UCSpv: Principled Voting in UCS Rule Populations

Gavin Brown
University of Manchester
School of Computer Science
Kilburn Building, Oxford Road,
Manchester M13 9PL
gbrown@cs.man.ac.uk

Tim Kovacs
University of Bristol
Dept. of Computer Science
Merchant Venturers Building
Bristol BS8 1UB
kovacs@cs.bris.ac.uk

James Marshall
University of Bristol
Dept. of Computer Science
Merchant Venturers Building
Bristol BS8 1UB
marshall@cs.bris.ac.uk

ABSTRACT

Credit assignment is a fundamental issue for the Learning Classifier Systems literature. We engage in a detailed investigation of credit assignment in one recent system called UCS, and in the process uncover two previously *undocumented* features. We draw on techniques from the classical pattern recognition literature, showing how to *analytically derive* an optimal credit assignment system, given certain assumptions. Our aim is not primarily to improve accuracy, but to better understand the system and put it on a more solid theoretical foundation. Nonetheless, empirical results on benign data demonstrate our new system, called UCSpv (UCS with principled voting), can match or exceed the original UCS. Further, its fitness function is principled, and, unlike that of UCS, requires no tuning. However, on more difficult data it seems UCSpv does need some form of tuning or correction. We believe the framework we adopt offers a promising new direction for LCS research, providing principled methods for action selection and bringing LCS closer to the mainstream pattern recognition literature.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Concept learning*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Algorithms

Keywords

Classifier systems, Fitness evaluation, Pattern recognition and classification, Ensembles, AdaBoost

1. INTRODUCTION

Learning Classifier Systems (LCS) evolve populations of rules to solve a given problem. These rulesets tend to contain many overlapping and contradictory rules, and as such the issue of conflict resolution – or how to combine the predictions of rules – is a significant one. Traditionally, the majority of LCS research has focused on *reinforcement learning* systems, while *supervised learning* systems have received less attention. We focus on a relatively recent supervised LCS called *UCS* [1] and investigate its credit assignment system. UCS is derived from the extremely popular XCS [9] but adapted for supervised learning.

In UCS each rule predicts a particular class for datapoints it matches. This class is fixed when the rule is created. Each rule's accuracy is calculated as the proportion of datapoints correctly classified. Each rule also has a fitness which determines both its weight in conflict resolution and its reproductive influence in the genetic algorithm. For each datapoint to be classified a *match set* is constructed consisting of all matching rules. UCS then uses the match set to construct a *prediction array* which is a set of *system predictions* for the various classes. The prediction array can be seen as storing the *support* for each possible class. As UCS is a supervised system it greedily selects the class with the highest system prediction. The system prediction for class c on a new input $\mathbf{x}$ is calculated as a normalized fitness-weighted vote:

$$P(c) = \frac{\sum f_i g_i(\mathbf{x})}{\sum g_i(\mathbf{x})} \quad (1)$$

where the summation is over the entire population, f_i is the fitness of rule i, and the function $g_i(\mathbf{x})$ returns 1 if rule i matches input $\mathbf{x}$, or 0 otherwise. The UCS literature (e.g. [1, 7]) is ambiguous regarding the exact form of eq(1) but we have confirmed it [5]. The fitness of rule i is

$$f_i = \gamma_{exp}(1 - \epsilon_i)^v \quad (2)$$

where ϵ_i is the error rate (proportion of datapoints incorrectly classified) of rule i on the datapoints it has matched so far and v is a parameter which affects the relative weighting of rules. The UCS literature does not contain any careful studies of how to set v but recommends $v = 10$ as a reasonable value [7]. γ_{exp} is an *inexperience discount* set to $\gamma_{exp} = 0.01$ if the number of datapoints matched so far is less than 10, or $\gamma_{exp} = 1$ otherwise. This discount does not appear in published descriptions of UCS [1, 7] but is used in UCS [5]. It is not a standard part of XCS but has been used in it [3]. In later sections we show it has a significant impact on performance by moderating the effect of rules that may have an erroneous estimate of their own accuracy.

In this paper we investigate the weighting of rules in UCS. In section 2 we study how to set the v parameter for best

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

performance. Then in section 3 we introduce a theoretical framework under which a novel UCS rule fitness function, which is optimal under certain assumptions, can be *analytically* derived. In the following three sections we compare the alternative fitness functions on benign data, class imbalanced data and noisy data respectively. In section 8 we discuss future work and then we conclude in the final section.

2. REPLICATING UCS

We implemented UCS from scratch. In this section we compare results with our implementation to the original and study the effect of varying UCS' υ parameter on both classification accuracy and %[B], the proportion of the best action map evolved [1]. We use the widely studied 11 multiplexer problem. This is defined on binary strings of length 11, and treats the string as being composed of an index segment (the first 3 bits) and a data segment (the remaining 8 bits). The value of the function is the value of the indexed bit in the data segment; so, for example, the value of input 10111110101 is 1, since the index bits 101 point to data bit 5[1] which has the value 1. Following the XCS tradition we conducted our replication experiment in an *online* learning fashion, i.e. at each iteration a single datapoint is selected with uniform random probability to present to the learning system. Figures report the accuracy over the last 50 datapoints seen, averaged over 20 repeat runs.

Figure 1 shows the accuracy of UCS on the 11-multiplexer, for different values of υ. Our implementation of UCS converges to accuracy 1.0 in approximately the same number of iterations as the original when a high υ is used [6]. We note that a setting of $\upsilon = 5$ results in very poor performance while values up to approximately $\upsilon = 30$ result in improvements, after which increasing υ has no major effects. In later sections we will evaluate the tuning of υ on other problems. The UCS literature recommends setting $\upsilon = 10$, but no studies or motivation for this value are given. As UCS is based on XCS the υ is likely to have been derived from XCS' updates, which have a related form. In XCS υ is used to separate the fitness of rules which otherwise would have similar fitness ([3] section 2.3.5.5). Exploration of the similarities between XCS and UCS beyond the scope of this work, but we note that υ provides a means of *tuning* the UCS accuracy function. Figure 5 illustrates the effect of tuning; increasing υ gives more weight to the most accurate rules. We leave it to future work to investigate how υ has this effect. An important point to note, illustrated in figure 2, is that UCS does not quite converge to accuracy 1.0 if we remove the (previously undocumented) inexperience discount in eq(2).

Figure 3 shows UCS' %[B] for various values of υ. For all values shown %[B] converges to 1. As υ has little effect on %[B] we will not report further results with it. However, we note that the rate of convergence of %[B] is likely to be more sensitive to selective pressure in the genetic algorithm than convergence of accuracy, which is likely to be more sensitive to parameterisation of voting. In UCS both are parameterised by υ. That υ affects accuracy and %[B] differently suggests they should be parameterised independently, but we leave this for future work. One of XCS' innovations was to separate a rule's prediction from its fitness. Curiously, UCS combines the two, despite being based on XCS.

[1]Being computer scientists we start counting at position 0.

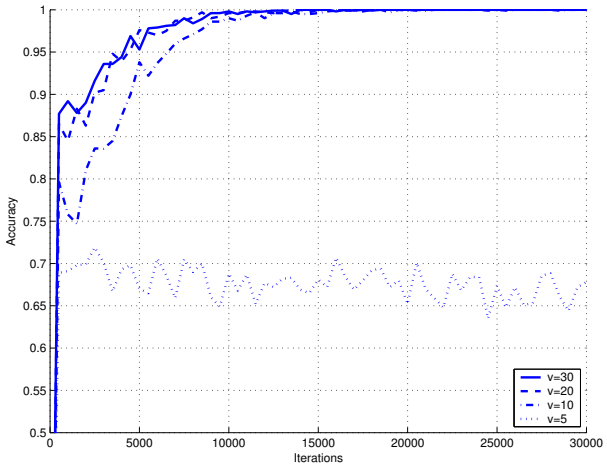


Figure 1: Accuracy of UCS on 11mux, illustrating the effect of the υ parameter. In this case, going beyond $\upsilon = 30$ produces no significant further improvements.

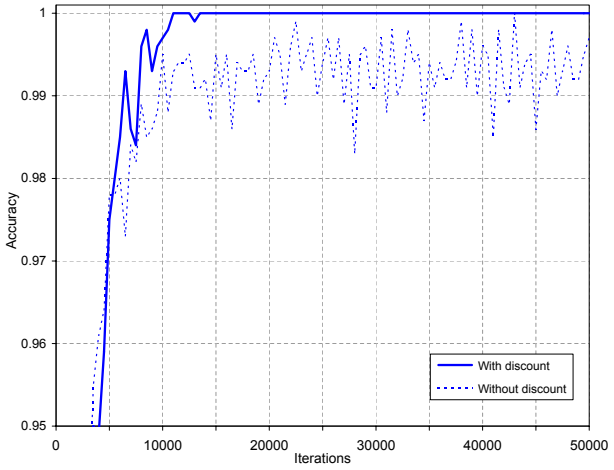


Figure 2: Accuracy of UCS($\upsilon = 30$), illustrating that UCS requires the previously undocumented inexperience discount to converge.

3. UCSPV: PRINCIPLED VOTING

In this section we demonstrate a principled method of weighting votes, derived from the literature on computational learning theory. The weighting is optimal given the assumptions of i) a particular exponential bound on the cost function and ii) that there is no error in accuracy estimates. Both assumptions are likely to make the derived weights suboptimal in practice. While the first is outside the scope of this work we will see that the inexperience discount and related corrections may help address the second. A further limitation is that the derivation is limited to two-class classification tasks. Despite these limitations the derived weights have advantages over the original UCS weights in terms of performance and ease of parameterisation. In being theoretically well-grounded this approach is a good basis for further work. In subsequent sections we will compare our new vote to the original UCS.

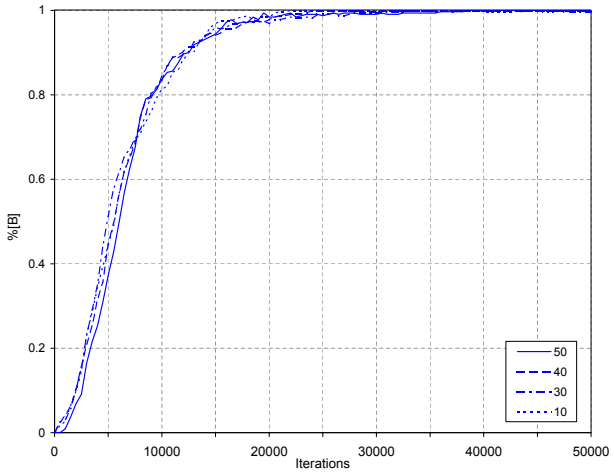


Figure 3: %[B] of UCS on 11mux, illustrating that the v parameter has little effect.

3.1 Minimizing a Cost Function

The ultimate measure we wish to minimize with UCS is the *classification error rate*,

$$P(error) = \frac{1}{|D|} \sum_{(\mathbf{x},c)\in D} I(H(\mathbf{x}) \neq c) \tag{3}$$

where $H(\mathbf{x})$ is the prediction issued by the system, and D is a training dataset. The unfortunate problem with this is that the indicator function $I(a \neq b)$ is discontinuous and non-differentiable, so does not permit an analytic solution for the individual rule fitnesses.

Instead, we adopt a methodology from *Boosting*, a powerful learning system with roots in computational learning theory and PAC-learnability [2, 8]. For binary problems, we assume rule i implements a function $h_i(x)$ that provides a label from $\{-1, 0, +1\}$, where a label of 0 indicates abstention by the rule as it does not match the presented instance. In this case the predicted class is given by

$$H(\mathbf{x}) = sign\Big(\sum_i \alpha_i h_i(\mathbf{x})\Big) \tag{4}$$

where α_i is a function proportional to the rule accuracy, in our LCS framework corresponding to the rule fitness—we would like to identify the optimal values for α_i that minimize the probability of system error. This implements a set of *localised weighted votes*; it should be noted that this is unnormalized, unlike eq(1). Each vote corresponds to the area of input space matched by the rule, weighted by its fitness α_i. Now we assume a cost function that provides a continuous, tight bound on the classification loss function, but retains the desirable properties. This is shown in figure 4; the inequality described as $I(H(\mathbf{x}) \neq c) \leq exp(-cd(\mathbf{x}))$ where $d(\mathbf{x}) = \sum_i \alpha_i h_i(\mathbf{x})$.

Assuming this exponential cost, it can easily be seen that

$$P(error) \leq \frac{1}{|D|} \sum_{(\mathbf{x},c)\in D} \prod_i exp(-\alpha_i h_i(\mathbf{x})c) \tag{5}$$

This says that the probability of error[2] is bounded by a function of the individual fitnesses, α_i. If we minimize the right hand side of equation 5 by independently minimising these fitnesses, we are guaranteed to exponentially reduce training error over the entire dataset. Taking the partial derivative of eq(5) with respect to each α_i, and solving for α_i at the minimum, we obtain an analytic solution for the fitness,

$$\alpha_i = \frac{1}{2} ln\Big(\frac{1-\epsilon_i}{\epsilon_i}\Big) \tag{6}$$

This is plotted in figure 5, compared to the UCS formulation from eq(2); it should be noted that our formulation does not naturally use the inexperience discount factor γ_{exp}. We find that the functions follow similar trends, but the gradients are quite different over the range of possible accuracies (figure 6). We refer to UCS with the derived weights as UCSpv (UCS with principled votes).

A complication is that the weighting function tends to infinity as the accuracy tends to 1. To avoid overflow errors when accuracy is 1 we set $\alpha_i = \alpha'$ where α' is a parameter we can tune. α' is like v in that it affects the slope of the weighting curve (figure 5). LCS are notorious for the large number of parameters they contain and we are reluctant to introduce any new ones. However, we will have much more to say about α' in later sections.

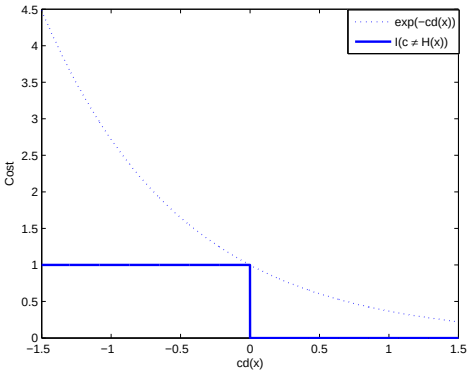


Figure 4: Using an exponential cost function to bound the classification error.

4. BALANCED NOISELESS DATA

We now compare UCS to our new UCSpv on well-behaved data. From this point forward we adopt an *offline* learning regime. The 11-multiplexer has 2048 datapoints which we uniform randomly split into 50% train and 50% test sets on each of 20 repeat runs. In figures we plot accuracy on the complete test set after every 500 training examples (which we select uniform randomly with replacement). Figure 7 shows UCSpv with an arbitrary setting of $\alpha' = 100$ and UCS with a range of v values. UCSpv dominates UCS, with no need to tune α'. Our next step is to study the sensitivity of each system to its weight-tuning parameter.

From this point on we generally dispense with learning curves show the test set accuracy after 50,000 training iterations. Figure 8 shows UCS' accuracy over a range of v values, with and without the inexperience discount γ_{exp}. We

[2]On the *training* set—extensions of this can apply to testing data and provide (loose) bounds on generalisation error, but this is outside the scope of this paper.

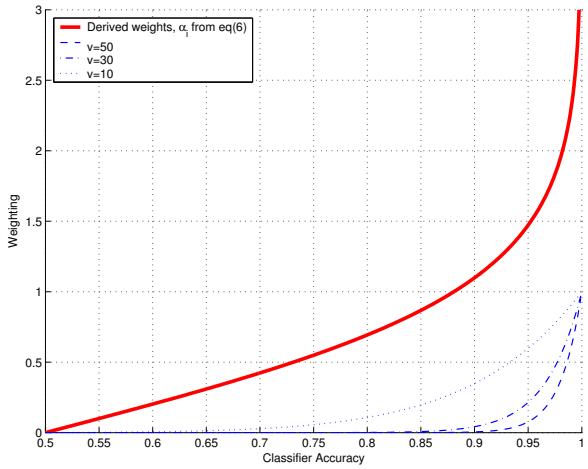


Figure 5: Derived vs UCS rule weightings

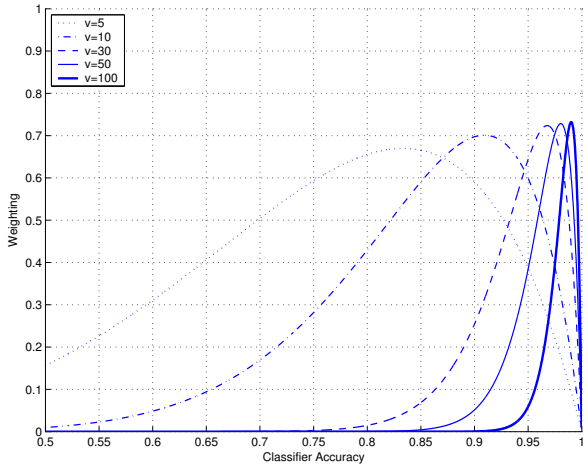


Figure 6: Ratio of UCS gradient to derived weight gradients

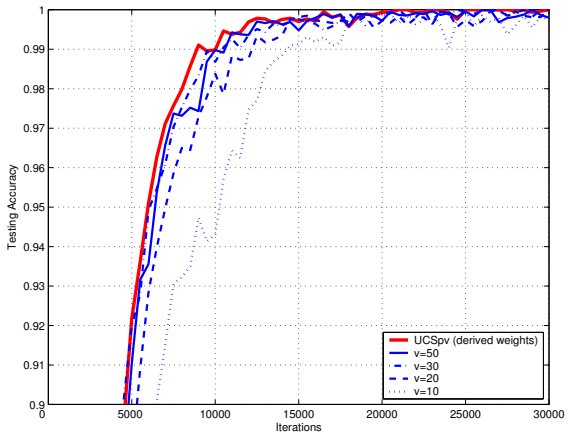


Figure 7: UCS and UCSpv on the 11mux using 50/50 train/test.

note i) low v results in very poor performance but performance suddenly reaches a plateau as v rises, ii) the inexperience discount improves performance except when v is suboptimally low, and iii) UCS generally, but not always,

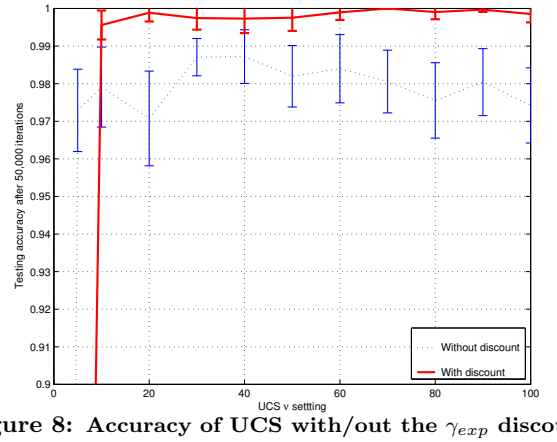


Figure 8: Accuracy of UCS with/out the γ_{exp} discount. The discount helps UCS significantly though convergence to accuracy 1.0 is not guaranteed.

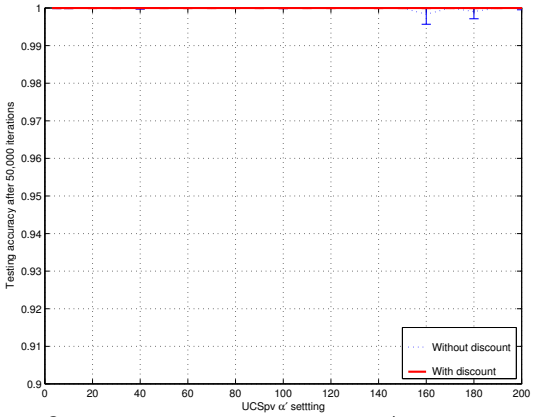


Figure 9: Accuracy of UCSpv with/out the γ_{exp} discount. Compare to performance of UCS in fig 8.

converges to accuracy 1 when v is high enough. The equivalent results for UCSpv are in figure 9, where for completeness we also incorporated the inexperience discount – we simply multiply the α_i value in eq(6) by 0.01 if the rule has matched less than 10 examples. With and without the discount, and over all values of α', UCSpv converges to accuracy 1.0 on virtually every run.

Results thus far are excellent. The accuracy of UCSpv is marginally better than that of UCS, UCSpv does not need the inexperience discount and its α' parameter needs no tuning. Further empirical study on this kind of benign data is needed to confirm these advantages. However, we turn instead to situations where the training data is less representative of the testing data. First we skew the class distribution, then we add random noise to the class labels.

5. CLASS-IMBALANCED TRAINING SET

We now alter the 11 multiplexer in order to undersample one of the two classes during training. UCS has been studied on class imbalanced datasets before [1] and found to have difficulties. In this case, however, we only imbalance the training set, which makes it less representative of the test set, and thus makes the problem even more difficult. At each training iteration we select a class with a certain probability and sample the dataset repeatedly until an example of that class is found.

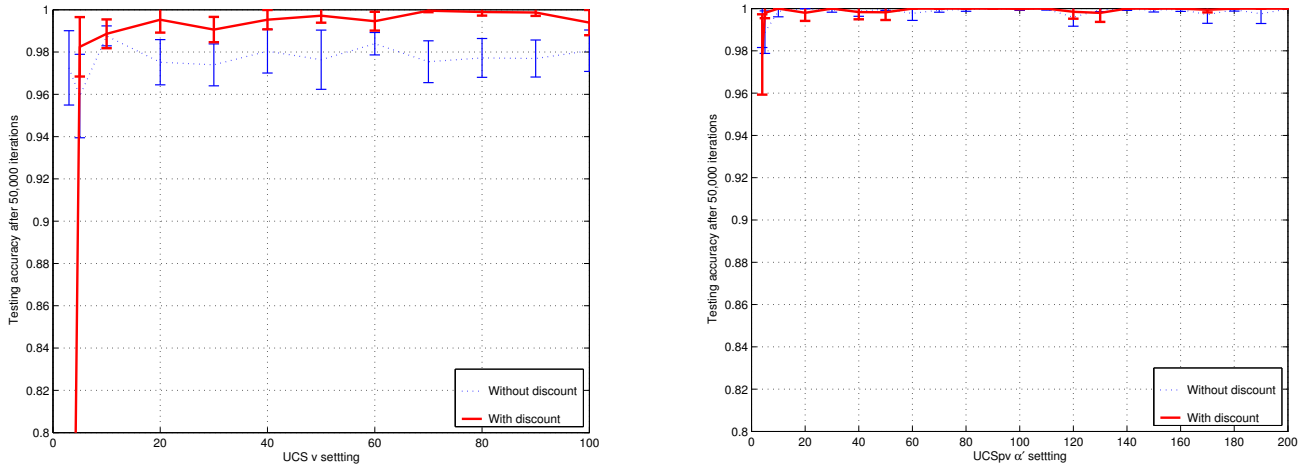


Figure 10: Accuracy on *2:1 unbalanced data* for UCS (left) and UCSpv (right) with/out discount

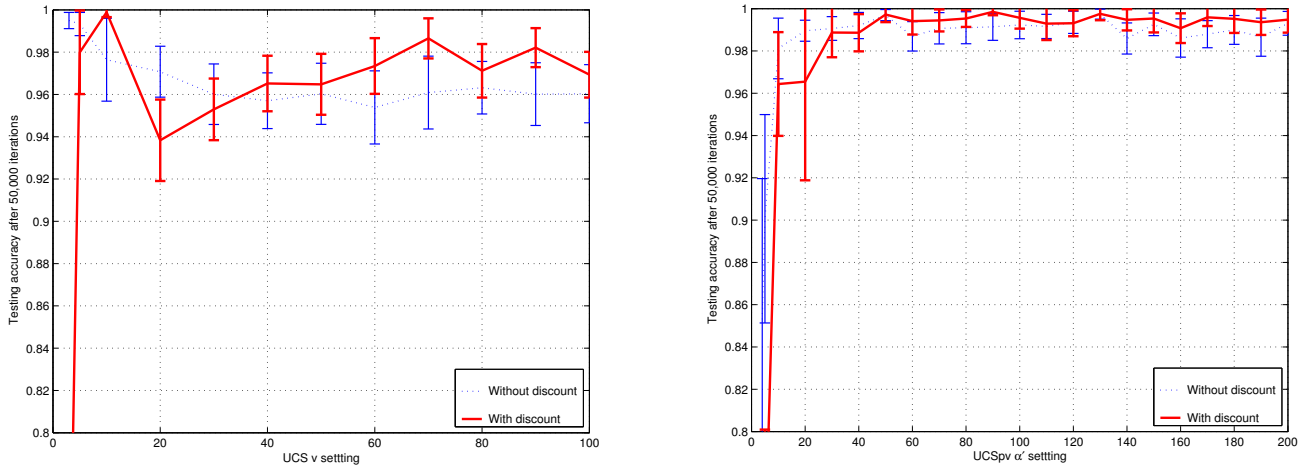


Figure 11: Accuracy on *4:1 unbalanced data* for UCS (left) and UCSpv (right) with/out discount

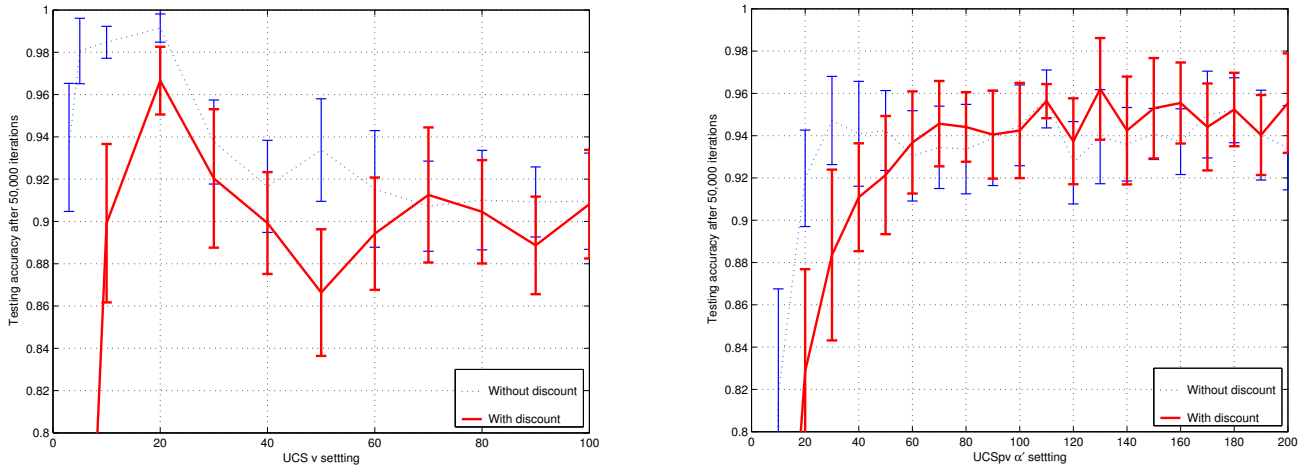
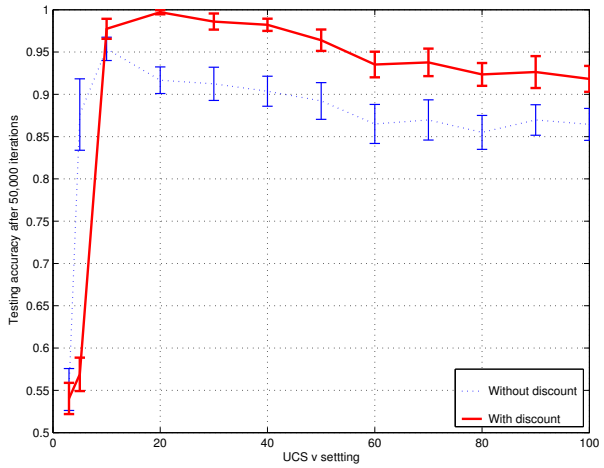


Figure 12: Accuracy on *8:1 unbalanced data* for UCS (left) and UCSpv (right) with/out discount



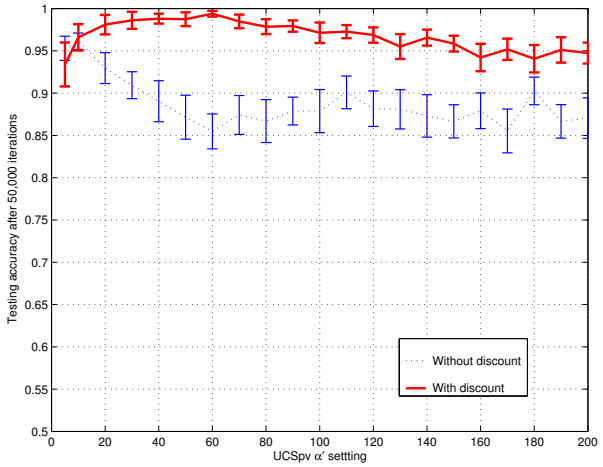
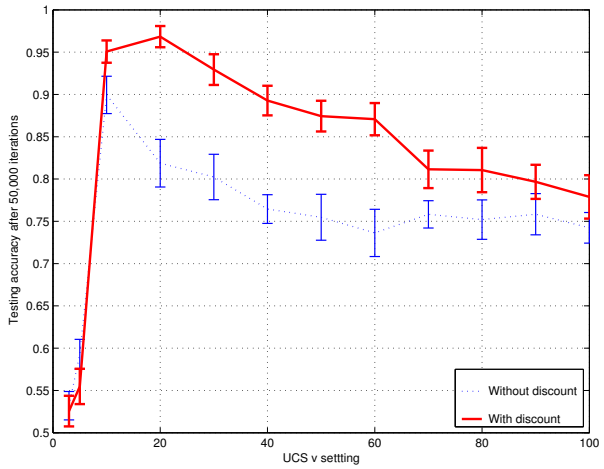


Figure 13: Accuracy on multiplexer with *5% noise* for UCS (left) and UCSpv (right) with/out discount



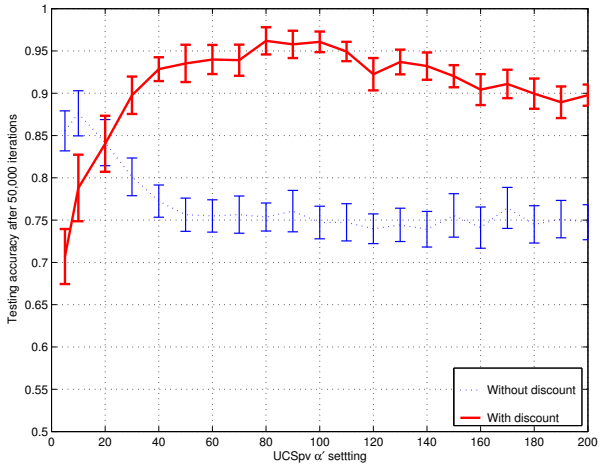
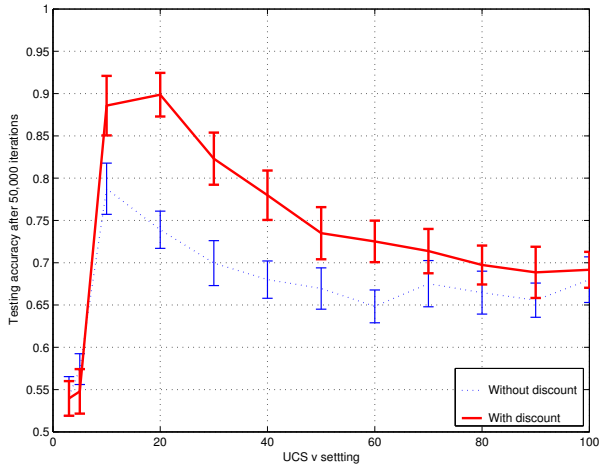


Figure 14: Accuracy on multiplexer with *10% noise* for UCS (left) and UCSpv (right) with/out discount



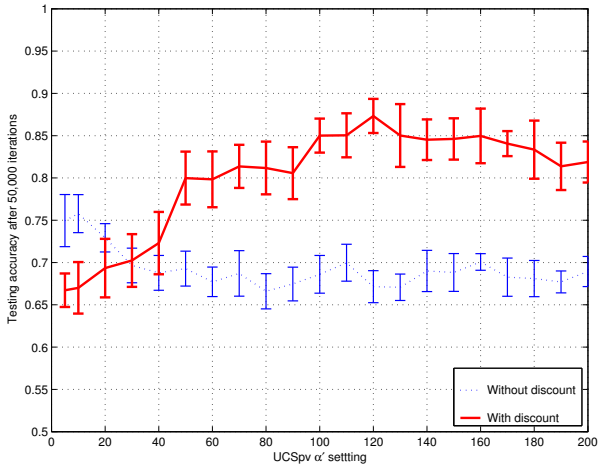


Figure 15: Accuracy on multiplexer with *15% noise* for UCS (left) and UCSpv (right) with/out discount

Figures 10 to 12 show the performance of UCS on the left and UCSpv on the right on a variety of imbalances. Error bars in these and following figures show 95% confidence intervals. Observations to be drawn from these figures are i) as imbalance increases performance decreases for all algorithms, ii) best performance for UCSpv is as good or better than that of UCS in all cases except in the 8:1 condition (and then only when UCS is carefully parameterised), iii) for imbalanced data α' has an effect; low values result in very poor performance, iv) α' is nonetheless easier to set than v as v has a narrow intermediate range of optimal values while α' reaches a plateau, v) the inexperience discount has no statistically reliable effect on UCSpv, and vi) at higher levels of imbalance (4:1 and 8:1) the discount only improves UCS at very low v whereas for 2:1 and 1:1 data it improves it at higher v. In summary, for all cases examined so far, UCSpv is easier to parameterise than UCS, and (with one exception noted above), performs as well or better than UCS. Note also that UCS requires the discount for best performance while UCSpv does not. We defer analysis until section 7.

6. NOISY DATA

In the previous section we evaluated the effect of class imbalances on UCS and UCSpv and also on the utility of the γ_{exp} inexperience discount. In this section we report the corresponding experiments with noisy data. We begin with a noisy 11 multiplexer in which we have flipped the class labels of 5%, 10% or 15% of the training data. We then consider the Monks3 problem which has 5% class noise.

6.1 The Noisy Multiplexer

Results on the multiplexer with 0% noise have already been given in figures 8 and 9. Figures 13 to 15 show the corresponding experiments with 5, 10 and 15% noise. Conclusions to be drawn from these results are i) as noise increases accuracy decreases for all algorithms, ii) best performance for UCSpv is as good or better than that of UCS in all cases (but see note below on the 15% noise condition) iii) for noisy data α' has an effect; low values are suboptimal and (unlike for imbalanced data) high values are also suboptimal, iv) α' is more robust than v (but the difference is not as great as with the imbalanced data), v) unlike the imbalanced case the inexperience discount now has a major benefit for UCSpv, and vi) the discount uniformly improves UCS and improves UCSpv except when α' is suboptimally low.

We now return to the point noted in ii) above. Although the accuracies of the optimally parameterised UCS and UCSpv are statistically indistinguishable given our 95% confidence intervals, there is some suggestion that UCS may outperform UCSpv on the most noisy data. This is similar to the case of highest class imbalance where optimally parameterised UCS does outperform UCSpv. Thus we suspect that on the most difficult data there is a narrow range of parameterisation which results in better performance for UCS.

To summarise results on all versions of the multiplexer, most of our conclusions from the end of section 5 hold; in all cases UCSpv is easier to parameterise than UCS, and, with the exception discussed above, performs as well or better than UCS. However, we have now found that on noisy data (only) the inexperience discount helps UCSpv.

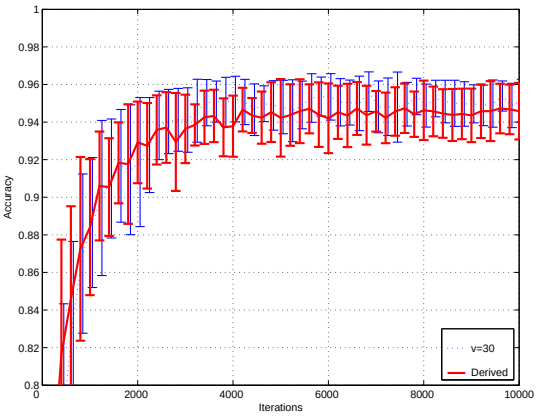


Figure 16: UCS and UCSpv on Monks3. Accuracy is statistically indistinguishable at the 95% confidence level.

6.2 The Monks3 problem

On the noisy multiplexer we found UCSpv performs best when the inexperience discount is used and that a robust setting for α' is approximately 100. We tested these settings on the Monks3 problem, which has 5% noise. For comparison we evaluated UCS with various v values and selected the best ($v = 30$). Figure 16 illustrates that these settings match the performance of UCS with tuned v.

7. ANALYSIS

At the start of section 3 we noted that UCSpv's weights are optimal only given certain assumptions We hypothesise that the second assumption, that accuracy estimates are error-free, is violated on the more difficult data. As we increase either imbalance or noise the training set becomes less representative of the test set and hence accuracy estimates obtained on the training set increase in error when classifiers are applied to the test set. This would explain why the inexperience discount becomes helpful for UCSpv on the noisy data though why it is not on the imbalanced data is unclear. We leave this to future work to resolve.

On class-balanced, noiseless data UCSpv's α' had no effect but as we increased problem difficulty the need to tune α' also increased. Our interpretation is that the UCSpv weights need significant corrections for data which violate their assumptions. Indeed, Freund and Schapire have already noted this weighting scheme has difficulties with noisy data [2]. An important observation is that α' *only affects one point on the weighting curve and is thus a very poor means of tuning it.* (Despite this, α' has a major effect on UCSpv's accuracy on the harder problems.) Consequently, the v parameter allows us to tune UCS more effectively than α' allows us to tune UCSpv. We hypothesise this accounts for the higher accuracy of the optimally parameterised UCS on the most highly class-imbalanced problem. In future we will investigate more powerful alternatives to α' and base our weights on other exponential bounds. In the next section we will see that α' can actually become irrelevant.

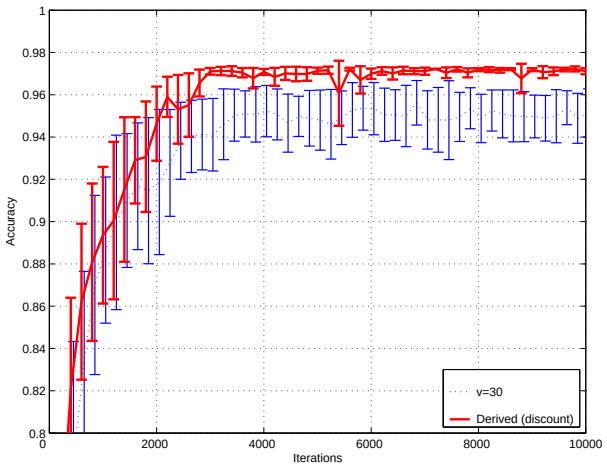


Figure 17: UCS and UCSpv(κ) accuracy on the Monks3 problem, $\kappa = 0.95$

8. CORRECTING ACCURACY ESTIMATES

If we suspect our training data contains noise we should expect rule accuracies estimated on it to be overestimates. This is a problem because rules with overestimated accuracy are given too much weight, which should result in reduced test set accuracy. The extreme case illustrates this principle nicely; a rule with accuracy 1 should, theoretically, have infinite weight. This is sensible if the rule really does have accuracy 1, but if we have any doubt of its accuracy we should give it a finite weight.

There are various means of correcting accuracy estimates. One is the γ_{exp} inexperience discount, which presumes that rules with less than 10 matches will have inaccurate accuracy estimates. In sections 4–6 we found this discount was sometimes beneficial and we now briefly experiment with another form of correction. Since we have a diminished trust in the accuracy estimate ϵ_i, we can use the following:

$$\alpha_i = \frac{1}{2} ln\Big(\frac{\kappa(1-\epsilon_i)}{1-\kappa(1-\epsilon_i)}\Big) \tag{7}$$

where κ is a parameter we tune based on our anticipation of noise. κ can be interpreted as our "trust" in the accuracy estimate—the more noise in the data, the less trust we have.

We experimented on the Monks3 problem where we know there is noise in the data and so set a discount of $\kappa = 0.95$. Figure 17 demonstrates this correction results in significantly higher accuracy and lower variance for UCSpv(κ) than UCS. This contrasts with figure 16 in which there is no significant difference between UCSpv and UCS. In addition to improving performance, using κ makes α' irrelevant since with $\kappa < 1$, equation (7) does not overflow.

Future work will explore more principled ways of incorporating statistical confidence into κ. Elsewhere, we have begun to investigate Bayesian estimation of rule accuracy [4] as a principled approach to discounting, and we will develop this approach further in future work.

9. CONCLUSIONS

First, through careful replication of results and with the help of others we identified two *undocumented* features of UCS. These were (1) a normalisation of the system prediction, and (2) a *discount* for low experience rules. Second, we derived a principled voting method for UCS, which we called UCSpv. Drawing on the pattern recognition literature on the minimization of exponential cost functionals, we replaced the original weighting function with a more principled one – essentially bounding the classification error with a continuous function and minimizing that function, thereby minimizing the classification error *indirectly*. We derived an *analytic solution* for rule weighting. This is currently used for both voting and genetic fitness, but in future work we will optimise them independently.

In section 4 we found UCSpv superior to UCS on benign (noiseless and class-balanced) data; UCSpv's accuracy was higher and, unlike UCS, its weighting function needed no tuning. In sections 5 and 6 we showed that more difficult data, where the training set is less representative of the test set, degrades the performance of both UCS and UCSpv. There was evidence that optimally parameterised UCS could outperform the optimally parameterised UCSpv. In section 7 we concluded the more difficult data violated the assumptions under which UCSpv's weights are optimal, and that UCS' tuning parameter was more effective than UCSpv's. In section 8 we presented preliminary results indicating that a correction to UCSpv's weights to account for noise in the data can be effective at improving accuracy.

As a final note, an advantage of the view we adopted is that it permits other cost functionals than pure exponential, enabling a framework to *design* the fitness function in a principled manner. Future work will explore this promising new direction for supervised LCS, and attempt to extend the principles to reinforcement learners like XCS.

10. ACKNOWLEDGEMENTS

We thank Albert Orriols-Puig and Kamran Shafi for help in implementing UCS and Alwyn Barry for many comments.

11. REFERENCES

[1] E. Bernadó-Mansilla and J. M. Garrell-Guiu. Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. *Evolutionary Computation*, 11(3):209–238, 2003.

[2] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proc. 13 Int. Conf. on Machine Learning*, pp. 148–156. Morgan Kaufmann, 1996.

[3] T. Kovacs. *Strength or Accuracy: Credit Assignment in Learning Classifier Systems*. Springer, 2002.

[4] J. A. R. Marshall, G. Brown and T. Kovacs. Bayesian estimation of rule accuracy in UCS. In Tina Yu, ed., *Proceedings of the 2007 workshops on genetic and evolutionary computation*. ACM Press, 2007.

[5] A. Orriols-Puig. Personal Communication 15 Oct. 2006.

[6] A. Orriols-Puig. Personal Communication 26 Oct. 2006.

[7] A. Orriols-Puig and E. Bernadó-Mansilla. A further look at UCS classifier system. In J. van Hemert ed., *Proceedings of the 2007 workshops on genetic and evolutionary computation*. ACM Press, 2006.

[8] R. E. Schapire. Theoretical views of boosting and applications. In *Algorithmic Learning Theory, 10th Int. Conf.*, LNAI 1720, pp. 13–25. Springer, 1999.

[9] S. W. Wilson. Classifier Fitness Based on Accuracy. Evolutionary Computation, 2(3):149–175, 1995.