PCA for Improving the Performance of XCSR in Classification of High-dimensional Problems

Mohammad Behdad
University of Western Australia
35 Stirling Highway Crawley
6009 WA, Australia
behdad@csse.uwa.edu.au

Tim French
University of Western Australia
35 Stirling Highway Crawley
6009 WA, Australia
tim@csse.uwa.edu.au

Luigi Barone
University of Western Australia
35 Stirling Highway Crawley
6009 WA, Australia
luigi@csse.uwa.edu.au

Mohammed Bennamoun
University of Western Australia
35 Stirling Highway Crawley
6009 WA, Australia
bennamou@csse.uwa.edu.au

ABSTRACT

XCSR is an accuracy-based learning classifier system (LCS) which can handle classification problems with real-value features. However, as the number of features increases, a high classification accuracy comes at the cost of more resources: larger population sizes and longer computational running times. In this research, we present a PCA-enhanced LCS, which uses principal component analysis (PCA) as a preprocessing step for XCSR, and examine how it performs on complex multi-dimensional real-world data. The experiments show that this technique, in addition to significantly reducing the computational resources and time requirements of XCSR, maintains its high accuracy and even occasionally improves it. In addition to that, it reduces the required population size needed by XCSR.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Concept learning*

General Terms

Experimentation, Performance

Keywords

Genetics-Based Machine Learning, Learning Classifier Systems, XCSR, Network Intrusion Detection, Principal Component Analysis

GECCO'11, July 12–16, 2011, Dublin, Ireland.

1. INTRODUCTION

First proposed by John Holland [7] in 1976, a learning classifier system (LCS) is a machine learning technique that combines reinforcement learning, evolutionary computing, and other heuristics to produce an adaptive system capable of learning in dynamic environments. It uses a population of condition-action-prediction rules called classifiers to encode appropriate actions for different input states. Learning classifier systems (LCSs) are good at solving complex adaptive problems and have been successfully used in data mining applications [3]. However, when they are used for high-dimensional problems, they slow down significantly [9]. XCS [21] 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.

KDD'99 [6] is a popular network intrusion data-set. Each element of this set describes a network connection using 41 features describing properties about a connection (such as how many bytes are transferred, number of files access, or the number of times a login was attempted). Many of these connections are legitimate, but a number are malicious (such as trying to gain root access to a system). The classifier must try to determine whether a connection is benign, or whether it corresponds to one of several different attack types. This data-set has some interesting characteristics: it is large in both the number of instances and number of dimensions, and its dimensions are of different types, including strings, Booleans and numbers.

In order to be able to use the XCSR for KDD'99 data-set, the data must first be normalized. The normalization process increases the number of features from 42 to 118 (see Section 4 for details). One important property of the normalized data is that most of its features have value 0. In other words, we are dealing with sparse data with a high number of dimensions. Hence, there is a high chance that some of the features in the data are correlated or redundant. In order to exploit this characteristic, we can use a dimensionality reduction technique to "compress" the problem and reduce the search space for the learning algorithm.

Principal Component Analysis (PCA) is a linear dimensionality transformation method, capable of reducing the number of dimensions of a data-set by transforming the problem into a new search space. In this paper, we investigate if using PCA as a preprocessing step can reduce the time and computational resources needed by XCSR in such high-dimensional problems while maintaining its classification accuracy.

In the next section, we review the related works done in this area and present background information about LCSs and PCA. Then, in Section 3, we introduce the use of PCA alongside XCSR. In Section 4, we present experiments that explore the performance of XCSR on a KDD'99-like data-set, with and without PCA with different parameter settings and discuss the results. Then, in Section 4.3 we briefly discuss the criteria in setting the number of principal components. Finally, Section 5 concludes the work.

2. RELATED WORKS

There have been some previous studies on the effects of higher numbers of dimensions on the performance of machine learning methods [8], as well as LCS [9]. Howley et al. [8] discuss the effect of high-dimensional data on machine learning accuracy, concluding that such data will degrade classification accuracy. They propose using PCA in order to improve the performance of machine learning classification in problems with high number of dimensions. The techniques considered include support vector machines, distance based methods, decision trees, and naive Bayes. Xu and Wang [23] improve the performance of support vector machines both in speed and accuracy by using PCA.

Ferrandi et al. [5] use PCA as a linear classifier to compare its performance with that of LCS. Ravichandran et al. in [13] do a similar thing. They compare the accuracy and robustness of LCS against a PCA-based distance classifier and find that LCS on its own can outperform PCA on its own. However, no analysis and detail are provided. It should be noted that these works examine XCS which can only handle binary strings.

In order to test the performance of a PCA-enhanced LCS, we use a real-world problem: the KDD'99 intrusion detection data-set [6]. This data-set exhibits many of the complexities and features of real-world applications in one composite problem. The data-set contains $4,898,430$ experiences (connections) in a "training" file and $311,029$ connections in the "test" file. A smaller version of the training file (called training10p), which contains a 10% subset of the training data at the same distribution as the complete file is also included in the data-set. For each file, every "connection" has a vector of 41 features and a label. The label determines if the connection is normal or an attack. There are 22 types of attack in the training file and 39 in the testing file. Every attack type belongs to one of four attack categories. Classifiers are trained to identify whether a connection is an attack, and if so, what category the attack is, so there are effectively five classes a connection may belong to (the four attack types plus non-attack).

For this data-set, there are two metrics which are traditionally used for evaluation of performance. The first one is *accuracy*, which is the percentage of the correct predictions made by the system and is calculated using true positive, true negative, false positive and false negative values: $(TP+TN)/(TP+FN+FP+TN)$. The second metric is *Cost Per Example* (CPE) and is calculated using the following formula [19]: $CPE = \frac{1}{N}\sum_{i=1}^{m}\sum_{j=1}^{m} CM(i,j) * C(i,j)$ where CM is the Confusion Matrix (the number of instances of class i classified as class j), C is the Cost Matrix (the cost of misclassifying class i as class j), N the total number of test instances, and m the number of classes. In the context of network intrusion detection, a smaller CPE means a better classification. While CPE is an important metric to show a techniques applicability to the real world problem of network intrusion detection, accuracy is a better comparison between two techniques in the general case.

Few examples exist in applying a LCS to the KDD'99 data-set. Shafi et al. [14] analyse two learning classifier systems, namely XCS and UCS on a subset of KDD'99 data-set, suggesting some improvements for these two algorithms in the areas of class imbalances and rarity. They also compare the performance of these methods with other machine learning algorithms including decision tree methods, naive Bayes, and distance based methods, and conclude that LCSs are a competitive approach to the problem. Tamee et al. [18] proposes using Self-Organizing Maps (SOMs) with LCS for network intrusion detection. They analyse the performance of their method and show that the proposed system is able to perform significantly better than twelve machine learning algorithms. Marín-Blázquez and Martínez Pérez [10] use a modified version of XCS called linguistic hedged fuzzy-XCS to tackle the KDD'99 problem; the advantage of their approach is that their system returns a set of human interpretable knowledge (rules). In [2], Behdad et al. apply XCSR on the KDD'99 data-set as an example of a real-world fraud detection problem and report competitive performance.

2.1 Learning Classifier Systems

Learning classifier systems use a population of condition-action-prediction rules called classifiers to encode appropriate actions for different input states. Rules are of the If-Then type: if a condition is true, then the corresponding action will be performed. The decision of which rule to use depend on the type of LCS used. Some use reward (the prediction in the classifier triple) the system expects to receive from the environment as a result of performing the chosen action. In some methods, this decision is based on the accuracy of the classifiers matching the current input. In essence, an LCS attempts to enhance its understanding of the environment through improving its classifiers over time.

2.1.1 XCS

XCS [21] 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. Another important difference of XCS over its predecessors

is that the GA is applied to A (or niches) rather than the whole classifier population. This narrows evolutionary competition to between classifiers which can be applied in similar scenarios rather than all the classifiers in the LCS [15].

2.1.2 XCSR

Traditionally, XCS has been used for problems with binary strings as input. However, many real-world problems have continuous values and can not be represented by the ternary representation used by XCS. A continuous version of XCS, called XCSR, was introduced by Wilson [22] which handles these kinds of problems using an interval-based representation. For this work, we use the Unordered Bound Representation (UBR) [17] representation in which the interval predicate takes the form of (p_i, q_i) where either p_i or q_i can be the maximum or minimum bound.

2.2 Principal Component Analysis

Principal component analysis is a feature transformation technique whose purpose is to find the most important information; so called compaction through dimensionality reduction. It analyses a set of feature vectors (original variables) and represents them as a set of new orthogonal feature vectors (new 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 [11].

The values of the new variables (called factor scores) are calculated by projecting the original variables onto the principal components. Each principal component has an eigenvalue which is calculated by summing the squared factor scores for that component. Dimensionality reduction is achieved by using only a subset of the principal components (the first n) instead of using all of them. The choice of n is subjective, and different criteria has been suggested by different researchers, but a general rule of thumb is to use only the principal components whose eigen-values are larger than 1. However, this rule does not work in all domains [12].

For a comprehensive introduction to PCA, the reader is referred to [1].

3. PCXCSR

In our previous work [2], we used XCSR as a network intrusion detection system. We considered both online learning (where each new instance from the test set may influence the set of classifiers) and offline learning (where the set of classifiers is derived from a training set and then statically applied to a test set). Our resultant system was evaluated against the KDD'99 data-set [6], with results indicating performance was comparable to that achieved by the winner of the KDD'99 competition without online learning, and superior to the performance of the KDD'99 winner when online learning was used [2].

One problem with using XCSR on this data-set is the high number of dimensions or features (118 after normalization): the time performance of XCSR is relatively slow (as a rough benchmark, these results took more than 20 minutes to run on a relatively new desktop machine). Hence, the main purpose of this research is to improve the speed of XCSR in such high-dimensional data-sets.

Our solution is to use PCA to compact the data through dimensionality reduction. So, after normalizing the data, it is firstly compacted (transformed) by PCA before given to XCSR to learn from. XCSR hence learns 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. We call this combination of PCA-preprocessing and XCSR, PCXCSR. This presents a number of potential advantages:

1. By examining only the most significant principal components, we reduce the dimensionality of the search space.
2. As the principal components are linear combinations of dimensions, the classifier rules may naturally exploit existing dependencies among features rather than having to randomly search out such dependencies.
3. By retaining only the components with the highest eigenvalues, we eliminate noise caused by less relevant components.

4. EXPERIMENTS

In order to evaluate the utility of PCXCSR, we compare its performance with the performance of vanilla XCSR on a data-set based on KDD'99. Our main interest is exploring the trade-off between accuracy and time, varying the number of principal components used by the learning classifier system.

In order to analyse the effects of using PCA with XCSR, a random subset of the KDD'99 training10p file was created. This subset, which we call "kdd-sub", contains $100,000$ connections (feature vectors). Twenty percent of this data-set ($20,000$ connections) is used for training and the remaining eighty percent of instances are used for testing. It should be noted that we are not interested in getting the best results for KDD'99; we are instead interested in observing the behaviour of PCXCSR. Since this paper is focused on classification accuracy, and the fitness function of the learning classifier system ignores cost, we only use accuracy for measuring overall performance on the data-set.

To start, the data in the kdd-sub data-set was firstly normalized. This involved scaling all quantitative fields to lie between -1 and 1, and representing all enumerated types with independent scalar features (so if there were 6 different values for a type, we would replace this with 6 fields between -1 and 1). This process resulted in the number of features growing from 42 to 118, but also ensured the data was very sparse.

The underlying learning classifier system is based on Butz's implementation of XCS [4]. For representation of intervals, we use the Unordered Bound Representation (UBR) without limiting the ranges to be between 0 and 1. The values of important XCSR parameters, if not stated differently, are as follows: $N = 50,000$, $\alpha = 0.1$, $\beta = 0.2$, $\nu = 5$, $\epsilon_0 = 0.01$, $\theta_{GA} = 25$, and $_{Sub} = 12$. In these experiments, we only consider off-line learning, and thus the system does not learn during the test phase.

MATLAB is used for the PCA pre-processing step. Firstly, MATLAB's PCA routine is used to calculate the eigen-vectors (coefficients in MATLAB) and eigen-values (or latents in

MATLAB) for the training data-set. Then, each connection is normalized and transformed using PCA using n principle components. Experiments below vary n, with results reporting the trade-off in performance (accuracy) versus running time.

All experiments are repeated 10 times and the results averaged. To keep the consistency of the conditions (especially the time each experiment takes), experiments are all run on the same machine (an Intel Core2 Duo 2.50 GHz machine with 4 GB RAM) under the same load conditions. Reported times include the time needed to perform the initial PCA pre-processing.

4.1 Experiment 1: Accuracy vs. Time

In our first experiment different numbers of Principal Components (PCs) between 1 and 118 (the number of features after normalization) are analysed to see what effect the number of PCs has on both time and accuracy. In our previous work [2], the population size that gave the best results for XCSR on KDD'99 was $50,000$. So, we use the same population size for XCSR. The other parameter settings for XCSR are the same as the one in [2]. All the experiments are repeated 10 times, and the results averaged. The summary of the results of these experiments can be seen in Figure 1, where the horizontal access presents the number of principal components and the accuracy and time are represented by the left and right axis respectively. As the results for the vanilla XCSR are independent of the number of principal components the time and accuracy are represented by two horizontal lines.

As Figure 1 shows, XCSR can get very high accuracies (above 99%) by only using a small number of PCs (principal components). This reduces the amount of information (size of a single classifier) and consequently all the related computations (such as finding matchsets) of XCSR. As a result, when using PCA with only 5 PCs for example, the time needed for the whole training and testing phases is reduced to less then 7% of the time needed when vanilla XCSR (without PCA) is used (93 seconds vs. 1343 seconds) while still having the same high accuracy (above 99%).

It is interesting that the accuracy of the XCSR when using between 5 and 29 PCs is even higher than that of XCSR without PCA. However, as we continue adding PCs, the accuracy starts to deteriorate. The best explanation for this behavior is that PCA compacts the data. That is, it stores the first PC represents the most important feature in the new space (the feature with highest variance), and the second PC is the second important feature and so on. So, when XCSR is learning in this new feature space when only the first few "important" PCs are presented to it, the algorithm focuses its learning process (creating and improving its classifiers) on these few dimensions. But, when more PCs are introduced, the attention of XCSR, or at least a part of its attention, is consumed by these less important (redundant or even misleading) dimensions. XCSR intrinsically treats all the dimensions equally and does not give different weights to the dimensions. Hence, it must effectively learn which dimensions to "listen" to, slowing down the rate of its learning. We plan to examine this problem in future works.

When using 50,000 population size for "kdd-sub", the time vanilla XCSR takes is longer than that of PCXCSR with 118 principal components. As these two methods are working on a data of the same size (118 dimensions), and PCXCSR requires some additional PCA calculations steps, it is expected to see the opposite. It is curious and creates a new avenue for research in this area. The reason may lie in the linear transformation of PCA which creates a more efficient decision making space. There are many factors involved in determining the time PCXCSR requires for classification, such as match-sets creation, subsumption process, etc. In future works we will study each aspect of PCXCSR in order to find the reason for this odd and interesting behaviour.

4.2 Experiment 2: Population vs. Accuracy vs. Time

As a secondary investigation, we examine the effect of population size on the accuracy and time required by XCSR when used with PCA. We know from out previous experiments in [2] that population size directly influences accuracy of XCSR and the time it needs. In the next set of experiments, we reduce the population size to $4,000$ which is expected to give very poor accuracies. Again, experiments are done for both vanilla XCSR and PCXCSR. Figure 2 shows the results of these experiments, contrasting the time needed and accuracy achieved by each different setting, namely different number of PCs.

This time, the effect of PCA is much more pronounced. As expected, vanilla XCSR can not perform as effectively as it did with population size of $50,000$ and its accuracy drops from 99% to 33%. But, when PCA is used alongside the XCSR, we get accuracies as high as 99%. Such high accuracy is achieved by using as few as 5 PCs, and even smaller numbers yield reasonable performances. Here, again we see the same behavior of losing accuracy when adding PCs to our PCA calculations; for instance, when 20 PCs are used, the accuracy drops to 28%.

The last set of experiments are similar to the previous one, but this time we use many different population sizes ranging from 50 to $50,000$. These experiments show that PCXCSR is able to give very good results with only a small population, provided only a small number of principal components are used. PCXCSR is able to achieve an accuracy in excess of 98% with a population of 500 (2-4 PCs), and is able to achieve an accuracy in excess of 99% with a population of $4,000$ (5-11 PCs). In contrast, without principal component analysis XCSR is only able to achieve an accuracy of 20% with a population of 500 and 32% with a population of $4,000$. We see that increasing the population beyond $4,000$ allows PCXCSR to *tolerate* more principal components, but not significantly improve the result. For example, the best accuracy achieved with the each of the populations $4,000$, $25,000$ and $50,000$ was 99.5%, and each achieved this accuracy using only 9 principal components. The time taken to achieve this best accuracy was 65 seconds for the $4,000$ population, 140 seconds for the $25,000$ population and 144 seconds for the $50,000$ population.

Figure 3 shows the summary of these experiments in terms of accuracy.

4.3 Setting the number of principal components

The eigen-values for all the principal components of the training portion of "kdd-sub" is demonstrated in Figure 4. As can be seen in this figure, roughly after 10 principal components, the contribution of each new eigen-value drops to below 1. This corresponds to Figures 1 and 2, in which the

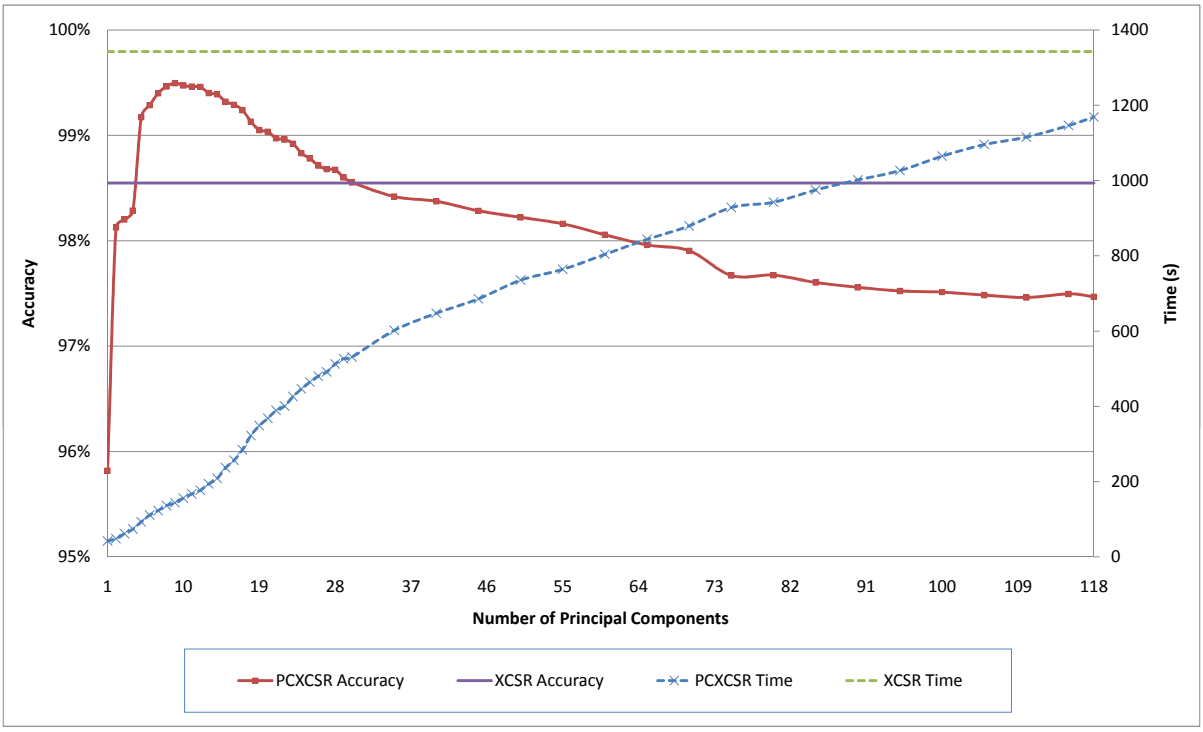

Figure 1: Accuracy and running time for vanilla XCSR vs. PCXCSR with population sizes 50,000.

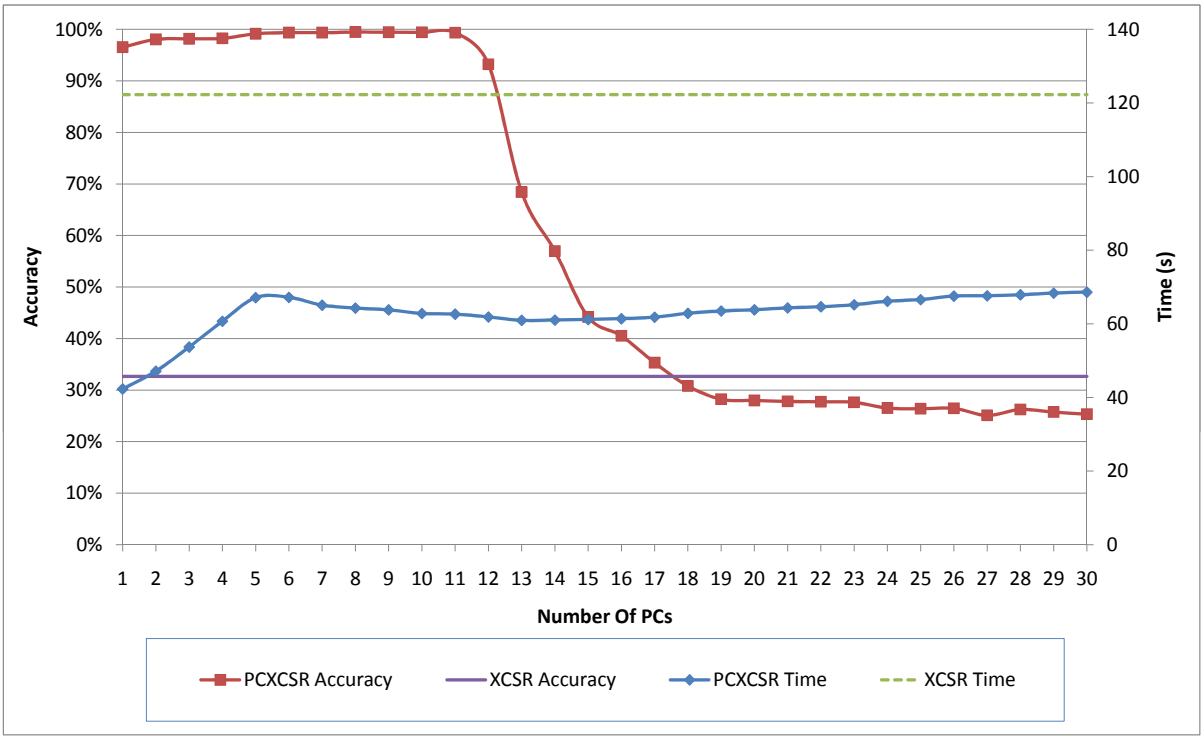

Figure 2: Accuracy and running time for XCSR vs. PCXCSR with population sizes 4,000.

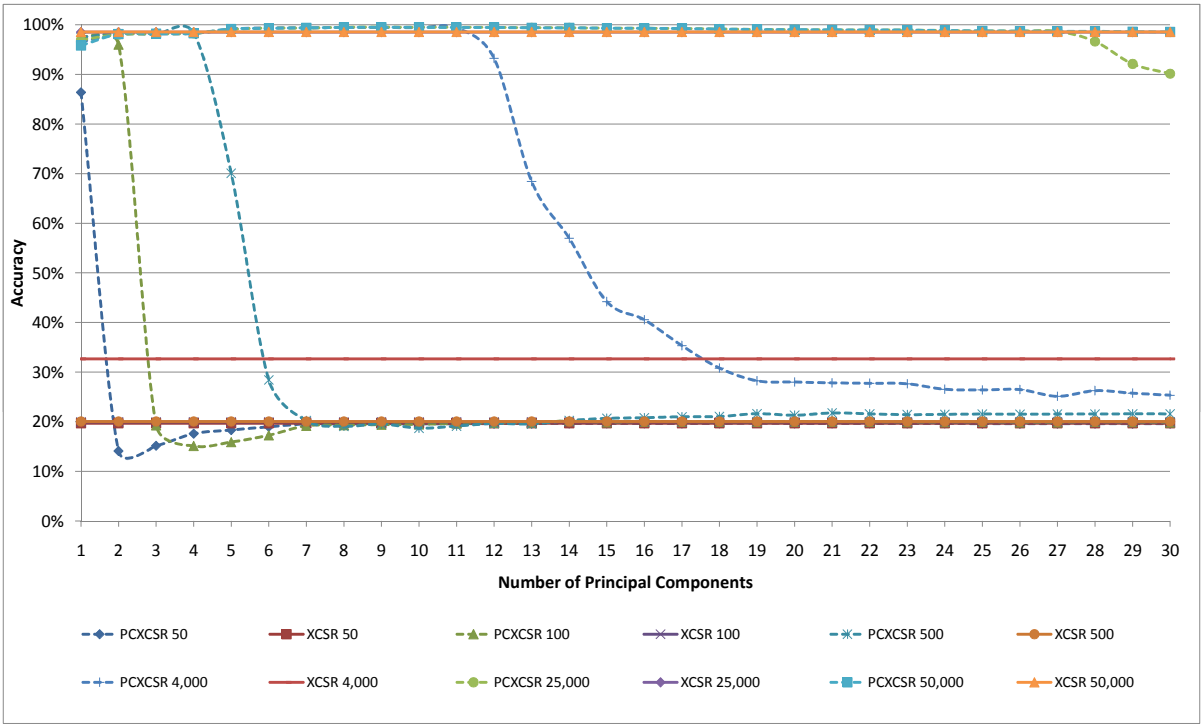

Figure 3: Accuracy of XCSR vs. PCXCSR using different population sizes.

accuracy reaches its peak and then starts to deteriorate. Future work will seek to identify an appropriate mechanism to select the number if principal components, given the distribution of the corresponding eigenvalues. Although we note, even a single principal component can yield reasonable results.

Finally, we note the trade off between dimension, population and time:

1. XCSR (population 50,000) will achieve an accuracy of 98.5% in 1343 seconds;
2. PCA-XCSR (population of 50,000) with 5 principal components will achieve an accuracy of 99% in 100 seconds;
3. PCA-XCSR (population 4,000) with 9 principal components will achieve an accuracy of 99% in 68 seconds;
4. XCSR (population 4,000) will only achieve an accuracy of 33%.

We see that augmenting XCSR with PCA allows us to achieve the same high accuracy, but with a reduced population and dimension resulting in much faster learning (almost twenty times as fast). This, of course, depends on us finding an appropriate combination of population size and number of principal components. Further investigation is required to determine effective heuristics to do this.

5. CONCLUSION AND FUTURE WORKS

In this work, we examined the challenge of high dimensionality for LCSs and investigated the use of PCA as a pre-processing step for XCSR in order to compact the search space the LCS must explore. Our experiments on a KDD'99-related data-set showed that XCSR can use the compacted data produced by PCA very effectively, obtaining very high accuracies in a much shorter running time, at least in this problem domain. This PCA-enhanced XCSR also reduces the population size parameter of LCS significantly.

We note that as PCXCSR uses PCA as a preprocessing step, it is not innately capable of online learning, which presents an interesting challenge to explore. In the future, we will examine the effects of running PCA on just a sample of the population, and explore how one might transform a linear basis during (online with) the learning process.

As discussed earlier, work by Howley et al. [8] used seven different machine learning methods for classifying high dimensional spectra data. They analysed these methods with and without PCA, and based on a degragation of performance for C4.5 and RIPPER when used with PCA, they conclude "poor results can be achieved when PCA is used in combination with rule-based learners" [8]. However, this work shows that this statement is not necessarily correct in all cases. XCSR, which is a rule-based classification method, combined with a population based approach can perform better when it is used with PCA.

It would also be interesting to compare PCXCSR with XCSF [20]. XCSF attempts to learn a given function by sampling points and producing rules that define a piecewise linear function. As XCSF uses a linear transformation within the rules, it is possible that it may be able to identify the relationships between features that PCXCSR so efficiently exploits. A detailed study of scalability and resource management for XCSF is given in [16].

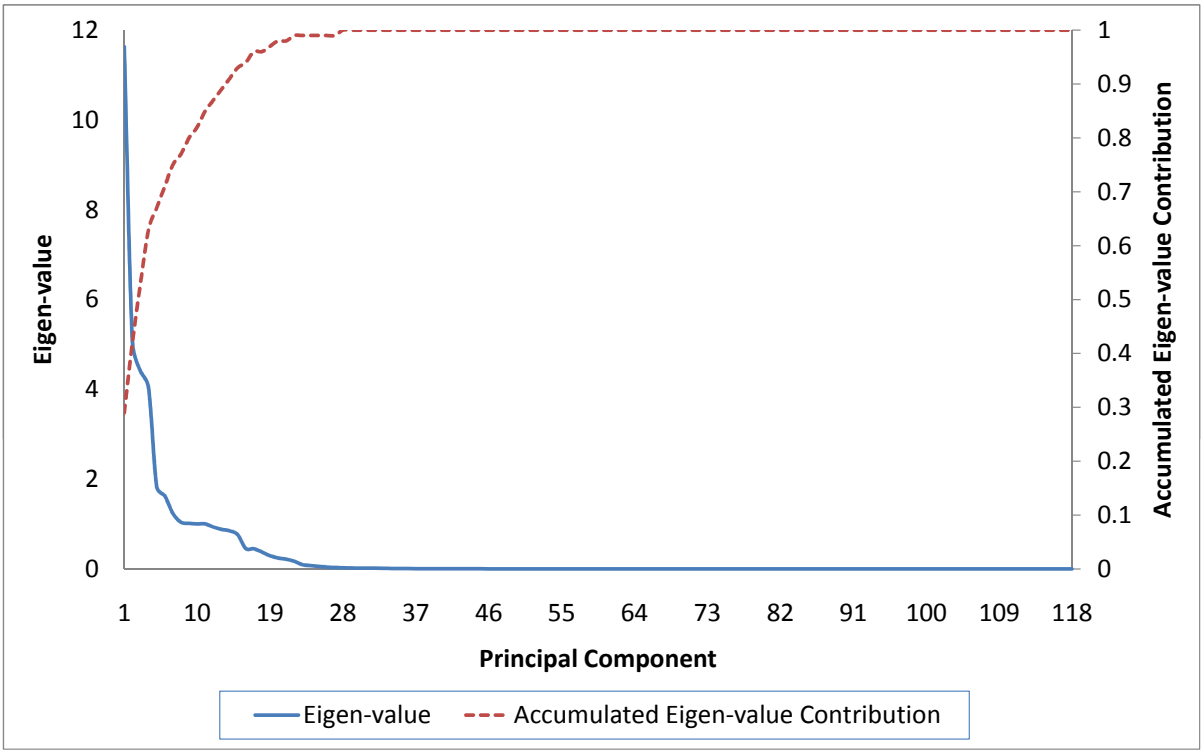

Figure 4: Size of eigen-values for principal components and their accumulated contribution.

Future work includes finding the best combination of smallest population size and lowest number of principal components which yield the highest accuracy for XCSR. Further, other high-dimensional problems in domains other than network intrusion detection should be investigated. Finally, XCSR treats all the features (dimensions) of its input equally and does not have a weighing mechanism to favour one dimension over another. Adding the capability for different weights may lead to obtaining the same high accuracies and low population sizes that PCA provides, without actually needing PCA. We plan to explore this in our next paper.

6. REFERENCES

[1] H. Abdi and L. J. Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.

[2] M. Behdad, L. Barone, T. French, and M. Bennamoun. An investigation of real-valued accuracy-based learning classifier systems for electronic fraud detection. In *GECCO (Companion)*, pages 1893–1900, 2010.

[3] L. Bull. *Applications of Learning Classifier Systems*. Springer, 2004.

[4] M. V. Butz and S. W. Wilson. An algorithmic descrtipion of xcs, 2000. Technical report 2000017, Illinois Genetics Algorithms Laboratory.

[5] F. Ferrandi, P. Lanzi, D. Sciuto, and M. Tanelli. System-level metrics for hardware/software architectural mapping. In *Proceedings of the Second IEEE International Workshop on Electronic Design, Test and Applications*, DELTA '04, pages 231–236, Washington, DC, USA, 2004. IEEE Computer Society.

[6] S. Hettich and S. D. Bay. The UCI KDD archive [http://kdd.ics.uci.edu]. Irvine, CA: University of California, Department of Information and Computer Science, 1999.

[7] J. H. Holland. Adaptation. *Progress in Theoretical Biology*, 4:263–293, 1976.

[8] T. Howley, M. G. Madden, M.-L. O'Connell, and A. G. Ryder. The effect of principal component analysis on machine learning accuracy with high-dimensional spectral data. *Knowledge-Based Systems*, 19(5):363–370, 2006.

[9] M. Kirley and M.Abedini. CoXCS: a coevolutionary learning classifier based on feature space partitioning. In *Lecture Notes in Computer Science*, volume 5866, pages 360–369, 2009.

[10] J. Marín-Blázquez and G. Martínez Pérez. Intrusion detection using a linguistic hedged fuzzy-XCS classifier system. *Soft Computing — A Fusion of Foundations, Methodologies and Applications*, 13:273–290, 2009.

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

[12] A. J. O'Toole, H. Abdi, K. A. Deffenbacher, and D. Valentin. Low-dimensional representation of faces in higher dimensions of the face space. *Journal of the Optical Society of America*, 10(3):405–411, Mar 1993.

[13] B. Ravichandran, A. Gandhe, R. Smith, and R. Mehra. Robust automatic target recognition using learning classifier systems. *Information Fusion*, 8(3):252 – 265, 2007.

[14] K. Shafi, T. Kovacs, H. A. Abbass, and W. Zhu. Intrusion detection with evolutionary learning classifier systems. *Natural Computing*, 8(1):3–27, 2009.

[15] O. Sigaud and S. W. Wilson. Learning classifier systems: a survey. *Soft Computing — A Fusion of Foundations, Methodologies and Applications*, 11(11):1065–1078, 2007.

[16] P. Stalph, X. Llora, D. Goldberg, and M. V. Butz. Resource management and scalability of the XCSF learning classifier system. *Theoretical Computer Science*, 2010.

[17] C. Stone and L. Bull. For real! XCS with continuous-valued inputs. *Evolutionary Computation*, 11(3):299–336, 2003.

[18] K. Tamee, P. Rojanavasu, S. Udomthanapong, and O. Pinngern. Using self-organizing maps with learning classifier system for intrusion detection. In *PRICAI '08*, pages 1071–1076. Springer, 2008.

[19] A. N. Toosi and M. Kahani. A new approach to intrusion detection based on an evolutionary soft computing model using neuro-fuzzy classifiers. *Computer Communications*, 30(10):2201–2212, 2007.

[20] S. Wilson. Function approximation with a classifier system. In *Proceedings of GECCO'01*, pages 974–981, 2001.

[21] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[22] S. W. Wilson. Get real! XCS with continuous-valued inputs. In *Learning Classifier Systems, From Foundations to Applications*, pages 209–222. Springer, 2000.

[23] X. Xu and X. Wang. An adaptive network intrusion detection method based on pca and support vector machines. In X. Li, S. Wang, and Z. Dong, editors, *Advanced Data Mining and Applications*, volume 3584 of *Lecture Notes in Computer Science*, pages 731–731. Springer, 2005.