Data Mining in Learning Classifier Systems: Comparing XCS with GAssist

Jaume Bacardit[1] and Martin V. Butz[2]

[1] ASAP, School of Computer Science and IT, University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK,
jqb@cs.nott.ac.uk,
WWW home page: http://www.cs.nott.ac.uk/ jqb/

[2] Department of Cognitive Psychology, University of Würzburg, 97070 Würzburg, Germany,
butz@psychologie.uni-wuerzburg.de,
WWW home page: http://www-illigal.ge.uiuc.edu/ butz/

Abstract. This paper compares performance of the Pittsburgh-style system GAssist with the Michigan-style system XCS on several datamining problems. Our analysis shows that both systems are suitable for datamining but have different advantages and disadvantages. The study does not only reveal important differences between the two systems but also suggests several structural properties of the underlying datasets.

1 Introduction

Successful data mining applications are important for modern-day learning classifier systems (LCSs). Additionally, the study and comparison of different types of data miners on various data sets may enable the identification of strengths and weaknesses of the respective data miners. Several types of problem difficulty can be distinguished in data mining including data volume, search space size and type, complexity of the concept, noise in the data, the handling of missing values, or the problem of over-fitting.

Successful datamining applications of learning classifier systems have been shown in the past (**?**) investigating and comparing performance of the accuracy-based Michigan-style LCS XCS (**?**) and the Pittsburgh-style LCS GALE (**?**). Both systems showed competent performance in comparison to six other machine learning systems.

Recently, new systems have appeared in the LCS field, like the Pitt-style LCS GAssist (**?**). Also, there are improved versions of already established systems, like the XCS with tournament selection (**?**). The objectives of this paper are two-fold: (1) We provide further performance results of GAssist and XCS on several interesting datasets. (2) We compare and investigate performance of the two systems revealing problem dependencies, suitability of the respective approaches, as well as over-fitting or over-generalization tendencies.

2 Framework

Before we start with the datamining analysis, this section provides a short introduction to the LCSs under investigation.

2.1 GAssist

GAssist (?) is a Pittsburgh genetic-based machine learning system descendant of *GABIL* (?). The system applies a near-standard *GA* that evolves individuals that represent complete problem solutions. An individual consists of an ordered, variable-length rule set. Bloat control is achieved by a combination of a fitness function based on the minimum description length (*MDL*) principle and a rule deletion operator (?).

The knowledge representation used for real-valued attributes is called *adaptive discretization intervals* rule representation (*ADI*) (?). This representation uses the semantics of the *GABIL* rules (conjunctive normal form predicates), but applies non-static intervals formed by joining several neighbor discretization intervals. These intervals can evolve through the learning process splitting or merging among them potentially using several discretizers at the same time.

The system also uses a windowing scheme called *ILAS* (incremental learning with alternating strata) (?). This scheme stratifies the training set into s subsets of equal size and approximately uniform class distribution. Each *GA* iteration uses a different strata to perform its fitness computation, using a round-robin policy. This method showed to introduce an additional implicit generalization pressure to GAssist. [3]

Figure 1 presents the pseudocode of *ILAS*. This kind of scheme is reported to apply some extra generalization pressure to the system, which is an interesting feature for data mining domains.

2.2 XCS

The XCS classifier system (?; ?; ?) evolves online a set of condition-action rules, that is, a *population* of *classifiers*. In difference to GAssist, in XCS the population as a whole represents the problem solution. XCS differs in two fundamental ways to other Michigan-style LCSs: (1) Rule fitness is derived from rule accuracy

[3] GAssist's parameters were set as follows: Crossover probability 0.6; tournament selection; tournament size 3; population size 400; probability of mutating an individual 0.6; initial number of rules per individual 20; probability of "1" in initialization 0.75; Rule Deletion Operator: Iteration of activation: 5; minimum number of rules: number of classes of domain +3; MDL-based fitness function: Iteration of activation 25; initial theory length ratio: 0.075; weight relax factor: 0.9. ADI knowledge representation: split and merge probability: 0.05; reinitialize probability at initial iteration: 0.02; reinitialize probability at final iteration: 0; merge restriction probability: 0.5; maximum number of intervals: 5; set of uniform discretizers used: 4, 5, 6, 7, 8, 10, 15, 20 and 25 bins; iterations: maximum of 1500. Results are averaged over 150 experiments.

Fig. 1. Pseudocode of the incremental learning with alternating strata (ILAS) scheme

```
**Procedure** Incremental Learning with Alternating Strata
**Input** : $Examples, NumStrata, NumIterations$
Initialize $GA$
Reorder $Examples$ in $NumStrata$ parts of approximately
equal class distribution
$Iteration = 0$
$StrataSize = size(Examples)/NumStrata$
**While** $Iteration < NumIterations$
    **If** $Iteration = NumIterations - 1$ **Then**
        $TrainingSet = Examples$
    **Else**
        $CurrentStrata = Iteration$ mod $NumStrata$
        $TrainingSet=$ examples from
            $Examples[CurrentStrata \cdot StrataSize]$ to
            $Examples[(CurrentStrata + 1) \cdot StrataSize]$
    **EndIf**
    Run one iteration of the $GA$ with $TrainingSet$
    $Iteration = Iteration + 1$
**EndWhile**
**Output** : Best individual (set of rules) from $GA$ population
```

instead of rule reward prediction. (2) GA selection is applied in the subsets of currently active classifiers resulting in an implicit pressure towards more general rules.

Due to the variable properties of the investigated datasets including real values, nominals, and binary features, we use a hybrid XCS/XCSR approach that can handle any feature combination as done before in (**?**). Additionally, we apply tournament selection, which proved to result in more robust fitness pressure toward accurate rules (**?**). In the investigated problems, a reward of 1000 is provided if the classification is correct, and 0 otherwise. [4]

3 Experiments

3.1 Setup

In Table 1 we show the most important properties of the datasets we have selected from the University of California at Irvine (UCI) repository (**?**). The selected datasets are:

[4] XCS's parameters are set as follows: $N = 6400$, $r_0 = 4(100)$, $P_{\#} = 0.6$, $\beta = 0.2$, $\chi = 1.0$ applying uniform crossover, $\mu = 0.04$, $m_0 = 0.2$, $\theta_{GA} = 48$, $\tau = 0.4$, $\varepsilon_0 = 1$, $\delta = 0.1$, $\theta_{del} = 50$, GA Subsumption is applied with $\theta_{sub} = 50$. Experiments are run applying either 100,000 learning steps (averaging over 150 experiments) or 500,000 learning steps (averaging over 20 experiments).

- Annealing Data (*ann*)
- 1985 Auto Imports Database (*aut*)
- Balance Scale Weight & Distance (*bal*)
- Contraceptive Method Choice (*cmc*)
- Horse Colic (*col*)
- German Credit (*cr-g*)
- Glass Identification (*gls*)
- Cleveland Heart Disease (*h-c*)
- Hungarian Heart Disease (*h-h*)
- Johns Hopkins University Ionosphere database (*ion*)
- Sonar, Mines vs. Rocks database (*son*)
- Wisconsin Breast Cancer database (*wbcd*)
- Wisconsin Diagnostic Breast Cancer (*wdbc*)

The selection of datasets gives a representative overview over the phenomena we were able to detect while comparing GAssist with XCS.

Table 1. The dataset properties indicate complexity, size, and data distributions in the respective datasets. #Inst. = Number of Instances, #Attr. = Number of attributes, #Real = Number of real-valued attributes, #Nom. = Number of nominal attributes, #Cla. = Number of classes, Dev.C = Deviation of class distribution, Maj.C. = Percentage of instances belonging to the majority class, Min.C. = Percentage of instances belonging to the minority class, MV I. = Percentage of instance with missing values, MV A. = Number of attributes with missing values, MV V. = Percentage of values ($\#instances \cdot \#attr$) with missing values

Dataset Properties											
Name	#Inst	#Attr	#Real	#Nom	#Cla	Dev.C	Maj.C	Min.C	MV I	MV A	MV V
ann	898	38	6	32	5	28.28	76.17	0.89	—	—	—
aut	205	25	15	10	6	10.25	32.68	1.46	22.44	7	1.11
bal	625	4	4	—	3	18.03	46.08	7.84	—	—	—
cmc	1473	9	2	7	3	8.26	42.70	22.61	—	—	—
col	368	22	7	15	2	13.04	63.04	36.96	98.10	21	22.77
cr-g	1000	20	8	12	2	20.00	70.00	30.00	—	—	—
gls	214	9	9	—	6	12.69	35.51	4.21	—	—	—
h-c1	303	13	6	7	2	4.46	54.46	45.54	2.31	2	0.17
h-h	294	13	6	7	2	13.95	63.95	36.05	99.66	9	19.00
ion	351	34	34	—	2	14.10	64.10	35.90	—	—	—
son	208	60	60	—	2	3.37	53.37	46.63	—	—	—
wbcd	699	9	9	—	2	15.52	65.52	34.48	2.29	1	0.23
wdbc	569	30	30	—	2	12.74	62.74	37.26	—	—	—

The test design for GAssist has two goals: Comparing the effect of using both different number of iterations and different degrees of generalization pressure. The latter goal is achieved by using the *ILAS* windowing scheme. However, our goal is not run-time reduction, but rather the maximization of the generalization pressure introduced by the *ILAS* scheme. Thus, we will increase the number of iterations when using windowing proportional to the number of strata used. This means having constant number of learning steps (using the Michigan-LCS meaning of the term). We will also test another stratified setup using a number of iterations that makes it equivalent in run-time compared to the non-windowed setting (1 strata).

3.2 Results

Results of GAssist and XCS are shown in Table 2. The comparison is not meant to determine which system is better in general but rather to show in which problem types which system appears to have advantages. Our comparison starts with a general data observation and then investigates separate datasets with respect to specific phenomena.

A look at the overall performance shows that XCS and GAssist show comparative performance results indicating the general difficulty of the respective datasets. XCS tends to learn the training data much more precisely which however is not necessarily advantageous for performance on the test data (using stratified ten-fold cross-validation). The solution representation differs (as expected) very significantly between GAssist and XCS: The number of rules in the best individual in GAssist is much smaller than the number of rules in XCS. However, it should be noted that GAssist maintains 400 individuals and thus the overall number of rules is actually similar to the number of rules in XCS. While we did not make explicit speed comparisons it appears that XCS runs take longer than GAssist's. Again, this is expectable since XCS is an online learner that learns from each problem instance separately and iteratively. Thus, the number of necessary learning iterations are higher.

Taking a closer look at the particular datasets we see that in the anneal (ann) dataset, performance of both systems reaches a similar level if XCS is run long enough. As also indicated by XCS's smaller population size in longer runs, generalization appears important and requires sufficient learning time. Generalization is even more important in the autos (aut) problem indicated by XCS's poor performance when starting specific and its improved test performance and smaller population size in longer runs as well as in GAssist's slight performance improvement and rule number decrease when using three strata. Additionally, the higher population size of XCS compared to the anneal problem indicates a general higher complexity of the problem. Balance-scale (bal) is a typical problem which can be over-fitted easily: XCS's performance is worse when starting more specific and when performing longer runs. Note that the population size of XCS actually increases when starting general and running more iterations—a clear indication of over-fitting. GAssist's performance points in the same direction in that generalization can slightly improve performance but longer runs are not helpful. The cmc problem appears to be a tough problem in general. XCS over-fits the data more than GAssist showing higher train performance but worse test performance. In the colic (col) as well as in the heart-h (h-h) problem, performance of XCS is significantly worse compared to GAssist. The major reason for this appears to be the missing value policy. While in GAssist a missing value is replaced by the majority value for the nominal case or by the average value in the real-valued case, XCS assumes a match in the missing value case. The

Table 2. Train and test performance results of GAssist and XCS using 10-folded cross-validation. Besides the performance results, we show the number of rules in the best individual of GAssist and the number of (macro-)classifiers in XCS (at the end of a run). The different GAssist runs distinguish a different application of strata (1 vs. 3 strata) as well as number of iterations (609, 1827, and 1447, respectively). In XCS, we compare long (500,000 learning iterations) and short learning runs (100,000 learning iterations) as well as a general ($r_0 = 100$) and specific ($r_0 = 4$) initialization of the population.

		GAssist			XCS (500,000)		XCS (100,000)	
Data	Res.	1 strata	3 s.(steps)	3 s.(time)	$r_0 = 100$	$r_0 = 4$	$r_0 = 100$	$r_0 = 4$
ann	Train	97.4±2.2	97.8±3.3	97.9±2.5	99.6±.46	100±.18	94.3±2.0	98.9±.61
	Test	97.0±2.6	97.4±3.5	97.5±2.8	98.4±1.6	98.6±1.5	91.2±2.7	91.7±2.9
	#rules	6.9±.9	6.3±.7	6.3±.5	2507±232	3211±146	4440±87	5426±51
aut	Train	85.5±2.9	84.7±3.2	82.8±3.7	99.8±.23	99.6±.39	99.3±.67	99.4±.56
	Test	67.5±9.8	68.8±9.7	67.5±9.5	71.5±9.5	68.8±12	64.7±9.6	13.4±6.9
	#rules	12.8±2.7	7.8±1.1	7.8±1.0	3403±98.5	4679±217	4281±87.3	5426±36.9
bal	Train	87.7±.49	86.0±.69	85.9±.73	98.4±.72	98.6±.64	90.6±2.2	97.9±.86
	Test	79.0±4.2	78.8±3.8	79.2±4.4	81.4±3.6	81.0±3.8	84.6±3.3	82.0±3.5
	#rules	13.1±2.0	9.6±1.6	9.8±1.6	2061±73.2	2014±59.8	1611±169	2465±65.9
cmc	Train	59.8±.96	59.6±1.1	59.8±1.1	70.5±1.9	77.6±2.0	57.0±1.8	71.5±2.2
	Test	54.8±4.2	54.6±4.0	54.9±4.1	53.6±4.0	52.9±4.7	50.1±4.7	53.6±3.6
	#rules	7.7±1.4	9.3±3.0	9.1±2.9	3261±88.1	3210±84.3	3958±91.4	3929±64.7
col	Train	99.7±.34	99.6±.48	99.5±.50	94.6±1.2	95.5±1.3	91.7±1.6	95.0±1.1
	Test	93.0±4.7	93.8±4.6	94.1±4.3	84.4±5.0	83.7±5.8	84.5±5.8	84.8±5.6
	#rules	7.4±1.6	7.0±1.4	7.0±1.4	3102±156	3685±84.2	3612±169	4100±96.3
cr-g	Train	82.0±.76	83.7±.94	84.3±.83	98.2±1.2	99.6±.34	89.7±3.2	94.4±1.4
	Test	72.3±3.6	72.0±4.2	72.2±3.8	70.2±3.6	72.3±4.2	71.4±3.9	72.5±3.1
	#rules	6.8±1.5	11.3±3.0	13.1±2.1	2016±69.2	2623±75.4	3217±106	4401±104
gls	Train	82.1±1.8	80.4±1.9	79.9±1.8	98.8±.64	99.6±.67	89.7±2.8	96.6±1.4
	Test	68.2±9.3	69.4±9.2	68.4±9.9	74.7±7.7	71.2±8.7	70.7±8.2	70.7±8.4
	#rules	8.8±1.4	6.6±0.8	6.6±0.8	1808±86.5	2143±78.4	3093±134	3137±92.7
h-c1	Train	93.4±.82	91.4±.98	92.6±.86	99.9±.25	100±0.0	99.5±.46	100±0.0
	Test	80.2±7.0	80.0±6.8	80.28±6.5	76.4±6.7	79.6±6.5	77.7±6.8	68.9±8.6
	#rules	9.3±1.5	6.9±1.1	7.4±1.2	2043±69.2	2808±89.6	2854±99.6	2907±68.2
h-h	Train	99.7±.32	99.0±.48	99.0±.50	99.7±.44	100±0.0	95.4±2.2	100±0.0
	Test	95.5±4.4	95.7±4.4	95.8±3.3	78.7±9.0	76.6±6.9	79.4±7.7	70.8±6.9
	#rules	6.1±0.7	6.3±0.5	6.0±0.2	2072±103	2686±71.9	3091±136	2861±67.5
ion	Train	98.2±.46	96.8±.63	96.8±.59	99.9±.19	99.7±.41	99.7±.32	99.8±.34
	Test	92.5±4.9	92.7±4.7	93.0±4.8	89.3±4.8	57.4±6.4	90.7±5.3	57.1±6.8
	#rules	3.9±0.8	2.2±0.7	2.2±0.8	2935±93.5	5613±28.9	3479±97.6	5685±31.4
son	Train	97.0±1.0	96.6±1.2	96.3±1.2	100±0.0	100±0.0	99.9±.30	100±0.0
	Test	74.4±8.9	76.8±9.0	77.5±9.2	78.4±7.4	82.6±8.3	77.3±8.1	81.6±7.9
	#rules	8.3±1.4	6.8±1.1	6.9±1.1	4959±120	4168±142	5148±107	4473±89.8
wbcd	Train	99.1±.27	97.8±.50	97.9±.47	99.8±.24	100±0.0	97.7±.89	99.9±.13
	Test	95.2±2.9	96.1±2.6	96.0±2.4	96.1±2.8	96.2±2.2	96.2±2.2	96.5±1.9
	#rules	5.0±1.0	2.4±0.6	2.4±0.6	1562±96.8	2131±52.9	1108±144	3137±81.8
wdbc	Train	98.6±.5	97.6±.68	97.6±.78	100±.09	100±0.0	99.8±.22	99.9±.24
	Test	94.1±3.0	94.2±2.9	94.1±2.8	96.1±2.5	96.7±2.2	95.9±2.6	92.9±3.3
	#rules	6.0±1.3	3.8±0.7	3.9±0.9	4104±112	5051±50.9	4485±85.5	5551±87.7

latter strategy appears mediocre in the investigated data mining experiments explaining XCS's poor performance in these settings.

Performance in the credit-g problem (cr-g) indicates that over-fitting is unlikely but in order to reach higher performance more specific initialization is helpful. Again, XCS reaches a much higher train performance but test performance is hardly influenced.

XCS's behavior in the glass problem (gls) is similar to that of credit-g. However, generalization is more important as also indicated by the performance improvement in GAssist when using three strata. Similar to the autos problem, XCS outperforms GAssist in the glass problem indicating higher problem complexity which might partially stem from the large number of classes in the problem.

XCS's performance in heart-c1 (h-c1) is actually very similar to the performance in in heart-h (h-h) suggesting that besides the problem of missing values in heart-h, XCS tends to strongly over-fit the training data. GAssist does not suffer from this problem in these datasets.

Another interesting observation was made in the ionosphere problem (ion) in which the automatic default rule detection mechanism in GAssist is actually able to discover that the minority class results in a better problem performance. XCS tends to over-fit as indicated by the poor performance and large population size when starting too specific.

On the other hand, in the sonar problem (son) a start from the specific side is actually beneficial for XCS suggesting small special-case niches which can be separated only if the population is initialized more specific. The more generalized representation of GAssist is not advantageous in this dataset.

In the Wisconsin breast-cancer dataset (wbcd) performance of both systems is similar and the problem appears to be generally easy as indicated by the small number of rules in both systems.

Finally, wdbc is another problem in which the complexity of the problem makes it hard for GAssist to reach XCS's performance level. XCS needs a large number of classifiers to solve the problem but is able to evolve the appropriate set. Slight generalizations are possible. GAssist on the other hand learns a very general—but slightly over–general solution.

4 Summary and Conclusions

In sum, both LCS systems showed that they are suitable for data-mining applications developing very different problem solutions that nonetheless perform similarly well on the test sets. Additionally, the comparison showed that regardless of offline (GAssist) or online (XCS) learning, LCSs are suitable data-miners.

The results allowed us to infer problem properties as well as problem difficulties. We saw that the current policy of handling missing values in XCS can affect performance negatively. Also, while GAssist has the tendency to ignore additional problem complexity, XCS tends to over-fit the training data more often (dependent on the nature of the data). Additionally, GAssist has slight problems with handling many output classes as well as a huge search space suggesting the addition of special covering operators that could ensure that each individual in GAssist differentiates at least all classes in the problem at hand. On the other hand, XCS's generalization tendency needs to be revisited in the data-mining domain. Especially in smaller datasets, XCS clearly tends to over-fit the data. Due to the small size of the datasets, the natural generalization pressure due to the niche reproduction mechanism hardly applies. Thus, additional pressure towards syntactic generality becomes more important and may be reconsidered in these problem domains.

The insights gained from our study prepare the systems for a more general problem application suggesting initial testing with each learning approach for suitability and appropriate initialization. XCS may need to be improved in terms of generalization to avoid over-fitting. GAssist may be endowed with further covering mechanism to ensure that all problem classes are covered by each individual and that it is able to detect additional small but significant problem subspaces.

Acknowledgments

The authors would like to thank Professor David E. Goldberg and the whole IlliGAL lab for their support and advise during this work. Support from the following sources is acknowledged: the Spanish Research Agency (CICYT) under grant numbers TIC2002-04160-C02-02 and TIC 2002-04036-C05-03; the Department of Universities, Research and Information Society (DURSI) of the Autonomous Government of Catalonia under grants 2002SGR 00155 and 2001FI 00514; the German research foundation (DFG) under grant DFG HO1301/4-3; the European commission contract no. FP6-511931; the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-03-1-0129; the Computational Science and Engineering graduate option program (CSE) at the University of Illinois at Urbana-Champaign.