

Modeling UCS as a Mixture of Experts

Narayanan U. Edakunni
Dept. of Computer Science
University of Bristol
nara@cs.bris.ac.uk

Tim Kovacs
Dept. of Computer Science
University of Bristol
kovacs@cs.bris.ac.uk

Gavin Brown
School of Computer Science
University of Manchester
gavin.brown@manchester.ac.uk

James A. R. Marshall
Dept. of Computer Science
University of Bristol
marshall@cs.bris.ac.uk

ABSTRACT

We present a probabilistic formulation of UCS (a sUpervised Classifier System). UCS is shown to be a special case of mixture of experts where the experts are learned independently and later combined during prediction. In this work, we develop the links between the constituent components of UCS and a mixture of experts, thus lending UCS a strong analytical background. We find during our analysis that mixture of experts is a more generic formulation of UCS and possesses more generalization capability and flexibility than UCS, which is also verified using empirical evaluations. This is the first time that a simple probabilistic model has been proposed for UCS and we believe that this work will form a useful tool to analyse Learning Classifier Systems and gain useful insights into their working.

Categories and Subject Descriptors

G3 [Mathematics of Computing]: Probability and Statistics

General Terms

Algorithm

Keywords

Learning Classifier System, probabilistic modeling, mixture of experts, UCS

1. INTRODUCTION

Recently there has been a growing interest in probabilistic models for Learning Classifier Systems (LCSs), the genetics-based machine learning models introduced by Holland. Such models would help integrate the LCS field into mainstream machine learning research by drawing parallels to other existing machinery in probabilistic machine learning. In this

paper we provide an explicit probabilistic model for a specific Learning Classifier System called UCS, which stands for sUpervised Classifier System [1].

When we try to establish the equivalence of two seemingly disparate learning algorithms it is important to establish the characteristics of the learning machines that we are to compare. In this paper we establish the equivalence in representation of data by the learned machines and do not concern ourselves with the algorithm that is used to learn the particular representation. The logic behind this reasoning being that the representation is more important while making decisions using the learned machine rather than the actual method used to learn the representation. Thus, while UCS uses a genetic algorithm to learn the optimal representation, a mixture of experts learns through a non-stochastic Expectation Maximisation algorithm [3], although both of them share similar prediction routines for classification.

In this paper we compare UCS with a mixture of experts using a classification problem with an input space made up of binary strings of length D . The number of classes is taken to be K and the classification is performed using an ensemble (also called a population) of classifiers (called individuals or rules) each made up of a *condition* and a *class* (or *action*). A condition is a ternary string composed of characters from $\{0, 1, \#\}$ where $\#$ is a *don't care* symbol which expresses generalisation over input strings. Finally, we denote the space of all D -length ternary strings by \mathcal{S} and note that its cardinality is 3^D .

The paper starts with a brief introduction to UCS followed by an introduction to the probabilistic model for the mixture of experts. The equivalence between a mixture of experts and UCS is then established in Section 4 by deriving the UCS classification rules from a two component mixture of experts. Following this the two systems are compared in terms of their components. Finally, they are empirically compared on toy data sets to support the theoretical analysis derived in this paper. It must be noted that the aim of this paper is to come up with a probabilistic model of UCS and *not to advocate a novel method* for classification, although the probabilistic model developed in this paper has better learning abilities than UCS.

2. UCS

We now describe UCS [1] in a way which should be comprehensible to those with a background in either LCS or ensembles. UCS is an ensemble of classifiers which combine

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.
Copyright 2009 ACM 978-1-60558-325-9/09/07 ...\$5.00.

their individual decisions to produce a unified output. UCS can be applied to supervised learning problems in which a labelled training set of (input,class) pairs is available. UCS uses a Genetic Algorithm (GA) to evolve (create and delete) classifiers using the Michigan approach [7] in which each classifier applies to (matches) a subset of the input space. A classifier's condition and action is set when the classifier is created by the genetic algorithm and do not change.

We will not describe the operation of the genetic algorithm as it is of no consequence to our analysis, but we will outline its function, which is to evolve a population of classifiers which make good quality predictions for the inputs they match. The quality of prediction is determined by a fitness function which is based on the number of correct and incorrect predictions made on the training data. Consequently, evaluation of classifier fitness on training data is embedded within the evolutionary cycle; the evolution and evaluation of classifiers interleave.

In order to evaluate the fitness of the current population on the training data we first select a training input. The subset of the population which matches the input, called the *match set*, is identified using a match function. The match set is composed of *action sets* consisting of classifiers which predict the same class (action). Only classifiers in the match set contribute to the prediction for the current input and only they have their fitness updated based on their prediction.

The match function $M(., .)$ is defined by a simple bitwise comparison of the input string with the classifier condition. A 1 bit in the condition matches with a 1 bit in the corresponding position in the input string, 0 with a 0 bit and a # with both 0 and 1 bits of the input string. Given the condition of the j^{th} rule r_j , and the input string \mathbf{x} , the match function is defined as -

$$M(j, \mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \text{ matches } r_j \\ 0 & \text{otherwise} \end{cases}$$

For example condition 00# matches two input strings: 000 and 001, while ### matches all 3-bit input strings.

In UCS the *fitness function* $F(.)$ is -

$$F = \left(\frac{\text{number of correct classifications}}{\text{number of matches}} \right)^v$$

where the number of matches is the number of times the classifier matched an input string from the training set, and the number of correct classifications is the number of times it matched and its class agreed with the class specified by the training set. If \mathbf{x}_i is one of the training input strings and c_i the corresponding class label then the fitness of a rule j with class k is given by -

$$F(j, k) = \left(\frac{\sum_i M(j, \mathbf{x}_i) \mathcal{I}(c_i == k)}{\sum_i M(j, \mathbf{x}_i)} \right)^v \quad (1)$$

where \mathcal{I} is an indicator function which takes value 1 when its argument evaluates to true and 0 otherwise and $v \geq 1$ is an arbitrary parameter which is used to tune the selective pressure of the genetic algorithm. Since our analysis will not include the operation of the genetic algorithm we set v to 1 for simplicity. An alternative definition of rule accuracy has been provided by [8] which uses a Bayesian rule to derive the fitness. At convergence with large amounts of data the Bayesian fitness rule corresponds to the definition in Eq. (1).

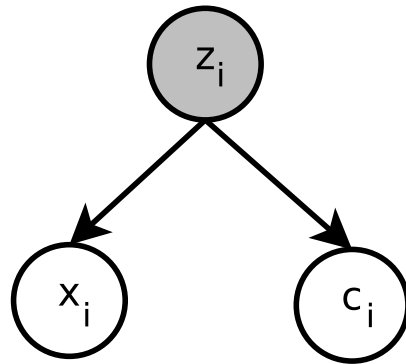


Figure 1: The graphical model of the mixture of experts

The rules that define the match is pre-defined and fixed for a particular implementation of UCS. In contrast, the classifiers of a population are evolved for a particular training set, and the fitness of the population is evaluated/learned from that training set. Rather than model the evolutionary process we simply model its output: the population. The population is composed of condition-action rules. Recall from Section 1 that the set of possible conditions is denoted \mathcal{S} and that there are K classes. Any population, then, is composed of elements from the set $\mathcal{S} \times \{1 \dots K\}$. We model the current population using an *inclusion function* $I(., .)$ which is an indicator function defined over the set $\mathcal{S} \times \{1 \dots K\}$. I takes values of 1 or 0 depending on whether a particular classifier is included in the population or not.

The prediction algorithm of UCS uses the trained classifiers to predict the class label for a test input \mathbf{x}_q . The test input defines a match set and action sets. The sum of fitnesses of the classifiers in each of the action sets is computed and the action corresponding to the action set with the maximum sum of fitness is taken as the action of the system as a whole. The process can be represented as -

$$S(\mathbf{x}_q, c = k) = \sum_{j=1}^T F(j, k) \times M(j, \mathbf{x}_q) \times I(j, k) \quad (2)$$

where $S(\mathbf{x}_q, c = k)$ is the support for class k for a test input string \mathbf{x}_q , $F(j, k)$ is the fitness for the j^{th} classifier with class(action) k , M the match function, I the inclusion function and $T = |\mathcal{S}| = 3^D$ where D is the length of the binary input string. The support S is computed for all of $k = 1 \dots K$ classes and the class with the maximum support is taken as the prediction of the ensemble.

3. MIXTURE OF EXPERTS (MOE)

A mixture of experts is a probabilistic model associated with a stochastic process where the data is assumed to be generated from distinct experts with various probabilities [6]. The generative process can be graphically illustrated as in Fig. (1) where \mathbf{x} is input, c the class label corresponding to the input and z is the hidden multinomial variable that points to the expert which is responsible for the particular input-class pair. Associated with each edge of the graph is a conditional probability that relates the two variables represented by the respective nodes. Equivalently, the graphical

model given by Fig. (1) can be written down as a factorisation of the joint probability of the class variable c and the input \mathbf{x} -

$$P(\mathbf{x}, c) = \sum_{j=1}^T P(c|z=j)P(\mathbf{x}|z=j)P(z=j) \quad (3)$$

In our case, an expert would be a ternary coded string *condition* with as many experts as there are conditions. Hence T in Eq. (3) corresponds to the number of experts and is equal to the cardinality of set \mathcal{S} . The probabilistic model given by Eq. (3) and illustrated by Fig. (1) assumes that data are generated by first choosing a condition with probability $P(z)$ and this condition in turn chooses an input string from all possible strings that it matches with a probability of $P(\mathbf{x}|z)$. For example, if the chosen condition is 10# then the possible inputs that it can generate (or match) are {100, 101}. In the absence of any prior knowledge, each of the matching inputs have equal probability of being generated. Finally, a class label is chosen according to $P(c|z)$. The input-class pair thus generated would follow a joint probability distribution given by Eq. (3). In Eq. (3) there are three important distributions that determine the likelihood of data -

1. The conditional class probability $P(c|z=j)$ which is an unknown constant for a given j
2. Conditional probability of the data given the condition string is a known constant and is given by -

$$P(\mathbf{x}|z=j) = \begin{cases} 2^{-(\text{no. of \# in } z)} & \text{if } \mathbf{x} \text{ matches } z, \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

3. The probability of the individuals in the population $P(z=j)$ which is an unknown constant.

The unknown probability values in the model need to be determined by maximising the likelihood of the training data. The simplest method to optimise the likelihood is through an Expectation Maximisation (EM) algorithm. Assuming an independently and identically distributed data, log likelihood is given by -

$$\begin{aligned} \mathcal{L} &= \ln \prod_{i=1}^N P(c_i, \mathbf{x}_i) \\ &= \sum_{i=1}^N \ln P(c_i, \mathbf{x}_i) \\ &\quad \text{and, using Eq. (3),} \\ &= \sum_{i=1}^N \ln \sum_j P(c_i|z_i=j)P(\mathbf{x}_i|z_i=j)P(z_i=j) \end{aligned}$$

where N is the number of training instances. Using Jensen's inequality, the lower bound for \mathcal{L} is given by -

$$\begin{aligned} \mathcal{L} \geq \mathcal{J} &= \sum_i \sum_j P(z_i=j|\mathbf{x}_i, c_i) \ln P(c_i, \mathbf{x}_i, z_i=j) \\ \mathcal{J} &= \sum_i \sum_j P(z_i=j|\mathbf{x}_i, c_i) \ln P(c_i|z_i=j) \\ &+ \sum_i \sum_j P(z_i=j|\mathbf{x}_i, c_i) \ln P(\mathbf{x}_i|z_i=j) \\ &+ \sum_i \sum_j P(z_i=j|\mathbf{x}_i, c_i) \ln P(z_i=j) \end{aligned}$$

The two quantities that we need to determine are the conditional probability of the class given a condition string ($P(c_i|z_i=j)$) and the probability of the condition itself ($P(z_i=j)$). We denote the former as $\alpha_{j,k} = P(c_i=k|z_i=j)$ and the latter as $\gamma_j = P(z_i=j)$. We can now use the lower bound given by \mathcal{J} to optimise the log likelihood given by \mathcal{L} . An EM algorithm to optimise the likelihood consists of two different steps.

E-step : In this step we compute the posterior of the hidden variables as

$$P(z_i=j|c_i, \mathbf{x}_i) = \frac{P(c_i|z_i=j)P(\mathbf{x}_i|z_i=j)P(z_i=j)}{\sum_j P(c_i|z_i=j)P(\mathbf{x}_i|z_i=j)P(z_i=j)} \quad (5)$$

M-step : In this step we compute the values of the parameters (α and γ) by taking the derivative of \mathcal{J} with respect to these parameters and setting them to zero -

$$\begin{aligned} \mathcal{J}(\alpha_{j,k}) &= \sum_i \sum_j P(z_i=j|\mathbf{x}_i, c_i) \ln \alpha_{j,k}^{\mathcal{I}(c_i==k)} \\ \frac{\partial \mathcal{J}}{\partial \alpha_{j,k}} &= \sum_i \frac{P(z_i=j|\mathbf{x}_i, c_i) \mathcal{I}(c_i==k)}{\alpha_{j,k}} \end{aligned}$$

Equating $\frac{\partial \mathcal{J}}{\partial \alpha_{j,k}}$ to zero with the constraint of $\sum_k \alpha_{j,k} = 1$ yields -

$$\alpha_{j,k} = \frac{\sum_i P(z_i=j|\mathbf{x}_i, c_i) \mathcal{I}(c_i==k)}{\sum_k \sum_i P(z_i=j|\mathbf{x}_i, c_i) \mathcal{I}(c_i==k)} \quad (6)$$

We can use a similar procedure to estimate γ and this results in -

$$\gamma_j = \sum_{i=1}^N P(z_i=j|\mathbf{x}_i, c_i) / N \quad (7)$$

Having trained a mixture of experts we can use the model to predict the class for an unseen test input \mathbf{x}_q by evaluating the joint probability of the data for different classes -

$$P(\mathbf{x}_q, c=k) = \sum_{j=1}^T P(c=k|z=j)P(\mathbf{x}_q|z=j)P(z=j) \quad (8)$$

The joint probability is evaluated for $k = 1 \dots K$ classes and the class with the maximum value is chosen as the prediction for \mathbf{x}_q . In this paper we propose that the form of class prediction defined by Eq. (8) is equivalent to UCS (Eq. (2)) with $P(c|z=j)$ serving as the *fitness function* for rule j , $P(\mathbf{x}_q|z=j)$ the *matching function* for the UCS and $P(z=j)$ the *inclusion function* as summarised in Table 1. The proposed equivalence between components of MOE and UCS might seem ad hoc especially since the training rule for fitness as given by Eq. (1) is different from the training rule for $P(c|z)$ given by Eq. (6). The other discrepancy is between the matching function of UCS which takes only values from $\{0, 1\}$ whereas $P(\mathbf{x}|z)$ takes on real values in the interval $[0, 1]$. These discrepancies can be attributed to the difference in training of MOE and UCS as explained in the following sections.

4. UCS VS MOE

In this section we look at the similarities and differences between UCS and MOE formulations in terms of their corresponding constituents of fitness, matching and inclusion

MOE	UCS
$P(c z)$	fitness
$P(x z)$	matching function
$P(z)$	inclusion function

Table 1: Equivalence between UCS and MOE

functions. One of the difficulties while comparing UCS and MOE is their differing training routines. To set a common ground we switch off the GA and deletion heuristics of UCS and include all the condition-action pairs from $\mathcal{S} \times \{1 \dots K\}$ in the population. It must be noted that this does not affect the learning of the fitness function of an individual classifier since it is independent of others in the population.

4.1 Fitness function vs class probability

In an MOE, mixture probabilities of all the experts are learned simultaneously with one being dependent on others. This is in contrast to a UCS where the fitness of classifiers are determined independent of each other. This results in different learning rules for the fitness function of UCS and MOE. The fitness rule for UCS can still be derived from the probabilistic model given by Fig. (1), if we slightly change the training methodology of a mixture of experts. Instead of training all the classifiers together in a single MOE we form *complement mixtures* for each individual rule of the population. A *complement mixture* model can be written down as -

$$P(c, \mathbf{x}) = P(c|z'=1)P(\mathbf{x}|z'=1)P(z'=1) + P(c|z'=0)P(\mathbf{x}|z'=0)P(z'=0)$$

where $z' = 1$ corresponds to the $z = j$ condition of the original model and $z' = 0$ would be its complement. It must be noted that although we have changed the model to a two component MOE, the constituents of the mixture have the same probabilistic model as Fig. (1). The matching probability in a complement mixture model has the property that $\forall \mathbf{x} P(\mathbf{x}|z' = 1) = 0 \Rightarrow P(\mathbf{x}|z' = 0) \neq 0$ and vice versa. This property ensures that the complement of a condition matches all the input strings that the condition string itself does not match. We can now train all the rules in the rule set \mathcal{S} using complement mixtures. This would guarantee that the learning rule for each rule is independent of the others. The training and the prediction routines for a UCS formulated as a set of complement mixtures is illustrated in Fig. (2). For a complement MOE corresponding to rule j , the posterior probability derived in Eq. (5) will transform into -

$$\begin{aligned} P(z'_i = 1|c_i, \mathbf{x}_i) &= \frac{P(c_i|z'_i = 1)P(\mathbf{x}_i|z'_i = 1)P(z'_i = 1)}{\sum_{l=0,1} P(c_i|z'_i = l)P(\mathbf{x}_i|z'_i = l)P(z'_i = l)} \\ &= \begin{cases} 0 & P(\mathbf{x}_i|z'_i = 1) = 0 \\ 1 & P(\mathbf{x}_i|z'_i = 1) \neq 0 \end{cases} \\ &= \mathcal{I}(\mathbf{x}_i \text{ matches rule } j) \\ &= M(\mathbf{x}_i, r_j) \end{aligned} \quad (9)$$

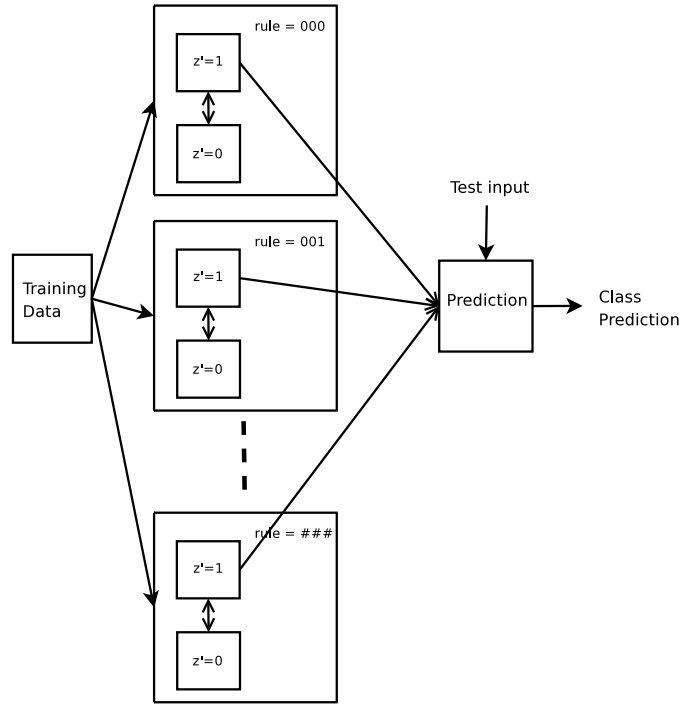


Figure 2: Training and prediction routines for UCS using complement mixtures

Substituting Eq. (9) in Eq. (6) we get -

$$\begin{aligned} \alpha_{j,k} &= \frac{\sum_i M(\mathbf{x}_i, r_j) \mathcal{I}(c_i == k)}{\sum_k \sum_i M(\mathbf{x}_i, r_j) \mathcal{I}(c_i == k)} \\ &= \frac{\sum_i M(\mathbf{x}_i, r_j) \mathcal{I}(c_i == k)}{\sum_i M(\mathbf{x}_i, r_j)} \end{aligned}$$

which is equivalent to the fitness measure as defined in Eq. (1). This demonstrates the link between an MOE and a UCS. In a UCS, complement mixture model is used during the training phase whereas the trained experts are combined as a conventional MOE during prediction. Hence, the model used for training is different from the model used for prediction and can result in inconsistencies. The issue of independent vs dependent training is well known in the field of ensemble learning and it has been shown that training an ensemble of experts without considering their correlations can lead to sub-optimal results [2]. A detailed explanation of independent learning in UCS and its potential inconsistencies is not within the scope of this paper but can be found in [5].

4.2 Matching function vs input probability

A matching function in UCS serves to select a classifier to include in the prediction process. The classifiers which match the test input string are included while calculating the sum of fitnesses. The matching function takes a value of 1 if the input string matches otherwise it takes value 0. On the other hand the input probability given by $P(\mathbf{x}|z = j)$ measures the degree of match of input string \mathbf{x} with rule j . This quantity measures the confidence of each rule in its prediction as a function of the input space. The match function is illustrated in Fig. (3), where a table of match probabilities is displayed. The rules form the column of the table and the

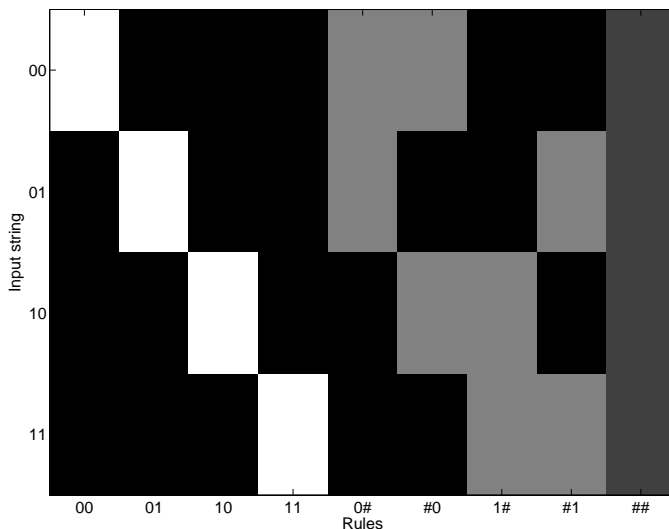


Figure 3: Illustration of the conditional probability of an input string given a rule

input string the rows. The intensity of the colour in the grid indicates the probability of match with white corresponding to probability 1 and black to 0. We can see that specific rules have high confidence over a small region of space in contrast to a generic rule which has a more diffused probability mass. Hence while calculating the support for a class using a sum of fitness, the fitness needs to be weighted by the confidence of each expert. The absence of confidence weighting in UCS is another of its drawbacks. To illustrate the difference with an example, let us consider a population having two rules - 001, 00#. Given a single training example of 001, both will have equal fitness, but, when used for prediction the more specific rule will inherently be more trustworthy than the generic one and hence must be weighted accordingly. This can be explained by the fact that while the generic rule is trained by all the different matching inputs, the concept represented by a specific rule is an unadulterated version of the original. It can also be related to the idea of underfitting (generic rule) and overfitting (specific rule).

4.3 Inclusion function vs rule probability

We now turn our attention to the other important quantity of the mixture model namely the probability of a rule being included as an expert given by $P(z)$. This quantity is the probabilistic equivalent of the inclusion function of UCS. As we train the mixture of experts we can observe $P(z)$ for some of the experts tends to zero which effectively amounts to removing that expert from the mixture. Furthermore, the rule probability term helps an MOE adapt to different noise levels in the training examples. When the noise level is low an MOE will favour specialised rules over generic ones due to higher confidence in predictions, but as the noise increases the specific rules become poor predictors of the class and then the probability distribution of the rules tends to favour generic rules. UCS lacks this flexibility due to its impoverished representation of the inclusion function. A partial solution to this problem lies in having different numerosities for the individuals of the population. This would provide a weighted averaging of fitnesses instead of an unweighted

sum of fitness. The numerosity of an individual is determined by a stochastic GA and hence is not amenable to a principled analytical treatment. However, empirical results in [1] points to correlation between the numerosity of an individual, the generality of its condition and its fitness. This might then be able to provide a much richer representation of the inclusion function.

5. RELATED WORK

The probabilistic formulation of UCS as a mixture of experts is closely related to the work in [4] although, in contrast to [4], we have provided a more robust analytical link between UCS and MOE by comparing the constituents of the two classification systems in Section 4. In [4] the probabilistic model is again a mixture of experts obtained by treating the rules as hidden variables and factorising the joint probability of the input and class variables as a sum over the hidden variables. However, the probabilistic model used in [4] is different from Eq. (3) in that the probability of the rule is conditioned on the input rather than being the other way around. This difference, though subtle, results in complicated learning rules for the parameters of the model. Specifically in [4], $P(z|\mathbf{x})$ is modeled by a softmax function with complicated training rules for the parameters. Another advantage of factorising the joint probability as in Eq. (3) is that the rule probability $P(z)$ acts as a natural measure of goodness for a rule and does not need any complicated model selection routines (chapter 7 of [4]) to obtain the optimal rule combination.

6. EVALUATION

In this section we study the properties of the probabilistic model using some empirical evaluations and compare it with the behaviour of UCS. As mentioned in earlier sections, we have switched off the GA in the UCS and include all the condition-class combinations in the population of UCS.

In the first experiment we demonstrate the working of the mixture of experts using a simple example of the *mux-3* dataset. The *Mux-3* problem involves determining the class of 3 bit strings with the most significant bit standing for the address bit and the rest of the two bits for data. The class of a bit string is decided by the value of the data bit addressed by the address bit. With 3 bit strings there are $M = 27$ possible rules in the global ruleset \mathcal{S} . These are then used as the experts and are trained using all possible bit combinations and their classes. The trained mixture of experts is displayed in Fig. (4(a)). The first 3 columns denote the bits of the rule with a black representing bit 0, white representing 1 and grey #. The last column in the figure stands for the class probability of each rule and the rules themselves have been sorted according to the decreasing probability of their occurrence (given by $P(z)$). We can see from the last column of the figure that the class probabilities have been learned accurately for each rule. One of the advantages of using a probabilistic formulation is that when we learn the parameters of the mixture model the probabilities of some of the rules tend to zero indicating that these rules can be removed from the population. This is demonstrated in Fig. (4(b)) where the probabilities of the 27 rules of the mux problem have been plotted and some of them are seen to approach zero.

In the next experiment, we analyse the effect of noise on

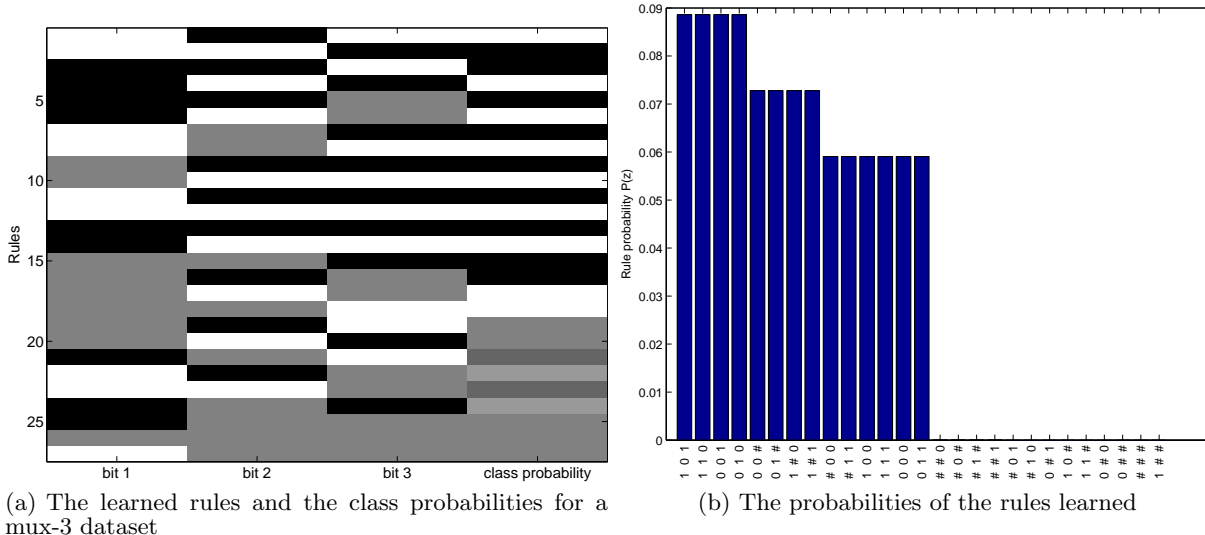


Figure 4: Demonstration of MOE learning mux-3 rules

the performance of the classifiers. We use the Mux-3 dataset which consists of 8 possible instances. We take all the 8 instances and replicate them 10 times to obtain a training set of size 800. The test data consists of the original 8 instances of Mux. Noise is modelled in the problem by choosing a random instance of the training data and flipping its class. The number of instances whose class is flipped is gradually increased from 0 to 800 thus corresponding to an increase in noise in the system. Here we have defined noise as possible conflicts in class labels between two instances of the same bit strings. At each noise level both the learners are trained on training data and tested on the test data. The performance of the classifiers are measured using two different criteria. Firstly we use the conventional classification accuracy to measure the percentage of correctly classified instances in the test set. The second measure is termed as the class ratio. This is the ratio of support for class 1 and support for class 0. For the classifiers if the support for class 1 is given by $S_1(\mathbf{x})$ and for class 0 as $S_0(\mathbf{x})$ then the class of \mathbf{x} is given by -

$$Class(\mathbf{x}) = \begin{cases} 0 & \text{if } S_1 < S_0, \\ 1 & \text{if } S_1 \geq S_0 \end{cases} \quad (10)$$

and the class ratio is defined as -

$$class\ ratio(\mathbf{x}) = \frac{\min(S_1(\mathbf{x}), S_0(\mathbf{x}))}{\max(S_1(\mathbf{x}), S_0(\mathbf{x}))} \quad (11)$$

From the equation, we can see that the class ratio measures the margin of separation of the two classes by the classifier. Maximum uncertainty corresponds to $class\ ratio = 1$ while minimum uncertainty corresponds to $class\ ratio = 0$. We use this quantity to compare the two classifiers. The class ratio and the accuracies of the two classifiers are plotted as a function of noise in Fig. (5(a)). As the noise increases both the classifiers start making mistakes but we find that MOE is able to tolerate more noise as compared to UCS. It must be noted that due to the construction of the training data as we cross the 50% mark for the noise, the problem becomes a mirror symmetry of the original problem and in

the case of 100% noise the training data is just the complement of the original problem. Hence we have restricted the graphs to “maximum” noise of 50%. The significant aspect of the result, averaged over 10 randomised train-test splits illustrated in Fig. (5(a)), is the evolution of the class ratio. We find that for the MOE, class ratio increases with noise and reaches a maximum when the noise is at its maximum at the 50% mark. The performance of UCS is much worse and is hardly able to adapt to the noise levels. This difference in performance can be mainly attributed to the probabilistic formulation and dependent learning of the mixture model as explained in Section 4.2. To further illustrate the internal workings of an MOE, Fig. (5(b)) plots the average rule probability grouped according to the generality of rules. There are four plots in the figure with one corresponding to the most specific rules which do not have any # in them, another one corresponds to rules with 1# in them and so on. In Fig. (5(b)), as the noise increases, the specific rules become less accurate and consequently the weight given to them by the rule probabilities decrease and more generic rules are favoured. On the other hand, effect of noise on UCS is modulated by the parameter v that we had earlier set to 1. In Fig. (6) we have plotted the class ratio of UCS for different values of v for varying noise levels. We can see that at high noise levels a low value of v is optimal and for low noise, a high value of v . Therefore, to obtain an optimal UCS classifier the v parameter must be manually tuned in accordance with the noise levels in the training data. This is a cumbersome process when compared to an MOE which adjusts its parameters automatically according to the level of noise in the data.

In the final experiment, we look at the effectiveness of a probabilistic representation of the inclusion function in an MOE and compare it with that of UCS. In this experiment we make use of mux-6 dataset. We generate all possible bit strings of length 6 and its corresponding classes. We then split this into training and testing datasets. A mixture of experts is trained with the training data and the performance measured on the test data. A UCS is constructed

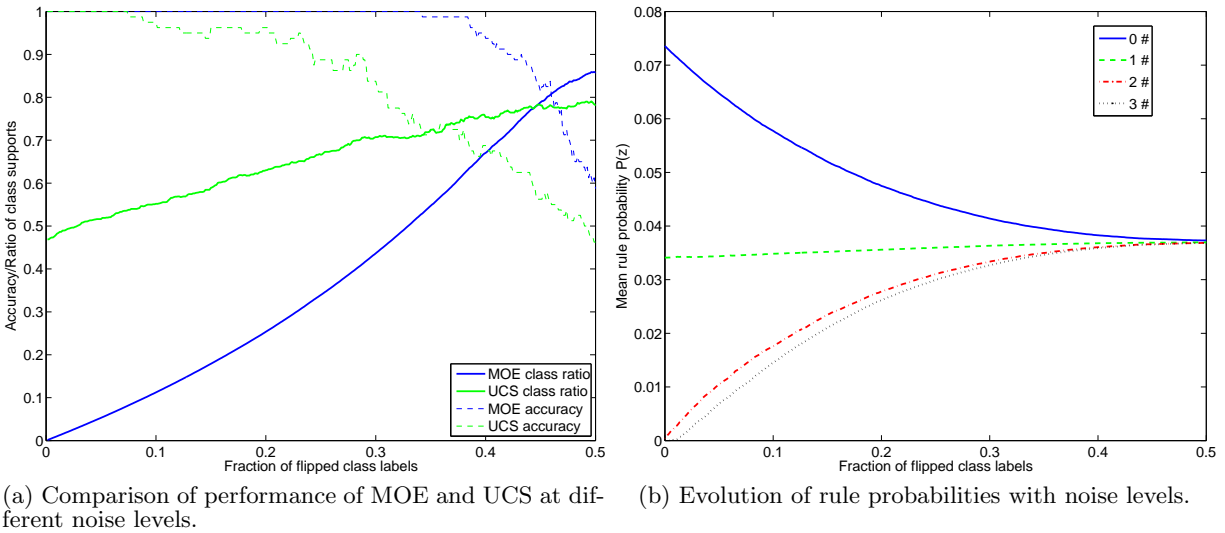


Figure 5: Effect of noise on learning

from all the rules of the rule superset \mathcal{S} and trained on the same training data and tested on the test data. During this process the GA is switched off so that the population sizes of a UCS and MOE are the same. Once the training is complete, the deletion heuristic of UCS is used to delete rules wherein the individuals which take part in the largest *correct sets*[1] get higher priority for deletion. After each deletion the accuracy of the prediction over the test set is measured. The same procedure is repeated with the mixture of experts where the probability of the rules is used as the basis for pruning. The accuracies for the UCS and MOE are shown in Fig. (7) where the plot corresponding to UCS is marked UCS(heuristic). Under ideal circumstances, as the quality of the rules in the population decreases the classification accuracy should fall, but the fall in accuracy should be gradual and graceful. By comparing the two learning methods we can see that the decrease in accuracy is much more regular in MOE than a UCS suggesting the importance of having a principled measure for the fitness of the rules. This experiment also serves to justify our choice of switching off the GA during comparisons between UCS and MOE. This choice was mainly motivated to factor out the effects of a stochastic learning method like GA and establish a level field for the learning algorithms. The main objective of a GA is to evolve populations with maximum fitness. When individuals are deleted based on their fitness, the population at any stage would contain the best individuals in them. Such a population would represent the best that a GA can learn. The evolution of error using fitness to delete individuals is shown in Fig. (7) as UCS(fitness). This particular procedure also underperforms when compared to an MOE. For the purpose of illustration we have also plotted in dashed line, the evolution of the scaled sum of fitness of the individuals of the population during the deletion process. These experiments strongly support the theoretical analysis that we have presented in this paper and illustrate the advantages of a principled probabilistic model for classifier combination.

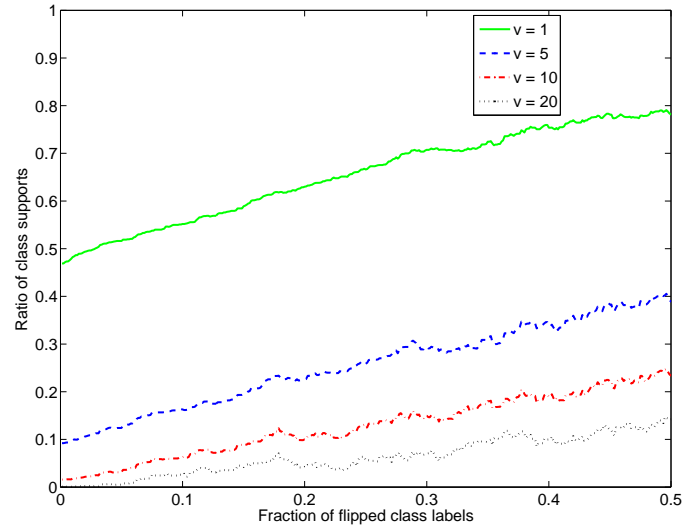


Figure 6: Effect of parameter v on the class ratio with noisy data

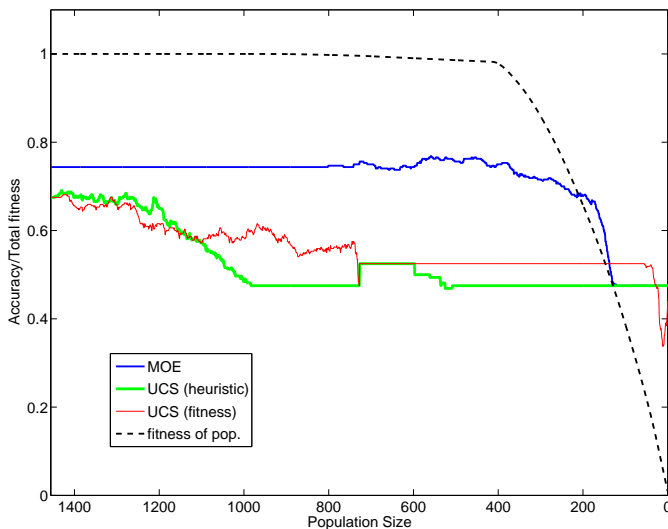


Figure 7: Comparison of UCS and MOE’s pruning rules on a mux-6 problem. The plot is an average over 10 different train-test splits.

7. DISCUSSION

In this work, we have developed a novel probabilistic model to explain the workings of a supervised Classifier System. For the first time a probabilistic model has been developed that matches the learning rules of UCS closely. The most significant contribution of this work is summarised in Section 4 where each component of UCS has been provided with a probabilistic counterpart in the mixture of experts model. Subsequent discussion in Section 4 proves that the MOE model is a richer formulation of the UCS model and allows the model to be more flexible as demonstrated through empirical evaluations in Section 6. In this work we have mainly concentrated on providing a principled probabilistic model for the supervised classifier system, but we find that the mixture of experts model is much more generic and powerful than a UCS. One of the drawbacks of MOE is that it uses all of the rules in \mathcal{S} which results in lots of open parameters that need to be learned. This results in an increased space complexity of the system. UCS copes with this situation by sampling from \mathcal{S} using a GA and needs to maintain only a subset of \mathcal{S} . The same strategy can also be used for a MOE wherein Monte-Carlo sampling methods can be used to learn the optimal rule probability when the size of \mathcal{S} is large. These are largely practical issues and is not of much concern in this paper which places emphasis on the theoretical aspects of the problem.

Supervised classifier system belongs to the family of Learning classifier systems and derives their rules of learning and prediction from the members of the family. This paper has been developed as a first step towards a probabilistic formulation covering the entire gamut of Learning Classifier Systems including reinforcement learners like XCS and is not an attempt to improve UCS.

8. 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] Gavin Brown, Jeremy Wyatt, and Peter Tino. Managing diversity in regression ensembles. *Journal of Machine Learning Research*, 6:1621–1650, 2006.
- [3] Arthur Dempster, Nan Laird, and Donald Rubin. Likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.
- [4] Jan Drugowitsch. *Design and Analysis of Learning Classifier Systems*. Springer-Verlag, 2008.
- [5] Narayanan U. Edakunni and Tim Kovacs. Probabilistic modeling of UCS : a theoretical study. Technical report, University of Bristol, 2009.
- [6] R. Jacobs, M. I. Jordan, Nowlan. S. J., and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.
- [7] Tim Kovacs. Genetics-based machine learning. In Grzegorz Rozenberg, Thomas Bäck, and Joost Kok, editors, *Handbook of Natural Computing: Theory, Experiments, and Applications*. Springer Verlag, 2009.
- [8] James Marshall, Gavin Brown, and Tim Kovacs. Bayesian estimation of rule accuracy in UCS. In *Proceedings of the 2007 GECCO Conference Companion on Genetic and Evolutionary Computation*, pages 2831–2834, 2007.