FACIL: Incremental Rule Learner for Classification of Data Streams

Francisco J. Ferrer–Troyano\textsuperscript{1}, Jesús S. Aguilar–Ruiz\textsuperscript{2}, and José C. Riquelme\textsuperscript{1}

\textsuperscript{1} Computer Science Department, University of Seville, Spain
\{ferrer,riquelme\}@lsi.us.es
WWW home page: http://www.lsi.us.es/~ferrer

\textsuperscript{2} School of Engineering, University Pablo de Olavide, Spain
jsagurui@upo.es

Abstract. This paper describes an incremental classifier that provides a model formed by decision rules and border examples. Such examples are only linked with impure rules, so that they are instance–by–instance updated in order to describe decision boundaries. By means of this model, our approach can be seen as a multiple–window–based learning algorithm, using one window per impure rule. In addition, new examples can be classified by covering or distance such as the nearest neighbour algorithm. Furthermore, the algorithm can use an implicit forgetting heuristic so that positive and negative examples are removed from an impure rule when they are not the nearest neighbours one another.

Keywords: Classification, incremental learning, decision rules, data streams

1 Introduction

Over more than 40 years, a large number of learning methods based on very different approaches have been developed and successfully applied to a great deal of real–world problems of classification. A significant family of classifiers is that of inductive rule based methods for which all input training examples are required to be loaded in main memory before learning begins. Techniques belonging to this family are known as batch learners or multi–pass algorithms, so that rules are obtained by several passes on the training set. The problem with batch learning is that it makes the knowledge extraction from data streams hard, at least impossible. Examples of data streams include networks event logs, phone calls and telecommunications records, spatial data, financial transactions, and credit card operations. Formally, a data stream is just an ordered sequence of items read in increasing order of the indices. In practice, data stream classification implies an unbounded sequence of examples received at a so high rate that each one can be read at most once. This scenario compels learning algorithms to give an approximate answer from few scans (ideally only one), using small and constant time per example, and assuring that both result and performance are not adversely affected by the order of the examples [3].
This paper describes FACIL\textsuperscript{3} (Fast and Adaptive Classifier by Incremental Learning), an incremental classifier that builds a model formed by decision rules and border examples. Such examples are only linked with impure rules, so that they are updated to describe decision boundaries. Pure or consistent rules have no examples associated, whereas impure or inconsistent rules retain both positive and negative examples. In this sense, FACIL can be seen as an incremental rule learner based on multiple windows, one per impure rule. In addition, FACIL uses several strategies to classify new examples according to where they lie. Pure rules classify new examples by covering and inconsistent rules classify them by distance as the nearest neighbour algorithm. Furthermore, our proposal provides an implicit forgetting heuristic so that positive and negative examples are removed when they are not near one another.

The paper is organized as follows. The next section outlines a background and related work of classification, incremental learning, concept drift and data streams classification systems. In Section 3, we motivate and describe the basis of our algorithm. Section 4 describes the data sets used in our experiments and shows the results achieved. In Section 5, we discuss the conclusions we reached based on the experimental results and outline possible directions for future works.

2 Background and Related Work

Within batch learning, the problem of classification is generally defined as follows. An input data set of training examples \( T = \{e_1, \ldots, e_n\} \) is given. Every training example \( e_i = (\vec{x}_i, y_i) \) is formed by a vector \( \vec{x}_i \) and a discrete value \( y_i \), named label and taken of a finite set \( \mathcal{Y} \) named class. Every vector \( \vec{x}_i \) has the same dimensionality, each dimension is named attribute and each component \( x_{ij} \) is an attribute value (numeric or symbolic). Under the assumption there is an underlying mapping function \( f \) so that \( y = f(\vec{x}) \), the goal is to obtain a model from \( T \) that approximates \( f \) as \( \hat{f} \) in order to classify or decide the label of non–labelled examples (tests), so that \( \hat{f} \) maximizes the prediction accuracy.

A large family of classification algorithms is that of rule based approaches. A decision rule is a logic predicate of the form \( \text{if antecedent then label} \). The antecedent is a conjunction of conditions of the form \( \text{Attribute} \models \text{Values} \), with \( \models \) being an operator that states a relation between an attribute and values of its domain. If the vector \( x \) of an example \( e \) fulfills the antecedent of a rule \( \mathcal{R} \), then \( e \) is said described or covered by \( \mathcal{R} \). Regions modelled by rule learners may not describe the whole attribute space, so that new test examples may not be covered by any rule. Fundamental batch learning classifiers based on decision rules include the AQ family [8], CN2 [1], and RIPPER [2].

Within incremental learning, a whole training set is not available a priori but examples arrives over time, normally one at a time \( t \) and not time–dependent necessarily. According to the taxonomy in [7], if \( T_t = \{(\vec{x}, y) : y = f(\vec{x})\} \) for

\textsuperscript{3} \( \text{fácil} \) is the spanish word for easy
\( t = < 1, \ldots, \infty > \), then now \( \hat{f}_t \) approximates \( f \). In this context, if an algorithm discards \( \hat{f}_{t-1} \) and generates \( \hat{f}_t \) from \( T_i \), for \( i = < 1, \ldots, t > \), then it is on-line batch or temporal batch with full instance memory. If the algorithm modifies \( \hat{f}_t \) using \( \hat{f}_{t-1} \) and \( T_t \), then it is purely incremental with no instance memory. A third approach is that of systems with partial instance memory, which select and retain a subset of past training examples to use them in future training episodes. Our proposal belongs to this latter family, where fundamental incremental rule learners include STAGGER [9] (the first system designed expressly for coping with concept drift), the FLORA family of algorithms [12] (with FLORA3 being the first system able to deal with recurring contexts), and the AQ–PM family [7]. Since pure incremental rule learners take into account every training example, many of them have not still adapted to a data streams environment, especially those featuring numerical attributes.

Along with the ordering effects, incremental learning from real-world domains faces two problems known as hidden context and concept drift, respectively [12]. The problem of hidden context is when the target concept may depend on unknown variables, which are not given as explicit attributes. The problem of concept drift is when changes in the hidden context induce changes in the target concept. In addition, changes in the hidden context may change the underlying data distribution, making incremental algorithms to review the current model in every learning episode. This latter problem is called virtual concept drift.

Above problems make incremental learning be more complex than batch learning, so effective learners should be able to distinguish noise from actual concept drift. There are two common approaches that can be applied altogether to detect changes in the target concept [6]. An approach consists in repeatedly applying the learner to a single window of training examples whose size can be dynamically adjusted whenever target function starts to drift. The other one is to apply weighting for the training examples according to the time they arrive, reducing the influence of old examples. Weighting based approaches are partial instance memory methods.

Data Streams Mining can be seen as an extreme case in incremental learning. Formally, a data stream is an ordered sequence of data items read in increasing order of the indices. 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 [3]. Recent works on data streams classification has been mainly addressed by two approaches: decision trees [4, 5] and ensemble methods [10, 11].

As pointed out in [11], 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 sequen-
ces usually results in large, complex and incomprehensible models, so we claim interactive, parameterized 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 the user need to explore paths of several dozen of levels to know interesting patterns.

3 FACIL

3.1 Knowledge Model

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 \( \Omega \) 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 \( A_j (j \in \{1, \ldots, m\}) \). 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 \( I_j \) that defines a region inside the multidimensional attribute space. \( I_j \) is a closed interval \([I_{jl}, I_{ju}]\) when \( A_j \) is a numerical attribute so that \( l \) denotes lower bound and \( u \) upper bound. If \( A_j \) is symbolic, then \( I_j \) is a set of values \( a_j \in D(A_j) \) belonging to the attribute domain \( D(A_j) \) 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 \( e_i = (\overrightarrow{x_i}, y_i) \) is said negative for a rule \( \mathcal{R} \) with label \( y' \) if \( y' \neq y_i \).

Definition 3 (Purity of a rule (\( \omega \))) 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 \( \Omega \) is reached by a rule, i.e. \( \omega(\mathcal{R}) < \Omega \), 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. Along with $\Omega$, FACIL takes two user parameters $\Phi$ and $\Psi$ that are described later on.

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 are linked with examples, which are positive and negative. Such examples are not necessary extreme and the rules are not repaired every time they become inconsistent. The more number $pe$ of positive examples covered by a rule $R$, the more number $ne$ of negative examples that $R$ can store, so every time $ne$ increases by one unit, a new positive example is stored. 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.

### 3.2 Instance by Instance Learning

FACIL is based on instance by instance learning instead of the more usual block by block learning approach. Every time a new example $e = (\vec{x}_i, y_i)$ is read, FACIL updates the set of rules. Rules are stored in different sets $\mathcal{M}_{y_i}$ depending on the associated label. 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 instance**: $x_i$ is not covered by any rule.

**Positive–covering**. First, the rules in $\mathcal{M}_{y_i}$ associated with the label $y_i$ of the new example are visited. While visiting the rules, FACIL computes the generalization necessary to describe the new example $x_i$, according to Eq. 1.

**Definition 4 (Growth of a rule)** Let $\mathcal{R}$ be a rule whose antecedent is formed by $m$ conditions $I_j$. 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:

$$\mathcal{G}(\mathcal{R}, x) = \sum_{j=1}^{m} \Delta(I_j, x_j);$$

$$\Delta(I_j, x_j) = \begin{cases} \delta(x_j, I_j), & \text{if } A_j \text{ is numerical}; \\ \partial(x_j, I_j), & \text{if } A_j \text{ is symbolic}. \end{cases}$$

$$\delta(x_j, I_j) = \min(|I_{jl} - x_j|, |x_j - I_{ju}|);$$

$$\partial(x_j, I_j) = \begin{cases} \frac{1}{\max(I_j)}, & \text{if } x_j \notin I_j; \\ 0, & \text{if } x_j \in I_j. \end{cases}$$
This metric gives a rough estimate of the quantity of changes required in the conditions forming the antecedent of a rule to describe a new 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. The growth of a rule is computed as the sum of the increases in all the input attributes (numeric and symbolic). Increment in a symbolic attribute $A_j$ is proportional to the number of values of its domain $D(A_j)$.

After visiting the rules in $M_{y_i}$, the one with the minimum growth is marked as final candidate. Henceforth, we denote this rule as $R_c$. 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 $G$ of a rule $R$ requires to describe an example $e = (x, y)$ is said moderate if:

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

with $\Phi$ being the second user parameter. 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 $M_{y_i}$, then the rest of rules of different label $y' \neq y$ are visited. If a different label rule $R'$ does not cover $x_i$, the intersection $\cap_c'$ between $R'$ and the final candidate $R_c$ is computed. If $\cap_c' \neq \emptyset$, then $R_c$ is rejected and no positive covering is possible. When the first different label rule $R'$ covering $x_i$ is found, its negative coverage is increased by one unit, and $x_i$ is added to its window. If $\omega(R') < \Omega$, then new consistent rules according to the examples in its window are included in the model. Then $R'$ 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 $|M_{y_i}| < \Psi$.

### 3.3 Concurrent Pruning

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.

3.4 Forgetting Heuristics

Similarly to AQ–PM, our approach also involves a forgetting mechanism that can be either explicit or implicit. Explicit forgetting takes place 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. Thus, positive examples linked with an impure rule are updated for them to be nearer to negative covered examples than positive ones.

3.5 Computational Complexity

In worst case, a new example involves a negative-covering after visiting all the rules in each set \( M_y \). The computational complexity associated with this case is \( O(m \cdot s \cdot \tau) \), with \( m \) being the number of attributes, and \( s \) as the model size or total number of rules. \( \tau \) estimates the average number of examples per rule.

3.6 Classification

Finally, to classify a new test example, FACIL 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.

4 Empirical Evaluation

As in [4, 11], accuracy, learning time, and memory requirements were evaluated using synthetic data streams generated from a moving hyperplane. All the experiments were conducted on a PC with CPU 1.7GHz and 512 MB of RAM running Windows XP. First, examples are randomly generated and uniformly distributed in multidimensional space \([0, 1]^m\). The examples satisfying \( \sum_{i=1}^{m} a_i x_i \geq a_0 \) are labelled as positive, and examples satisfying \( \sum_{i=1}^{m} a_i x_i < a_0 \) as negative. Weights \( a_i \) (\( 1 \leq i \leq m \)) are initialized by random values in the range of \([0, 1]\).
The value of \( a_0 \) is chosen so that the hyperplane cuts the multi-dimensional space in two parts of the same volume, that is: \( a_0 = \frac{1}{2} \sum_{i=1}^{m} a_i \). Thus, roughly half of the examples are positive, and the other half are negative. Concept drift is simulated with three parameters: \( \alpha, \beta, \) and \( \gamma \). Parameter \( \alpha \) specifies the total number of dimensions whose weights are involved in changing. Parameter \( \beta \in \mathbb{R} \) 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. 40% dimensions’ weights are changing at \( \pm 0.10 \) per 10000 examples.

In a first set of experiments, the goal was to evaluate capacity to detect drift in the distribution of the examples. 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. Figures 1, 2, and 3 show the results with explicit forgetting and 10 numeric attributes. In this case, the minimum purity threshold \( \Omega \) was set to 95%. Since running time depends on the number of rules, the maximum growth \( \Phi \) is alternately limited to 75% and 100%, and the parameter \( \Psi \) is set to 25 and 10 rules per label, respectively.

Figure 1 shows the prediction accuracy as the number of training examples increases, which seems to respond to a logarithmic function. Figure 2 shows the time in seconds, and Figure 3 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. Therefore, 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 high accuracy without exceeding severe memory limitations.

In a second set of experiments, we evaluated the sensitivity of FACIL as a function of the number of attributes. Figure 4 shows the final number of rules per label with explicit forgetting after 100000 examples were processed. The minimum purity threshold \( \Omega \) was set to 90%. Analogously to the previous experiments, the parameter \( \Psi \) is alternately limited to 50 and 100 rules per label. In both cases, the memory requirements seems to fluctuate sinusoidally without a similar period.

5 Conclusions and Future Work

FACIL is a IIIL rule learner with partial instance memory that works online, processes each new example in constant time, and takes drift into account. Similarly to AQ11–PM, our approach is not based on a global window policy. However, FACIL retains border examples that are rejected when changes in the target concept are detected. Experimental results show a satisfactory performance as a
high-speed data streams classification method. Our future work is oriented to drop irrelevant attributes, and recover dropped attributes turned relevant later.

References

Fig. 3. Number of border examples per rule as a function of the number of examples.

Fig. 4. Number of rules as a function of the number of attributes.

5. R. Jin and G. Agrawal. Efficient decision tree construction on streaming data. In Proc. of the 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining - KDD’03.