2006 International Joint Conference on Neural Networks
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

SOM-Based Sparse Binary Encoding for AURA Classifier

Simon O'Keefe and Oleksiy K. Dekhtyarenko, *Member, IEEE*

***Abstract*—The AURA *k*-Nearest Neighbour classifier associates binary input and output vectors, forming a compact binary Correlation Matrix Memory (CMM). For a new input vector, matching vectors are retrieved and classification is performed on the basis of these recalled vectors. Real-world data is not binary and must therefore be encoded to form the required binary input. Efficient operation of the CMM requires that these binary input vectors are sparse. Current encoding of high dimensional data requires large vectors in order to remain sparse, reducing efficiency. This paper explores an alternative approach that produces shorter sparse codes, allowing more efficient storage of information without degrading the recall performance of the system.**

I. Introduction

THE AURA (Advanced Uncertain Reasoning Architecture) system provides (via a C++ library) classes and methods for rapid partial matching of large data sets [1]. The conceptual building blocks for AURA systems are Correlation Matrix Memories (CMMs), which are matrices of strictly binary elements. Records are stored as associations in a CMM between a binary input and a binary output vector. Information is retrieved from the CMM by presentation of an input vector and production of the corresponding output vector or vectors by a matrix-vector multiplication and a threshold operation. As the number of associations or patterns stored increases, the CMM will *saturate* – that is, the vectors retrieved will not faithfully reproduce the output vectors used in the pattern storage phase. The rate at which the CMM saturates is determined in part by the number of bits set in each input and output pattern – sparse patterns result in fewer bits used to store each association, and hence reduce the rate of saturation. Thus for efficient operation of the CMM (and hence AURA systems) we need a data coding for both inputs and outputs that uses as few bits as possible. Conventional approaches to encoding of inputs and outputs, for example that used in the *k*-NN classifier described in [2], use one set bit for each variable in the input data. High dimensional data sets use many bits, and therefore need large vectors for them to be sparse. Even with intelligent compression of the CMM, this is an inefficient use of storage. This paper explores an alternative approach to coding high-dimensional input data that will produce shorter sparse codes, and demonstrates that the new encoding does not degrade the recall performance of the system.

The remainder of this section describes how AURA is used to construct a classifier, which may be used to classify previously unseen data based on the associations between known data and their classes stored in a CMM. Section II proposes a new encoding algorithm that is intended to produce shorter and therefore more efficient input codes for the CMMs used in AURA systems. The experimental evaluation of the proposed system is presented in Section III, with results in Section IV. Conclusions about the efficacy of the proposed encoding algorithm are presented in Section V.

A. The Importance of Encoding for CMMs

The purpose of input encoding is to transform real data into the form required by the CMM part of AURA. The objective of this work is to reduce the size of input vectors produced by the encoding as compared to those produced by current methods. The encoding of the vectors interacts with the effective storage and recall of data. For example, it has been shown in [3] and elsewhere that there is a trade-off between the encoding and performance of k-nearest neighbor classifier systems built around the CMM core, using AURA to recall vectors similar to those presented to the system.

One of the characteristics of the CMM is that it handles recall from partial input data well, allowing the system to generalize easily from stored data to correctly classify partial or corrupt input data. However, this property is lost if the input encoding is such that it does not map similar input data (typically multidimensional real data) to similar encoded inputs (sparse binary vectors).

Another important feature of AURA systems (important from the point of view of ease and efficiency of implementation) is that they operate best when the input encoding is sparse *with a fixed number of bits set to one*. Thus we are restricted in the number of valid binary vectors that we can map the real data to.

When we create an AURA application or system, the CMM encoding parameters that we have are the length p of the binary vector that forms the input to the CMM, and the number q of bits that we are going to set in that binary vector. The values selected for p and q affect performance. As is discussed in Section I.C below, with the current "quantized reals" approach the number of bins we choose for quantizing each input variable determines the length of the input vector and hence the size of the CMM. However, this

This work was supported in part by the INTAS Young Scientist Fellowship YSF 03-55-795.

S. O'Keefe is with the University of York, Department of Computer Science, Heslington, York, YO10 5DD, United Kingdom (phone: +44-1904-432762; fax: +44-1904-432767; e-mail: sok@cs.york.ac.uk).

O. K. Dekhtyarenko is with the Institute of Mathematical Machines and Systems, Department of Neurotechnologies, 42 Glushkov Ave., Kyiv, 03187, Ukraine (e-mail: olexii@mail.ru).

is not necessarily the most effective or efficient approach to the input encoding.

Analysis of the performance of CMMs in the *general* case (in which random data are stored and retrieved) shows that the most efficient operation of the CMM is obtained when we choose p and q such that $q \approx log_2 p$ (see [1-3]) without restriction on the selection of the q bits for each input vector. That is, we obtain the most efficient information storage when we choose the number of bits to set in each input vector as $\log_2$ of the length of the input vector (in bits), and are able to choose freely which q of the p bits in the vector we will set to represent the data. However, the data in real applications is usually very non-random, so the relationship between p and q for most efficient storage of a particular set of (non-random) data may not be $q \approx log_2 p$, and we should therefore determine experimentally what combination of p and q gives the best results in practice.

The specific condition that we have no restriction on the selection of the q bits is not met by current encoding methods, which usually restrict the selection greatly. Current encoding methods are therefore not optimal, and there is thus the possibility that we can improve on the efficiency of current encoding methods if we can find a mapping from data to binary vectors that allows us to choose the q bits with more freedom.

B. *AURA* k-*NN Classifier*

The AURA *k*-NN classifier [3] is an implementation of a k-nearest neighbour classifier that has been shown to be efficient in storage of data and rapid in recall (and therefore classification). The system stores a complete set of training data in a CMM, with the CMM input representing the attributes or values of a piece of data and the corresponding CMM output representing the class of that piece of data. Each test example is classified on the basis of the classes of the k most similar data in the training set. Two data are similar (in the simplest case) when their representations as binary vectors have a small Hamming distance.

Storing training data is a single epoch process with one training step for each input-output association. The j^{th} input datum is converted to a binary vector I_j and the corresponding output to a binary vector O_j . The cross products $O_j \times I^T_j$ of these pairs of binary vectors represents their association, and there is one cross product for each record in the data set. The “training” process is the creation of the CMM from these cross products. As can be seen from equation 1, each association is added to the CMM by forming the bitwise logical OR between the current CMM (CMM_c) and the new association, causing additional bits in the CMM to be set to 1:

$$CMM = CMM_c \vee (O_j \times I_j^T) \tag{1}$$

where $\vee$ is bitwise logical OR .

Recall of the O vector when presented with the corresponding I vector is achieved by multiplication of CMM and I and thresholding of the result, as show in equation 2.

$$O_t = \theta(r) = \theta(CMM \cdot I_t) \tag{2}$$

where θ is a thresholding function, and r is referred to as the raw output. The formation of the CMM guarantees that, for a previously seen input, the raw output r will contain values that are a maximum for the positions in O_t at which bits were set during the training phase, but there will be “noise values” in the vector as well. The threshold process removes the noise and returns a binary output vector. The form of θ depends on the application. Typically this will be “Willshaw thresholding”, with the threshold set at a pre-determined level, which is convenient when the inputs have a fixed number of bits set and the expected output is therefore known. Alternatively “LMAX thresholding” adapts the threshold so that L bits remain set, and is useful when the input is previously unseen, is corrupted or is incomplete. By selecting the threshold function and and adjusting the threshold value, the output vectors for one or more matching or partially matching records may be retrieved from the CMM for any particular input.

Similarity of the CMM input and the data stored in the CMM is computed in one step in the recall process. The AURA *k*-NN classifier relies on this similarity between bit patterns coding input data as a proxy for the neighbourhood calculation necessary for the *k*-NN approach. Any new coding must therefore not only be more efficient, but have a similarity metric between patterns that follows the similarity between the actual data, whether this is based on Euclidean distance or some other metric.

C. *Current Methods of Binary Encoding*

The conversion of real-valued data into data with discrete attributes is referred to as binary encoding, and equivalently as discretisation, quantization or binning. As well as enabling the usage of discrete learning algorithms (such as the AURA classifier) it has been shown elsewhere that data discretisation may speed up or improve the accuracy of other learning algorithms [4].

The current method used to construct the binary input vector corresponding to a record or data, as used in [3], transforms each variable or field in that record into a binary vector separately, by quantisation of the range of values taken by that field over the set of records, and assignment of a pattern of bits to represent each quantum. The final input vector is formed by concatenation of the vectors for each field. If each field is represented by a vector with one bit set to one, then the number of bits set in the final vector will equal the number of fields in the record. However, the placement of each set bit is restricted to those positions in the final vector corresponding to its particular field.

As an example, consider a record of 5 fields, each a real number in the range 1..10. A possible quantisation of the real data is into five subranges or bins. If each field is represented by 5 bits, of which only one is set to 1, then the concatenation of the 5 fields yields 25 bits with 5 bits set. For a record comprising the values {7,2,5,1,10} this could be represented by the set of binary vectors $\{[00010]^T, [10000]^T,$

$[00100]^T$, $[10000]^T$, $[00001]^T\}$, and from the concatenation of these we form the final input vector:

$$[0001010000001001000000001]^T$$

If we had an unrestricted choice of which bits to set, then by setting any 5 in 25 bits we have ${}^{25}C_5$ = 53,130 possible representations available. However, the restriction on the setting of bits to be each within its own subvector means that only 5^5 = 3,125 different combinations of values may be represented.

The converse of this is that, by making a suitable choice of encoding, we could represent the 3,125 possible vectors by setting only 4 bits in 25, as ${}^{25}C_4$ = 12,650. This reduces the rate at which the memory saturates, allowing us to store more information is a given number of bits, or to use fewer bits to store the information we have. The gain to be had from a more efficient coding increases as the size of the input vectors increases.

Depending on whether or not the class labels are used by the algorithm to determine encoding, discretisation methods can be divided into supervised and unsupervised approaches.

Equi-width and equi-frequency approaches are the most typical representatives of unsupervised methods [3]. Equi-width discretisation (or fixed-width discretisation) subdivides the range of an attribute into adjacent equal-width bins, distributed uniformly across the range of the attribute. In contrast, equi-frequency discretisation (also called histogram equalization) aligns the bin boundaries so that approximately the same number of data points falls into each bin.

Error-based or entropy based methods fall into the category of supervised discretisation approaches. Error-based methods, such as error-based Holte's 1R discretisation [5], determine the bin boundaries using an error function and aim to minimise the number of misclassified records within the training set. Fayyad & Irani's entropy-based discretisation [6] technique uses information gain evaluations to determine the boundaries and aims to maximise the information gain by minimising the entropy. However, these methods are still binning individual attributes which are then concatenated, so they have the same disadvantages as the unsupervised methods.

D. *Kohonen's SOM*

Kohonen's SOM (Self Organizing Map) [7] paradigm maps complex multidimensional data to relatively simple low dimensional representation. This mapping preserves the topological relationship between original and mapped data points and therefore it can be used for efficient similarity preserving data encoding. The next section discusses how we can derive an encoding of data that meets the requirements of AURA (sparse codes, fixed number of bits set in each vector) from a Kohonen map of our data.

II. PROPOSED ENCODING ALGORITHM

A. *Algorithm*

Suppose we have a classification problem with real valued n-dimensional input data and l classes, with a dataset size of m, such that $\{X_i, c_i\}_{i=1..m}$, $X_i \in R^n$, $c_i \in C = \{C_1,...,C_l\}$.

To apply the AURA k-NN classifier to this problem we have to find a transformation f that would map the input data into the B_q^p binary space, $0 \le q \le p$. B_q^p is a set of p-dimensional binary vectors with q components set to 1 (and the remaining p-q components set to 0), so $f : R^n \to B_q^p$. Ideally this encoding should: preserve the data features important for classification; have small values for p and q; and have low computational complexity.

The proposed encoding algorithm works as following:

AURA Input Encoding Algorithm:

1. The value for binary space dimension p is set;
2. The SOM network with $P \ge p$ neurons is trained using the available input data $\{X_i\}_{i=1..m}$;
3. The subset of p neurons is chosen from all P SOM neurons (see below for the possible ways of selection);
4. The value q for binary space density is set;
5. For every input vector X to be encoded, q Best Matching Units (BMUs) are found among the selected p neurons. The input vector is mapped to the binary sequence of length p with bits set to 1 or 0 depending on whether the corresponding neuron is chosen as one of BMUs.

As was noted in Section I.A, the parameters p and q may describe the length and density of a binary vector used as input to a CMM. The optimal relationship between p and q for non-random data is not known, so the appropriate values of p and q for a particular dataset are to be determined by experimentation.

B. *Neuron Subset Selection*

From the trained SOM network, a subset of neurons is selected because some neurons do not add much information to the classification process, and we want the input vector created from the neurons to be as short as possible while consistent with good classification performance.

Method 1 (Selection of random subset)

From P SOM neurons a subset of $p \le P$ neurons is selected with each neuron having equal chances of being selected.

Method 2 (Rank based selection)

For every input vector from the training dataset $\{X_i\}_{i=1..m}$ the BMU is found. All neurons of the network are ranked in the decreasing order of their activity of being BMUs and the subset is selected as first p neurons from this ordered sequence.

We note that selecting SOM units based on proximity to the training data (Method 2) may introduce a bias into the representation of the data. Hence we have included the random selection of SOM units (Method 1) as a comparison.

III. Experimental Set-up

A. SOM Network

To create and train SOM networks, the MATLAB SOM Toolbox 2.0 [8] was used. The map had a rectangular sheet shape, and neurons were located on a hexagonal grid.

The initial neuron weight values were assigned by linear initialisation algorithm [9], so they belong to the subspace spanning the two largest eigenvectors calculated for the data set. The ratio of the SOM sheet sides is set equal to the ratio of these eigenvalues as suggested in [9] for linear initialisation. The fast batch algorithm was used for network training. In this algorithm the whole dataset is presented to the map before any weight adjustments are made. One pass through the dataset and the following network modifications represents one training epoch. The training process starts with $10P/m$ epochs of coarse training, that serve to orient and scale the SOM, followed by $40P/m$ epochs of fine training that fit the SOM more closely to the data.

The neuron neighbourhood function was Gaussian. The initial neighbourhood size was set to 1/8 of the maximum length of the side of the map and was linearly decreased throughout the coarse training stage to the value of 1. It was then held constant for the fine training stage.

B. AURA Classifier

We used the AURA software library v2.4.0 [10] to construct the classifier system.

The input vectors to the classifier were constructed, for each data vector, from the indices of the SOM best matching units activated on presentation of that data vector to the SOM. Each bit in the vector corresponds to one unit in the SOM, and for each data vector, the bits corresponding to the BMUs are set to 1, all other bits are zero. The correspondence between SOM units and bits in the data vector was (arbitrarily) made in the order that the SOM units are held in the software.

The output vectors index stored records, with each bit corresponding to one record.

On recall, LMAX thresholding is used to determine the k nearest neighbours to the query point (with in this case L corresponding to k). LMAX thresholding allows matching against partial or corrupt data. The decision about query class was taken using the unweighted majority voting.

C. Dataset

To test the proposed encoding algorithm we used the Glass Identification Database from the UCI Machine Learning Repository [11]. The dataset consists of $m = 214$ records; each with 9 real-valued attributes (refractive index and 8 percentage by weight measurements for each of the elements Na, Mg, Al, Si, K, Ca, Ba, Fe) and belongs to one of 6 classes. Linear normalization (zero mean, unit variance) was applied to each attribute.

All the available input data vectors were used for unsupervised SOM training and further data encoding. After the binary encoding of the dataset we performed a random split of the data into 80%-20% training and testing subsets. The SOM encoding of the complete dataset does not use the class labels, so the 20% testing subset is independent with respect to the output data. The training subset was stored in AURA (as associations between data values and record indices), and recall values were obtained using the testing subset (on recall, indices of the k nearest neighbors to the test record were obtained). In a manner similar to the method in [3], for every classifier configuration that was tested we averaged the recall value using 5 different train/test splits (*n*-folding).

An experimental evaluation of the AURA system operating as a classifier is given in [3]. This paper presents a comparison of a number of input encoding methods (including equi-width and equi-frequency) for UCI datasets [11], including the Glass Identification Database. The classification results produced using AURA for the Glass Identification Database vary considerably with the choice of input encoding algorithm. At the low end, classification performance was only 0.5 for equi-width discretisation with 5 bins per component ($p = 45$, $q = 5$) and $k = 10$ neighbours – that is, the 9 variables in the data are each split into 5 equal-width bins, giving a total of 45 bits to encode each record; each encoded record is then used in a k-nearest neighbour classifier with k = 10. At the high end, performance was 0.72 for error-based Holte's 1R discretisation with the Standard Deviation ranging in *[0.04, 0.1]*. These results from the examination in [3] of the Glass Identification Database, are the basis for comparison with the method of input encoding for the AURA system presented in this paper. Other datasets were examined, with the same behaviour observed.

IV. Experimental results

Each experiment constructed a SOM to represent the training data. The SOM was then used to encode the data – that is, to provide a mapping from real-valued data to binary vectors, as proposed above. Encoded data was stored in a CMM to form the AURA k-NN classifier, and data which had not been stored was presented to determine whether the class could be correctly found. The result of each experiment shows the classification ability of the CMM-based AURA system, given the encoding provided by the SOM.

A. Varying the Network Size and the Encoding Density

The first experiment examines the question of the dependence of recall quality on the encoding dimension p and the encoding density q. We did not select subsets of neurons at this stage, so the encoding dimension is equal to the number of neurons in the network $(p = P)$.

The average values of recall obtained using 5-fold cross validation are shown in Fig. 1. Each individual recall had an average standard deviation of 0.05. The general trend is towards higher recall with larger SOM network sizes. As we increase the size of the SOM network, we increase the size of the input to the CMM. We are therefore increasing the size of the CMM and are using more memory to store the data, so better recall for larger networks is not surprising. However, as we are interested in the relatively low encoding density (q less than around 10), we can observe that with the network size P above approximately 150, the recall performance reaches its highest and most stable values in that region. The network with $P = 300$ neurons can be chosen as a typical network exhibiting such behaviour (several network sizes were examined, with the same behaviour observed – it is seen fairly clearly for $P = 300$). In the next sections we consider this network in more detail.

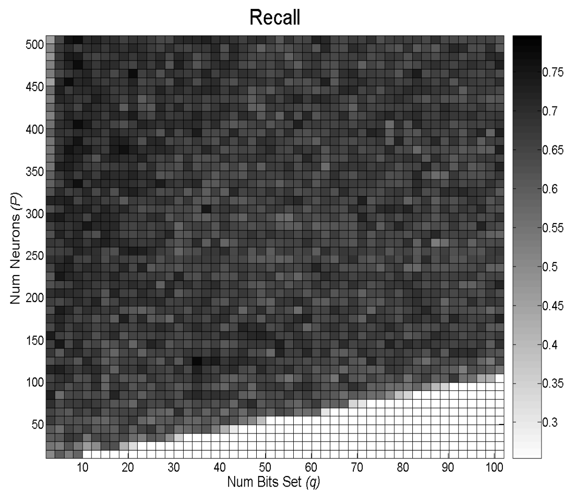

Fig. 1. Average values of recall accuracy from the CMM-based classifier, obtained using 5-fold cross validation, when varying p and q. The size of input to the CMM (p) is determined by the number of neurons selected from the SOM, and the number of bits set (q) is determined by the number of best matching units in the SOM that are used to determine the encoding of the input data.

B. Varying the Encoding Density and the Number of AURA's Nearest Neighbours

For the chosen 300-neuron SOM network (a rectangular map of 20×15 neurons), 5-fold classification recall performance is obtained for different values of q ("encoding density" or number of bits set in each input record) and k (number of neighbours in the k-NN algorithm). That is, we are looking for interactions between q, the number of bits set in the input vector for the CMM, and k, the number of records that we wish to retrieve from the CMM to drive the k-NN algorithm. As in the previous experiment (Section IV.A), all network neurons are used for encoding ($p = P = 300$).

The 5-fold recall average performance is shown in Fig. 2.

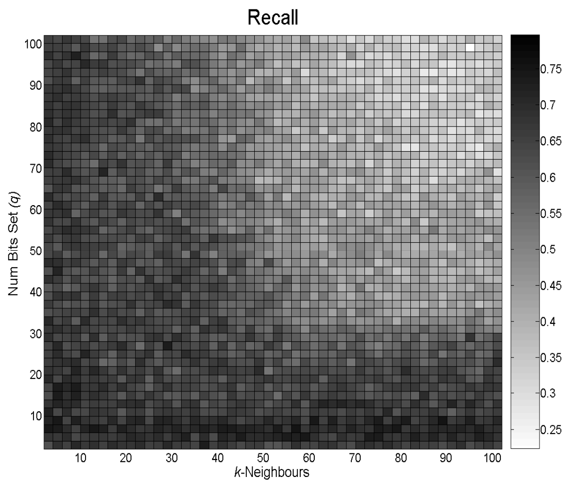

Fig. 2. Average values of recall accuracy from the CMM-based classifier, obtained using 5-fold cross validation, when varying q and k. The number of bits set (q) is determined by the number of best matching units in the SOM that are used to determine the encoding of the input data. The number of nearest neighbors (k) used by the AURA k-NN is set by selecting the appropriate thresholding function and level.

It is interesting to observe that for the values of encoding density in the range $q \in [3..20]$ there is no significant dependence of recall on the number of nearest neighbours used by the k-NN algorithm. For values of q larger than this, preference should be given to smaller values of k. This may indicate that for a given size of CMM input, as the number of bits set on each input increases, the saturation of the CMM increases. This reduces the reliability of the recall process, and therefore looking at only the closest neighbours (small values of k) may include less noise in the classification process. This is in accordance with the results presented in [12, 13] for the variation of recall performance with the ratio q/p. The average standard deviation of each individual 5-fold recall is 0.06.

C. Selecting the Subset of Neurons Used for Data Encoding

In this experiment we want to test the methods of selection of the representative subset of neurons for further data encoding. That is, given a SOM that has been constructed to represent the data, we want to use a subset of the neurons as a further approximation to the data, if we can do so without reducing performance in the k-NN classifier. Ideally the subset should be as small as possible, as its size determines the size of the input to the CMM and therefore the complexity of the encoding process. We should also attempt to select this subset such that we get good performance by using a reasonably small number of BMUs (small value of q) for encoding, thus avoiding the memory saturation problem in the AURA classification system.

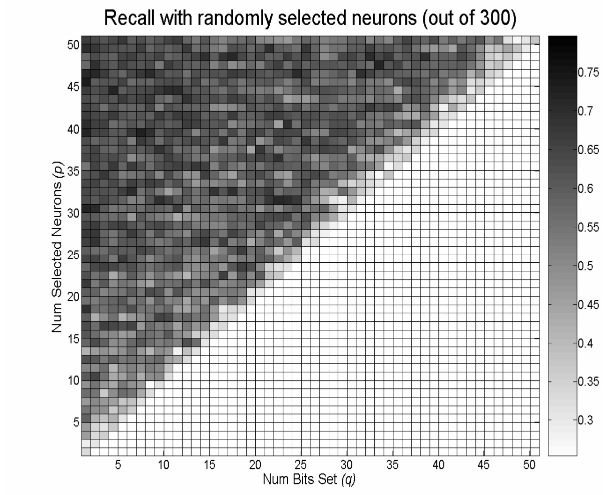

Fig. 3. Average values of recall accuracy from the CMM-based classifier, obtained using 5-fold cross validation, when varying *p* and *q*. The size of input to the CMM (*p*) is determined by the number of neurons selected from the SOM, which contains 300 neurons. Selections of neurons to form the encoder for CMM inputs are random. The number of bits set (*q*) is determined by the number of best matching units selected from the SOM.

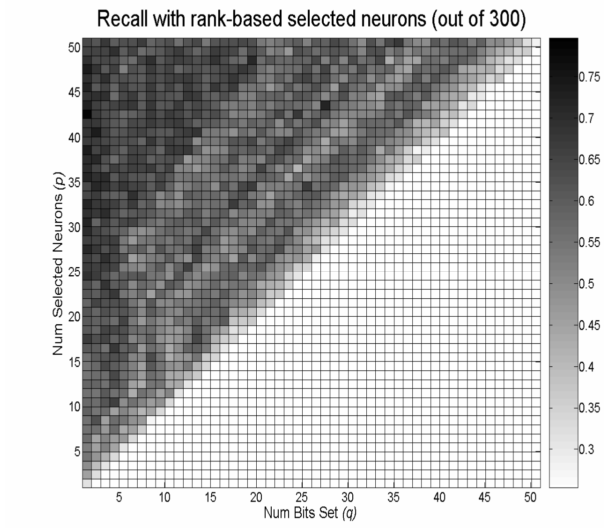

Fig. 4. Average values of recall accuracy from the CMM-based classifier, obtained using 5-fold cross validation, when varying *p* and *q*. The size of input to the CMM (*p*) is determined by the number of neurons selected from the SOM, which contains 300 neurons. Selections of neurons to form the encoder for CMM inputs are based on ranking neurons according to frequency with which they are the "best matching unit". The number of bits set (*q*) is determined by the number of best matching units selected from the SOM.

Fig. 3 shows how the average recall value varies depending on the size of neuron subset size (*p*) and the number of bits set in the encoded data (number of selected BMUs, *q*). The results are obtained by selecting subsets of neurons from the 300-neuron SOM network, using random selection of neurons in the subset (Method 1). The recall results obtained using the randomly selected subset of neurons are not stable. This is due to the higher chance of selection of between-cluster neurons, and these provide less information for the encoding than their in-cluster counterparts.

The performance using a rank-based selection of neurons (Method 2) is better. From Fig. 4 we can see that high and stable classification results are obtained using a small subset of neurons, with the size *p* of the subset in the range *[*25..35*]*. It is sufficient to use values of $q \in [1..5]$ bits set in the encoded data to achieve these results. The average classification result for this range of parameters is 0.65, with a min-max range of *[*0.53, 0.72*]*.

D. Encoding Timing

All tests were carried out on a Windows XP workstation with a 2.4GHz Intel Pentium 4 processor and 512MB RAM. We used Matlab 7.0 (Release 14) for SOM training and Microsoft Visual C++ .NET 7.1 to compile encoding, training and testing procedures.

Training of the SOM network with 10…500 neurons took 0.5 to 23 sec (quadratic dependence on the number of neurons).

For the network with 300 neurons it took about 500 ms to select the subset of 30 neurons used for encoding; 50 ms to do the encoding of all dataset with $q \in [1,5]$ bits set, and 10/15 ms for training/testing of AURA *k*-NN classifier.

V. Conclusion

The objective of this work was to improve on the current method for encoding real data into binary vectors for use with AURA systems, particularly classifier systems. Using the proposed SOM-based encoding method, we can get classification results that are comparable with those obtained in [3].

As was noted in Section III.C, the maximum performance reported in [3] for a conventional binning approach to encoding the input data for this dataset was around 0.72. The average performance we obtain is slightly less at 0.65 than the best encoding, but there is sensitivity to the parameters and the best performance at 0.72 equals the best conventional performance. It does however use far fewer bits – we can represent each record using 25 bits, as compared to the 45 bits used in [3]. As AURA classifiers are typically used as a first, fast, coarse classifier, to be followed by a slower but more accurate classifier [2], this potential small drop in performance does not compromise the applicability of the system, while the reduction in memory requirement from a better encoding may substantially increase the range of useful applications. We conclude that the proposed encoding may offer an advantage in application of the AURA system.

As noted above, other datasets from [10] were also examined. The effect was present, but was less apparent as the overall performance was high for all methods of encoding and little variation between methods was observed. The importance of encoding does depend on the dataset, but in a way which is not predictable.

Of concern is the stability of the results. As can be seen in Figures 1-4, the performance of the system fluctuates as the parameters are varied. Selection of an acceptable set of parameters currently requires some experimentation, although estimates of the range over which parameters need to be experimented with may be obtained from the optimal relationship between p and q: $q \approx log_2\, p$. In particular, the effect of data on the effectiveness of an encoding inevitably will require some experimentation to determine the optimal values of p and q for any particular data set. The next stage of the work will be to attempt reduce this variability and make more specific recommendations about parameters.

REFERENCES

[1] J. Austin, “Distributed associative memories for high speed symbolic reasoning,” *Int. J. on Fuzzy Sets and Systems*, vol.82, pp. 223–233, 1995.

[2] P. Zhou, J. Austin and J.A. Kennedy, “High performance k-NN classifier using a binary Correlation Matrix Memory,” *Advances in Neural Information Processing Systems*, vol. 11, pp. 713–722, 1999.

[3] V. Hodge and J. Austin, “Discretisation of real-valued data in a binary neural k-nearest neighbour algorithm,” *Data Mining and Knowledge Discovery*, submitted for publication.

[4] I.. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools with Java Implementations*. Morgan Kaufmann, San Francisco, 1999.

[5] R.C. Holte, “Very simple classification rules perform well on most commonly used datasets,” *Machine Learning*, vol. 11, no. 1, pp. 63–90, 1993.

[6] U. Fayyad and K. Irani, “Multi-interval discretisation of continuous-valued attributes for classification learning,” in Proc. of the 13th Int. Joint Conf. on Artificial Intelligence (IJCAI-93), Chambery (France), 1993, pp. 1022–1027.

[7] T. Kohonen, “The self-organising map,” *Proc. of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[8] MATLAB SOM Toolbox 2.0, Helsinki University of Technology, The Laboratory of Computer and Information Science (CIS). Available: http://www.cis.hut.fi/projects/somtoolbox.

[9] J. Vesanto, J Himberg, E. Alhoniemi, J. Parhankangas, “SOM Toolbox for Matlab 5,” Helsinki University of Technology, Tech. Rep. A57 FIN-02015, Apr. 2000.

[10] AURA home page, University of York, Advanced Computer Architecture Group, Department of Computer Science. Available: http://www.cs.york.ac.uk/arch/NeuralNetworks/AURA/aura.html.

[11] UCI Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html.

[12] G. Palm, “On Associative Memory,” *Biological Cybernetics*, vol. 36, pp. 19–31, 1980.

[13] M. Turner and J. Austin, “Matching Performance of Binary Correlation Matrix Memories,” *Neural Networks*, vol. 10, no. 9, pp. 1637–1648, 1997.