Incremental Rule Learning and Border Examples Selection from Numerical Data Streams

Francisco J. Ferrer–Troyano Jesús S. Aguilar–Ruiz José C. Riquelme
Computer Science Dept., Univ. of Seville, 41012 Sevilla, Spain
{ferrer, aguilar, riquelme}@lsi.us.es

Abstract: Mining data streams is a challenging task that requires online systems based on incremental learning approaches. This paper describes a classification system based on decision rules that may store up–to–date border examples to avoid unnecessary revisions when virtual drifts are present in data. Consistent rules classify new test examples by covering and inconsistent rules classify them by distance as the nearest neighbour 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.

Key Words: Classification, decision rules, incremental learning, concept drift, data streams

Category: H.2.8, I.2.6, I.5.2

1 Introduction

Classification and rule learning are important, well–studied tasks in machine learning and data mining. In order to classify and model large–scale databases, important works have been recently addressed to scale up inductive classifiers and learning algorithms [3, 16]. However, a growing number of emerging business and scientific applications, where high–rate streams of detailed data are constantly generated, is frequently challenging the scalability of such methods. Examples of such data streams include networks event logs, telecommunications records, and financial and retail chain transactions. Applications of such streams include credit card fraud protection, target marketing, and intrusion detection, for which it is not possible to collect all relevant input data before applying the learning process. Under these circumstances, KDD systems have to operate continuously - online - and process each item in real–time [4]. In these environments, memory and time limitations make multi–pass scalable algorithms unfeasible, since data are received at a higher rate than they can be repeatedly analyzed. Furthermore, real–world data streams are not generated in stationary environments, requiring incremental learning approaches to track trends and adapt to changes in the target concept.

This paper describes FACIL[1], an incremental learning algorithm that provides a set of decision rules induced from numerical data streams. Our proposal extends

[1] **F**ast and **A**daptive **C**lassifier by **I**ncremental **L**earning

previous work [1] by filtering the examples that lie near to decision boundaries, so that every rule may retain a particular set of positive and negative examples. This information makes possible to ignore false alarms with respect to virtual drifts and avoid hasty modifications.

Paper Organization. The rest of 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

In the problem of classification, an input data set of training examples $T = \{e_1, \ldots, e_n\}$ is given. Every training example $e_i = (\overrightarrow{x_i}, y_i)$ is a pair formed by a vector $\overrightarrow{x_i}$ and a discrete value y_i, named class label and taken of a finite set Y. Every vector $\overrightarrow{x_i}$ 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 $y = f(\overrightarrow{x})$, 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.

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 (e.g., time series). Despite online learning systems continuously review, update, and improve the model, not every online system is based on an incremental approach. According to the taxonomy in [13], if $T_t = \{(\overrightarrow{x}, y) : y = f(\overrightarrow{x})\}$ for $t =< 1, \ldots, \infty >$, 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.

Along with the ordering effects, incremental learning from real-world domains faces two problems known as *hidden context* and *concept drift*, respectively [20]. The problem of hidden context is when the target concept may depend on unknown variables, which are not given as explicit attributes. In addition, hidden contexts may be expected to recur due to cyclic or regular phenomena (aka *recurring contexts*) [5]. The problem of concept drift is when changes in the hidden context induce changes in the target concept. In general, two kinds of concept

drift depending on the rate of the changes are distinguished in the literature: sudden (abrupt) and gradual. 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 [20]. In [14] virtual concept drift is referred to as sampling shift, and real concept drift is referred to as concept shift.

Formally, in [10] concept drift is defined in terms of consistency and persistence. Consistency refers to the change $\epsilon_t = \theta_t - \theta_{t-1}$ that occurs between consecutive examples of the target concept from time $t-1$ to t, with θ_t being the state of the target function in time t. A concept is *consistent* if ϵ_t is smaller or equal than a consistency threshold ϵ_c. A concept is *persistent* if it is consistent during p times, where $p \geq \frac{w}{2}$ and w is the size of the window. The drift is therefore considered *permanent* (real) if it is both consistent and persistent. Virtual drift is consistent but it is not persistent. Noise has neither consistency nor persistence. In practice, the output model needs to be updated independently the concept drift is real or virtual.

Above problems make incremental learning be more complex than batch learning, so effective learners should be able to distinguish noise from actual concept drift and quickly adapt the model to new target concept or recurring contexts. There are two common approaches that can be applied altogether to detect changes in the target concept [8]. 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. In [10] problems with this approach are studied and an unsupervised algorithm that uses three windows of different sizes is proposed. Another approach 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.

Formally, a data stream is an ordered sequence of data items $\ldots e_i \ldots$ 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 [4]. Thus, data streams contexts compel to learning systems to give approximate answers using small and constant time per example [6]. Recent works on data streams classification has been mainly addressed by two different approaches: decision trees [2, 6, 7] and ensemble methods [9, 17, 19].

Domingos & Hulten's VFDT and CVFDT systems [6] 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 [7] a numerical interval pruning approach to reduce the processing time for

numerical attributes, without loss in accuracy. Gama et al.'s VFDTc system [2] extends the VFDT properties in two directions: the ability to deal with numerical attributes and the ability to apply nave 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 [17] 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. [19] propose using ensemble of classifiers weighted based on their expected classification accuracy on the test examples. In [9] Kolter & Maloof propose DWM, an ensemble method based on the Weighted Majority algorithm [11].

As pointed out in [19], 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. A decision rule is a logic predicate of the form *if antecedent then label*. The antecedent is a conjunction of conditions of the form Attribute$\models$Values, and $\models$ is a operator that states a relation between a particular attribute and values of its domain. Within rule learning, each training example is said a maximally specific rule. Contrary to partitions obtained with decision tree based approaches, the regions given by decision rules do not model the whole space. Thus, new test examples may not satisfy - be covered by - any rule.

Fundamental incremental rule learners include STAGGER [15] (the first system designed expressly for coping with concept drift), the FLORA family of algorithms [20] (with FLORA3 being the first system able to deal with recurring contexts), and the AQ–PM family [13]. 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.

3 Border Examples inside Rules

The core of our approach is that rules may be inconsistent by storing positive and negative examples which are very near one another (border examples). A rule is said consistent when does not cover any negative (different label) example. The aim is to seize border examples up to a threshold is reached. This threshold is given as an user parameter and sets the minimum purity of a rule. 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 is reached, the examples associated with the rule are used to generate new positive and negative consistent rules. This approach is similar to the AQ11–PM system [12, 13], 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).

Our approach differs from AQ11–PM in that a rule stores two positive example per negative example covered. The stored examples are not necessary extreme and the rules are not repaired every time they become inconsistent, reducing the computational complexity. Since the number ne of negative examples that a rule can store increases as the number of covered positive examples does, every time ne increases by one unit, a new positive example is stored. Although this approach suffers the ordering effects, it does not compromise the learning efficiency and guarantees that an impure rule is always modified from as positive as negative examples.

3.1 Moderate Generalization

Henceforth, the next notation is used to describe our proposal. Let m be the number of numerical attributes. Let $Y = \{y_1, \ldots, y_z\}$ be the set of class labels. Let $e_i = (\overrightarrow{x_i}, y_i)$ be the new i^{th} training example arriving, where $\overrightarrow{x_i}$ is a normalized vector in $[0, 1]^m$ and y_i is a discrete value in Y. A decision rule r is given by a set of m closed intervals $[I_{jl}, I_{ju}]$ ($j \in \{1, \ldots, m\}$) which define an hyper–rectangle inside the space. l denotes lower bound and u upper bound. Rules are stored in different sets according to the associated label. Since no global training window is used but each rule handles a different set of examples (a window per rule), every time a new example arrives the model is updated. In this process, one of three tasks is at least 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 $y' \neq y_i$.

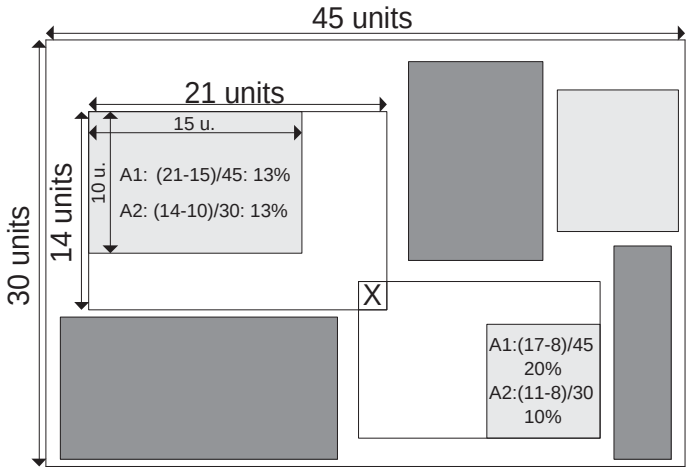

Figure 1: Moderate generalization prevents any rule can be selected as *candidate* to describe a new example(κ =10%).

3. ***New description***: x_i is not covered by any rule in the model.

Positive covering. First, the rules associated with y_i are visited and the generalization necessary to describe the new example x_i is measured according to definition 1.

Definition 1 (Growth of a rule) *Let r be a rule in $[0,1]^m$ formed by m closed intervals $[I_{jl}, I_{ju}]$. Let x be a point in $[0,1]^m$. The growth $\mathcal{G}(r, x_i)$ of the rule r to cover the point x_i is defined as:*

$$\mathcal{G}(r, x_i) = \sum_{j=1}^{m}(g_j - r_j);$$
$$g_j = u_j - l_j;\ r_j = I_{ju} - I_{jl};$$
$$u_j = \max(x_{ij}, I_{ju});\ l_j = \min(x_{ij}, I_{jl});$$

This heuristic gives a rough estimate of the new region of the search space that is taken, biasing in favour of the rule that involves the smallest changes in the minimum number of attributes. While visiting the rules associated with y_i, the one with the minimum growth is marked as *candidate*. However, a rule is taken into account as a possible candidate only if the new example can be seized with a moderate growth, so that:

$$\forall\ j \in \{1, \ldots, m\} :\ g_j - r_j \leq \kappa;\ \kappa \in (0, 1]$$

Figure 1 shows an example in which two rules do not satisfy this condition in one attribute using $\kappa = 0.1$. Since every example is previously normalized in $[0,1]^m$, the divisor factor for the domain of each attribute is omitted in definition

1. 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 a rule associated to y_i, then the rest of rules associated with a label $y' \neq y_i$ are visited. If a different label rule r' does not cover x_i, the intersection between r' and the candidate is computed. If such a intersection is not empty, the candidate is rejected. When the first different label rule r'' covering x_i is found, its negative support is increased by one unit, and x_i is added to its window. If the new purity of r'' is smaller than the minimum given by the user, then new consistent rules according to the examples in its window are included in the model. r'' is marked as *unreliable* so that it can not be generalized and has 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 $y' \neq y_i$. If no rule covers the new example and there is not a candidate that can be generalized to cover it, then a maximally specific rule to describe it is generated.

3.2 Refining and Forgetting Heuristic

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 extended rule associated with 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 support.

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.

Table 1: Means and standard deviations of prediction accuracy, learning time in seconds, and number of rules obtained by C4.5Rules from twelve UCI databases with numerical attributes.

	P. Accuracy		L. Time		Number of rules	
Database	mean	sd	mean	sd	mean	sd
Balance	83.17	3.97	0.07	0.04	38.08	3.81
Breast Cancer	94.69	2.51	0.03	0.02	9.93	1.96
Glass	68.75	10.6	0.04	0.02	15.25	1.56
Heart Statlog	77.33	7.81	0.05	0.02	17.61	2.41
Ionosphere	90.83	4.66	0.16	0.05	7.53	1.69
Iris	94.20	5.25	0.00	0.01	3.85	1.36
P–Diabetes	73.45	4.51	0.06	0.02	7.51	1.53
Sonar	77.40	9.43	0.15	0.04	7.54	0.96
Vehicle	72.21	3.61	0.39	0.09	33.26	4.16
Vowel	77.67	4.60	1.67	0.20	66.90	6.48
Waveform	77.62	1.77	22.46	1.85	87.66	7.40
Wine	92.04	6.07	0.01	0.01	4.46	0.59
Average	81.61	5.40	2.10	0.20	24.96	2.83

In worst case, a new example involves a new description, visiting therefore every rule in each set. The computational complexity associated with this case is $\mathcal{O}(m \cdot s \cdot \overline{e})$, with m being the number of attributes and s as the model size or total number of rules. $\overline{e}$ estimates the average number of examples per rule.

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.

4 Empirical Evaluation

Although the STAGGER concepts [15] provide a standard benchmark of tracking the drift from examples with symbolic attributes, data streams classifiers so far lacks a standard experimental method to evaluate them with numerical attributes. In [6, 19] both robustness and reliability of incremental classifiers are evaluated using synthetic data streams generated from a moving hyperplane.

Table 2: Means and standard deviations of prediction accuracy, learning time in seconds, and number of rules obtained by FACIL from twelve UCI databases with numerical attributes.

	P. Accuracy			L. Time			Number of rules		
Database	mean	b/w	sd	mean	b/w	sd	mean	b/w	sd
Balance	95.18	⋆	2.07	0.008	⋆	0.008	18.36	⋆	7.79
Breast Cancer	95.50		4.66	0.007	⋆	0.007	8.10		1.68
Glass	75.89	⋆	2.31	0.003	⋆	0.006	14.71		1.67
Heart Statlog	78.59		2.12	0.010	⋆	0.007	16.59		1.31
Ionosphere	90.31		2.34	0.013	⋆	0.005	7.75		1.20
Iris	98.10	⋆	0.60	0.000		0.000	3.00		0.00
P–Diabetes	90.42	⋆	3.59	0.014	⋆	0.006	8.51		0.99
Sonar	75.04		2.43	0.019	⋆	0.007	8.57	•	1.04
Vehicle	76.28	⋆	3.85	0.050	⋆	0.008	33.89		2.77
Vowel	77.53		3.91	0.060	⋆	0.006	65.90		0.30
Waveform	84.77	⋆	7.92	1.220	⋆	0.070	40.30	⋆	11.15
Wine	95.05		0.93	0.002	⋆	0.005	7.70	•	1.24
Average	86.11	⋆	3.09	0.12	⋆	0.01	19.43	⋆	2.59

In [18] an framework for incremental learning with SVMs is proposed and two incremental variants of the cross–validation experimental method are presented to evaluate them using real databases available at the UCI repository. The problem here is that both methods are designed to evaluate BBL algorithms (*block by block learning*). FACIL is based on *instance by instance learning* (IIL) where the algorithm does not wait for receive a block of examples - or to complete the window - to update the model, but every time a new example arrives it is processed online. Precisely, that is why standard cross validation can be applied to evaluate (IIL) learning algorithms similarly to *multi–pass* methods so that one–pass processing of the training examples in a sequential manner according to the order they arrive is enough.

Similarly to [18], we also evaluate our algorithm as general purpose classifier using 10–folds cross validation. Both experiments were conducted on a PC with CPU 1.7GHz and 512 MB of RAM running Windows XP.

4.1 Real databases from the UCI repository

In this set of experiments, concept drift is not present in data and all attributes are numerical. For the sake of comparison, decision lists generated by C4.5Rules are included. For every database, 10–fold cross validation is repeated ten times

Table 3: Number of examples per rule and maximum growth in FACIL according to Table 2.

	Number of examples		Maximum growth
Database	mean	sd	κ
Balance Scale	8.37	16.35	100
Breast Cancer	3.26	4.58	60
Glass	2.99	7.37	40
Heart Statlog	1.72	2.41	40
Ionosphere	2.68	5.36	75
Iris	3.75	2.25	80
Pima Diabetes	16.74	18.15	75
Sonar	1.59	2.31	40
Vehicle	3.67	21.98	55
Vowel	2.19	4.1	100
Waveform	10.79	52.31	35
Wine	2.01	2.5	80
Average	5.03	11.84	64.17

shuffling the examples so that they are ordered randomly each time. Since noise is not present in data, the minimum purity per rule is set with an user parameter from the prediction accuracy obtained by C4.5Rules with values between 80% and 100%. The maximum growth is increased as prediction accuracy does, being it fixed when the number of rules surpasses the size of the model provided by C4.5Rules. Tables 1 and 2 show the average values and standard deviations for one hundred executions. The number of examples per rule and the maximum growth are showed in Table 3. Values about accuracy, learning time, and number of rules that are marked with $\star$ or $\bullet$ in Columns b/w - better/worse - involve an improvement or loss respectively according to t–student with significance $\alpha = 0.05$.

In six databases (*Balance–Scale, Glass, Iris, Pima–Diabetes, Vehicle*, and *Waveform*) FACIL obtained a significant increase with respect to prediction accuracy. This improvement excels in *Pima–Diabetes* database, for which the average value in accuracy (90.42%) exceeds almost 17 units in comparison to C4.5Rules (73.73%), that is, an improvement greater than 23% using only one more rule. In addition, model complexity and prediction accuracy are significantly improved at once in two databases. In *Balance–Scale* database, the reduction of the number of rules is greater than 50%, gaining 12 units in prediction accuracy (more than 14%). In *Waveform* database, for which the model size is reduced to 46% with respect to C4.5Rules, the accuracy surpasses 7 units,

that is, an improvement greater than 9% using 47 fewer rules. Since FACIL is a *single–pas* algorithm and C4.5Rules is *multi–pass*, learning time is improved in all the studied databases. In this respect, FACIL spent only 1.22 seconds to process *Waveform* database, whereas C4.5Rules exceeded 22 seconds, which involves an improvement greater than 1700%.

On the other hand, the last row in Table 3 shows that the number of examples stored in memory is small and a moderate generalization generally improves the global accuracy in different domains.

4.2 Moving Hyperplane

In a second experiment, we create synthetic data with drifting concepts based on a moving hyperplane as in [6, 19]. A hyperplane in m–dimensional space is denoted by equation:

$$\sum_{i=1}^{m} a_i x_i = a_0$$

First, examples are randomly generated and uniformly distributed in multi-dimensional 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.

As in [19], 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 [19], 40% dimensions' weights are changing at ± 0.10 per 10000 examples. Tables 4–7 show the results with both implicit and explicit forgetting after 100000 examples are processed. In both cases, minimum purity is 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.

Column Time shows the time in seconds spent on building the model and classifying new test examples. The final number of rules is indicated in Column Size. Since running time depends on the number of rules, this factor is alternately limited to 50 and 100 rules per label. The goal here is evaluate the computational

Table 4: Prediction accuracy, time in seconds, and number of rules obtained by FACIL using explicit forgetting, maximum growth $\kappa = 50\%$, and maximum number of rules equal to 50.

Attributes	Accuracy	Time	Size
10	96.36	27	10
20	93.25	125	9
30	91.04	248	3
40	89.70	440	2
50	83.88	553	4

Table 5: Prediction accuracy, time in seconds, and number of rules obtained by FACIL using implicit forgetting, maximum growth $\kappa = 50\%$, and maximum number of rules equal to 100.

Attributes	Accuracy	Time	Size
10	84.61	82	153
20	67.13	214	150
30	63.17	276	125
40	59.65	759	22
50	59.50	851	9

cost as a function of the number of attributes. In general, explicit forgetting heuristics provides a performance significantly higher than the implicit one. Average explicit 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:

- 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%.

5 Conclusions and Future Work

In this paper, we described and evaluated FACIL, an incremental rule learner with partial instance memory based on moderate generalization and example

Table 6: Prediction accuracy, time in seconds, and number of rules obtained by FACIL using explicit forgetting, maximum growth $\kappa = 75\%$, and maximum number of rules equal to 100.

Attributes	Accuracy	Time	Size
10	98.32	40	7
20	94.57	70	12
30	89.26	106	10
40	89.19	135	7
50	87.72	155	10

Table 7: Prediction accuracy, time in seconds, and number of rules obtained by FACIL using implicit forgetting, maximum growth $\kappa = 75\%$, and maximum number of rules equal to 50.

Attributes	Accuracy	Time	Size
10	87.93	70	29
20	55.61	262	27
30	53.48	248	53
40	52.25	469	65
50	51.21	495	36

nearness. Similarly to AQ–PM, our proposal is not based on a window policy but examples are rejected when they do not describe a decision boundary. On the contrary, FACIL builds and refines inconsistent rules simultaneously without adversely affecting the learning efficiency and avoiding unnecessary revisions when virtual drifts are present in data. Experimental results show an excellent performance of our approach as a general purpose classification method.

Our future research directions are oriented to drop irrelevant dimensions, and recover dropped attributes turned relevant later. We are also evaluating alternative growth measures for tackling with symbolic attributes in order to compare our system with others data stream classifiers as CVFDT [6] and VFDTc [2].

References

1. F. Ferrer-Troyano, J. Aguilar-Ruiz, and J. Riquelme. Discovering decision rules from numerical data streams. In *Proc. of the* 19^{th} *ACM Symposium on Applied Computing - SAC'04*, pages 649–653.

2. 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.
3. J. Gehrke, R. Ramakrishnan, and V. Ganti. Rainforest – a framework for fast decision tree construction of large datasets. In *Proc. of the* 24^{th} *Int. Conf. on Very Large Data Bases – VLDB'98*, pages 416–427, 1998.
4. L. Golab and M. Ozsu. Issues in data stream management. *SIGMOD Record*, 32(2):5–14, 2003.
5. M. Harries, C. Sammut, and K. Horn. Extracting hidden context. *Machine Learning*, 32(2):101–126, 1998.
6. 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.
7. 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*.
8. R. Klinkenberg. Learning drifting concepts: example selection vs. example weighting. *Intelligent Data Analysis, Special Issue on Incremental Learning Systems Capable of Dealing with Concept Drift*, 8(3), 2004.
9. J. Z. Kolter and M. Maloof. Dynamic weighted majority: A new ensemble method for tracking concept drift. In *Proc. of the* 3^{th} *IEEE Int. Conf. on Data Mining - ICDM'03*, pages 123–130, 2003.
10. M. Lazarescu, S. Venkatesh, and H. Bui. Using multiple windows to track concept drift. Technical report, Faculty of Computer Science, Curtin University, 2003.
11. N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994.
12. M. Maloof. Incremental rule learning with partial instance memory for changing concepts. In *Proc. of the* 15^{th} *IEEE Int. Joint Conf. on Neural Networks - IJCNN'03*, pages 2764–2769, 2003.
13. M. Maloof and R. Michalski. Incremental learning with partial instance memory. *Artificial Intelligence*, 154:95–126, 2004.
14. M. Salganicoff. Tolerating concept and sampling shift in lazy learning using prediction error context switching. *AI Review, Special Issue on Lazy Learning*, 11(1-5):133–155, 1997.
15. J. Schlimmer and R. Granger. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, 1986.
16. J. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proc. of the* 22^{th} *Int. Conf. on Very Large Databases – VLDB'96*, pages 544–555, 1996.
17. W. Street and Y. Kim. A streaming ensemble algorithm SEA for large-scale classification. In *Proc. of the* 7^{th} *ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining - KDD'01*, pages 377–382.
18. N. Syed, H. Liu, and K. Sung. Handling concept drifts in incremental learning with support vector machines. In *Proc. of the* 5^{th} *ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining - KDD'99*, pages 272–276. ACM Press, 1999.
19. H. Wang, W. Fan, P. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proc. of the* 9^{th} *ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining - KDD'03*, pages 226–235.
20. G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.