Gene Expression Classification with a novel coevolutionary based learning classifier system on Public Clouds

Christian Vecchiola[1], Mani Abedini, Michael Kirley, Xingchen Chu[1], and Rajkumar Buyya[1]
[1]Cloud Computing and Distributed Systems (CLOUDS) Laboratory
Dept. of Computer Science and Software Engineering
The University of Melbourne, Australia
{csve, mabedini, mkirley, xchu, raj}@csse.unimelb.edu.au

***Abstract*—Microarray technology allows for the simultaneous monitoring of thousands of genes expressions per sample. Unfortunately, the classification of these samples into distinct classes is often difficult as the number of genes (features) greatly exceeds the number of samples. Consequently, there is a need to investigate new, robust machine learning techniques capable of accurately classifying microarray data. In this paper, we present a coevolutionary learning classifier system based on feature set partitioning to classify gene expression data. A distributed implementation, which leverages Cloud computing technologies, is used to address the inherent computational costs of our model. The development and execution of this application was done using the Aneka middleware on the public Cloud (Amazon EC2) infrastructure. Experiments conducted using gene expression profiles demonstrates that the proposed implementation outperforms other well-known classifiers in terms of accuracy. Preliminary analysis into the impact of different Cloud setups on the performance of the classifier are also reported.**

I. Introduction

Gene expression technology using DNA microarrays, allows for the monitoring of the expression levels of thousands of genes at once. As such, they provide important insights into, and further our understanding of, biological processes. Consequently, they are key tools used in medical diagnosis, treatment and drug design [21].

The classification of gene expression data samples into distinct classes is a challenging task. The dimensionality of typical gene expression data sets ranges from several thousands to over ten thousands genes. However, only small sample sizes are typically available for analysis. [22]. " The curse of dimensionality" is the common issue in microarray gene expression data sets which may jeopardize the generalization ability of most machine learning approaches.

Learning classifier systems [12] are a widely used machine learning technique for classification problems. They generate a population of *condition-action rules*. The eXtended Classifier system (XCS) [20], a Michigan-style model, has been successfully tested on a variety of test case and real world applications. However, the effectiveness of XCS when confronted with high dimensional data sets (such as microarray gene expression data sets) has not been explored in detail. XCS like most other machine learning techniques, is vulnerable to the curse of dimensionality.

Cloud Computing [8] presents a cost effective approach for quickly harnessing the compute power required to carry out classification tasks without having a large distributed infrastructure in-house. It provides a wide collection of services that cover the entire computing stack from the hardware level to the software level, on a pay as you go basis. Users can elastically scale up and down their computing infrastructure and leverage the Cloud to conduct large scale experiments and use these facilities only for the time needed.

In this paper, we will describe a distributed implementation of the XCS classifier system – Cloud-CoXCS – and discuss how the system can be used to classify gene expression datasets on Public Clouds. Part of this paper has been presented in [17]. The contribution of the paper is twofold. Firstly, we investigate how our model can improve the accuracy of the classification for gene expression datasets. Secondly, we discuss the advantages of leveraging the Cloud, in particular the Amazon EC2 infrastructure, for computation by comparing different setups of testbed for our experiments.

The structure of the paper is as follows: Section 2 provides background information on the problem of classifying gene expression datasets, and a brief overview of Cloud Computing technologies. Section 3 describes Cloud-CoXCS and the components that characterize it. Section 4 describes the experiments conducted to evaluate the performance of Cloud-CoXCS and a discussion of the results obtained on two real datasets with different Cloud setups. Section 5 presents a brief overview of the related works and conclusions follow.

II. Background and related work

The reference context of this work is the field of multi-population learning classifier system. In this section we will briefly review the background context and the more relevant work to the Cloud-CoXCS.

Ritcher *et al.* [15] investigate the performance gain of decomposing the problem space in different sub tasks and assigning them to different XCS instances. In a similar study,

978-0-7695-4295-9/10 $26.00 © 2010 IEEE
DOI 10.1109/eScienceW.2010.24

a multi-population parallel XCS for classification of electroencephalographic signals was introduced by Skinner *et al.* [16]. The specific focus of that study was to investigate the effectiveness of migration strategies between sub-populations mapped to ring topologies.

Zhu and Guan [23] took the decomposition approach to the extreme. In the proposed coevolutionary model, individuals in isolated sub-populations encode *if–then* rules for each feature in the data set and are used to classify the partially masked training data corresponding to the feature in focus.

A. Classification for Gene Data Expressions

Gene-expression profiling using DNA microarrays can analyze multiple gene markers simultaneously. Consequently, it is widely used for cancer prediction.

Recently, there have been a number of investigations for class discovery of gene expression data sets using machine learning techniques: Decision Tree [4], [11], Support Vector Machines (SVM) [5], [13] and k-Nearest Neighbor (k-NN) [3].

B. XCS overview

The eXtended Classifier system (XCS) [20] is the most successful learning classifier systems based on an accuracy model. XCS maintains a population of classifiers and each classifier consist of a *condition-action-prediction* rule, which maps input features to the output signal (or class).

A ternary representation of the form 0,1,# (where # is don't care) for the condition and 0,1 for the action can be used. In addition, real encoding can also be used to accurately describe the environment states. Input, in the form of data instances (a vector of features or genes), is passed to the XCS. A match set $[M]$ is created consisting of rules (classifiers) that can be "triggered" by the given data instance. A covering operator is used to create new matching classifiers when $[M]$ is empty. A prediction array is calculated for $[M]$ that contains an estimation of the corresponding rewards for each of the possible actions. Based on the values in the prediction array, an action, a (the output signal), is selected. In response to a, the reinforcement mechanism is invoked and the prediction, p, prediction error, ϵ, accuracy, k, and fitness, F, of the classifier are updated [6] (see Figure 1).

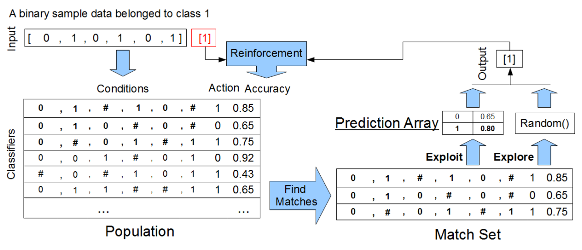

Fig. 1. XCS learning scenario for learning a binary row values.

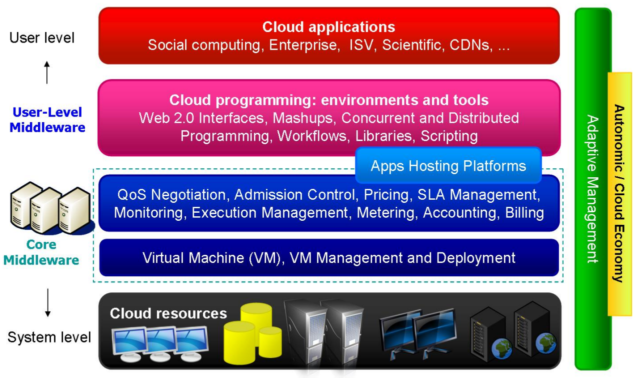
User level
User-Level Middleware
Core Middleware
System level
Cloud applications
Social computing, Enterprise, ISV, Scientific, CDNs, ...
Cloud programming: environments and tools
Web 2.0 Interfaces, Mashups, Concurrent and Distributed Programming, Workflows, Libraries, Scripting
Apps Hosting Platforms
QoS Negotiation, Admission Control, Pricing, SLA Management, Monitoring, Execution Management, Metering, Accounting, Billing
Virtual Machine (VM), VM Management and Deployment
Cloud resources
Adaptive Management
Autonomic / Cloud Economy

Fig. 2. Cloud Computing Architecture.

C. Cloud Computing

Cloud Computing is a broad term that describes how IT resources and software services are delivered to end users. Even though there is no widely accepted definition, a *Cloud* can be defined as a *type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreement* [8].

Figure 2, gives a layered architecture of the Cloud Computing. The lowest layer is characterized by the physical resources on top of which the infrastructure is deployed. These can be clusters, datacenters, and spare desktop machines. This level provides the *horse power* of the Cloud. The physical infrastructure is managed by the core middleware layer whose objectives are to provide an appropriate run time environment for applications and the maximum utilization of the physical resources. In order to provide advanced services, such as application isolation, quality of service, and sandboxing, the core middleware can rely on virtualization technologies. Together with the physical infrastructure the core middleware represents the platform on top of which the applications are deployed in the Cloud. This provides environments and tools simplifying the development and the deployment of applications in the Cloud: web 2.0 interfaces, command line tools, libraries, and programming languages. The user level middleware constitutes the access point of applications to the Cloud.

The commercial offerings for Cloud Computing are heterogeneous and address different customer needs. Among the major players in the field we can mention Google AppEngine, Microsoft Azure and Amazon EC2 and S3. Google AppEngine, and Microsoft Azure are integrated solutions providing both a computing infrastructure and a platform for developing applications.

The Cloud Computing model introduces several benefits for applications and enterprises: applications can dynamically acquire more resource to host their services in order to handle peak workloads and release when the load decreases. Enterprises do not have to plan for the peak capacity anymore, but they can provision additional resources on demand and for the time needed. Moreover, reduced administration and maintenance costs are implied by moving the IT infrastructure to the Cloud. On the other hand, the Cloud model introduces new challenges for what concerns the location of the information and the policies that are applied to maintain their confidentiality.

III. Cloud-CoXCS

Cloud-CoXCS, is a machine learning classification system which divide the the searching space into multiple sub searching spaces and assign an independent XCS to each one. Cloud-CoXCS benefits from running parallel XCSs on the Cloud infrastructure to speed up the learning process. Cloud-CoXCS is composed of three components: *CoXCS*, *Aneka*, and *Offspring*. In the remainder of the section, a brief overview of all these three components will be provided.

A. CoXCS

CoXCS is a coevolutionary learning classifier based on feature space partitioning [2]. It extends the XCS model by introducing a coevolutionary approach. Figure 3, provides a schematic example of how different classifiers learn from the feature space and interact with each other. The CoXCS architecture is based on a collection of independent populations of classifiers that are trained using different partitions of the feature space within the training dataset. The model uses a modified covering operator and crossover operators, which improves the generation of new classifiers during the evolutionary process. After a fixed number of iterations, selected classifiers from each of the independent populations are transferred to a different population, the evolutionary cycle is then repeated. This process continues until a specific accuracy threshold is reached.

B. Aneka

Aneka [18] is a platform for developing applications and deploying them on Clouds. It provides a runtime environment and a set of APIs that allow developers to build .NET applications that offload their computation on both public and private clouds. One of the key features of Aneka is the ability to support multiple programming models (ways of expressing the execution logic of applications by using specific abstractions). This is accomplished by creating a customizable and extensible service oriented runtime environment represented by a collection of software containers connected together. By leveraging this architecture, advanced services including resource reservation, persistence, storage management, security, and performance monitoring have been implemented. On top of this infrastructure, different programming models can be plugged to provide support for different scenarios such as engineering, life science, and business applications.

Figure 4, provides an overall view of the services and the internal architecture of the Aneka Container. A container is the building block of Aneka Clouds. It provides a collection of services that perform all the operations required by the

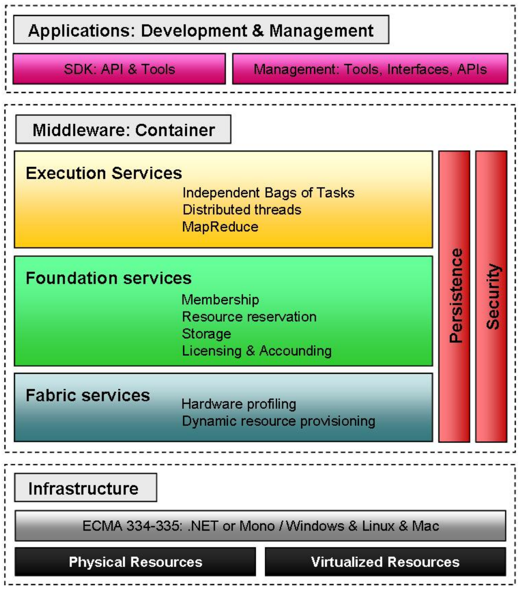

Fig. 4. Aneka Features Overview.

system: security, scheduling, job execution, and storage. The container can be deployed on either physical machine or virtual resources that are dynamically provisioned on demand by interacting virtual machine managers such as Amazon, VMWare, and Xen. On top of this architecture, three programming models are supported: independent bag of tasks (Task Model), distributed threads (Thread Model), and mapreduce (MapReduce Model). Developers can define their own abstraction for programming distributed applications with Aneka and simply configure the services required for the scheduling and the execution of the units of work.

The setup prepared for Cloud-CoXCS has been configured with the Task Model for the execution of the classification jobs. The Task Model provides a very simple set of abstractions that allows developers to define a sequence of unrelated tasks that do not have precedence or sequencing constraints. By using the Task Model it is possible to wrap existing legacy applications or also implement new tasks with any language supported by the .NET runtime. In the case of Cloud-CoXCS the existing CoXCS application has been packaged into a legacy task and remotely executed.

C. Offspring

Offspring [19] is a software tool that allows scientists and developers to quickly prototype distributed applications. By using the APIs provided by Offspring, developers can: i) define the concept of task that will be remotely executed; ii) define and implement the logic that coordinates the distributed execution of tasks; and iii) offload the execution of distributed applications on different distributed systems. Offspring provides a simple model based on the independent bag of tasks for structuring distributed applications. It encapsulates the logic of creating and coordinating the execution of tasks into *strategies*.

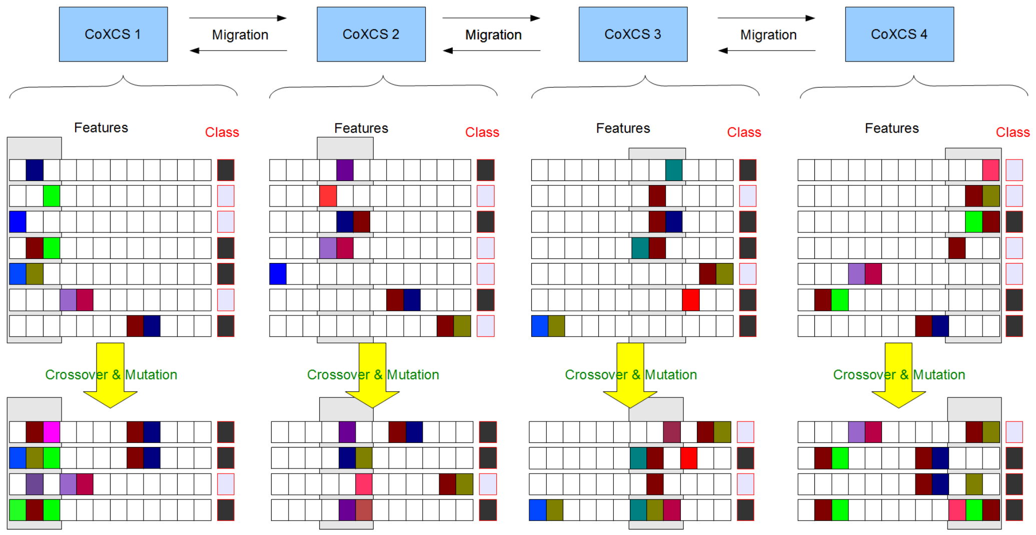

Fig. 3. High level overview of feature paritioning policy in the CoXCS model.

Strategies are programmable client-side workflows that developers can define and plug into the environment. By defining a strategy, developers can coordinate the execution of existing legacy applications, as in the case of Cloud-CoXCS, or implement more sophisticated models by implementing their own tasks. A strategy is composed of a sequence of phases in which a collection of tasks is generated. Each of these tasks are submitted through Offspring and executed remotely. Their successful completion (or failure) can trigger the generation of additional tasks within the same phase or move strategy to the next phase. It is possible to model either simple parameter sweeping applications or complex dynamic workflows.

In the case of Cloud-CoXCS, a multi-phase strategy has been implemented. In each phase, a number of parallel learning tasks are generated. The output of a learning task is a population of classifiers that have been trained against a given dataset. Once all the learning tasks complete, an additional task that applies migration among the population of classifiers will be submitted. It sets the completion of the phase once its execution finishes. This process is repeated for a specified number of iterations decided by the user. Algorithm 1 describes in detail the strategy.

IV. Experiments

In order to evaluate the performance of the Cloud-CoXCS, a set of experiments have been conducted using different gene expression datasets and Cloud setups.

A. Datasets

Two datasets were considered in this study: **BRCA** (Breast Cancer gene profiles) data set contains BRCA (15 samples) and Sporadic (7 samples) which each sample is described by 24,481 genes (features) [10], [11], and **Prostate** that is collected from 21 prostate cancer patients with 12600 genes [11].

B. Methods

1) CoXCS parameters: CoXCS with a hybrid feature encoding scheme was implemented and integrated in Aneka and Offspring frameworks. The parameter settings for our modified XCS were based on the default XCS settings recommended in [7]. The parameter values that were different include: population sizes of 5000; the exploration/exploitation rate was set to 0.3. The partitioning scheme used was a simply equal linear division of the feature space. In this study, we have employed 20 separate partitions (islands) for all data sets. The migration ratio was set to 10% of the population size. Five separate migration stages were used, where the number of iterations between migration episodes was fixed at 100.

2) Cloud setup: The experiments have been carried out via distributed infrastructure managed by Aneka and deployed on Amazon EC2 virtual instances. Two different Amazon images have been used to configure the system: a master image and a slave image. The master image features an instance of the Aneka container hosting the scheduling and file staging services for the Task Model on a Windows Server 2003 operating system, while the slave image hosts a container configured with the corresponding execution services deployed on a Red Hat Linux 4.1.2 (kernel: 2.6.1.7) .

The Aneka Cloud deployed for the experiments is composed by one master node and multiple slave nodes that have been added to the cloud on demand. Experiments have been done to compare different cloud setups. In particular two different image types have been tested to deploy slave instances:

Algorithm 1 XCSStrategy

```
Require: $p$: number of phases (migration stages)
         $e$: number of parallel evaluations (XCS instances)
         $d$: gene expression dataset
         $failed$: list of failed executions
         $input$: list of input populations
         $output$: list of output populations
         $AvgAUC$: average accuracy
         $MaxAUC$: minimu accuracy
         $MaxAUC$: max accuracy
         $best$: best population
 1: for $i$ = 0 to $p$ do
 2:     clear $output$
 3:     create $e$ instances of classifiers task $cj$
 4:     partition $d$ into $e$ sets and assigns each set to $cj$
 5:     if $i > 0$ then
 6:         for $j$ = 0 to $e$ do
 7:             configure $cj$ with population $pj$ in $input$
 8:         end for
 9:     end if
10:     submit the list of classifiers to the cloud.
11:     for $j$ = 0 to $e$ do
12:         if $cj$.Success then
13:             add output population $pj$ to $output$
14:             update $AvgAUC$, $MaxAUC$, $MinAUC$
15:             if $cj$.AUC $\neq MaxAUC$ then
16:                 $best \leftarrow pj$
17:             end if
18:         else
19:             add $cj$ to $failed$
20:         end if
21:     end for
22:     if $i < p$ then
23:         create mixer task $mi$ that takes as input all the $pj$ stored
            in $output$.
24:         submit the mixer task to the cloud
25:         collect $pj$ generated and add it to (or replace it into) $input$
26:     end if
27: end for
28: return $best$
```

m1.small and *c1.medium*. For what concerns the master node, the *m1.small* image has been used in both cases.

Table I describes the characteristics of the two different clouds used for the experiments. It can be noticed that *c1.medium* instances provide a computational power that is double compared with the one provided by *m1.small* and exposed as a two core machine. The computing power is expressed in EC2 Compute Units[1]. In both cases a complete parallelism at each stage is obtained because Aneka scheduler dispatches one task per core. Hence *c1.medium* instances will receive two tasks to process each time.

[1]An EC2 Compute Unit is a virtual metric that is used to express the computational power of an instance. One EC2 Compute Unit (ECU) provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.

Experiment	Image Type	Cores	EC2 Units	Memory	Slave Instances	Cost/Hour
Exp 1	m1.small	1	2.5	1.7 GB	20	0.10 USD
Exp 2	c1.medium	2	5	1.7 GB	10	0.20 USD

TABLE I
EXPERIMENTS SETUP. VIRTUAL MACHINE CHARACTERISTICS FOR SLAVE NODES.

3) Validation: Cross-validation is a standard approach when running experiments for both bioinformatics and machine learning tasks (2 fold for the BRCA dataset and 4 fold for the prostate dataset). For each fold, the area under the ROC curve (AUC) is calculated (this is a well-known machine learning technique used to compare the accuracy of different techniques).

We have also included results generated using other well-known classifiers. There results were generated using the Weka package [1].

C. Result and Analysis

Table II lists the accuracy results obtained for each of the datasets examined. The results obtained using other well-known classification methods are also listed. For each of the classifiers, the average AUC value obtained against the test data has been included. The relative performance of the base-line XCS and the other classifiers were very similar. The accuracy performance of the CoXCS was generally better than other classifiers. However, there is still room for improvement.

Table III, shows the average execution time comparison over different Cloud setups. The CoXCS execution times recorded for the Prostate dataset are approximately four times longer than the execution time for the BRCA dataset. This may be attributed to the different number of features that characterize the two gene profiles, giving the approximately equal number of samples. It is interesting to note, that the setup using the dual core machines performs slightly worse in the case of BRCA while it provides a significant drawback in the case of the Prostate profile. As the number of attributes per partition is approximately four times larger in the second case, the single CoXCS task requires more time to complete, and in the case of a dual core machines, the presence of two learning tasks

Classifier	Mode	BRCA	Prostate
j48	Train	0.92±0.06	1.00
	Test	0.35±0.01	0.60±0.10
NBTree	Train	1.00	1.00
	Test	0.65±0.12	0.46±0.04
Random Forest	Train	1.00	1.00
	Test	0.51±0.01	0.60±0.09
Logistic Regression	Train	1.00	0.50
	Test	0.85±0.17	0.50
Naive Bayes Classifier	Train	0.99±0.01	1.00
	Test	0.90±0.05	0.35±0.04
SVM	Train	1.00	1.00
	Test	0.53±0.04	0.51±0.07
XCS	Train	0.50	0.50
	Test	0.50	0.50
Cloud-CoXCS	Train	1.00	1.00
	Test	**0.98±0.02**	**0.70±0.02**

TABLE II
AUC RESULTS. BOLD VALUES INDICATE THE THE CLOUD-COXCS MODEL WAS SIGNIFICANTLY BETTER WHEN COMPARED TO ALL OF THE OTHER CLASSIFIERS.

Setup	BRCA				Prostate	
	Fold 0	Fold 1	Fold 2	Fold 3	Fold 0	Fold 1
m1.small	08:26	10:00	10:00	09:00	35:13	40:44
c1.medium	10:42	10:04	15:17	11:42	52:48	53:48

TABLE III
EXPERIMENTS RESULT. EXECUTION TIME (MINUTES).

executing at the same time implies a longer execution time for both of them. This effect is not noticeable in the case of the BRCA profile, where the single learning task is very quick. Since the overall cost of the two setups is the same, it is possible to conclude that in case of long running computing intensive tasks, given the same number of cores, it is better to rely on a setup that uses as many as *m1.small* instances rather than half *c1.medium* instances.

V. CONCLUSIONS

In this paper, we have presented Cloud-CoXCS, a system for performing gene expression dataset classification on Public Clouds. Cloud-CoXCS is a system that provides a distributed implementation of the CoXCS (coevolutionary learning classifier based on feature space partitioning). It relies on the Aneka Cloud Computing platform and the Offspring environment in order to harness on demand the computing power offered by Public Clouds. The Offspring environment provides a mechanism to prototype distribution strategies, which coordinate the logic of the execution and connect them with the selected distribution middleware.

The experiments performed, using the Amazon Elastic Compute Cloud (EC2) infrastructure, have demonstrated that by using Cloud-CoXCS it is possible to obtaining a suitable Computing Cloud platform for high-dimensional classification problems.

In order to investigate the advantage of performance on a cloud characterized by the same number of *m1.small* instances, and on a cloud of *c1.medium* instances whose number was half of the previous one. The experimental results have demonstrated that in case of computationally intensive tasks, the execution time plays a critical role in determining the performance of the cloud. More precisely, for very short tasks there is no different between the two setups, but for long running tasks the setup characterized by dual core machines is less performance.

In future work, we will conduct more detailed experiments investigating distributed classification based on CoXCS by considering different Cloud computing resource pools created using technologies such as Xen and VMWare, and the integration as being supported by Aneka.

REFERENCES

[1] Weka 3: Data Mining Software in Java. http://www.cs.waikato.ac.nz/ml/weka/.
[2] M. Abedini and M. Kirley. CoXCS: A Coevolutionary Learning Classifier Based on Feature Space Partitioning. Volume 5866/2009:360–369, 2009.
[3] D. W. Aha, D. F. Kibler, and M. K. Albert. Instance-Based Learning Algorithms. *Machine Learning*, 6:37–66, 1991.
[4] M. Beibel. Selection of Informative Genes in Gene Expression Based Diagnosis: A Nonparametric Approach. In *ISMDA '00: Proceedings of the First International Symposium on Medical Data Analysis*, pages 300–307, London, UK, 2000. Springer-Verlag.
[5] M. P. Brown, W. N. Grundy, D. Lin, N. Cristianini, C. W. Sugnet, T. S. Furey, M. Ares, and D. Haussler. Knowledge-based analysis of microarray gene expression data by using support vector machines. *PNAS*, 97(1):262–267, January 2000.
[6] M. V. Butz, T. Kovacs, P. L. Lanzi, and S. W. Wilson. Toward a theory of generalization and learning in XCS. *Evolutionary Computation, IEEE Transactions on*, 8(1):28–46, 2004.
[7] M. V. Butz and S. W. Wilson. An Algorithmic Description of XCS. In *Advances in Learning Classifier Systems*, volume 1996/2001 of *Lecture Notes in Computer Science*, pages 267–274. Springer Berlin / Heidelberg, 2001.
[8] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud Computing and emerging IT platforms: vision, hype, and reality for delivering IT services as the 5th utility. *Future Generation of Computer Systems*, 25:599–616, 2009.
[9] M. Gershoff and S. Schulenburg. Collective behavior based hierarchical XCS. In *Genetic and Evolutionary Computation Conference, GECCO 2007*, volume ACM, pages 2695–2700, New York, NY, USA, 2007. ACM.
[10] I. Hedenfalk, D. Duggan, Y. Chen, M. Radmacher, M. Bittner, R. Simon, P. Meltzer, B. Gusterson, M. Esteller, O. P. Kallioniemi, B. Wilfond, A. Borg, and J. Trent. Gene-Expression profiles in hereditary breast cancer. *N Engl J Med*, 344(8):539–548, February 2001.
[11] M. M. Hossain, M. R. Hassan, and J. Bailey. ROC-tree: A Novel Decision Tree Induction Algorithm Based on Receiver Operating Characteristics to Classify Gene Expression Data. In *Proceedings of the SIAM International Conference on Data Mining*, pages 455–465, 2008.
[12] P. L. Lanzi. Learning classifier systems: then and now. *Evolutionary Intelligence*, 1(1):63–82, March 2008.
[13] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. pages 185–208, 1999.
[14] M. A. Potter and K. A. D. Jong. Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.
[15] U. Richter, H. Prothmann, and H. Schmeck. Improving XCS Performance by Distribution. In *Simulated Evolution and Learning, 7th International Conference, SEAL 2008*, volume 5361 of *Lecture Notes in Computer Science*, pages 111–120, December 2008.
[16] B. Skinner, H. Nguyen, and D. Liu. Distributed classifier migration in XCS for classification of electroencephalographic signals. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007*, pages 2829–2836. IEEE, September 2007.
[17] C. Vecchiola, M. Abedini, M. Kirley, X. Chu, and R. Buyya. Classification of Gene Expression Data on Public Clouds . Technical report, 2009.
[18] C. Vecchiola, X. Chu, and R. Buyya. *High Performance & Large Scale Computing*, chapter Aneka: A Software Platform for .NET-based Cloud Computing. IOS Press, 2009.
[19] C. Vecchiola, M. Kirley, and R. Buyya. Multi-Objective Problem Solving With Offspring on Enterprise Clouds. In *Proc. of the 10th International Conf. on High-Performance Computing in Asia-Pacific Region (HPC Asia'09)*, 2009.
[20] S. W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995. http://prediction-dynamics.com/.
[21] F.-X. Wu, W. Zhang, and A. Kusalik. On Determination of Minimum Sample Size for Discovery of Temporal Gene Expression Patterns. In *First International Multi-Symposiums on Computer and Computational Sciences*, pages 96–103, June 2006.
[22] Y. Zhang and J. C. Rajapakse. *Machine Learning in Bioinformatics*. Wiley Series in Bioinformatics. 1'st edition, 2008.
[23] F. Zhu and S. Guan. Cooperative co-evolution of GA-based classifiers based on input decomposition. *Engineering Applications of Artificial Intelligence*, 21:1360–1369, 2008.