Online Learning Classifiers in Dynamic Environments with Incomplete Feedback

Conference Paper · June 2013

DOI: 10.1109/CEC.2013.6557777

2 authors:

Mohammad Behdad
University of Western Australia
8 PUBLICATIONS **56** CITATIONS

SEE PROFILE

Tim French
University of Western Australia
84 PUBLICATIONS **489** CITATIONS

SEE PROFILE

2013 IEEE Congress on Evolutionary Computation
June 20-23, Cancún, México

Online Learning Classifiers in Dynamic Environments with Incomplete Feedback

Mohammad Behdad
School of Computer Science and Software Engineering
The University of Western Australia
35 Stirling Highway CRAWLEY WA 6009 Australia
Email: mbehdad@gmail.com

Tim French
School of Computer Science and Software Engineering
The University of Western Australia
35 Stirling Highway CRAWLEY WA 6009 Australia
Email: tim.french@uwa.edu.au

***Abstract*—In this paper we investigate the performance of XCSR (a real-valued genetics-based machine learning method) in an online environment in which the feedbacks are received with a delay and not for all the instances. The importance of such environments lies in the fact that many real world environments have these characteristics. For instance, in spam detection some of the undetected spam messages which are delivered to the user may be flagged as spam by user after a while. Hence, the feedback is both delayed and partial in this context. Similar situation can easily be imagined in other fraud detection contexts such as network intrusion and credit card fraud.**

We also present an architecture for an adaptable online XCSR and present two heuristics to deal with biased partial feedback environments. The heuristics use the information about the environment and their observations and create artificial feedbacks for the classifications that do not receive any feedback. We show that these heuristics always help XCSR learn better and perform more accurately in such situations.

I. Introduction

Learning in XCSR, a real-valued learning classifiers system, depends heavily on the feedbacks it receives for its classifications. Typically, research assures that the feedback is provided to XCSR immediately after each classification. In this paper, we investigate the performance of XCSR in real-world environments in which the feedback is incomplete. That is, it is provided with a delay —from classification to receiving the delay— and also not for all the classifications.

Such characteristics are of more importance in environments that have concept drift. Concept drift is when in an environment a target learning concept is changing over time. In practice, it means a classifier that is using its rule-set to classify the incoming instances, now finds its rule-set "old". It requires feedbacks for all its classifications as soon as possible up update its recently out-dated rule-set. However, these feedbacks are not provided for all of the classifications in most real-world cases and also they come with a delay.

In Section II, some research works that are related to the current paper are discussed. Then, in Section III the learning techniques used in this paper such as LCS, XCS, XCSR and PCA-XCSR are briefly explained. Enron processed email corpus that is the data set used in the experiments of this paper is introduced in Section IV. Next, in Section V we examine and confirm the existence of concept drift in our data set. Online XCSR framework in introduced in Section VI. Also, we find the optimum values for some key parameters of online PCA-XCSR for Enron processed data set in this section.

In Section VII the performance of online XCSR in delayed feedback situations is analysed. Then, we examine the behaviour of online XCSR when feedback is received stochastically in Section VIII. In Section IX, the concept of biased partial feedback is introduced in which feedback is not provided uniformly for all different types of classifications. That is, for some types of classifications, e.g. false negative ones, there is a higher chance of receiving a feedback than other types. Three heuristics are presented for dealing with such situations. Finally, Section X concludes the paper along with presenting some future directions for this research.

II. Related Works

Effect of delay in learning process has been studied before. Walsh et al. [1] investigate the constant delay between observation and reward and its impact on planning and learning. They propose a model-based simulation algorithm for planning in such environments and a model-based reinforcement learning to enable it to work in both finite and continuous environments. Their experiments show the success of the proposed method in mentioned situations.

Rahmandad et al. [2] examine the effect of delay between taking an action and observing the results in learning process. This time, the delay is not limited to the constant ones. They present four learning heuristics and analyse their performance in an experiment which attempts to solve resource allocation problem. They show that those heuristics work well when the delay magnitude is correctly calculated and it is either short or none. Otherwise, the performance drops significantly.

The former paper does not address the effect of delay when the delay changes, and the latter paper cannot perform well when the magnitude of delay is not known precisely. We present an architecture that handle both situations well. The reason is that in our architecture we do not store the classifications for future feedbacks. In this XCSR architecture, when a feedback is provided, the relevant instance is also provided alongside it, as these two pieces of information are all that XCSR needs to update its classifiers.

Finally, we mention two LCS related works. The first one is by Shafi and Abbass [3] in which a supervised version of LCS is enhanced in order to handle an online, noisy and imbalanced

environment more efficiently. The second paper is by Dam et al. [4] that investigates how XCS learns online in dynamic noisy environments with concept drift. It concludes that XCS can handle small changes in the data well and adapt quickly. The authors also propose some changes to XCS in order to make it more robust in such dynamic environments. However, the paper does not address incomplete or delayed feedback.

III. XCSR

LCSs (Learning classifier systems) are a group of genetics-based machine learning algorithms that use a population of condition-action-prediction rules called classifiers to encode appropriate actions for different inputs: if a condition is true, then performing the corresponding action is expected to lead to receiving the anticipated reward. The decision of which rule to use depends on reward the system expects to receive from the environment as a result of performing the chosen action.

XCS [5] is an accuracy-based LCS in which it is not the classifiers which produce the greatest rewards that have the best chance of propagating their genes into subsequent generations, but rather the classifiers which predict the reward more accurately, independent of the magnitude of the reward. When a state is perceived by an LCS, the rule-base is scanned and any rule whose condition matches the current state is added as a member of the current *match set* M. In XCS —specifically in exploit phases— once M has been formed, an action which has the highest weighted fitness prediction is chosen. All members of M that propose the same action as the chosen rule then form the *action set* A. The accuracy of all members of the action set are then updated when the reward from the environment is determined.

XCSR [6] is a variation of XCS [7] which can handle inputs having real-value features. In an Unordered Bound Representation (UBR) [8] encoding, the condition part of classifiers consists of a set if intervals. Each takes the form of (p_i, q_i) where either p_i or q_i can be the maximum or minimum bound. When XCSR receives an instance, if the values of the features of the instance are in the range of a classifier's condition, then that classifier is used in the classification process.

The performance of LCSs degrades when they are dealing with problems of high dimensionality [9], [10]. PCA-XCSR [11] is the method we use to improve the performance of XCSR in such high-dimensional data sets, both by using less time and computational resources.

PCA (Principal Component Analysis) is a feature transformation technique whose purpose is to find the most important information through dimensionality reduction. It analyses a set of original variables and represents them as a set of new orthogonal variables called principal components. These principal components are calculated using linear combinations of the original variables. The principal components are listed in the order of importance. That is, the first one will have the highest variance, the second one, which is orthogonal to the first, has the second largest variance, and so on [12]. Dimensionality reduction is achieved by using only a subset of the principal components (the first n) instead of using all of them.

In PCA-XCSR, PCA is used as a pre-processing step to compact the data set through dimensionality reduction. First, the data is normalised so that XCSR can handle every type of feature in the data correctly. Then, it is compacted (transformed) by PCA so that XCSR earns in a smaller search space —the space that results from selecting only a fixed number of dimensions (the n best principal components in the transformed space). The basis for doing this is that as these dimensions add the most variance to the input space, they will most accurately predict the class of each input.

In this paper, we assume that the environment is a binary class one. That is, every instance belongs to either positive or negative class. Therefore, every classification can be one of the four following types: (1) True positive (TP): an instance is correctly classified as positive. (2) False positive (FP): an instance is falsely classified as positive. (3) True negative (TN): an instance is correctly classified as negative. (4) False negative (FN): an instance is falsely classified as negative.

Two accuracy performance measures are used in the experiments of this paper. First one is the common accuracy which is the ratio of total instances classified correctly to all the classifications. The second one is TPR.TNR which is the product of true positive rate $TPR = \frac{TP}{TP+FN}$ and true negative rate $TNR = \frac{TN}{TN+FP}$. The reason for using it in addition to the common accuracy is that in data sets which are biased towards one class, for example in a data set that 90% of the instances belong to one class, if a system classifies everything as the majority class its accuracy will be 90% which seems as if the systems is doing a good job while it actually has not learned anything about the minority class which may be of higher importance.

IV. DATA

For the experiments in this paper we require a real-world data set which has concept drift in it. Enron corpus was selected for this reason. It contains email messages that belong to the senior managers of Enron corporation and were made public during the investigations of this corporation. The significance of this data set in addition to the large number of messages it contains, is the temporal aspect of it. Most messages have an attribute which shows the date they have been sent. This makes it possible to use this data for a close-to-reality online learning experiments with potential concept drift.

We used a random subset of a cleaned and labeled version of this corpus [13]. The final data set used in the experiments was processed because XCSR can only analyse and use instances that are in the form of real value vectors and Enron data consists of email messages in text format. So, we used SpamAssassin [14] which receives an email message, extract its important values and produces a vector of real values as its output. It was used to transform Enron messages into a CSV (comma-separated values) format file, in which each line corresponds to one email message and contains a set of real values. Next, the messages were sorted in ascending order on date and time so that the temporal aspect of the data is preserved. Finally, the date and time attributes of email messages were removed from the training and test files.

Total number of instances (email messages) in the final data set was $13,543$. There are $11,608$ spam messages in the data set (85.71% of the whole instances) and the remaining

$1,935$ messages are legitimate messages or hams (14.29%). The number of features for each instance is 712.

V. Concept Drift in the Data Set

As mentioned before, concept drift is present in an environment if the characteristics of the target learning concept is changing over time. Here we examine the existence of concept drift in our data set. It is done using an empirical approach by carrying out experiments comparing the performance of XCSR when it is trained only on the first half of spam and ham messages against training it on arbitrary messages.

In order to determine empirically the existence of concept drift in the Enron processed email data set, the following experiment was carried out. First, two scenarios were designed: In the first one, PCA-XCSR is trained on some of the instances coming from the early periods (the first half of the data set) and then tested on the instances coming later (the second half the data set). In the second scenario, PCA-XCSR is trained on email messages coming from any period (all the parts of the data set) and then tested on the same test messages as the first scenario. If PCA-XCSR performs better in the latter case, it means the messages (instances) have changed as time has passed, or in other words there is concept drift in the data set. The detailed steps of the experiment is explained below.

- "Enron processed email data set" which contains $13,543$ spam and ham messages, each sorted on date and merged evenly, is split into two halves: first (old) period and second (new) period.
- Training1 file is created by taking $5,000$ emails randomly from the instances in the first period
- Training2 file is created by taking $5,000$ emails randomly from the instances in both periods
- Test file is created by taking $3,000$ emails randomly from the instances in the second period. These emails should not be in Training2 —none of the instances in the Test file are seen in the training phase.
- In the first scenario, PCA-XCSR is trained using Training1 and tested on the Test file. The number of components is varied between 1 and 9 as the rule of thumb suggests that the optimum number of components is less than 10.
- In the second scenario, PCA-XCSR is trained using Training2 and tested on the Test file. The number of components for the PCA part is varied between 1 and 9.
- The previous two experiments are repeated 10 times and the average performances are compared.

Table I shows the result of these experiments. Based on these observations, best accuracies are achieved when three or four components are used. More importantly, scenario 2 always achieves better results. We conclude, when PCA-XCSR is trained on samples taken from the full period, it is able to perform (classify) better in comparison to when it is trained on samples taken from only the first period. It means a change in the data distribution from the first period to the second one and can be interpreted as a sign of concept drift in data.

TABLE I. Enron Concept Drift Experiment

Number of components	Scenario 1 TPR.TNR	Scenario 2 TPR.TNR
XCSR	6.92%	17.29%
1	14.39%	50.86%
2	37.25%	76.93%
3	93.04%	93.25%
4	91.19%	93.36%
5	80.92%	88.49%
6	75.95%	86.36%
7	74.12%	85.55%
8	74.18%	81.84%
9	64.16%	76.15%

VI. Online XCSR

Typically, the use of XCSR consists of two phases. The first phase is called the training phase during which the XCSR receives instances from training data set and learns the solution space through the feedbacks it receives. The second phase is called the test phase during which the performance of XCSR is evaluated on classification of instances from test data set (no feedbacks). Upon receiving each instance, XCSR has two options: explore or exploit. "Explore" entails selecting a random class for the instance when classifying, but "exploit" means selecting the best class according to its knowledge.

During training phase, in order to enhance its knowledge map, XCSR purely explores. That is, it assigns a random class to the instance it receives. Then it receives a feedback (reward) for its classification and based on that it updates the classifiers involved in that decision. Throughout this phase, the performance is not measured. However, in the next phase — the test phase— XCSR purely exploits: it selects the best class for each instance it receives. It does not receive any feedback and thus does not learn anything. The performance of XCSR is measured during this phase.

In this paper we present an architecture for XCSR called "online XCSR" which assumes an online environment. It attempts to incorporate realistic parameters of the feedback process. In online XCSR, instead of having two separate phases and two separate sets of instances, there is only one phase and one continuous set of instances. After receiving each instance, XCSR classifies it. Then, it receives a feedback for its decision and updates its rules accordingly. Next, it receives the next instance, classifies it, receives feedback and so on. Therefore, in online XCSR, training and testing occur in reverse order and constantly for every instance. To make things more complicated, in the real-world applications, feedback is not received for every classification and not immediately after each classification. The latter aspects will be discussed in Sections VII and VIII in detail.

As mentioned before, in non-online XCSR during the training phase solely "explore" is used and during the test phase only "exploit". What about online XCSR? If only "explore" is to be used, random classifications lead to a very poor performance and if only "exploit" is to be used learning rate and population diversity will be low and after a while XCSR may get stuck with a static population of classifiers. So, what is a good ratio? The hypothesis here is that a small fraction of decisions should use explore and the remaining ones exploit. This way, the accuracy performance of XCSR is kept high by mostly using exploit, while rare explore cycles

helps diversifying and maintaining a complete knowledge map of the solution space.

In the rest of this section, we carry two sets of experiments in order to find the optimum settings for an online XCSR for Enron processed email data set. As the data set is a high dimensional one, having 712 real-value features, it is expected that XCSR will not perform well on it and PCA-XCSR would be the suitable version of XCSR to use. We examine this theory and find the optimum number of components for PCA-XCSR in this setting in the first set of experiments. In the second set of experiments, we try to find the best explore probability value for online XCSR.

In the first set of experiments, online XCSR is used to classify the email messages of the Enron processed email messages. XCSR and PCA-XCSR (using different number of components ranging from 1 to 9) are used. A small value, that is 0.01, is used for explore probability and population size is limited to $10,000$. All the experiments are repeated 10 times and the average results are shown in Table II. The performance is measured mainly by TPR.TNR.

TABLE II. FINDING THE OPTIMUM NUMBER OF COMPONENTS

Number of Components	Accuracy	TPR	TNR	TPR.TNR
XCSR	37.17%	26.79%	99.40%	26.63%
1	91.53%	98.25%	51.23%	50.29%
2	94.05%	98.64%	66.52%	65.62%
3	96.77%	97.37%	93.15%	90.70%
4	96.12%	96.07%	96.37%	92.58%
5	91.23%	90.10%	97.97%	88.27%
6	89.80%	88.38%	98.34%	86.91%
7	89.46%	87.95%	98.49%	86.62%
8	83.88%	81.38%	98.84%	80.44%
9	79.22%	75.93%	98.90%	75.10%

As Table II shows, XCSR performs poorly and achieves only 27% TPR.TNR. Apparently its population of classifiers do not represent the solution model. One reason for its poor performance is predicted to be its need for huge population size. The preliminary experiments support this idea. But, PCA-XCSR especially when using 4 components performs very well and achieves TPR.TNR value of 92.98%.

Now that we have found the optimum number of components for the current data set, we are able to experiment on the effect of explore probability on the performance of online XCSR. Explore probability is a way to denote the ratio of explore trials versus exploit ones. If explore probability is equal to 0.01, then there is 1% chance of using explore versus 99% chance of using exploit for a given instance. different values of explore probability ranging from 0 which virtually is "only exploit", to 1 which means "only explore". Table III shows the results of these experiments.

TABLE III. EFFECT OF EXPLORE PROBABILITY VALUE ON THE PERFORMANCE OF ONLINE PCA-XCSR

Explore Probability	0	0.001	0.01	0.05	0.1	0.5	1
Median TPR.TNR	94%	94%	93%	90%	86%	54%	25%
Min TPR.TNR	70%	84%	88%	85%	84%	54%	24%
Max TPR.TNR	95%	96%	95%	92%	87%	56%	26%
Average TPR.TNR	92%	93%	93%	90%	86%	54%	25%

As Table III demonstrates, using a value between 0.01 and 0.001 for explore probability, PCA-XCSR achieves its highest accuracies (TPR.TNR value of 93%). Even, when we only exploit (explore probability is equal to 0), we get a good accuracy. However, in order to prevent formation of a static population, we will use the value 0.001 for explore probability from now on.

One thing worth mentioning here is a potentially better way of setting a value for explore probability. It is better to choose a larger value such as 0.05 for it during the early periods of using XCSR to enable it to diversify and expedite the learning, and after that reducing it to a smaller value (even to 0) in order to not compensate performance for exploration. Also, another recommendation is to make explore probability a dynamic parameter that will be reduced when XCSR is performing well and increase when its performance drops which may indicate a concept drift. These recommendations are not investigated here but can be considered in future research and implementations.

VII. DELAYED FEEDBACK

In the previous chapter, online XCSR was introduced which does not have separate training and testing phases, but both happen at the same time. In other words, after receiving each instance, XCSR classifies the instance and then immediately receives feedback for its decision based on which it enhances its classifiers. However, in real-world applications, feedback is not provided immediately after each classification. Often the feedback is received with a delay. This delay is measured in the number of instances received between classifying a certain instance and receiving the feedback for it. In this section, the effect of delayed feedback on XCSR is experimented and discussed.

The experiments are designed as follows: A value for feedback delay, n , is chosen for each experiment (for instance 100). Then two familiar phases of training and testing are used but in a different way. Here, testing is done first on the first n instances, and then training is done on the same data. After that, testing is done on the second n instances, and then training is done on them and so on until all data is processed.

During testing phases, performance of XCSR is recorded, but no feedback is provided and thus no learning occurs. In such phases, only exploit is used in order to maximise the performance of XCSR. However, during training phases, feedback is provided and XCSR is allowed to learn, but its performance is not measured. Therefore, in order to enable the "accuracy-based" XCSR learn the complete map of the solution space, only explore is used.

At the beginning of the experiments, the optimum number of components for PCA-XCSR must be found. So, for different feedback delays, ranging from 1 to 500, PCA-XCSR with different number of components is used. Table IV shows the results of these experiments.

As Table IV demonstrates, similar to the previous section where feedbacks were immediate, pure XCSR does not perform well in any delay setting. However, when 3 to 5 components are used, in all different feedback delays PCA-XCSR performs well. For instance, when delay is 1, the best TPR.TNR is 92.94% which is achieved when 3 components are used. Similarly, for 500 instance delay, PCA-XCSR performs best when 5 components are used —it achieves TPR.TNR value of 84.34%.

TABLE IV. FINDING THE OPTIMUM NUMBER OF COMPONENTS FOR PCA-XCSR IN THE CONTEXTS WHERE FEEDBACK COMES WITH A DELAY. THE NUMBERS SHOW THE PERFORMANCE IN TERMS OF TPR.TNR.

Components	Delay 1	Delay 10	Delay 100	Delay 200	Delay 500
XCSR	26.40%	17.86%	14.75%	13.91%	12.55%
1	54.10%	55.14%	42.53%	45.67%	30.23%
2	65.90%	55.60%	43.23%	48.76%	43.70%
3	92.94%	91.35%	84.94%	85.82%	76.40%
4	92.86%	93.00%	87.99%	85.62%	83.83%
5	90.14%	89.48%	88.14%	87.07%	84.34%
6	88.77%	87.67%	86.32%	85.51%	82.98%
7	88.12%	86.97%	85.66%	84.62%	82.18%
8	81.57%	79.75%	78.10%	76.77%	74.05%
9	75.74%	73.16%	71.01%	70.01%	67.56%

It is important to note that in delayed version of XCSR, for every instance, both exploit and explore are done. Exploit will give the best classification (highest accuracy when performance is recorded) and explore will increase the learning rate when performance is not recorded. Therefore, every instance is effectively used twice, accounting for the higher TPR.TNR values seen in Table IV in comparison to Table II.

Now that the best settings for PCA-XCSR are found, let us analyse its behaviour when facing different levels of feedback delay. Fig. 1 contains a graph demonstrating the performance (TPR.TNR) of PCA-XCSR with different delay feedbacks. The graph is split into two halves. The top half shows the performance when the first half of the instances in the data set are received and processed and the bottom graph does the same for the second half of the data set. Each point in the graph shows the average TPR.TNR of PCA-XCSR for the previous 50 instances. For instance, on the line which shows the "100 delay", the point (3550, 86.49%) means that in the experiments that feedback delay has been 100, the TPR.TNR for instances 3551 to 3550 on average has been 86.49%.

These are the observations from Fig. 1:

1) After receiving about 4000 instances, all different delay settings achieve above 90% TPR.TNR value, majority of which is above 97%. The longest delay which is 500 performs slightly worse and its performance occasionally falls to 80%.
2) There are dips in the performance along the way, such as in the point 6150, 12100 and 13150 in which the performance significantly drops and then rises again in the next period. In the point 12600 all the performances drop to 90% and below. The worst one belongs to delay 10 with 81.75% and then to delay 1 with 82.46%. The whole experiments were repeated three more times and similar drops at similar point occurred. Spam emails stemming from new types of attack/virus are the source of these dips.
3) Smaller delays such as 1 and 10 have a shorter learning curve. That is, as expected, when XCSR receives feedback with shorter delay, it updates the rule-set faster. Therefore, its accuracy will be higher in dealing with the potentially changed environment.

It should be noted that XCSR does not need to store the instances that have not received their feedback, along with their meta-information such as the selected label, the match set or action set, in a buffer. The reason is that when the feedback is received (with the delay), the instance and correct class are enough to update the population. Also, the match set and action set at the time of classification are not valid any more, because the classifiers in those sets are probably updated.

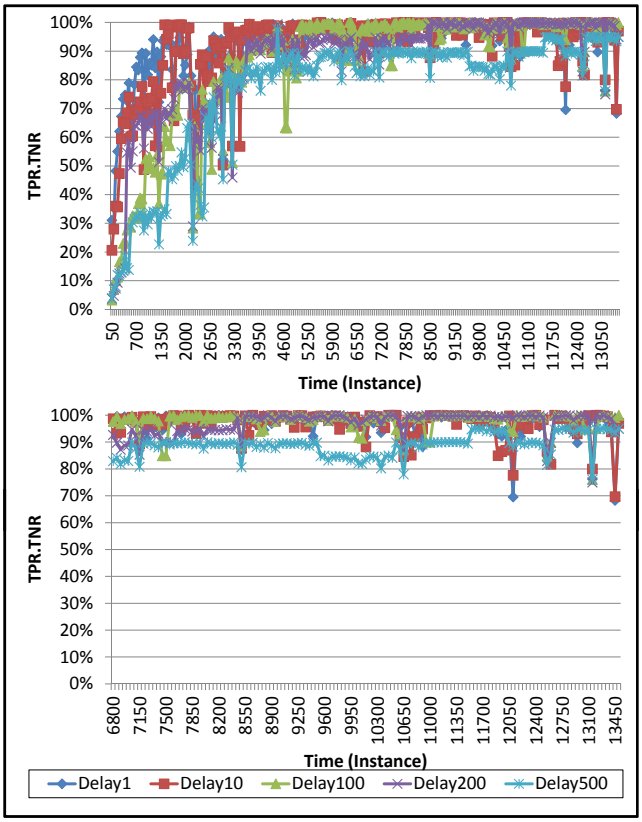

Fig. 1. Effect of feedback delay on the performance of online PCA-XCSR. Performance is shown by TPR.TNR for the last fifty instances at each point.

VIII. PARTIAL FEEDBACK

In this section, the robustness of XCSR when it faces partial feedback is studied. Partial feedback occurs when classification feedbacks are received stochastically. In order to observe the behaviour of XCSR when it receives partial feedbacks, the following experiments are designed and carried out.

Similar to the other sections, online XCSR is used. As previous sections showed, when using online PCA-XCSR, you get the best performance with 4 components and a small explore probability, here 0.001. Next, the PCA-XCSR with mentioned parameters is run on the whole Enron processed data set which contains $13,543$ instances. In each set of experiments, a different feedback probability is used, starting from 0.01% to 100% —when the feedback probability is 10%, after classifying each single instance, there is 10% chance of receiving a feedback. Table V demonstrates the result of these experiments as a table.

Let us start examining the results of the experiments depicted in Table V by looking at the values in the final column which is TPR.TNR, the usual performance measure. It can be seen that having a feedback probability larger than approximately 30% yields TPR.TNR value around 90% and below 5% feedback probability that is receiving fewer than

600 feedbacks, the performance drops significantly to less than 60%. The other thing to notice is that receiving more feedback does not improve the performance significantly. In other words, it is enough for XCSR to achieve its best performance by only receiving about 4,000 feedbacks —in case of this data set— and additional feedbacks do not have a significant effect.

Next thing to look at is the values in TPR column. Unlike TPR.TNR column, the values in this column increase steadily. It starts with very low values of 23% and 38% for 0.01% and 0.1% feedback probabilities respectively, and gets to it maximum of 96.4% for 100% feedback rate. This means every single additional feedback is helpful for XCSR to learn the majority class (positive instances make 86% of the data set) better. The values in the TNR column do not follow the same smooth increasing trend, causing the fluctuating increase in the TPR.TNR values.

When XCSR receives an instance, regardless of receiving a feedback or not for that instance, through covering mechanism it creates classifiers which match the instance. This way, XCSR makes use of even the instances without feedback as the classifiers created for such instances may be used by the next instances. In other words, when another instance is received, XCSR first searches its population of classifiers to find the matching classifiers. When it finds some classifiers which are already in the population, it uses them instead of creating some other random classifiers. This way, XCSR reaches a high level of generalisation which is one of its main goals and techniques, even in presence of very low feedback rates.

TABLE V. ANALYSING THE EFFECT OF PARTIAL FEEDBACK

Feedback Probability	Feedbacks Received	TPR	TNR	TPR.TNR
0.01%	1	23.0%	88.0%	20.5%
0.1%	14	37.8%	88.9%	33.3%
1%	135	71.8%	75.2%	53.6%
5%	677	85.5%	95.7%	81.8%
10%	1,354	89.8%	88.2%	79.2%
20%	2,709	92.3%	86.6%	79.9%
30%	4,063	93.7%	97.4%	91.2%
40%	5,417	94.2%	92.7%	87.4%
50%	6,772	94.9%	96.5%	91.6%
60%	8,126	95.2%	97.6%	93.0%
70%	9,480	95.6%	94.7%	90.5%
80%	10,834	95.9%	96.9%	92.9%
90%	12,189	96.1%	94.1%	90.4%
100%	13,543	96.4%	95.1%	91.6%

Finally, having an environment where feedbacks are both partial and also come with a delay, that is the combination of situations in Sections VIII and VII give the expected result of the deteriorating performance when feedback probability decreases and/or feedback delay increases. A complete set of experiments have been carried out, confirming the above statement. The details of the experiments are not mentioned here for the sake of paper space limitation.

IX. BIASED PARTIAL FEEDBACK

As discussed in Section VIII, in real-world applications, feedback is not given for every single classification. That is what we call partial feedback. When XCSR receives a feedback, it updates the classifiers in its population that contributed to that classification. But, there are classifications that do not receive any feedback. XCSR does not learn anything from such instances. In this section, we examine if we can use "no-feedback" to update the classifiers.

If we have some information about the environment such as class balance and probability of receiving feedbacks for each type of classification (that is true positive, false positive, true negative, false negative) we may be able to interpret a no-feedback as a feedback. Let us elaborate the concept with an example. If we can calculate that when an instance is classified as positive and no feedback is received for that classification, then the probability of the classification being done correctly is 90%, it makes sense that in such situations sometimes we assume a feedback confirming the correctness of the classification.

Based on the above concept, three heuristics are proposed in this section. The first one, ignores all the classifications for which no feedback is received. However, the next two heuristics attempt to make the best guess in such situation and make an inferred/artificial feedback based on the environment characteristics and the observations. The details of the heuristics are as follows.

A. Heuristic 0 (ignore-all)

This is what happens in a typical XCSR: When a classification does not receive any feedback, it is ignored. Nothing further is done to make any adjustments to the classifiers.

B. Heuristic 1

Here we use the information about the environment to calculate the relevant conditional probabilities. The information consists of the probabilities of receiving a feedback for certain types of classifications ($TPfPr$, $FPfPr$, $TNfPr$ and $FNfPr$) and class ratios (PR and NR).

When an instance is classified (as either positive or negative) and no feedback is received, the conditional probabilities of the classification being true or false is calculated using formulas (1 and 2).

Here are the definitions of the terms used in the formulas of Heuristics 1 and 2:

T	True or correct
F	False or wrong
CP	Classified as positive
CN	Classified as negative
nf	No feedback is received
$P(A\|B)$	The conditional probability of A given B
PR	Positive ratio. The distribution of positive instances in the environment
NR	Negative ratio. The distribution of negative instances in the environment
$TPfPr$	Probability of receiving a feedback when a TP classification is done
$FPfPr$	Probability of receiving a feedback when a FP classification is done
$TNfPr$	Probability of receiving a feedback when a TN classification is done
$FNfPr$	Probability of receiving a feedback when a FN classification is done
CPs	Number of instances classified as positive
CNs	Number of instances classified as negative

$TPswf$	Number of instances correctly classified as positive
$FPswf$	Number of instances wrongly classified as positive
$TNswf$	Number of instances correctly classified as negative
$FNswf$	Number of instances wrongly classified as negative

If an instance is classified as positive and no feedback is received, then with probability $P(P|CP\&nf)$, the classifications is assumed correct, otherwise it is assumed false. For negative classifications, $P(N|CN\&nf)$ is used instead. Then, an artificial feedback is accordingly created and finally the relevant classifiers are updated.

$$P(P|CP\&nf) = \frac{PR.(1 - TPfPr)}{PR.(1 - TPfPr) + NR.(1 - FPfPr)} \quad (1)$$

$$P(N|CN\&nf) = \frac{NR.(1 - TNfPr)}{NR.(1 - TNfPr) + PR.(1 - FNfPr)} \quad (2)$$

C. Heuristic 2

This estimation-based heuristic uses less information about the environment in comparison to Heuristic 1; It does not have class ratios. Instead, it estimates them by keeping track of feedbacks it receives for every type of classification. Therefore, this heuristic has the advantage of being able to work better in dynamic environments where the class ratio changes over time, or we don't have that information at the first place.

The conditional probabilities of a classification being true or false given no feedback is received is calculated using formulas (3, 4, 5 and 6). The rest is similar to Heuristic 1.

$$P(P|CP\&nf) = \frac{TPswf.(1 - TPfPr)}{TPfPr.(CPs - (TPswf + FPswf))} \quad (3)$$

$$P(N|CP\&nf) = \frac{FPswf.(1 - FPfPr)}{FPfPr.(CPs - (TPswf + FPswf))} \quad (4)$$

$$P(N|CN\&nf) = \frac{TNswf.(1 - TNfPr)}{TNfPr.(CNs - (TNswf + FNswf))} \quad (5)$$

$$P(P|CN\&nf) = \frac{FNswf.(1 - FNfPr)}{FNfPr.(CNs - (TNswf + FNswf))} \quad (6)$$

Both Heuristics 1 and 2 create artificial feedbacks for any classification without a feedback. This can be done in a more conservative way by using some filters and limiting the artificial feedback creation. For example, a filter can examine if the number of artificial updates is fewer than a ratio (e.g. 20%) of all the classifications. In the filtered version of a heuristic, if the filter condition is passed, the artificial feedback is created and classifiers are updated.

In order to run the experiments, five different environment settings for feedback probabilities are used. The first three settings, namely A, B and C are chosen to represent realistic scenarios. That is, the majority of the feedbacks are in regards to the errors in the classifications. Setting A, represents a fraud detection environment in which when an attack is not recognised (FN), feedback is provided in 90% of the times, but when a legitimate instance is categorised as fraud (FP), feedback probability is 10%. In regards to correct classifications (TP or TN), probability of feedback is low (1%). Setting B is similar to A with one difference: probability of feedback in case of a FP is as high as FN (90%). Finally, setting C is similar to B, but here no feedback is received in case of correct classifications.

Settings D and E are two extreme, unrealistic situations where feedbacks are received mostly (setting D) or only (setting E) for correct classifications. They are used in the experiments to study the effects of the heuristics in such extreme conditions.

As usual, in the experiments, we run PCA-XCSR with four components and explore probability of 0.001 on the Enron processed data set. Each experiment is repeated 10 times and the average is shown in the results. Each heuristics is applied to each setting, one at a time. The results of the experiments are summarised in Table VI. As the Accuracy and TPR.TNR columns under "Unfiltered" title show, except setting A, unfiltered Heuristics 1 and 2 are both performing as good as or better than Heuristic 0. The most impressive result can be seen in the setting E where the only feedbacks received are for correct classifications. Heuristic 0 only achieves 20% TPR.TNR while Heuristic 1 achieves 80%.

The second set of experiments use the the filtered versions of Heuristics 1 and 2. Filters we use are: (1) Limit the number of artificial updates to 20% of all the classifications, and (2) limit the artificial-feedback-updates to only the classifications for which when no feedback is received, the difference between probability of it being correct and false is greater than 0.3 (i.e. $|P(P|CP\&nf) - P(N|CP\&nf)| > 0.3$). The results of these experiments are shown in Table VI in the columns titled "Filtered". This time, the results of Heuristics 1 and 2 are always better than Heuristic 0.

When Heuristics 1 and 2 are used in comparison to Heuristic 0, improvements are not seen in all situations (for example, unfiltered heuristics on setting A). First reasons is that although beneficial adjustments are made to the classifiers based on the heuristics, some small fraction of the updates are also incorrect and this has a detrimental effect. The second reason may be that the Enron processed data set is not a suitable data set for these experiments (e.g. it has class imbalance: 86% positive instances v 14% negative ones). Using a more suitable data set may give more meaningful results. Even a fully controllable synthetic data set can give us the ability to analyse the behaviour more accurately.

To summarise the findings of this section, always use filtered Heuristics 1 and 2 which perform better than Heuristic 0, unless you are dealing with an extreme environment where

TABLE VI. BIASED PARTIAL FEEDBACK EXPERIMENTS RESULTS

Setting	Feedback Probability TP	FP	TN	FN	Heuristic	Unfiltered Accuracy	Unfiltered TPR.TNR	Filtered Accuracy	Filtered TPR.TNR
A	0.01	0.1	0.01	0.9	0	91%	65%	91%	65%
					1	83%	7%	94%	81%
					2	86%	31%	93%	75%
B	0.01	0.9	0.01	0.9	0	94%	87%	94%	87%
					1	95%	93%	96%	93%
					2	95%	91%	96%	93%
C	0	0.9	0	0.9	0	91%	89%	91%	89%
					1	95%	90%	96%	93%
					2	95%	89%	96%	93%
D	0.9	0.01	0.9	0.01	0	87%	68%	87%	68%
					1	92%	74%	92%	68%
					2	91%	72%	92%	69%
E	0.9	0	0.9	0	0	78%	20%	78%	20%
					1	93%	80%	86%	29%
					2	91%	67%	87%	39%

feedbacks are only for correct classifications (settings D and E) in which case unfiltered Heuristics 1 or 2 yield even better results.

X. CONCLUSION

In this paper we analysed the behaviour of XCSR in dynamic environments in which the feedback is delayed and partial. It was shown that XCSR can handle such environments well. Online XCSR, a framework for using XCSR in these situations was proposed gradually throughout the paper.

As an important aspect of environments with incomplete feedbacks, we studied the ones which provide biased partial feedback. That is, feedback probability is different based on the type of classification, be it class or correctness. Instead of ignoring classifications that receive no feedback, we proposed two heuristics that based on the environment characteristics and observations can create artificial feedbacks for such classifications. The results of the experiments on five different settings showed that we always get a higher accuracy when we use the proposed filtered heuristics in comparison to ignoring the no-feedback classifications.

Future work includes investigating improvements in the mentioned heuristics for dealing with biased partial feedback. The filtered version of the heuristics can be further improved by investigating other options. For example, limiting the artificial updates to the classifiers that: (1) have a low classification certainty (as indicated by the values in the prediction array), (2) have been updated less than a certain number of times (by adding a counter parameter to each classifier), (3) are young (whose experience is smaller than a threshold), and (4) are not fit enough (whose fitness is smaller than a threshold).

REFERENCES

[1] T. J. Walsh, A. Nouri, L. Li, and M. Littman, "Learning and planning in environments with delayed feedback," *Autonomous Agents and Multi-Agent Systems*, vol. 18, pp. 83–105, 2009.

[2] H. Rahmandad, N. Repenning, and J. Sterman, "Effects of feedback delay on learning," *System Dynamics Review*, vol. 25, no. 4, pp. 309–338, 2009.

[3] K. Shafi and H. A. Abbass, "An adaptive genetic-based signature learning system for intrusion detection," *Expert Systems with Applications*, vol. 36, no. 10, pp. 12 036 – 12 043, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417409002589

[4] H. H. Dam, C. Lokan, and H. A. Abbass, "Evolutionary online data mining: An investigation in a dynamic environment," *Studies in Computational Intelligence*, vol. 51, pp. 153–178, 2007.

[5] S. W. Wilson, "Classifier fitness based on accuracy," *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, 1995.

[6] ——, "Get real! XCS with continuous-valued inputs," in *Learning Classifier Systems, From Foundations to Applications*. Springer, 2000, pp. 209–222.

[7] R. J. Urbanowicz and J. H. Moore, "Learning classifier systems: a complete introduction, review, and roadmap," *J. Artif. Evol. App.*, vol. 2009, pp. 1:1–1:25, January 2009.

[8] C. Stone and L. Bull, "For real! XCS with continuous-valued inputs," *Evolutionary Computation*, vol. 11, no. 3, pp. 299–336, 2003.

[9] M. Behdad, L. Barone, T. French, and M. Bennamoun, "An investigation of real-valued accuracy-based learning classifier systems for electronic fraud detection," in *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, 2010, pp. 1893–1900.

[10] M. Abedini and M. Kirley, "A multiple population XCS: Evolving condition-action rules based on feature space partitions," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*, July 2010, pp. 1–8.

[11] M. Behdad, T. French, L. Barone, and M. Bennamoun, "On principal component analysis for high-dimensional XCSR," *Evolutionary Intelligence*, vol. 5, pp. 129–138, 2012, 10.1007/s12065-012-0075-6.

[12] B. Moore, "Principal component analysis in linear systems: Controllability, observability, and model reduction," *IEEE Trans. on automatic control*, vol. 26, pp. 17–32, 1981.

[13] V. Metsis, I. Androutsopoulos, and G. Paliouras, "Spam filtering with naive bayes-which naive bayes," in *3rd Conference on Email and Anti-Spam*. Citeseer, 2006, pp. 125–134.

[14] Apache Software Foundation, "Spamassassin." [Online]. Available: http://spamassassin.apache.org