FACIL – An Approach for Classifying Data Streams by Decision Rules and Border Examples

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, riquelme}@lsi.us.es

Abstract. This paper describes FACIL, a classifier based on decision rules and border examples that avoids unnecessary revisions when virtual drifts are present in data. Rules in FACIL are both pure - consistent - and impure - inconsistent -. Pure rules classify new test examples by covering and impure rules classify them by distance as the nearest neighbor algorithm. In addition, the system provides an implicit forgetting heuristic so that positive and negative examples are removed from a rule when they are not near one another.

1 Introduction

Formally, a data stream is an ordered sequence of data items $< \ldots e_{i-1} \prec e_i \succ e_{i+1} \ldots >$ read in increasing order of the indices i. In practice, a data stream is an unbounded sequence of items liable to both noise and concept drift, and received at a so high rate that each one can be read at most once by a real time application [2]. Thus, data streams contexts compel to learning systems to give approximate answers using small and constant time per example [3]. Recent works on data streams classification has been mainly addressed by two different approaches: decision trees [1, 3, 4] and ensemble methods [5, 8, 9].

Domingos & Hulten's VFDT and CVFDT systems [3] build a decision tree based on Hoeffding bounds, which guarantee constant time and memory per example and an output model asymptotically nearly identical to that given by a batch conventional learner from enough examples. Since VFDT and CVFDT are evaluated for data streams with symbolic attributes, Jin & Agrawal propose in [4] a numerical interval pruning approach to reduce the processing time for numerical attributes, without loss in accuracy. Gama et al.'s VFDTc system [1] extends the VFDT properties in two directions: the ability to deal with numerical attributes and the ability to apply naive Bayes classifiers in tree leaves.

Ensemble batch learning algorithms such as Boosting and Bagging have proven to be highly effective from disk–resident data sets. These techniques perform repeated resampling of the training set, making them a priori inappropriate in a data streams environment. Despite what might be expected, novel ensemble methods are increasingly gaining attention because of they have proved to offer an improvement in prediction accuracy. In general, every incremental ensemble approach uses some criteria to dynamically delete, reactivate, or create

new ensemble learners in response to the base models' consistency with the current data. SEA [8] is a fast algorithm that requires approximately constant memory. It builds separate classifiers on sequential chunks of training examples, combining them into a fixed–size ensemble according to a heuristic replacement strategy. From sequential blocks as well, Wang et al. [9] propose using ensemble of classifiers weighted based on their expected classification accuracy on the test examples. In [5] Kolter & Maloof propose DWM, an ensemble method based on the Weighted Majority algorithm [6].

As pointed out in [9], a drawback of decision trees is that even a slight drift of the target function may trigger several changes in the model and severely compromise learning efficiency. On the other hand, ensemble methods avoid expensive revisions by weighting the members, but may run the risk of building unnecessary learners when virtual drifts are present in data. Rule sets take advantage of not being hierarchically structured, so concept descriptions can be updated or removed when becoming out of date without hardly affecting the learning efficiency. Mining potentially infinite data sequences usually results in large, complex and incomprehensible models, so we claim interactive, parametrized algorithms for moving on the expert's priorities to less accurate but more comprehensible answers. In this sense, rule sets could be a more useful knowledge representation than disjointed and hierarchically structured hypercubes given by decision trees, from which of the user need to explore paths of several dozen of levels to know interesting patterns.

2 The FACIL approach

The core of our approach lies in avoiding specific rules and allowing they may not be consistent. Within rule learning, each training example is said a maximally specific rule. On the other hand, a rule is said pure or consistent when does not cover any example of different label. In addition, impure rules are linked with border examples, i.e. different label examples which are very near one another. The goal is to seize border examples up to a threshold Ω is reached. This threshold is given as an user parameter and sets the minimum purity of a rule. Since FACIL is aimed at multi–class problems, let us extend the concepts of positive and negative example according to the next notation.

Let m be the number of attributes $\mathcal{A}_j$ ($j \in \{1, \ldots, m\}$). Let $\mathcal{Y} = \{y_1, \ldots, y_z\}$ be the set of class labels. Let $e_i = (\overrightarrow{x_i}, y_i)$ be the i^{th} example arriving, where $\overrightarrow{x_i}$ is a vector with m attribute values and y_i is a discrete value in $\mathcal{Y}$. The antecedent of a rule $\mathcal{R}$ in FACIL is given by a conjunction of m conditions $\mathcal{I}_j$ that defines a region inside the multidimensional attribute space. $\mathcal{I}_j$ is a closed interval $[I_{jl}, I_{ju}]$ when $\mathcal{A}_j$ is a numerical attribute so that l denotes lower bound and u upper bound. If $\mathcal{A}_j$ is symbolic, then $\mathcal{I}_j$ is a set of values $a_j \in \mathcal{D}(\mathcal{A}_j)$ belonging to the attribute domain $\mathcal{D}(\mathcal{A}_j)$ and standing for a disjunction of all those values.

Definition 1 (Positive Coverage of a rule (pe)) *The positive coverage pe of a rule $\mathcal{R}$ is the number of same label examples covered by the rule $\mathcal{R}$. Thus, an example $e_i = (\overrightarrow{x_i}, y_i)$ is said positive for a rule $\mathcal{R}$ with label y' if $y' = y_i$.*

Definition 2 (Negative Coverage of a rule (ne)) *The negative coverage ne of a rule $\mathcal{R}$ is the number of different label examples covered by the rule $\mathcal{R}$. Thus, an example $e_i = (x_i^{'}, y_i)$ is said negative for a rule $\mathcal{R}$ with label y' if $y' \neq y_i$.*

Definition 3 (Purity of a rule (ω)) *Let pe and ne be the positive and negative coverage of a rule $\mathcal{R}$, respectively. The purity or confidence of $\mathcal{R}$ is defined as:*

$$0 < \omega(\mathcal{R}) = \frac{pe}{pe + ne} \leq 1$$

Thus, the purity of a rule is the ratio between the number of positive examples that it covers and its total number of covered examples, positive and negative. When the threshold Ω is reached by a rule $\mathcal{R}$ ($\omega(\mathcal{R}) < \Omega$), FACIL updates the model from the examples associated with $\mathcal{R}$, generating new consistent rules that describe both positive and negative examples that $\mathcal{R}$ has covered.

This approach is similar to the AQ11–PM algorithm [7], which selects positive examples from the boundaries of its rules (hyper rectangles) and stores them in memory. When new examples arrive, AQ11–PM combines them with those held in memory, applies the AQ11 algorithm to modify the current set of rules, and selects new positive examples from the corners, edges, or surfaces of such hyper rectangles (*extreme* examples).

FACIL differs from AQ11–PM in that only impure rules store examples, positive and negative ones. Such examples are not necessary extreme and the rules are not repaired every time they become inconsistent, reducing the computational complexity. The more number pe of positive examples covered by a rule $\mathcal{R}$, the more number ne of negative examples that $\mathcal{R}$ can store, so every time ne increases by one unit, a new positive example is stored. Positive examples of a rule are updated for them to be nearer to negative covered examples than positive ones. For example, let $\Omega = 0.9\%$ be the minimum purity threshold. When a rule $\mathcal{R}$ has covered 90 positive examples, the maximum number ne' of negative examples that $\mathcal{R}$ can store is 10. On covering its hundredth positive example e, ne' increases in one unit, so that FACIL maintains e in memory as a member of the window associated with $\mathcal{R}$. Therefore, every impure rule has an independent sliding window of recent border examples.

Although this approach suffers the ordering effects, it does not severely compromise the learning efficiency and guarantees that an impure rule is always revised from as positive as negative examples. In addition, FACIL is based on *instance by instance learning* instead of the most usual *block by block learning* approach. Therefore, every time a new example $e = (\overrightarrow{x_i}, y_i)$ arrives FACIL updates the set of rules. In this process, three tasks are at most performed in the next order:

1. ***Positive covering***: x_i is covered by a rule associated with the same label y_i.
2. ***Negative–covering***: x_i is covered by a rule associated with a different label $y' \neq y_i$.
3. ***New description***: x_i is not covered by any rule.

Algorithm 1 FACIL – IIL updating

INPUT Φ, Ψ, Ω: real; $e_i = (x_i, y_i)$: example
INPUT/OUTPUT $\mathcal{M}$: Set of rules;
begin
  $\mathcal{R}_c \leftarrow \varnothing$
  candidate$\leftarrow$**positive–covering**($\downarrow \Phi$, x_i, $\downarrow\uparrow \mathcal{M}_{y_i}$, $\mathcal{R}_c$)
  if $\mathcal{R}_c \neq \varnothing$ **then**
    $\mathcal{R}' \leftarrow$**negative covering**($\downarrow \Omega$, $\downarrow\uparrow \mathcal{M}_{y'}$, $\mathcal{R}_c$)
    if $\mathcal{R}' = \varnothing$ **then**
      if $\mathcal{R}_c \neq \varnothing$ **then**
        replace(*candidate*, $\mathcal{R}_c$)
    else
      revise($\downarrow x_i$, $\mathcal{R}'$, $\downarrow\uparrow \mathcal{M}$)
  if $\mathcal{R}_c = \varnothing$ **and** $|\mathcal{M}_{y_i}| < \Psi$ **then**
    $\mathcal{R}_c \leftarrow$**generalize**(x_i)
    $\mathcal{M}_{y_i} \leftarrow \mathcal{M}_{y_i} \cup \{\mathcal{R}_c\}$
end

Positive–covering. Algorithm 1 shows the global process to build the model, which takes three user parameters: Φ, Ψ, and Ω. Ω is the minimum purity threshold described above. Φ and Ψ are described later on. First, the rules associated with y_i are visited. Rules are stored in different sets $\mathcal{M}_{y_i}$ depending on the associated label. While visiting the rules in $\mathcal{M}_{y_i}$, FACIL computes the generalization necessary to describe the new example x_i, according to Equation 1.

Definition 4 (Growth of a rule) *Let $\mathcal{R}$ be a rule whose antecedent is formed by m conditions $\mathcal{I}_j$. Let $e = (x, y)$ be an example. The growth $\mathcal{G}(\mathcal{R}, x)$ of the rule $\mathcal{R}$ to cover the point x is defined according to Equation 1:*

$$\mathcal{G}(\mathcal{R}, x) - \sum_{j-1}^{m} \Delta(\mathcal{I}_j, x_j); \tag{1}$$

$$\Delta(\mathcal{I}_j, x_j) = \begin{cases} \delta(x_j, \mathcal{I}_j), \text{ if } \mathcal{A}_j \text{ is numerical;} \\ \partial(x_j, \mathcal{I}_j), \text{ if } \mathcal{A}_j \text{ is symbolic.} \end{cases} \tag{2}$$

$$\delta(x_j, \mathcal{I}_j) = \min(\mid I_{jl} - x_j \mid, \mid x_j - I_{ju} \mid); \tag{3}$$

$$\partial(x_j, \mathcal{I}_j) = \begin{cases} \frac{1}{|\mathcal{D}(\mathcal{A}_j)|}, \text{ if } x_j \notin \mathcal{I}_j; \\ 0, \qquad \text{if } x_j \in \mathcal{I}_j; \end{cases} \tag{4}$$

This metric gives a rough estimate of the fraction of total space that a rule takes in order to cover a new positive example. Normalization is necessary to avoid numerical attributes with a large range (e.g. a real domain) outweigh attributes with a small range. The main gain of this simple generalization heuristics is that it biases in favour of the rule that involves smallest changes in least number of attributes. Increment in a symbolic attribute $\mathcal{A}_j$ is proportional to the number of values of its domain $\mathcal{D}(\mathcal{A}_j)$.

After visiting the rules in $\mathcal{M}_{y_i}$, the one with the minimum growth is marked as *final candidate*. Henceforth, we denote this rule as $\mathcal{R}_c$. However, a rule is only taken into account as a *possible candidate* if it can cover the new example under a moderate growth, according to Definition 5.

Definition 5 (Moderate Growth) *Let $\Phi \in (0, 1]$ be a real value given as an user parameter. The growth $\mathcal{G}$ of a rule $\mathcal{R}$ requires to describe an example $e = (x, y)$ is said moderate if:*

$$\forall j \in \{1, \ldots, m\} \cdot \ \Delta(\mathcal{I}_j, x_j) \leq \Phi$$

Since every numeric value is previously normalized in $[0, 1]$, the divisor factor for the domain of each attribute is omitted in definition 4. When the first rule covering x_i is found - the resulting growth is therefore 0 - its support is increased by one unit and the index of the last covered example is updated as i. If the number of negative examples that such a rule can store increases by one unit, then the example is added to its window.

Negative covering. If x_i is not covered by any rule in $\mathcal{M}_{y_i}$, then the rest of rules of different label $y' \neq y$ are visited. If a different label rule $\mathcal{R}'$ does not cover x_i, the intersection $\cap'_c$ between $\mathcal{R}'$ and the final *candidate* $\mathcal{R}_c$ is computed. If $\cap'_c \neq \emptyset$, then $\mathcal{R}_c$ is rejected and no positive covering is possible. When the first different label rule $\mathcal{R}'$ covering x_i is found, its negative coverage is increased by one unit, and x_i is added to its window. If $\omega(\mathcal{R}') < \Omega$, then new consistent rules according to the examples in its window are included in the model. Then $\mathcal{R}'$ is marked as *unreliable* so that it can not be generalized and is not taken into account to generalize other rules associated with a different label. In addition, its window is reset.

New description. After above tasks, the candidate rule is generalized if does not intersect with any other rule associated with a label $y' \neq y_i$. When no rule covers the new example and there is not a candidate to be generalized, then a maximally specific rule to describe it is generated, provided that $|\mathcal{M}_{y_i}| < \Psi$.

Furthermore, the set of rules is simultaneously refined while the first two tasks are accomplished. Before computing a rule covers the new example, it is removed if the last generalized rule of the same label (the last candidate) covers it. After computing a rule does not cover the new example, it is removed if satisfies one of two conditions:

- It is an unreliable rule whose support is smaller than the support of any rule generated from it.
- The number of times the rule hindered a different label rule to be generalized is greater than its positive coverage.

2.1 Forgetting Heuristics

Similarly to AQ–PM, our approach also involves a forgetting mechanism that can be either explicit or implicit. Explicit forgetting takes places when the examples are older than an user defined threshold. Implicit forgetting is performed by

removing examples that are no longer relevant as they do not enforce any concept description boundary. When a negative example x in a rule r has not a same label example as the nearest one after the number pe of positive examples that r can store is increased two times since x was covered, the system removes it. Analogously, a positive example is removed if it has not a different label example as the nearest one after pe is increased by two units.

2.2 Classification Heuristics

Finally, to classify a new test example, the systems searches the rules that cover it. If there are reliable and unreliable rules covering it, the latter ones are rejected. Consistent rules classify new test examples by covering and inconsistent rules classify them by distance as the nearest neighbour algorithm. If there is no rule covering it, the example is classified based on the label associated with the reliable rule that involves the minimum growth and does not intersect with any different label rule.

3 Empirical Evaluation

In [3, 9] both robustness and reliability of incremental classifiers are evaluated using synthetic data streams generated from a moving hyperplane. As in [9], here concept drifts are simulated with three parameters. Parameter α specifies the total number of dimensions whose weights are involved in changing. Parameter $\beta \in \mathcal{R}$ specifies the magnitude of the change (every N examples) for weights $a_1, \ldots, a_\alpha$, and $\gamma_i \in \{-1, 1\}$ specifies the direction of change for each weight. Each time the weights are updated, $a_0 = \frac{1}{2}\sum_{i=1}^{m} a_i$ is recomputed so that the class distribution is not disturbed. In addition, class noise is introduced by randomly switching the labels of 5% of the examples. As in [9], 40% dimensions' weights are changing at ± 0.10 per 10000 examples.

In a fist set of experiments, we evaluated the computational cost as a function of the number of attributes. All the experiments were conducted on a PC with CPU 1.7GHz and 512 MB of RAM running Windows XP. Figures 1–3 show the results with explicit forgetting after 100000 examples are processed. The minimum purity threshold Ω was set to 90%. Training and test examples are generated on the fly and directly passed to the algorithm. After 900 training examples are generated, 100 test examples are used to evaluate the algorithm.

Figure 1 shows the prediction accuracy as a function of the number of attributes. Figure 2 shows the time in seconds spent on building the model and classifying new test examples. Figure 3 shows the final number of rules per label. Since running time depends on the number of rules, the parameter Ψ is alternately limited to 50 and 100 rules per label.

Average accuracy is higher than 90% and average running time is higher than 100 examples per second. However, the latter holds satisfactory trade–offs between learning time and model complexity from low dimensionality data, so that:

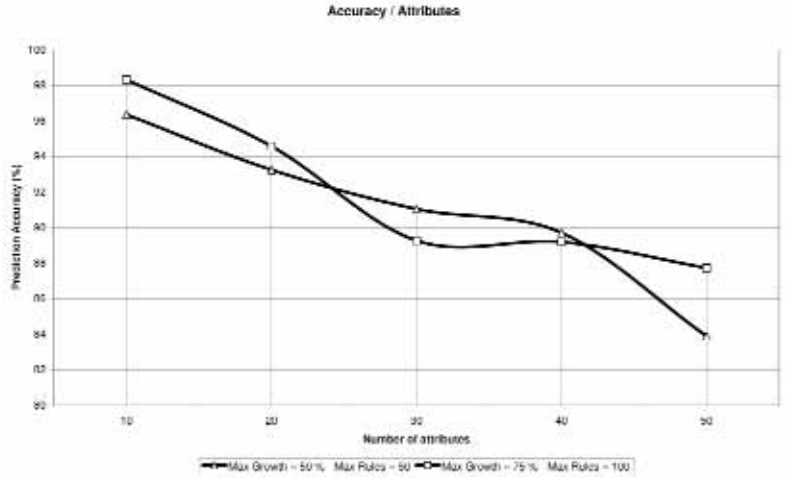

Fig. 1. Accuracy (%) as a function of the number of attributes.

- With ten attributes, learning time is greater than 3500 examples per second and accuracy exceeds 98%.
- With fifty attributes, learning time is greater than 600 examples per second and accuracy exceeds 88%.

In a second set of experiments, the goal was to evaluate capacity to detect drift in the distribution of the examples. Figures 4 6 show the results with explicit forgetting and 10 numeric attributes. In this case, the minimum purity threshold Ω was set to 95%. Analogously to the previous experiments, the maximum growth Φ is alternately limited to 75% and 100%, and the parameter Ψ is set to 25 and 10 rules per label, respectively.

Figure 4 shows the prediction accuracy as the number of training examples increases, which seems to respond to a logarithmic function. Figure 5 shows the time in seconds, and Figure 6 shows the average number of border examples - both positive and negative - per rule. The average accuracy is higher than 95% and the average running time is higher than 23000 examples per second. Again, FACIL shows a very satisfactory performance from low dimensionality data. With respect to the number of border examples per rule, it seems to fluctuate sinusoidally with a downward period. With $n < 30$ being the average number of examples that an impure rule stores, FACIL provides a few rules and high accuracy without exceeding severe memory limitations.

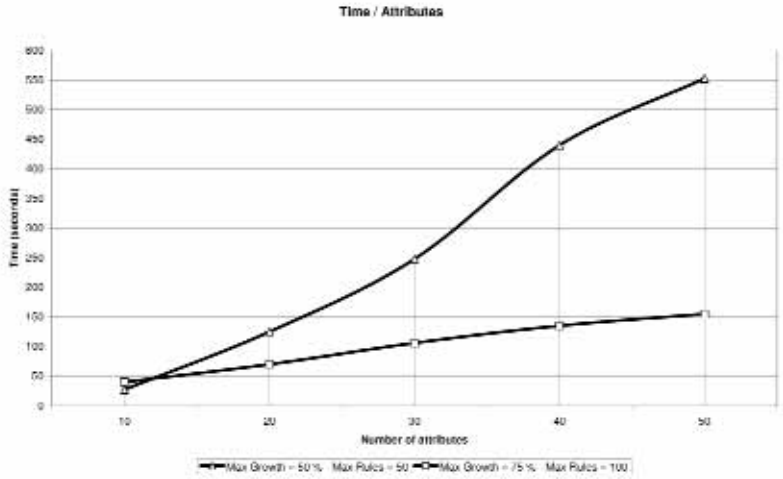

Fig. 2. Time (in seconds) as a function of the number of attributes.

4 Conclusions and Future Work

FACIL is an incremental rule learner with partial instance memory based on parameterized generalization and border examples. FACIL works online, processes each new example in constant time, performs a single scan over the training examples, takes drift into account. Experimental results show a satisfactory performance as a high speed data streams classification method. Our future research directions are oriented to drop irrelevant attributes, and recover dropped attributes turned relevant later.

References

1. J. Gama, P. Medas, and R. Rocha. Forest trees for on-line data. In *Proc. of the* 19^{th} *ACM Symposium on Applied Computing - SAC'04*, pages 632–636.
2. L. Golab and M.T. Ozsu. Issues in data stream management. *SIGMOD Record*, 32(2):5 14, 2003.
3. G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proc. of the* 7^{th} *ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining - KDD'01*, pages 97–106.
4. R. Jin and G. Agrawal. Efficient decision tree construction on streaming data. In *Proc. of the* 9^{th} *ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining - KDD'03.*

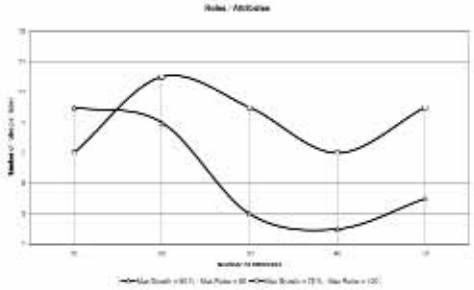

Fig. 3. Number of rules as a function of the number of attributes.

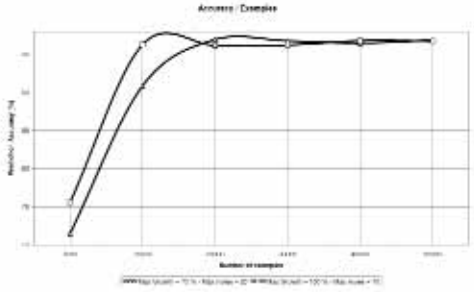

Fig. 4. Accuracy (%) as a function of the number of examples.

5. Jeremy Z. Kolter and M.A. Maloof. Dynamic weighted majority: A new ensemble method for tracking concept drift. In *Proc. of the 3th IEEE Int. Conf. on Data Mining - ICDM'03*, pages 123–130, 2003.
6. N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994.
7. M.A. Maloof and R.S. Michalski. Incremental learning with partial instance memory. *Artificial Intelligence*, 154:95–126, 2004.
8. W. Street and Y. Kim. A streaming ensemble algorithm SEA for large-scale classification. In *Proc. of the 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining - KDD'01*, pages 377 382.
9. H. Wang, W. Fan, P.S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proc. of the 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining - KDD'03*, pages 226–235.

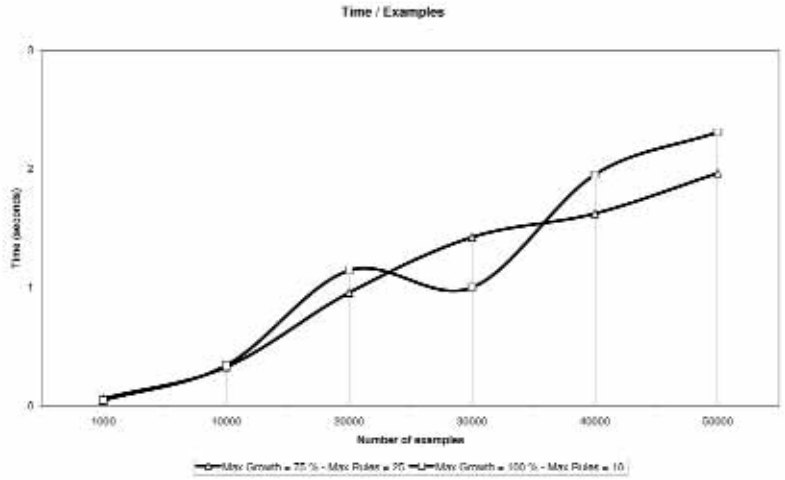

Fig. 5. Time (in seconds) as a function of the number of examples.

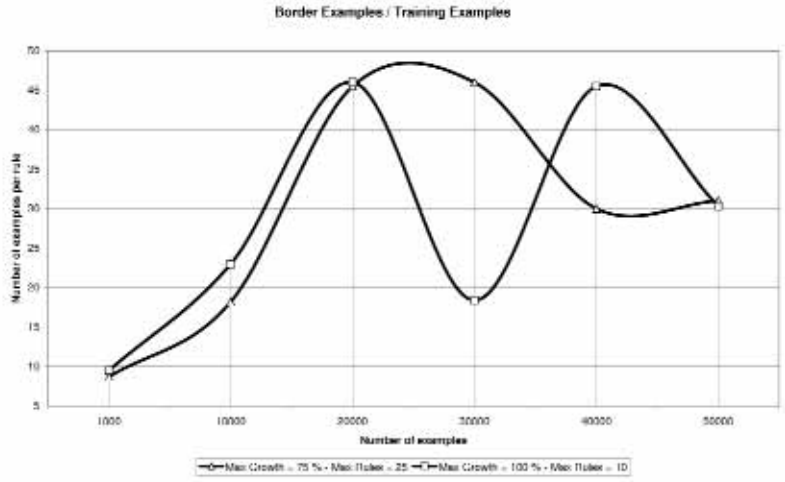

Fig. 6. Number of border examples per rule as a function of the number of examples.