, Canada, 2002. ACM Press.
7. P. Domingos and G. Hulten. Mining high-speed data streams. In *Proc.* 6^{th} *ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining*, pages 71–80, Boston, MA, 2000.
8. V. Ganti, J. Gehrke, and R. Ramakrishnan. Mining data streams under block evolution. *ACM SIGKDD Explorations*, 3(2):1–10, 2002.
9. M. Garofalakis and R. Rastogi. Scalable data mining with model constraints. *ACM SIGKDD Explorations*, 2(2):39–48, 2000.
10. J. Gehrke, V. Ganti, R. Ramakrishnan, and W.Y. Loh. BOAT – optimistic decision tree construction. In *ACM SIGMOD Conference*, pages 169–180, Philadelphia, Pennsylvania, 1999.
11. J. Gehrke, R. Ramakrishnan, and V. Ganti. Rainforest – a framework for fast decision tree construction of large datasets. In *Proc.* 24^{th} *Int. Conf. Very Large Data Bases, VLDB*, pages 416–427 , 1998.
12. S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *IEEE Symposium on Foundations of Computer Science*, pages 359–366, 2000.
13. S. Guha, R. Rastogi, and K. Shim. CURE: an efficient clustering algorithm for large databases. In *ACM SIGMOD International Conference on Management of Data*, pages 73–84, June 1998.
14. G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proc.* 7^{th} *ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining*, pages 97–106, San Francisco, CA, 2001. ACM Press.
15. G. Hulten, L. Spencer, and P. Domingos. Mining complex models from arbitrarily large databases in constant time. In *Proc.* 8^{th} *ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining*, Edmonton, Alberta, Canada, 2002. ACM Press.
16. M. Kelly, D. Hand, and N. Adams. The impact of changing populations on classier performance, 1999.
17. R. Rastogi M. Garofalakis, D. Hyun and K. Shim. Scalable data mining with model constraints. In *Proc.* 6^{th} *ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining*, pages 335–339, Boston, MA, 2000.
18. L. O'Callaghan, N. Mishra, A. Meyerson, and S. Guha. High–performance clustering of streams and large data sets. In *Proc.* 18^{th} *International Conf. on Data Engineering*, pages 359–366, 2000.
19. F. Provost and V. Kolluri. A survey of methods for scaling up inductive algorithms. *Data Mining and Knowledge Discovery*, 3(2):131–169, 1999.
20. J.C. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proc.* 22^{th} *International Conf. Very Large Databases, VLDB*, pages 544–555, 1996.
21. T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *ACM SIGMOD International Conf. on Management of Data*, pages 103–114, Montreal, Canada, June 1996.