Online, GA based Mixture of Experts : a Probabilistic Model of UCS

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

Gavin Brown
School of Computer Science
University of Manchester
gbrown@cs.man.ac.uk

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

ABSTRACT

In recent years there have been efforts to develop a probabilistic framework to explain the workings of a Learning Classifier System. This direction of research has met with limited success due to the intractability of complicated heuristic training rules used by the learning classifier systems. In this paper, we derive a learning classifier system from a mixture of experts that is similar to a sUpervised Classifier System (UCS) in terms of its training and prediction routines. We start by framing the learning model as a mixture of experts which uses an Expectation Maximisation (EM) procedure to learn its parameters. The batch updates of the EM is then converted into online updates and finally into a GA based sampled online update thus ending up with a classifier system similar to a sUpervised Classifier System. In this paper, we show the effectiveness of such a system as compared to UCS through a series of comparative studies on test datasets.

Track

Genetics Based Machine Learning

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

Recent research in Learning Classifier Systems(LCS) has concentrated on deriving a principled probabilistic model for LCS [3, 5]. This is in contrast to the largely heuristic approach that has been pursued within the field of learning classifier systems. A probabilistic framework for LCS allows us to formulate more efficient and accurate learning rules for the learning classifier system in addition to providing a sound statistical interpretation of the learned model. One of the recent works in this direction has been the use of Mixture of Experts(MOE) [6] framework to explain the workings of a sUpervised Learning Classifier System(UCS) [5]. In [5], authors have borrowed the probabilistic framework of a mixture of experts to explain the different components of UCS. The model formulated in [5] provides principled probabilistic rules for training the classifiers but it manages to do it only by switching off the GA and using the entire set of possible rules as component classifiers. This approach is not faithful to UCS and is not efficient in its use of the classifier rules. In this paper, we extend their approach to overcome the difficulties with their model and provide a truly comprehensive derivation of UCS-like learning rules starting from the probabilistic framework of a mixture of experts. The end result is a probabilistic model with principled learning rules that combines the *space efficiency* of UCS with the *robustness* of mixture of experts.

In [5], the probabilistic mixture of experts framework was shown to have close correspondence to various components of UCS. Specifically, the macroclassifiers in UCS were represented as latent variables in MOE with the class probability conditioned on the latent variable serving as the fitness function. In addition, the probability associated with a latent variable was used as a measure to decide if the particular rule can be included in the population of rules. The prediction routine was also shown to be similar to UCS where a weighted average of the individual predictions of rules was used to decide the class for a given input pattern. However, the similarity of MOE with UCS was established by using a version of UCS in which the GA was switched off with the entire universe of rules being used for classification. This approach is not suitable for real world problems where it is infeasible to store all possible rules in a computer's memory. In this paper, we rectify this deficiency by modifying the training rules of MOE to work with a reduced set of classifier rules and constructively build up the optimal set of rules.

This paper is organised as follows : we start with a brief review of UCS in Section 2 followed by MOE in Section 3. In Section 4, we derive online updates for MOE and in Section 5 we use stochastic sampling based MOE training updates to convert the MOE into a UCS-like classifier algorithm.

GECCO'11,

Finally, in Section 6 we provide an empirical comparison of GA based MOE with UCS to demonstrate the advantages of a classifier with a probabilistic formulation over a heuristics based classifier.

2. UCS

sUpervised Classifier Systems (UCS) [1] is a form of Learning Classifier System which learns to classify using supervisory signals. This method of classification is inspired by XCS and borrows a number of learning rules from XCS[7]. A typical UCS system operates in a problem domain that consists of input string - class tuple where the input string is a binary string made up of D bits and a class label is an element of the set $\{1 \dots K\}$. The UCS algorithm maintains a population of rules where a rule is defined as a condition-action pair. *Condition* is a ternary string made up of alphabets from $\{0, 1, \#\}$ and *action* is a class as predicted by that particular rule. When a binary input string is presented to the classifier, it triggers the rules whose condition matches the input. The match is computed as a bit match between the input and the rule condition with the symbol # matching both 0 and 1. For example, given an input string 001 it would match rules that have conditions 001, 0#1,0## and so on. The collection of rules that match the input string form the *match set*. The match set can be further partitioned into K disjoint sets corresponding to rules that advocate one of the K classes. These sets are termed as *action sets*. Each rule is associated with a fitness value that is proportional to the classification accuracy of the rule. The support for each class is then computed by taking the mean of the fitness of each rule in the action set and the class with the greatest support is output as prediction for the given input. The crux of UCS, however is the GA algorithm that samples from the set of rules to build a population of rules which yields the best classification accuracy over the training data.

3. MIXTURE OF EXPERTS (MOE)

Mixture of experts [6] as the name suggests is an ensemble of experts whose predictions are combined through a weighted sum to produce an overall prediction. In [5], it was established that the prediction rules of UCS are similar to that of an MOE and is the same as an MOE when we place restrictions on the correlation between the constituent experts. While there can be alternative probabilistic formulations of MOE for a Learning Classifier System - [4] and [5] being two of them, we use the latter one due to its simplicity of formulation and simple training rules. In this section, we provide a brief summary of the MOE model as formulated in [5]. We then extend this model in later sections of the paper.

A mixture of experts model is best expressed as the likelihood of observing a certain data tuple $\mathbf{x}, c$ where $\mathbf{x}$ is the input and c the class label. It can be framed as a random generative process consisting of a sequence of random events that finally outputs the observed data. In a mixture of experts model, we assume that there are M experts which generate the data. The first step in the generative process is to choose an expert. This process involves a random event whose outcome is represented as a multinomial random variable Z which can take a value from 1 to M. The probability that a particular expert, say j, is chosen is given by the probability $P(Z = j)$. Having chosen an expert, this expert then generates an input binary string $\mathbf{x}$ out of the 2^D possible strings with probability $P(\mathbf{X} = \mathbf{x}|Z = j)$ and chooses a class c out of K possible classes with probability $P(C = c|Z = j)$. Here $\mathbf{X}$ denotes the input random variable and C the random variable for class label with $\mathbf{x}$ and c being their respective realisations. The probability of an observed data tuple $\mathbf{x}, c$ being generated by an expert j is then given by :

$$P(\mathbf{X} = \mathbf{x}, C = c|Z = j) = P(\mathbf{X} = \mathbf{x}|Z = j)P(C = c|Z = j) \quad (1)$$

The random variable Z is unobserved and we cannot be sure of the identity of the expert, hence the probability of observing the data tuple is the probability that either of the M experts generated the data and is given by a summation of the probabilities of data generated by an expert weighted by the probability of the expert itself :

$$P(\mathbf{X} = \mathbf{x}, C = c) = \sum_{j=1}^{M} P(\mathbf{X} = \mathbf{x}|Z = j)P(C = c|Z = j)P(Z = j) \quad (2)$$

In a classification task we observe specific realisations of the input random variables and classes in the form of training data and we aim to fit a model that would maximise the joint probability of these events. Specifically, the training data given by the tuples $(\mathbf{x}_1, c_1), \dots (\mathbf{x}_N, c_N)$ are realisations of the random variables $(\mathbf{X}_1, C_1) \dots (\mathbf{X}_N, C_N)$ and we associate with each tuple $(\mathbf{X}_i, C_i)$ an unobserved random variable Z_i that can take values from $1 \dots M$ and indicates the expert responsible for producing the i^{th} data point. Assuming that the data points are independent and identically distributed we can express the likelihood of the observed data as the joint probability of the N events :

$$P(\mathbf{X}_1 = \mathbf{x}_1 \dots \mathbf{X}_N = \mathbf{x}_N, C_1 = c_1 \dots C_n = c_N) = \prod_{i=1}^{N} P(\mathbf{X}_i = \mathbf{x}_i, C_i = c_i) \quad (3)$$

and the log of the likelihood is given by :

$$\mathcal{L} = \sum_i \log P(\mathbf{x}_i, c_i) \quad (4)$$

$$= \sum_i \log \sum_j P(\mathbf{x}_i|Z_i = j)P(c_i|Z_i = j)P(Z_i = j) \quad (5)$$

where we have omitted the explicit representation of the random variables $\mathbf{X}_i, C_i$ and just denote it by its actual realisations $\mathbf{x}_i, c_i$.

3.1 Training

Having obtained the log likelihood, we now need to find the best set of parameter values by maximising the likelihood with respect to the parameters. The parameters in our model are :

$P(Z)$: the probability of an expert being chosen prior to observing the data. In the MOE model, latent variable Z is associated with the ternary condition string of the UCS and the probability of these experts can be interpreted as a measure of the overall usefulness of an expert in the ensemble.

$P(\mathbf{X}|Z)$: the probability of producing a particular binary string given an expert. This parameter is fixed and has a constant value such that it replicates the match operation of a UCS. The conditional probability is given by :

$$P(\mathbf{X}|Z=j) = \begin{cases} 2^{-(\text{no. of \# in j})} & \text{if } \mathbf{x} \text{ matches } j, \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

$P(C|Z)$: the probability of a class label given an expert.

Among these parameters, $P(X|Z)$ is fixed and we need to learn the optimal values of $P(Z)$ and $P(C|Z)$. Optimal values of these parameters can be determined by maximising a surrogate function of the log likelihood given by :

$$\begin{aligned} \mathcal{J} &= \sum_i \sum_j P(Z_i = j|\mathbf{x}_i, c_i) \ln P(c_i|Z_i = j) \\ &+ \sum_i \sum_j P(Z_i = j|\mathbf{x}_i, c_i) \ln P(\mathbf{x}_i|Z_i = j) \\ &+ \sum_i \sum_j P(Z_i = j|\mathbf{x}_i, c_i) \ln P(Z_i = j) \end{aligned} \quad (7)$$

where $P(Z_i = j|\mathbf{x}_i, c_i)$ is the probability that the i^{th} data point was produced by expert j and is usually termed as the posterior probability of the expert j. The quantity $\mathcal{J}$ serves as a surrogate for the log likelihood such that maximising the surrogate function is equivalent to maximising the log likelihood[6]. The surrogate function $\mathcal{J}$ can be maximised by using an EM algorithm which iteratively finds the optimal values of the parameters by alternating between estimating the posterior probabilities given the current value of the parameters and re-estimating the parameters with the posterior probabilities kept fixed. The details of the EM algorithm used to estimate the parameters in the MOE are given in [5]; in this paper we just list the learning updates for the parameters :

Compute the posterior : In the first step, we compute the posterior over the latent variables Z as :

$$P(Z_i = j|c_i, \mathbf{x}_i) = \frac{P(c_i|Z_i = j)P(\mathbf{x}_i|Z_i = j)P(Z_i = j)}{\sum_j P(c_i|Z_i = j)P(\mathbf{x}_i|Z_i = j)P(Z_i = j)} \quad (8)$$

Here we assume that the parameters used to compute the posteriors are known and constant.

Compute the parameters : In this step we compute the values of the parameters assuming that the posteriors are known :

$$P(C = k|Z = j) = \frac{\sum_i P(Z_i = j|\mathbf{x}_i, c_i)\mathcal{I}(c_i == k)}{\sum_k \sum_i P(Z_i = j|\mathbf{x}_i, c_i)\mathcal{I}(c_i == k)} \quad (9)$$

where $\mathcal{I}(c_i == k)$ is an indicator function which takes value 1 when its argument is true else takes on value 0. The probability of an expert in turn is computed as :

$$P(Z = j) = \sum_{i=1}^{N} P(Z_i = j|\mathbf{x}_i, c_i)/N \quad (10)$$

Eqs. (8), (9), (10) are cyclically repeated till the parameters converge and we reach a local maximum of the likelihood.

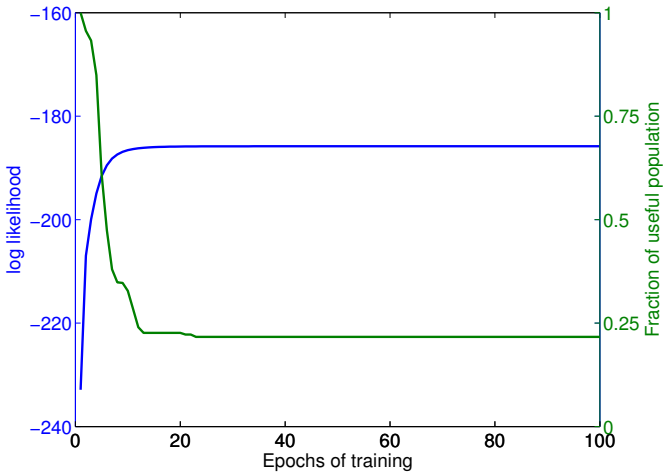

Figure 1: Illustration of the evolution of the log likelihood of the training data during the EM updates. Also illustrated is the evolution of proportion of the useful population during learning.

The process of iterative maximisation of the EM algorithm is illustrated in Fig. (1) where the evolution of log likelihood is shown as an MOE is trained on a typical dataset. We can see that this value increases monotonically and converges to a local maximum. The guaranteed monotonicity of EM is the main reason for using it. At the beginning of an EM the $P(Z)$ values of all the experts are initialised to the same value thus giving an unbiased starting point to all the experts, but as the model gets refined with each training epoch, some of the rules are less effective in prediction and their corresponding probabilities fall to zero. This leads to a sparse population as the EM procedure proceeds. This is illustrated in Fig. (1) where we plot the number of useful rules in the population. In this plot useful rules are defined as the first 95 percentile of the population. The percentile measure $R_{0.95}$ is computed by measuring the number of experts with the largest probability which jointly account for 95% of the cumulative probability of $P(Z)$ and is given by :

$$\begin{aligned} &R_{0.95} \in \{1 \ldots M\} \quad \text{such that} \\ &\sum_{j=1}^{R-1} P(Z = j) < 0.95 \quad \text{with} \\ &P(Z = 1) \geq P(Z = 2) \geq \ldots P(Z = M) \end{aligned} \quad (11)$$

As we can see from Fig. (1), the population gets sparser with each EM iteration and finally settles down to a fraction of the original population size thus retaining only the relevant rules in the population. This illustrates the role of $P(Z)$ as a sort of fitness function for the population of rules and will be used in later sections to select the best individuals in a population.

3.2 Prediction

In this section we look at the prediction routine for MOE. The trained MOE model includes parameters that maximise the likelihood of observed data. When presented with a previously unseen input point, MOE uses these parameters to predict the class of the test input. The class of the test

input is decided by comparing the joint probabilities of input and class evaluated over different classes. The class with the highest joint probability wins and is output as the prediction of the classifier. The joint probability of a class k and the test input $\mathbf{x}_q$ is computed as :

$$P(\mathbf{X} = \mathbf{x}_q, C = k) = \sum_j P(C = k|Z = j)_{learned} P(\mathbf{X} = \mathbf{x}_q|Z = j) P(Z = j)_{learned} \quad (12)$$

where $P(C = k|Z = j)_{learned}$ and $P(Z = j)_{learned}$ are the values of parameters learned by maximising the likelihood over the training data. The class prediction is then given by $Prediction = argmax_{k\in\{1\ldots K\}} P(\mathbf{X} = \mathbf{x}_q, C = k)$.

In this section, we have summarised the mixture of experts model as a probabilistic counterpart of UCS. The similarity of the MOE model to UCS, as derived so far, had already been demonstrated in [5] but there are still significant difference in the training paradigm employed in these algorithms. The differences in the training routines of UCS and MOE are listed below :

Online learning : The first difference is that UCS uses an *online training* routine which learns from a single data point at a time and discards it after using it. This reduces the space complexity of the algorithm by avoiding the need to store the entire data during the training process. This is in contrast with MOE which learns from a batch of data points.

Constructive addition of experts : The main contribution to the space complexity of an MOE algorithm is the storage of experts. For an input space of binary strings of length D the total number of ternary string rules would be 3^D. An MOE algorithm stores the parameters for each of these rules and computes the optimal values of these parameters. During the EM process some of the rules are found to be redundant as illustrated in Fig. (1) thus reducing the number of parameters, but the worst case space complexity remains $O(3^D)$. With increasing values of D, this space complexity becomes infeasible.

On the other hand, UCS uses a better approach to storing its experts. It *constructively* builds the population of experts by *sampling* the space of 3^D rule space and storing only the useful rules. Specific rules are sampled from the rule space, the parameters for these rules are computed and if found useful it is retained, else it is replaced by a more useful rule. This process of sampling from the rule space builds up the population incrementally resulting in an efficient space utilisation.

In following sections we reformulate the EM algorithm of an MOE to turn it into an online and constructive training algorithm similar to UCS with same or better space complexity as a UCS.

4. MOE WITH ONLINE TRAINING

In this section we adapt the training rules of the EM algorithm to learn from data in an online fashion. We start with training updates given by Eq. (9) and Eq. (10) where the parameter updates are expressed as a function of posterior probability of the experts and sufficient statistics of the data. To derive the online updates let us index the training data points by the time of its arrival t. At time t we must learn from the data tuple $\mathbf{x}_t, c_t$ and the parameter estimates learned from previous data points denoted as $P(C|Z)_{t-1}$ and $P(Z)_{t-1}$. Having observed the data point $\mathbf{x}_t, c_t$ at time t, we first compute the posterior distribution of expert j where $\forall j \in \{1\ldots M\}$:

$$P(Z_t = j|\mathbf{x}_t, c_t) = \frac{P(C = c_t|Z = j)_{t-1} P(\mathbf{X}_t = \mathbf{x}_t|Z = j) P(Z = j)_{t-1}}{\sum_j P(C = c_t|Z = j)_{t-1} P(\mathbf{X}_t = \mathbf{x}_t|Z = j) P(Z = j)_{t-1}} \quad (13)$$

We can now use this estimate to update a running statistic of the data which is subsequently used to update the value of the parameters. The running statistic for the conditional class probability is the numerator term of Eq. (9) denoted here by $sp_t(j, k)$:

$$sp_t(j, k) = P(Z_t = j|\mathbf{x}_t, c_t)\mathcal{I}(c_t == k) + sp_{t-1}(j, k) \quad (14)$$

where $sp_t(k)$ is the estimate for class k at time t and $P(Z_t = j|\mathbf{x}_t, c_t)$ is obtained from Eq. (13). The parameter value of the conditional class probability is then computed as :

$$P(C = k|Z = j)_t = \frac{sp_t(j, k)}{\sum_k sp_t(j, k)} \quad (15)$$

We can derive similar online update for the probability of the rule ($P(Z = j)$) by starting out with the batch update expression for t data points :

$$P(Z = j)_t = \sum_{i=1}^{t} P(Z_i = j|\mathbf{x}_i, c_i)/t \quad (16)$$

which can be rewritten as :

$$\begin{aligned} P(Z = j)_t &= \frac{\sum_{i=1}^{t-1} P(Z_i = j|\mathbf{x}_i, c_i)}{(t-1)} \frac{t-1}{t} \\ &+ \frac{P(Z_t = j|\mathbf{x}_t, c_t)}{t} \quad (17) \\ &= P(Z = j)_{t-1} \frac{t-1}{t} + \frac{P(Z_t = j|\mathbf{x}_t, c_t)}{t} \quad (18) \\ &= P(Z = j)_{t-1} \\ &+ \frac{1}{t}\left[P(Z_t = j|\mathbf{x}_t, c_t) - P(Z = j)_{t-1}\right] \quad (19) \end{aligned}$$

where we have used the definition of $P(Z = j)_{t-1}$ as $\frac{\sum_{i=1}^{t-1} P(Z_i=j|\mathbf{x}_i,c_i)}{(t-1)}$ to modify the equation into a recursive estimate.

The online estimates derived in Eqs. (13) - (19) can be used to learn the parameters that would incrementally maximise the likelihood of the observed data. This way of learning from a stream of data removes the necessity to store the training data during learning thus reducing the space complexity of the training routine. However, this does not reduce the space complexity of the learned model which still needs to store the parameters corresponding to the entire rule space and hence does not scale up with the length of the input strings. In the next section we transform the online EM learning rules into a sampling based learning such that the optimal set of rules can be built up incrementally.

5. ONLINE MOE WITH GA BASED TRAINING

In the preceding sections we have been using learning algorithms that pruned the experts in an ensemble by learning the usefulness of the rules based on the estimate of the probability $P(Z)$ for each individual rule in the rule space. This approach does not scale well with increasing size of rule space. In this section, we derive a set of learning rules for MOE such that it can constructively build an optimal subset of the rules and at the same time learn the parameters of these rule experts.

As we demonstrated in Section 3 through Fig. (1), the quantity $P(Z = j)$ gives the usefulness of a rule j amongst M possible rules. Probability given by $P(Z)$ is a discrete distribution and takes a value between 0 and 1 such that $\sum_j P(Z = j) = 1$. It is possible to represent a discrete probability distribution as a frequency distribution with the probability of an expert j given by its relative frequency of occurrence :

$$P(Z = j) = \frac{n_j}{\sum_j n_j} \quad \text{where } n_j \text{ is a positive integer} \quad (20)$$

The term n_j used in Eq. (20) is similar to the numerosity used in traditional Learning Classifier Systems. Similar to LCS, the numerosity of a classifier expert is proportional to its usefulness in predicting the classes correctly. The numerosity based representation of $P(Z)$ allows us to impose a condition of sparsity by bounding the sum of the numerosities. For instance, by bounding the total numerosity of all the rules in the population to some quantity, say Q, we are assured that only a maximum of Q experts with non-zero probabilities for $P(Z)$ can be represented. This also allows us to sample from the rule space by varying the values of n_j and computing the likelihood for a particular subset of the rule space. An addition of a rule from the rule space into the population would correspond to an increment in its numerosity and a deletion from the population space would correspond to a decrement in its numerosity. Through a series of addition and deletions we can find the optimal distribution of numerosities and hence the optimal values of $P(Z)$ for different rules. The effect of this sampling based approach is that we are able to learn the parameters of the experts in a population while still restricting the number of experts in the population.

We can reuse the same updates of the parameters as given by Eqs. (15) and (19) with the modification that the quantity M is now defined as the number of experts in the population which keeps changing in step with the sampling procedure. Here, we use GA to sample from the space of all possible 3^D conditions. The GA based sampling is directed by its fitness function defined over the space of conditions. In UCS the fitness of an individual is proportional to its classification accuracy whereas in the probabilistic setting we have seen that the usefulness of a ternary condition is decided by its probability $P(Z)$ computed by Eq. (19). We now introduce a new parameter for the fitness of an expert j - $F(j)$ which is updated as in Eq. (19) :

$$F_t(j) = F_{t-1}(j) + \frac{1}{t}\left[P(Z_t = j|\mathbf{x}_t, c_t) - F_{t-1}(j)\right] \quad (21)$$

The fitness values computed using the update in Eq. (21) is proportional to the probability of an expert and indicates its usefulness. The fitness value in turn can be used to modulate the numerosity of the experts in the population through sampling. Rules are sampled by adding and deleting experts as detailed below :

Addition : addition of experts happen through two different operations - *covering* and *crossover/mutation*. When training input string does not match any of the rules in the population, then the covering operation is invoked to introduce an expert that would match the given input string with its class probability $P(c = 0|z_{new}) = 0.5$ denoting total uncertainty. At frequent intervals of the learning process, GA is invoked which then introduces new experts into the population after possibly applying crossover and mutation to the condition part of the expert rules. The parent rules for the crossover are chosen based on its fitness. The covering, crossover and mutation operations are similar to the ones employed in XCS [2] and UCS [1] except for the definition of the fitness function.

Deletion : when the number of experts exceed a certain threshold ($\sum_j n_j > Population_{maxsize}$), the deletion operation is invoked to decrement numerosity of the experts with the lowest fitness. The deletion operation thus restores the sum of numerosities to a value less than the threshold. Here again the fitness of individuals play a major part to decide the numerosities of individual experts in the population. The deletion operation is also the same as in UCS and XCS.

After each run of sampling, the numerosities of the classifiers get updated and the updated numerosities are used to calculate new values for $P(Z)$ using Eq. (20). The introduction of numerosities to compute the expert probabilities makes the MOE training routine similar to UCS. This is clear if we write down the prediction rule given by Eq. (12) in terms of numerosities :

$$P(\mathbf{X} = \mathbf{x}_q, C = k) = \frac{\sum_j P(C = k|Z = j)P(\mathbf{X} = \mathbf{x}_q|Z = j)n_j}{\sum_j n_j} \quad (22)$$

Algorithm 1 Training routine for the online GA based mixture of experts

for $t = 1$ to N **do**
  Params : M, $P(C|Z)$, $P(Z)$, n_j, F
  $\forall j \in \{1 \dots M\}$ compute $P(Z = j) = n_j / \sum_j n_j$
  if $\sum_j P(\mathbf{x}_t|Z = j)P(Z = j) = 0$ **then**
    apply covering with $\mathbf{x}_t$
  end if
  for $j = 1$ to M **do**
    compute $P(Z_t = j|\mathbf{x}_t, c_t)$ using Eq. (13)
    $\forall k \in \{1 \dots K\}$ compute $sp_t(j, k)$ using Eq. (14)
    $\forall k \in \{1 \dots K\}$ compute $P(C = k|Z = j)_t = \frac{sp_t(j,k)}{\sum_k sp_t(j,k)}$
    compute $F_t(j)$ using Eq. (21)
  end for
  if $(t - recentGArun) > GA_{threshold}$ **then**
    apply crossover with probability χ (refer [2])
    apply mutation with probability μ (refer [2])
    delete individuals from the population if necessary.
    $recentGArun \leftarrow t$
  end if
end for

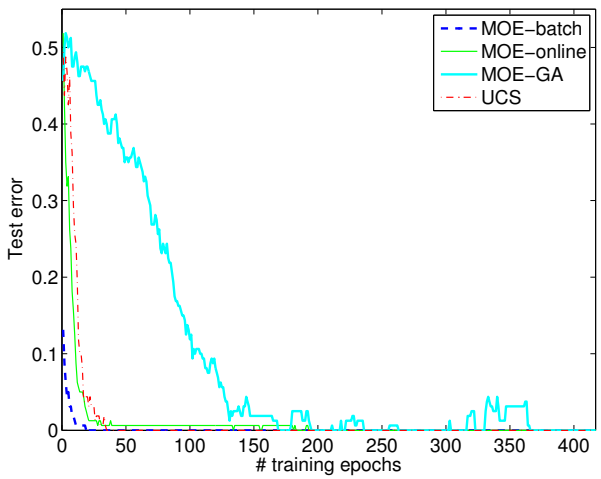

Figure 2: Comparison of the classification error on the test dataset for different learning algorithms. Each of the curves is an average of the performance over 10 disjoint train-test datasets.

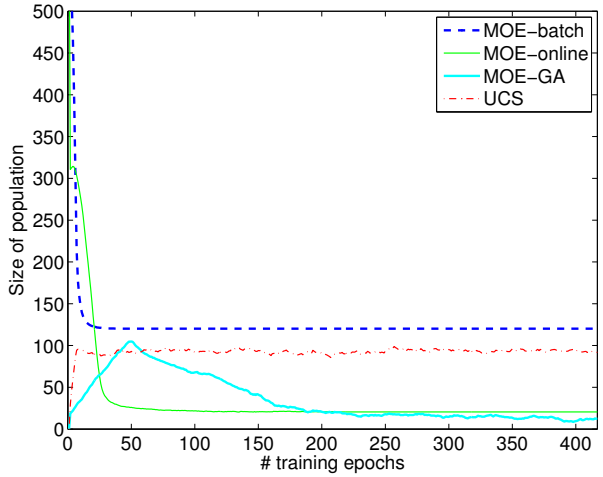

Figure 3: Comparison of the population sizes achieved for different learning algorithms. Each of the curves is an average of the performance over 10 disjoint train-test datasets.

which is similar to the class support computed by UCS :

$$S_k(\mathbf{x}_q) = \frac{\sum_j F_{UCS}(j) M_{UCS}(\mathbf{x}_q, j) n_j}{\sum_j n_j} \tag{23}$$

where $F_{UCS}(j)$ is the fitness function defined in UCS and $M_{UCS}(\mathbf{x}_q, j)$ is the matching function that takes value 1 if $\mathbf{x}_q$ matches condition of rule j else takes a value 0. The fitness function of UCS used to compute the prediction is equivalent to the conditional class probability, the match function corresponds to the match probability and the numerosity serving the same purpose in both the formulations. This correspondence had already been noted in [5] and the inclusion of the numerosity term in the prediction extends this correspondence. The pseudo-code for training the GA based online MOE is shown in Algorithm 1.

6. EVALUATION

In this section we use sample datasets to compare the learning characteristics of different learning algorithms developed in this paper with the UCS algorithm. In our evaluations[1] we compare the mixture of experts model learning from a batch of data (MOE-batch) explained in Section 3, mixture of experts that learns from data using an online training updates (MOE-online) introduced in Section 4, online mixture of experts with GA based sampling approach (MOE-GA) developed in Section 5 and UCS as developed in [1].

In our first evaluation, we compare the performance of each of the algorithms on a dataset that consists of 6 input bits with the class being determined by the *xor* of the first two bits. The remaining 4 bits are unused and are irrelevant. Each of the learning models are trained with 48 randomly chosen data points from this dataset and the remaining 16 data points are used to test the accuracy of prediction of each of the models. Fig. 2 plots the evolution of the test error with the epochs of training. Each epoch of training consists of a round of training on the training set of data.

We can see that the batch EM algorithm converges first to a zero error followed by MOE-online, UCS and MOE-GA. This is in line with our expectations since the batch algorithm is able to build a robust model with all of the training data whereas the online algorithms build models from sparse data in the beginning which causes a bias in the model learned and then it takes a number of epochs to reduce this bias. The difference between the convergence rates of UCS and MOE-GA can be attributed to the nature of the experts in these systems. In UCS, the experts are assumed to be independent and has independent learning rules. The addition and deletion of an expert does not affect the other experts, but on the downside it would need a lot of experts to cover the input space completely and produce accurate classification. The MOE-GA on the other hand uses training rules that produces correlated update rules for the experts, thus deleting or adding an expert through the sampling process can imbalance the correlations and would need relearning to reestablish the correlation for a new set of experts. The advantage of having the correlations is that the input space is divided amongst the experts efficiently and thus requires fewer number of experts for achieving comparable performance. This hypothesis is supported by the plot in Fig. (3), of the number of experts (number of rules with $n_j \geq 1$) in the population. We can see that the MOE-GA is able to achieve a sparser population compared to UCS while maintaining a zero classification error on the test data.

6.1 Effect of noise

In the next evaluation we compare the performance of the learning algorithms on datasets with noise. In particular we evaluate the ability of the different learning methods to estimate the generative process responsible for the data. We use the *mux-3* data to evaluate the performance of the learning methods. The *mux-3* data consists of binary strings of length 3, the first bit of the string is the address bit and

[1]The MATLAB implementation of the evaluations can be found at http://www.cs.bris.ac.uk/~nara/

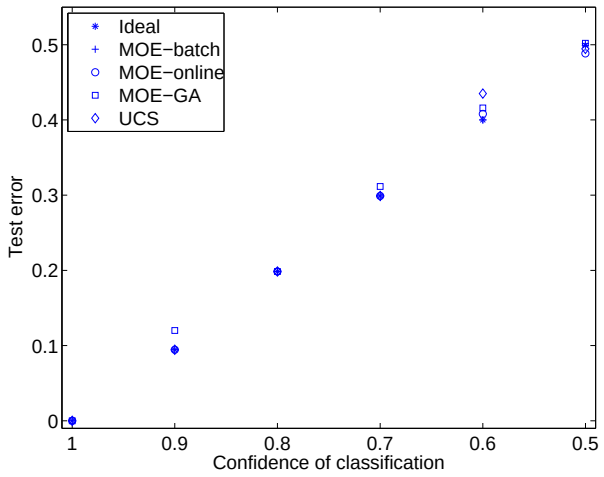

Figure 4: Comparison of the classification errors obtained on the noisy mux3 dataset. The plots shown are the average of 10 fold train-test evaluations.

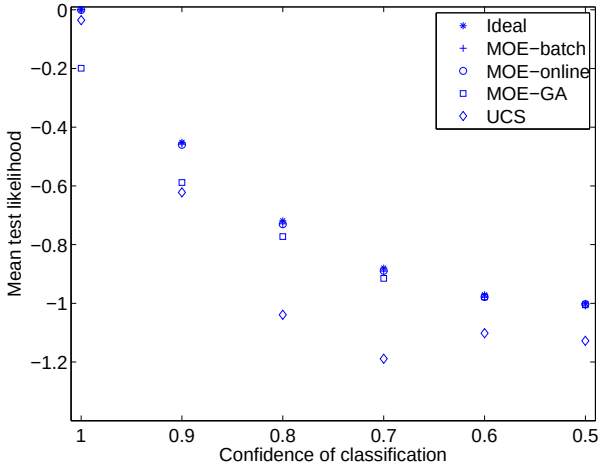

Figure 5: Comparison of the mean test log likelihood obtained on the noisy mux3 dataset. The plots shown are the average of 10 fold train-test evaluations

the rest of the bits are the data bits. In a noiseless case, the class is determined by the data bit which is addressed by the address bit. In a noisy case, the observed class label matches the actual class with a probability of p. For example, if the binary string is 010, the class label is 1 with probability p and is 0 with probability $1-p$. For the purpose of the evaluation, we generate 1000 noisy samples and split them into 10 disjoint train-test datasets. The learning algorithms are trained on the training data and the error of these algorithms are evaluated on the test data. An important characteristic of the evaluation setup is that the test data includes noise in it and to produce minimum risk classifications of the test samples, the learning algorithms must be capable of estimating the correct value of p. Hence, we evaluate the learning algorithms under differing values of p which ranges from 0 (noise-free dataset with complete certainty) to a value of 0.5 (random class assignments to the input data) to see if the algorithms are able to deal with the noise.

We evaluate the performance of the learning methods based on two different measures - the misclassification error and the log likelihood of the test data. The former measure is the commonly used criterion where we measure the classification error using the test set. The latter measure is defined as :

$$LogLikelihood = \sum_{i \in c_i=1} log_2(P(c_i = 1|\mathbf{x}_i)) + \sum_{i \in c_i=0} log_2(P(c_i = 0|\mathbf{x}_i)) \quad (24)$$

The log likelihood measures the confidence of a classifier in predicting a particular class. For instance, if a classifier predicts a class with complete confidence ($P(c|\mathbf{x}) = 1$) then its log likelihood will be 0. If the classifier is completely uncertain about its prediction ($P(c|\mathbf{x}) = 0.5$) its log likelihood will be -1 and if it predicts the wrong class with complete confidence ($P(c|\mathbf{x}) = 0$) then the log likelihood for that particular data point will be $-\infty$. An optimal classifier thus maximises this likelihood measure by predicting the correct class with maximum confidence. There is however a theoretical limit that can be reached when there is noise. A classifier cannot be expected to perform better than the theoretical limit.

Table 1: Description of the UCI datasets

Dataset	# train pts.	# test pts.	# features/bits
car	1296	432	12
mushroom	6093	2031	51
kr-vs-kp	2397	799	37

In Fig. (4) we plot the test error of different learning algorithms on the noisy *mux3* dataset for varying values of p on the x-axis. The noise in the test data set prevents any of the learning methods to achieve zero test error for $p < 1$, but the significant trend of the graph shows that the learning methods based on the MOE model is able to estimate the noise in the system better than UCS. The MOE based learners follow the trend of the "ideal" classifier (a classifier that knows the actual class labels and the probability p of generating the class labels) hence demonstrating the superiority of probabilistic classifiers in making confident predictions. We further obtain evidence of the difference between UCS and the MOE models of classification by plotting the mean test log-likelihoods for each of the p values as shown in Fig. (5). The plot shows that the UCS clearly departs from the ideal classifier and all the MOE based classifiers in estimating the noise levels in the dataset. These results are a confirmation of the trend witnessed in the experiments conducted in [5].

6.2 Performance on UCI dataset

In our final experiment, we compare the performance of the algorithms on real world datasets obtained from the UCI repository. The details of the data are given in Table 1 and the performance of MOE-GA and UCS are given in Table 2. It must be noted that batch algorithms which uses the entire set of rules for prediction are not feasible for these datasets, hence the comparison is limited to MOE-GA and

Table 2: Comparison of MOE-GA with UCS on various UCI datasets. The values in the parenthesis denote the standard deviation for the mean values given.

Dataset	MOE-GA			UCS		
	Accuracy	Likelihood	# macro classifiers	Accuracy	Likelihood	# macro classifiers
car	0.93(0.01)	-0.41(0.07)	71.6(65.8)	0.94(0.01)	-7.08(3.91)	224.1(12.6)
mushroom	0.64(0.02)	-0.85(0.02)	295(1.4)	0.67(0.03)	-1.08(0.66)	295(1.8)
kr-vs-kp	0.63(0.03)	-0.94(0.02)	479.2(4.05)	0.56(0.03)	-27.77(5.43)	488.7(5.1)

UCS. The parameters for the GA including the maximum population threshold were initialised to the same values for both MOE-GA and UCS. The performance is measured by splitting the dataset into training data and test data. The algorithms are trained on the train data and its performance measured on the test data. The mean of the performance metrics taken over 10 such train-test splits have been reported in Table 2. The performance of algorithms are again compared using the metrics of classification accuracy and the likelihood. For this particular experiment we have modified the likelihood measure such that $log_2(0)$ is truncated to a high but constant value so as to deal with situations that would otherwise yield $-\infty$. It is clear from the results in Table 2 that UCS and MOE-GA have a similar performance in terms of its classification accuracy, but there is a significant difference in the confidence that they attribute to their predictions. It can be seen from the log-likelihood that while MOE manages to attribute the right confidence levels to its predictions, the confidence levels of UCS is worse than a random classifier that chooses classes with equal probability. Another significant difference between the two is the number of macro classifiers each of them utilise to produce these results. The difference is significant in the *car* dataset where MOE is able to achieve a classification performance at par with UCS while using a significantly lesser number of rules.

7. DISCUSSION

In this paper, we have seen that a probabilistic model can be built that has the same characteristics of a sUpervised Classifier System and enjoys the same advantages of UCS while being strictly derived from mathematical principles. This is the first time that a paper has provided a comprehensive analogue of an LCS system. The statistical model of mixture of experts ties the UCS to mainstream machine learning paradigms that has a lot of research pertaining to such models of combining experts. We can conclude from this research that UCS is a special instance of a mixture of experts that is adapted to learning high dimensional discrete spaces using sampling to control the space complexity. The probabilistic model, apart from providing a firm theoretical ground for deriving training rules also scales up the learning algorithm to deal with noisy observations as demonstrated by the experiments in Section 6.

In this work, we have also shown that in cases of a small input space with sufficiently small D a batch algorithm of MOE is far better than a GA based online algorithm. This allows us to decide the optimum learning algorithm to use depending on the characteristics of the input space. For a UCS it is not easy to convert from a classification to regression and requires substantial effort in remodifying the learning updates to cover the continuous targets. For a MOE model it is fairly trivial to do the same by changing the probability distribution defined on the output variable $P(C|Z = j)$. The triviality of these modifications is aided due to the principled learning rules and the generative model used to learn from the data.

In future the same probabilistic model can be extended to provide an overarching framework to the whole family of Learning Classifier Systems thus unifying the different versions of the same designed to address different problems.

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] Martin V. Butz and Stewart W. Wilson. An algorithmic description of XCS. *Lecture Notes in Computer Science*, 2001.

[3] Jan Drugowitsch. *Design and Analysis of Learning Classifier Systems*. Springer-Verlag, 2008.

[4] Jan Drugowitsch and Alwyn M. Barry. A formal framework and extensions for function approximation in learning classifier systems. *Machine Learning*, 2008.

[5] Narayanan U. Edakunni, Tim Kovacs, Gavin Brown, and James Marshall. Modeling ucs as a mixture of experts. In *Proceedings of the 11th Annual conference on Genetic and Evolutionary Computation*, 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.