eRules: A Modular Adaptive Classification Rule Learning Algorithm for Data Streams

Frederic Stahl, Mohamed Medhat Gaber and Manuel Martin Salvador

Abstract Advances in hard and software in the past decade allow to capture, record and process fast data streams at a large scale. The research area of data stream mining has emerged as a consequence from these advances in order to cope with the real time analyis of potentially large and changing data streams. Examples of data streams include Goole searches, credit card transactions, telemetric data and data of continuous chemical production processes. In some cases the data can be processed in batches by traditional data mining approaches. However, in some applications it is required to analyise the data in real time as soon as it is being captured. Such cases are for example if the data stream is infinite, fast changing, or simply to large in size to be stored. One of the most important data mining techniques on data streams is classification. This involves training the classifier on the data stream in real time and adapting it to concept drifts. Most data stream classifiers are based on decision trees. However, it is well known in the data mining community that there is no single optimal algorithm. An algorithm may work well on one or several datasets but badly on others. This paper introduces *eRules*, a new rule based adaptive classifier for data streams, based on an **e**volving set of **Rules**. *eRules* induces a set of rules that is constantly evaluated and adapted to changes in the data stream by adding new and removing old rules. It is different from the more popular decision tree based classifiers as it tends to leave data instances rather unclassified than forcing a classification that could be wrong.

Frederic Stahl
Bournemouth University, School of Design, Engineering and Computing, Poole House, Talbot Campus,BH12 5BB e-mail: fstahl@bournemouth.ac.uk

Mohamed Medhat Gaber
University of Portsmouth, School of Computing, Buckingham Building, Lion Terrace, PO1 3HE e-mail: Mohamed.Gaber@port.ac.uk

Manuel Martin Salvador
Bournemouth University, School of Design, Engineering and Computing, Poole House, Talbot Campus,BH12 5BB e-mail: msalvador@bournemouth.ac.uk

1 Introduction

According to [13] and [2] streaming data is defined as high speed data instances/records that challenge our computational capabilities for traditional processing. It has greatly contributed to the recent problem of analysis of *Big Data*. Analysing these data streams can produce an important source of information for decision making in real time. In the past decade, many data stream mining techniques have been proposed, for a recent review and categorisation of data stream mining techniques the reader is referred to [12].

Several approaches to adapt existing batch learning data mining techniques to data streams exist, such as reservoir sampling [15] and sliding window [2, 9]. The basic idea of reservoir sampling is to maintain a representative unbiased sample of the data stream without prior knowledge of the total number of data stream instances. Data mining techniques are then applied on the reservoir sample. The basic idea of sliding windows is to only use recent data instances from the stream to build the data mining models.

Among the many techniques proposed for data stream mining, a notable group of these algorithms are the *Hoeffding bound* based techniques [11]. The *Hoeffding bound* is a statistical upper bound on the probability that the sum of random variables deviates from its expected value. The basic *Hoeffding bound* has been extended and adopted in successfully developing a number of classification and clustering techniques that were termed collectively as *Very Fast Machine Learning* (VFML). Despite its notable success, it faces a real problem when constructing the classification model. If there are ties of attributes and the *Hoeffding bound* is not satisfied, the algorithm tends to increase the sample size. This may not be desirable in a highly dynamic environment. The more flexible approach in addressing this problem is using the sliding window.

In this paper, we have used the sliding window to extend the rule-based technique *PRISM* [8] to function in the streaming environment. Motivated by the simplicity of *PRISM* and its explanatory power being a rule-based technique, we have developed and experimentally validated a novel data stream classification technique, termed *eRules*. The new technique is also able to adapt to concept drift, which is a well known problem in data stream classification. A concept drift occurs when the current data mining model is no longer valid, because of the change in the distribution of the streaming data. Our *eRules* classifier works on developing an initial batch model from a stored set of the data streams. This process is followed by an incremental approach for updating the model to concept drift. The model is reconstructed if the current model is unable to classify a high percentage of the incoming instances in the stream. This in fact may be due to a big concept drift. Although this may appear to be a drawback of our technique, it is one of its important strengths, as many other techniques fail to adapt to big drift in the concept when an incremental update is only used.

The aim of this work is to contribute an alternative method for the data stream classification, as it is a well-known observation that established algorithms may work well on one or several datasets, but may fail to provide high performance on

others. In fact eRules is being considered by the *INFER* project [1] to be included into its predictive methods toolbox. INFER software is an evolving data mining system that is able to handle data streams. INFER uses a pool of possible data mining and pre-processing methods including data stream classifiers and evolves in order to find the optimal combination and parameter setting for the current prediction task [16].

The paper is organised as follows. Section 2 gives the background on learning modular rules represented in this paper by the *Prism* algorithm. Our extension of *Prism* algorithm enabling it to function in the streaming environment, developing what we have termed as *eRules* technique, is detailed in Section 3. We have experimentally validated our proposed technique in Section 4. In Section 5, we discuss our ongoing and future work related to the *eRules* technique. Finally, the paper is concluded by a summary in Section 6.

2 Learning Modular Classification Rules

There are two main approaches to the representation of classification rules, the 'separate and conquer' approach, and the 'divide and conquer' approach, which is well known as the top down induction of decision trees. 'Divide and conquer' induces rules in the intermediate form decision trees such as the C4.5 classifier [18], and'separate and conquer' induces IF...THEN rules directly from the training data. Cendrowska claims that decision tree induction grows needlessly large and complex trees and proposes the Prism algorithm, a 'separate and conquer' algorithm that can induce modular rules that do not necessarily fit into a decision tree [8] such as for example the two rules below:

IF a = 1 and b = 2 THEN class = x

IF a = 1 and d = 3 THEN class = x

We propose to base the rule induction of eRules on the Prism algorithm for several reasons. The classifier generated by Prism does not necessarily cover all possible data instances whereas decision tree based classifiers tend to force a classification. In certain applications, leaving a data instance unclassified rather than risking a false classification may be desired, for example, in medical applications or the control of production processes in chemical plants. Also Prism tends to achieve a better classification accuracy compared with decision tree based classifiers, if there is considerable noise in the data [5].

For continuous data the algorithm can be summarised as follows, assuming that there are n (> 1) possible classes [20].

```
For each class i from 1 to n inclusive:
(a) working dataset W = initial Dataset;
    delete all records that match the rules that have
    been derived so far for class i;
(b) For each attribute A in W
```

```
    - sort the data according to A;
    - for each possible split value v of attribute A
      calculate the probability that the class is i
      for both subsets A < v and A >= v;
(c) Select the attribute that has the subset S with
    the overall highest probability;
(d) build a rule term describing S;
(e) W = S;
(f) Repeat b to e until the dataset contains only
    records of class i. The induced rule is then
    the conjunction of all the rule terms built at
    step d;
(g) Repeat a to f until all records of class i have
    been removed;
```

Even so Prism has been shown to be less vulnerable to overfitting compared with decision trees, it is not immune. Hence pruning methods for Prism have been developed such as J-pruning [6] in order to make Prism generalising better on the input data. Because of J-pruning's generalisation capabilities we use Prism with J-pruning for the induction of rules in eRules hence it is briefly described here. J-pruning is based on the J-measure which according to Smyth and Goodman [19] is the average information content of a rule *IF Y = y THEN X = x* and can be quantified as:

$$J(X;Y=y) = p(y) \cdot j(X;Y=y) \tag{1}$$

The first factor of the J-measure is *p(y)*, which is the probability that the antecedent of the rule will occur. The second factor is the cross entropy *j(X;Y=y)*, which measures the goodness-of-fit of a rule and is defined by:

$$j(X;Y=y) = p(x \mid y) \cdot log_2(\frac{p(x \mid y)}{p(x)}) + (1 - p(x \mid y)) \cdot log_2(\frac{(1 - p(x \mid y))}{(1 - p(x))}) \tag{2}$$

The basic interpretation of the J-value is that a rule with a high J-value also achieves a high predictive accuracy. If inducing further rule terms for the current rule results in a higher J-value then the rule term is appended as it is expected to increase the current rules predictive accuracy; and if a rule term decreases the current rule's J-value then the rule term is discarded and the rule is considered as being in its final form. That is because appending the rule term is expected to decrease the current rule's predictive accuracy.

Having discussed the Prism algorithm for rule induction and the J-pruning for avoiding the model overfitting problem, the following section is devoted for a detailed discussion of our proposed technique *eRules* which is based on the concepts presented in this section.

3 Incremental Induction of Prism Classification Rules with eRules

Prism is a batch learning algorithm that requires having the entire training data available. However, often we do not have the full training data at hand. In fact the data is generated in real time and needs to be analysed in real time as well. So we can only look at a data instance once. It is one of the data stream mining techniques is to be sublinear in the number of instances. This implies that the algorithm may only have one look at any data instance, or even some of the instances are discarded all together.

Data mining algorithms that incrementally generate classifiers have been developed, the most notable being Hoeffding Trees [10]. Inducing a Hoeffding Tree is a two stage process, first the basic tree is induced and second, once the tree is induced, it is updated periodically in order to adapt to possible concept drifts in the data. A concept drift is if the pattern in the data stream changes.

However, incremental classifiers for data streams are based almost exclusively on decision trees, even though it has been shown that general classification rule induction algorithms such as Prism are more robust in many cases as mentioned in Section 2. Hence the development of an incremental version of Prism, *eRules*, may well produce better classification accuracy in certain cases.

The pseudocode below describes the basic *eRules* algorithm. The basic algorithm consists of three processes. The first process learns a classifier in batch mode on a subset of the data stream, this is implemented in the very first stage when the algorithm is executed. This is done in order to learn the initial classifier. This process may also be repeated at any stage when the 'unclassification rate' is too high. The unclassification rate is discussed in 3.1. The second process adds new rules to the classifier in order to adapt to a concept drift, this is implemented in the *IF(inst is NOT covered by rules)* statement. The third process validates whether the existing rules still achieve an acceptable classification accuracy and removes rules if necessary, this is implemented in the *ELSE IF (inst wrongly classified)* statement. Removing rules aims to 'unlearn'/forget old concepts that are not valid anymore and adding rules aims to learn a newly appearing concept.

The algorithm starts with the first process, collecting the first incoming data instances from the stream and induces an initial rule set in batch mode using the Prism algorithm. The classifier is reset and a new batch classifier is trained as soon as too many incoming test instances remain unclassified. How many data instances are used for the batch learning and how the unclassification rate is measured is discussed in Section 3.1. eRules uses each incoming labelled data instance in order to validate the existing rules and to update the classifier if necessary. If the incoming data instance is not covered by any of the rules, then the second process is invoked, i.e., the unclassified instance is added to a buffer containing the latest unclassified instances. If there are enough instances in this buffer then new rules are generated and added to the rule set, using the instances in the buffer. How one can measure if there are enough instances in the buffer is discussed in Section 3.2. In contrary, if

Algorithm 1 eRules algorithm

```
ruleset ← new RuleSet
buffer ← new DataBuffer
LEARNBATCHCLASSIFIER( )

while more instances available do
    inst ← take next data instance
    if inst is NOT covered by rules then
        BUFFER.ADD(inst)
        if buffer fulfils Hoeffding Bound OR alternative metric then
            learn a new set of rules from buffer and add them to ruleset
            BUFFER.CLEAR( )
        end if
    else if inst wrongly classified then
        update rule information and delete rule if necessary
    else if inst is correctly classified then
        update rule information
    end if

    if unclassification rate too high then
        LEARNBATCHCLASSIFIER( )
    end if
end while

procedure LEARNBATCHCLASSIFIER( )
    ruleset ← train classifier on n first
    incoming data instances
    buffer ← new DataBuffer
end procedure
```

the incoming data instance is covered by one of the rules then two cases are possible. The rule predicts the class label correctly and the concerning rule's accuracy is updated. In the second case the rule misclassifies the data instance, again the concerning rules classification accuracy is updated, and then removed if the accuracy is below a certain threshold. Section 3.3 discusses how such a threshold could be defined.

We have given an overview of our *eRules* technique in this section. The following subsections provide further details with respect to each of the three processes of *eRules*.

3.1 First Process: Learn Rules in Batch Mode

At the start of eRules' execution, or if the unclassification rate is too high, a set of rules is learned in batch mode using the Prism classifier. At the moment the eRules implementation allows the user to specify the number of data instances that are used to train the classifier in batch mode. By default the number of data instances is

50 as it seems to work well for all the experiments we conducted including those presented in this paper. However more dynamic methods could and will be used in future implementations, such as the Hoeffding bound, which is also used in the Hoeffding tree classifier [10] and other Hoeffding bound machine learning methods. The Hoeffding bound highlighted in the equation below is used with different variations as an upper bound for the loss of accuracy dependent on the number of instances in each iteration of a machine learning algorithm.

$$\varepsilon = \sqrt{\frac{R^2 ln(\frac{1}{\delta})}{2n}} \tag{3}$$

The Hoeffding bound could be used in eRules to determine the optimal size of the batch of instances that minimises the error rate. This will be investigated by the authors for our future work in this area.

A second issue that remains to be discussed for the first process is how to determine the unclassification rate. A window of the most recent stream data instances is defined by the user, for which the current classifier's accuracy and its unclassification rate are defined. By default this window size is 50, however, it can be specified by the user. If there are for example 10 unclassified instances then the unclassification rate would be $\frac{10}{50}$. More dynamic methods to define the window size is currently being investigated.

3.2 Second Process: Adding new Rules

As discussed earlier in this section, the second process adds new rules to the rule set if there are 'enough' unclassified instances. If there are enough unclassified instances, then eRules will learn new rules only on the unclassified data instances using the Prism classifier. It will then add the new rules to the existing rule set. A window of unclassified instances with a certain size of the most recent unclassified data instances is defined in order to determine the current unclassification rate of the classifier. The oldest unclassified instance is always replaced by the newest unclassified instance. What remains to be discussed is the window size. The smaller the window size, the larger the risk that not the full concept describing the unclassified instances is represented in the window, and hence not learned. The larger the window size the higher the risk that older, no more valid concepts encoded in the unclassified instances, are learned. By default the algorithm uses window size 20. What needs to be determined as well is the optimal number of unclassified instances to induce new rules from, at the moment this is by default 30 unclassified instances. However, a possible improvement that is currently being investigated is the usage of the Hoeffding bound to determine the optimal number of unclassified instances to justify the induction of additional rules.

3.3 Third Process: Validation and Removal of Existing Rules

The removal of existing rules is determined by the classification accuracy of the individual rule but also on a minimum number of classification attempts the rule needs to fulfil in order to be discarded. Again both criteria can be predefined by the user but are in the current implementation by default 0.8 as the minimum accuracy and 5 as the minimum number of classifications attempted. The reason for not only using the minimum classification accuracy is the fact that a rule would be already discarded the first time it attempts to classify a test instance and actually assigns the wrong classification. For example, assuming that there are 10 test instances and the rule would be applied on the first one, it would have at this stage a classification accuracy of 0, and hence would be removed even if all following test instances would be classified correctly by this rule. However, if the minimum of classification attempts is 10 before the rule can be removed, then the rule would have a classification accuracy of 0.9 after 10 attempts.

4 Evaluation

For the evaluation of eRules, three different stream generators have been used, the SEA, LED and Waveform generator. The three data streams used for experimentation can be typically found in the literature of concept drift detection. We provide the details of the three generators in the following.

1. **SEA Concepts Generator**: this artificial dataset, presented in [23], is formed by four data blocks with different concepts. Each instance has 3 numerical attributes and 1 binary class. Only the two first attributes are used for classification, and the third one is irrelevant. The classification function is $f_1 + f_2 =< \theta$, where f_1 and f_2 are the two first attributes and θ is a threshold that changes for each of the four blocks in this order: $9, 8, 7, 9.5$. SEA data stream is the consecutive join of SEA_9, SEA_8, SEA_7 and $SEA_{9.5}$, where each block has 12500 instances and there is a sudden drift between blocks. It has been used in [4] [17] [14].
2. **LED Generator**: this data stream [7] has 24 binary attributes, but 17 of them are irrelevant for the classification. The goal is to predict the digit shown in a 7 segment LED display where each attribute has 10% of probability of being reversed. Concept drift can be included indicating a number d of attributes that change. It has been used in [24] [4].
3. **Waveform Generator**: this data stream was also presented in [7], and the goal is to distinguish between 3 different classes of waves. Each wave is generated by a combination of 2 or 3 basic waves. There are two variants of this problem: wave21 with 21 numerical attributes with noise, and wave40 with 19 extra irrelevant attributes. Concept drifts can be included by swapping a number d of attributes. It has been used in [24] [4] [14].

Taking these definitions as a starting point, and with the help of MOA software (Massive Online Analysis) [3], we have generated several datasets for the experimentation.

A gradual concept drift has been included between data instances 450 and 550. The following parameters have been used for all experiments highlighted in this section: 50 instances to learn the initial classifier, an accuracy below 0.8 and a minimum of 5 classification attempts for each rule to be removed, a minimum of 30 unclassified instances have been accumulated before they are used to induce new rules and a window size of the last 20 data instances from which the unclassification rate and accuracy of the classifier are calculated.

Two versions of eRules have been evaluated, both use the parameters outlined above. The first version just tries to adapt to concept drifts by removing and adding new rules to the classifier. The second version re-induces the entire classifier, if the unclassification rate is too high (40% in the last 20 instances observed).

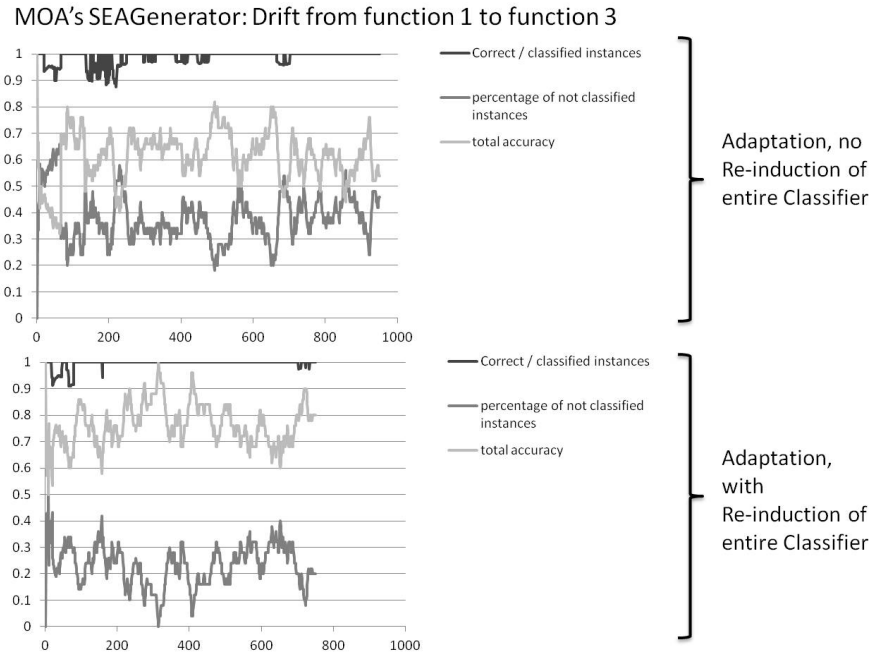

Fig. 1 Evaluation with SEAGenerator: The horizontal axises show the number of iterations (the time stamp of the data instance received) and the vertical axises show the accuracy and percentage of unclassified instances. The graph on the top shows how the algorithm behaves without re-induction of the entire classifier and the graph on the bottom shows how the algorithm behaves with re-induction of the entire classifier. The concept drift takes place between instances 450 and 550.

Figure 1 shows the unclassification rate, total accuracy and the accuracy on all instances for which eRules attempted a classification. It can be seen in both cases; eRules with and without re-induction, that there is a high unclassification rate and hence the total accuracy is relatively low. However, if we take only the classified instances into account it can be seen that the classification accuracy is relatively high. Also the concept drift is almost imperceptible. The unclassification rate for the version with re-induction is lower which in turn improves the total accuracy. However, the total classification accuracy for both versions of eRules is almost the same, only eRules with re-induction classifies more instances.

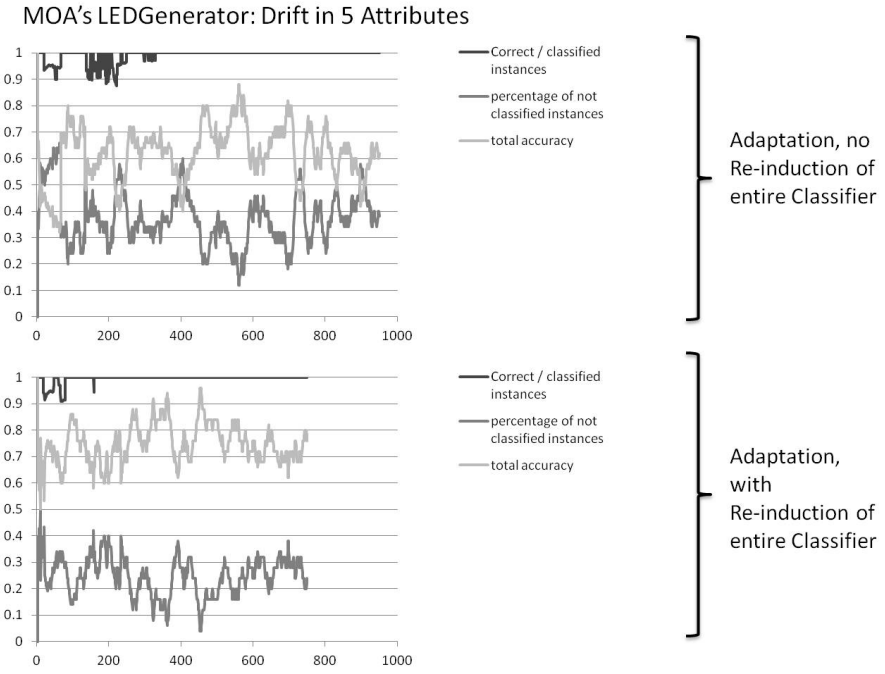

Fig. 2 Evaluation with LEDGenerator: The horizontal axises show the number of iterations (the time stamp of the data instance received) and the vertical axises show the accuracy and percentage of unclassified instances. The graph on the top shows how the algorithm behaves without re-induction of the entire classifier and the graph on the bottom shows how the algorithm behaves with re-induction of the entire classifier. The concept drift takes place between instances 450 and 550.

Figure 2 shows the unclassification rate, total accuracy and the accuracy on all instances for which eRules attempted a classification. eRules for the LED data stream exhibits a similar behaviour as for the SEA data stream. It can be seen in both cases; eRules with and without re-induction, that there is a high unclassification rate and hence the total accuracy is relatively low. However, if we take only the classified instances into account it can be seen that the classification accuracy is relatively high.

Also the concept drift is almost imperceptible. The unclassification rate for the version with re-induction is lower which in turn improves the total accuracy. However, the total classification accuracy for both versions of eRules is almost the same, only eRules with re-induction classifies more instances.

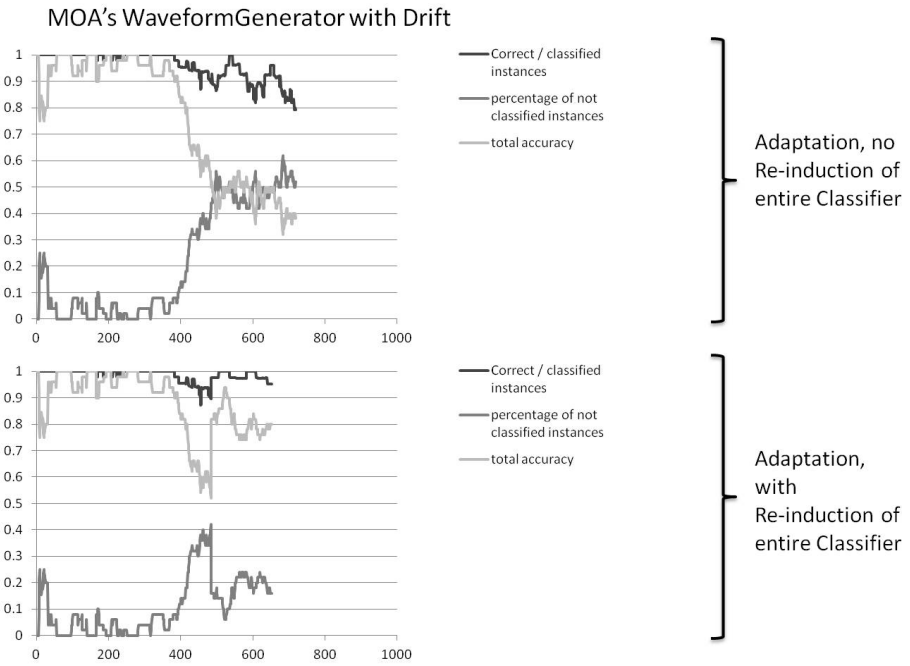

Fig. 3 Evaluation with WaveformGenerator: The horizontal axises show the number of iterations (the time stamp of the data instance received) and the vertical axises show the accuracy and percentage of unclassified instances. The graph on the top shows how the algorithm behaves without re-induction of the entire classifier and the graph on the bottom shows how the algorithm behaves with re-induction of the entire classifier. The concept drift takes place between instances 450 and 550.

Figure 3 shows the unclassification rate, the total accuracy and accuracy on all instances for which eRules attempted a classification. In this particular case, eRules does not cope well with the concept drift. For the version of eRules without re-induction the total classification accuracy drops considerably and the unclassification rate increases. However, if we take only the instances in consideration that have been assigned a class label, then the drop in classification accuracy is relatively low. This high classification accuracy whilst having a high unclassification rate can be explained by the fact that due to the concept drift bad performing rules are removed and the newly induced rules are not describing the unclassified instances entirely. The reason for this could be that simply 30 unclassified instances are not enough to generate adequate rules. This will be resolved in the future by using metrics such

as the Hoeffding bound in order to determine an adequate number of instances to induce new rules from. The version of eRules with re-induction performs better in this case as a completely new classifier is induced as soon as the unclassification rate reaches 40%. For the re-induction of the entire classifier the most recent 50 instances are used, which are 20 instances more compared with the induction of new rules covering unclassified cases. This again indicates that in this case more than 30 unclassified instances are needed for the induction of new rules.

In general it can be observed that eRules exhibits an excellent accuracy on the cases where classifications have been attempted. However, sometimes eRules may leave a large number of instances unclassified, because they are not covered by the existing rules. If the unclassification rate is too high, eRules reduces the unclassification rate by re-inducing the entire classifier. This strategy seems to have worked well in our experimental study.

As shown in the experiments, there are some improvements to eRules that are expected lead to a boost in its performance. These improvements are discussed in the following section.

5 Ongoing and Future Work

All evaluation experiments outlined in Section 4 have been performed with a fixed parameter setting that seemed to work well in most cases. However, as it has been observed in Section 4 for 3, there are cases in which the parameter setting could be improved. The ongoing work will comprise an investigation of metrics that could be used to dynamically adjust some of the parameters, for example the Hoeffding bound could be used to decrease the error by determining the optimal number of unclassified instances to be used to induce additional rules. Also the optimal size of the batch that is used to induce the initial classifier and to re-induce the entire classifier could be determined using the Hoeffding bound.

Further research will look into alternative implementations of the underlying modular classifier, with the aim to develop a rule induction method that delivers less unclassified cases. A possible candidate for that could be the PrismTCS classifier [6] which uses a default rule for unclassified instances.

Another possibility that will be investigated in future research is the usage of J-pruning. At the moment J-pruning is used for all rule induction processes in eRules, inducing the initial classifier in batch mode, re-induction of the entire classifier in batch mode and the induction of additional rules for the already existing classifier. However, J-pruning may not necessarily improve eRules's induction of additional classification rules. This is because the unclassified instances accumulated for inducing additional rules are polarised towards the concept that causes certain instances to be unclassified. Also further pruning facility for Prism algorithms, Jmax-pruning [22] will be investigated for its usage in the eRules system.

We also consider improving eRules's computational scalability by making use of parallel processing techniques. The underlying Prism batch classifier has been par-

allelised successfully in previous research [21], and hence indicates eRules potential to be parallelised as well. A parallelisation, and hence speed up of the rule adaption could enable eRules to be applicable on high speed data streams.

6 Conclusions

A novel data stream classifier, eRules, has been presented based on modular classification rule induction. Compared with decision tree based classifiers this approach may leave test instances unclassified rather than giving them a wrong classification. This feature of this approach is highly desired in critical applications. eRules is based on three basic processes. The first process learns a classifier in batch mode, this is done at the beginning of eRules's execution in order to induce an initial classifier. However, batch learning is also performed to re-induce the classifier if it leaves too many instances unclassified. The second and third processes are used to adapt the classifier on the fly to a possible concept drift. The second process removes bad performing rules from the classifier and the third process adds rules to the classifier to cover instances that are not classifiable by the current rule set.

A first version of the classifier has been evaluated on three standard data stream generators, and it has been observed that eRules in general delivers a high classification accuracy on the instances it attempted to classify. However, eRules also tends to leave many test instances unclassified, hence a version of eRules that re-induces the entire classifier, if the unclassification rate is too high has been implemented. eRules in general uses a fixed parameter setting, however, ongoing work on the classifier considers using the Hoeffding bound in order to dynamically adjust some of the parameters, while the algorithm is being executed.

Acknowledgements The research leading to these results has received funding from the European Commission within the Marie Curie Industry and Academia Partnerships & Pathways (IAPP) programme under grant agreement n^{o} 251617.

References

1. Computational intelligence platform for evolving and robust predictive systems, http://infer.eu/ 2012.
2. Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *In PODS*, pages 1–16, 2002.
3. Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *J. Mach. Learn. Res.*, 99:1601–1604, August 2010.
4. Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 139–148, New York, NY, USA, 2009. ACM.

5. M A Bramer. Automatic induction of classification rules from examples using N-Prism. In *Research and Development in Intelligent Systems XVI*, pages 99–121, Cambridge, 2000. Springer-Verlag.
6. M A Bramer. An information-theoretic approach to the pre-pruning of classification rules. In B Neumann M Musen and R Studer, editors, *Intelligent Information Processing*, pages 201–212. Kluwer, 2002.
7. Leo Breiman, Jerome Friedman, Charles J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Chapman & Hall/CRC, 1 edition, January 1984.
8. J. Cendrowska. PRISM: an algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4):349–370, 1987.
9. Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. In *ACM-SIAM Symposium on Discrete Algorithms (SODA 2002)*, 2002.
10. Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '00, pages 71–80, New York, NY, USA, 2000. ACM.
11. Pedro Domingos and Geoff Hulten. A general framework for mining massive data stream. *Journal of Computational and Graphical Statistics*, 12:2003, 2003.
12. Mohamed Medhat Gaber. Advances in data stream mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):79–85, 2012.
13. Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. Mining data streams: a review. *SIGMOD Rec.*, 34(2):18–26, 2005.
14. João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 329–338, New York, NY, USA, 2009. ACM.
15. Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.
16. Petr Kadlec and Bogdan Gabrys. Architecture for development of adaptive on-line prediction models. *Memetic Computing*, 1:241–269, 2009.
17. J. Zico Kolter and Marcus A. Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *J. Mach. Learn. Res.*, 8:2755–2790, December 2007.
18. Ross J Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.
19. P. Smyth and R M Goodman. An information theoretic approach to rule induction from databases. 4(4):301–316, 1992.
20. F. Stahl and M. Bramer. Towards a computationally efficient approach to modular classification rule induction. *Research and Development in Intelligent Systems XXIV*, pages 357–362, 2008.
21. F. Stahl and M. Bramer. Computationally efficient induction of classification rules with the pmcri and j-pmcri frameworks. *Knowledge-Based Systems*, 2012.
22. Frederic Stahl and Max Bramer. Jmax-pruning: A facility for the information theoretic pruning of modular classification rules. *Knowledge-Based Systems*, 29(0):12 – 19, 2012.
23. W. Nick Street and YongSeog Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '01, pages 377–382, New York, NY, USA, 2001. ACM.
24. Periasamy Vivekanandan and Raju Nedunchezhian. Mining data streams with concept drifts using genetic algorithm. *Artif. Intell. Rev.*, 36(3):163–178, October 2011.