A Binary Neural Decision Table Classifier

Victoria J. Hodge, Simon O'Keefe and Jim Austin
Advanced Computer Architecture Group
Department of Computer Science
University of York, York, YO10 5DD, UK

Abstract

In this paper, we introduce a neural network-based decision table algorithm. We focus on the implementation details of the decision table algorithm when it is constructed using the neural network. Decision tables are simple supervised classifiers which, Kohavi demonstrated, can outperform state-of-the-art classifiers such as C4.5. We couple this power with the efficiency and flexibility of a binary associative-memory neural network. Initially, we demonstrate how the binary associative-memory neural network can form the decision table index to map between attribute values and data records and subsequently we show how two attribute selection algorithms can be used to pre-select attributes for this decision table. The attribute selection algorithms are easily implemented within the same binary associative-memory framework producing a tightly-coupled, two-tier system allowing attribute selection and decision table indexing. The first attribute selector uses mutual information between attributes and classes to select the attributes that classify best. The second attribute selector uses a probabilistic approach to evaluate randomly selected attribute subsets.

1 Introduction

Supervised classifier algorithms aim to predict the class of an unseen data item. They induce a hypothesis using the training data to map inputs onto classified outputs (decisions). This hypothesis should then correctly classify previously unseen data items. There is a wide variety of classifiers including: decision trees, neural networks, Bayesian classifiers, Support Vector Machines and k-nearest neighbour.

We have previously developed a k-NN classifier [11] using an associative memory neural network called the Advanced Uncertain Reasoning Architecture (AURA) [6]. In this paper, we extend the approach to encompass a supervised decision table classifier, coupling the classification power of the decision table with the speed and storage efficiency of an associative memory neural network.

The decision table has two components: a schema and a body. The schema is the set of attributes pre-selected to represent the data and is usually a subset of the data's total attributes. There are various approaches for attribute selection; we discuss two later in this paper. The body is essentially a table of labelled data items where the attributes specified by the schema form the rows and the decisions (classifications) form the columns. Each column is mutually exclusive and represents an equivalence set of records as defined by the attributes of the schema. Kohavi [13] uses a Decision Table Majority (DTM) for classification whereby if an unseen item exactly matches a stored item in the body then the decision table assigns the stored item's decision to the unseen item. However, if there is no exact match then the decision table assigns the majority class across all items to the unseen item. Our decision table approach implements both DTM and proximity-based matching as implemented in our k-NN classifier whereby if there is no exact match then the decision table assigns the class of the nearest stored item to the unseen item.

2 RAM-based Neural Networks

The AURA C++ library provides a range of classes and methods for rapid partial matching of large data sets [6]. In this paper we define partial matching as the retrieval of those stored records that match some or all of the input record. In our AURA decision table, we use best partial matching to retrieve the records that are the top matches.

AURA belongs to a class of neural networks called Random Access Memory (RAM-based) networks. RAM-based networks were first developed by Bledsoe and Browning [8] and Aleksander and Albrow [1] for pattern recognition and led to the WISARD pattern recognition machine [2]. See also [4] for a detailed compilation of RAM methods.

RAMs are founded on the twin principles of matrices (usually called Correlation Matrix Memories (CMMs)) and n-tupling. Each matrix accepts m inputs as a vector or tuple addressing m rows and n outputs as a vector addressing n columns of the matrix. During the training phase, for the j-th instance the matrix weights M^{lk} are incremented if both the input row I_j^l and output column O_j^k are set. Therefore, training is a single epoch process with one training step for each input-output association preserving the performance. During recall, the presentation of vector I_j elicits the recall of vector O_j as vector I_j contains all of the addressing information necessary to access and retrieve vector O_j. This training and recall makes RAMs computationally simple and transparent with well-understood properties. RAMs are also able to partially match records during retrieval. Therefore, they can rapidly match records that are close to the input but do not match exactly.

2.1 AURA

The binary neural technique evaluated here uses AURA [6], which comprises a C++ library of classes and functions. AURA has been used in an information retrieval system [9], high speed rule matching systems [5], 3-D structure matching [17] and trademark searching [3]. AURA techniques have demonstrated superior performance with respect to speed compared to conventional data indexing approaches [10] such as hashing and inverted file lists which may be used for a decision table body. AURA trains 20 times faster than an inverted file list and 16 times faster than a hashing algorithm. It is up to 24 times faster than the inverted file list for recall and up to 14 times faster than the hashing algorithm. AURA techniques have demonstrated superior speed and accuracy compared to conventional neural classifiers [19].

The rapid training, computational simplicity, network transparency and partial match capability of RAM networks coupled with our robust quantisation and encoding method to map numeric attributes from the data set onto binary vectors for training and recall make AURA ideal to use as the basis of an efficient implementation. A more formal definition of AURA, its components and methods now follows.

Correlation Matrix Memories (CMMs) are the building blocks for AURA systems. AURA uses binary input I and output O vectors to store records in a CMM M as in equation 1. Training (construction of a CMM) is a single epoch process with one training step for each input-output association (each ${I_j O_j}^T$ in equation 1) which equates to one step for each record in the data set.

This process is illustrated in figure 1.

$$M = \bigvee_j ({I_j O_j}^T) \quad \text{where} \quad \bigvee \quad \text{is logical OR} \tag{1}$$

Recall of outputs associated with inputs to the CMM is performed by a product of a recall input vector I_k and CMM, as in equation 2. For an input I_k that appeared in the training set, we get a (non-binary) vector R, which is composed of the required output vector multiplied by a weight based on the dot product of the input vector with itself, plus a noise term ϵ formed from the cross products of the input with a reduced CMM. We constrain the number of bits set to one in the binary vectors to be always the same, so the weight is always known and can be used to set a threshold to convert R back to the binary vector O_k. If the inputs or outputs stored in M are orthogonal then the noise term ϵ disappears and we have exact recall. For sparse binary vectors that are not orthogonal, the noise is in most cases removed by the threshold process. The probability of failure can be calculated, and the system engineered accordingly. If the recall input I_k is not from the original training set, then the system will recall the output O_k associated with the closest stored input to I_k, based on the dot product between the test and training inputs. The threshold must be adjusted, and may either be fixed (Willshaw) or adaptive (L-max).

$$
\begin{aligned}
R^T &= {I_k}^T \cdot M && (2)\\
&= {I_k}^T I_k {O_k}^T + {I_k}^T (M - I_k {O_k}^T) && (3)\\
&= \|I_k\| {O_k}^T + \epsilon && (4)
\end{aligned}
$$

For the method described in this paper, we:

- Train the CMM (decision table body CMM) on the data set, so that it indexes all records in the data set and allows them to be matched.
- Select the attributes for the schema using schema CMMs. We describe two selection algorithms. The first uses a single CMM but the second algorithm uses two coupled CMMs.
- Match and classify unseen items using the trained decision table.

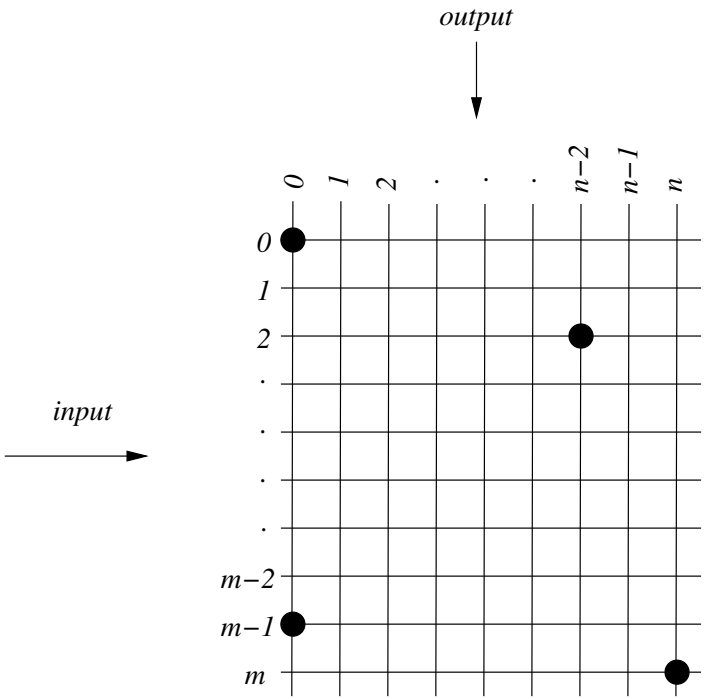

Figure 1: Showing a CMM with input vector i and output vector o. Four matrix locations are set following training – $i_0o_0, i_2o_{n-2}, i_{m-1}o_0$ and i_mo_n.

3 Data

For the data sets:

- Symbolic and numerical unordered attributes are enumerated and each separate token maps onto an integer ($Text \mapsto Integer$) which identifies the bit to set within the vector. For example, a SEX_TYPE attribute would map as ($F \mapsto 0$) and ($M \mapsto 1$).

Kohavi's DTM methodology is principally aimed at symbolic attributes but the AURA decision table can also handle continuous numeric attributes.

- Any real-valued or ordered numeric attributes are quantised (mapped to discrete bins), and each individual bin maps onto an integer which identifies the bit to set in the input vector. There are various quantisation techniques that may be used (see [12]) for an evaluation of four techniques in an AURA-based k-nearest neighbour algorithm. In this paper, we describe the simple equi-width quantisation.

A range of input values for attribute f map onto each bin which in turn maps to a unique integer to index the vector as in equation 5. The range of attribute values mapping to each bin is equal, i.e. the bins have equal width.

$$\Re_{fi} \rightarrow bins_{fk} \mapsto Integer_{fk} + offset_f \tag{5}$$

where

$$i \in FV_f \wedge cardinality(Integer) \equiv cardinality(bins_f)$$

In equation 5, $offset_f$ is a cumulative integer offset within the binary vector for each attribute f and $offset_{f+1} = offset_f + nBins_f$, where $nBins_f$ is the number of bins for attribute f, FV_f is the set of attribute values for attribute f, $\rightarrow$ is a many-to-one mapping and $\mapsto$ is a one-to-one mapping.

This equi-width quantisation or binning approach aims to subdivide the attributes uniformly across the range of each attribute. The range of values is divided into b bins such that each bin is of equal width. The even widths of the bins prevent distortion of the inter-bin distances, so the binned data can be used to calculate distances between data points that approximate the Euclidean distance.

3.1 Data Vectors

Once the bins and integer mappings have been determined, we map the records onto binary vectors. Each attribute maps onto a consecutive section of bits in the binary vector, thus:

```
For each record in the data set
  For each attribute
    Calculate bin for attribute value;
    Set bit in vector as in equation 5;
```

Each binary vector represents a record from the data set.

4 Body Training

The decision table body is an index of all contingencies and the decision to take for each. Input vectors represent quantised records and form an input I_j to the CMM during training. The CMM associates the input with a unique output vector ${O_j}^T$ during training which represents an equivalence set of records. This produces a CMM where the rows represent the attributes and their respective values and the columns represent equivalence sets of records (where equivalence is determined by the attributes designated by the schema). We use an array of linked lists to store the equivalence sets of records and a second array to store

the counts of each class for the equivalence set as a histogram. The algorithm is:

```
1) Input vector to CMM;
2) Threshold at value nF;
3)   If exact match
4)     Add the record to column list;
5)     Add class to histogram;
6)   Else train record as next column;
```

where nF is the number of attributes. Steps 1 and 2 are equivalent to testing for an exact match during body recall as described next. Training is a single epoch process with one training iteration (steps 1-6) for each input-output association (each ${I_j O_j}^T$ in equation 1) which equates to one iteration for each record in the data set. Figure 3 shows a trained CMM where each row is an attribute value and each column represents an equivalence set.

4.1 Example

We take 12 records from the Iris data set (available from [7]) as shown in table 1. The decision table with equi-width quantisation and 5 bins will be:

Sepal length has range 4.6 to 7.1 which is a spread of 2.5 and bin widths 0.5. Therefore, the bin boundaries are: $[4.6, 5.1), [5.1, 5.6), [5.6, 6.1), [6.1, 6.6), [6.6, 7.1]$.

Sepal width has range 2.3 to 3.5 which is a spread of 1.2 and bin widths 0.24. Therefore, the bin boundaries are: $[2.3, 2.54), [2.54, 2.78), [2.78, 3.02), [3.02, 3.26), [3.26, 3.5]$.

Petal length has range 1.3 to 6.0 which is a spread of 4.7 and bin widths 0.94. Therefore, the bin boundaries are: $[1.3, 2.24), [2.24, 3.18), [3.18, 4.12), [4.12, 5.06), [5.06, 6.0]$.

Sepal length has range 0.2 to 2.5 which is a spread of 2.3 and bin widths 0.46.

ID	Sepal len.	Sepal width	Petal len.	Petal width	Class
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	7.0	3.2	4.7	1.4	Iris-versicolor
6	6.4	3.2	4.5	1.5	Iris-versicolor
7	6.9	3.1	4.9	1.5	Iris-versicolor
8	5.5	2.3	4.0	1.3	Iris-versicolor
9	6.3	3.3	6.0	2.5	Iris-virginica
10	5.8	2.7	5.1	1.9	Iris-virginica
11	7.1	3.0	5.9	2.1	Iris-virginica
12	6.9	2.9	5.6	2.4	Iris-virginica

Table 1: Table listing 12 example records from the Iris data set.

Therefore, the bin boundaries are: $[0.2, 0.66), [0.66, 1.12), [1.12, 1.58), [1.58, 2.04), [2.04, 2.5]$.

The 12 records map on to the CMM (decision table body) as shown in figure 2. The rows represent the attribute value bins (the bin boundaries are listed in figure 2) and the columns represent the equivalence sets. Records 3 & 4 map to the same equivalence set. Records 5 & 7 map to the same equivalence set as do records 11 & 12. Each record has an associated class as given in table 1. The shaded arrows in figure 2 indicate the columns that the three classes map to.

5 Body Recall

The decision table classifies by finding the set of matching records. To recall the matches for a query record, we firstly produce an input vector by quantising the target values for each attribute to identify the bins and thus CMM rows to activate as in equation 3. To retrieve the matching records for a particular

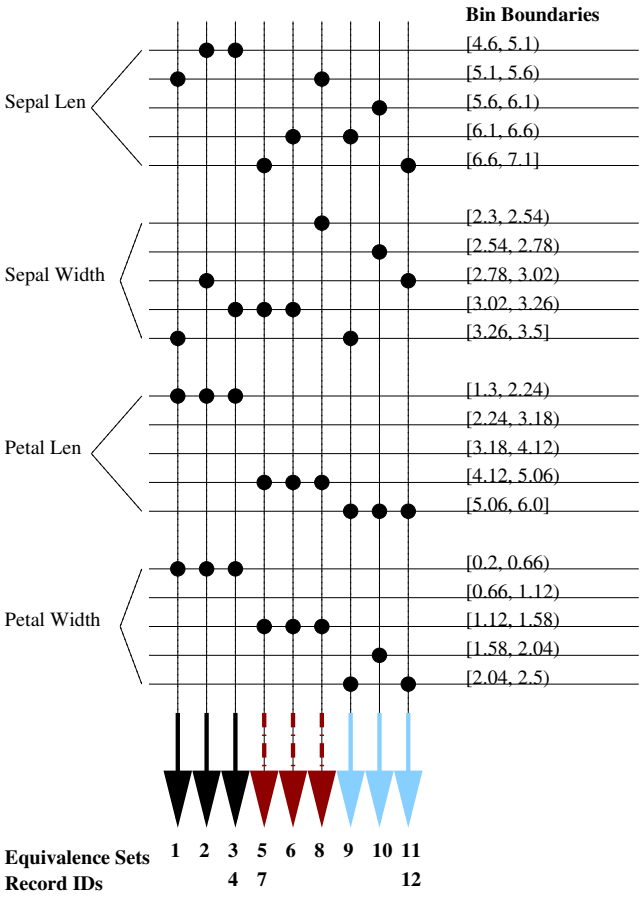

Figure 2: Showing the 12 example records from the Iris data set mapping to the CMM decision table body. We have separated the CMM rows across the attributes for easy viewing. The figure represents a 20 input row CMM (4 attributes with 5 bins (CMM rows) per attribute) with 9 columns (equivalence sets of records). The different shading on the arrows indicates the classes: iris-setosa is represented by the three leftmost columns; iris-versicolor is represented by the middle three columns; and iris-virginica is represented by the three rightmost columns.

record, AURA effectively calculates the dot product of the input vector I_k and the CMM, computing a positive integer-valued output vector R (the summed output vector) as in equation 6 and figure 3 and figure 4.

$$R^T = I_k \cdot M \tag{6}$$

The AURA technique thresholds the summed output R to produce a binary output vector as in figure 3 for exact match and figure 4 for a partial match.

For exact match (as in Kohavi's DTM), we use the Willshaw threshold. This sets a bit in the thresholded output vector for every location in the summed output vector that has a value higher than or equal to a threshold value. The threshold value is set to the number of attributes nF for an exact match. If there is an exact match there will be a bit set in the thresholded output vector indicating the matching equivalence set. It is then simply a case of looking up the class histogram for this equivalence set in the stored array and classifying the record by the majority class in the histogram. If there are no bits set in the thresholded output vector then we classify the unseen record according to the majority class across the data set.

For partial matching, we use the L-Max threshold. L-Max thresholding essentially retrieves *at least* L top matches. It sets a bit in the thresholded output vector for every location in the summed output vector that has a value higher than a threshold value. The AURA C++ library automatically sets the threshold value to the highest integer value that will retrieve at least L matches. For the AURA decision table, L is set to the value of 1. There will be a bit set in the thresholded output vector indicating the best matching equivalence set. It is then simply a case of looking up the class histogram for this equivalence set in the stored array and classifying the unseen record as the majority class. We note there may be more than one best matching equivalence set so the majority class across all best matching sets will need to be calculated.

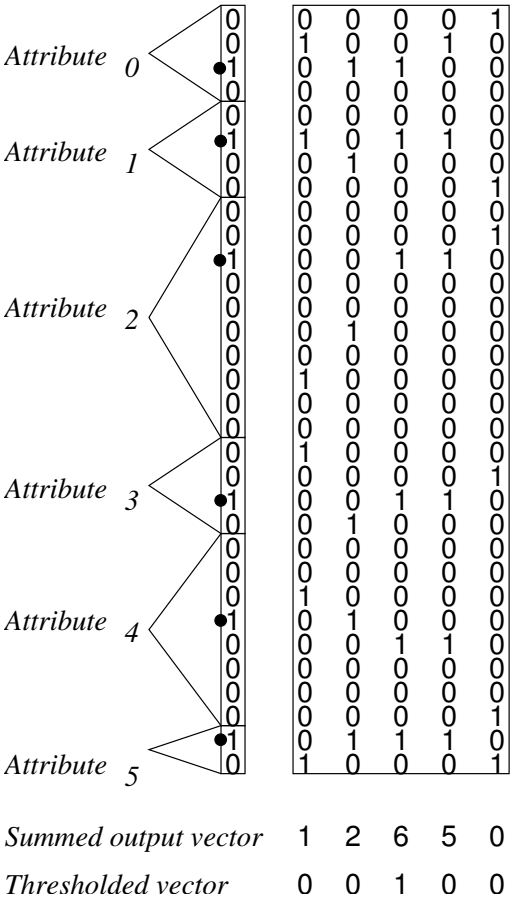

Figure 3: Diagram showing the CMM recall for an exact match. The left hand column is the input vector. The dot is the value for each attribute (a value for an unordered attribute or a bin for an ordered numeric attribute). Each column of the CMM matrix represents an equivalence set. AURA multiplies the input vector by the values in the matrix columns, using the dot product, sums each column to produce the summed output vector and then thresholds this vector at a value equivalent to the number of attributes in the input (6 here) to produce the thresholded attribute vector which indicates the matching column (the middle column here).

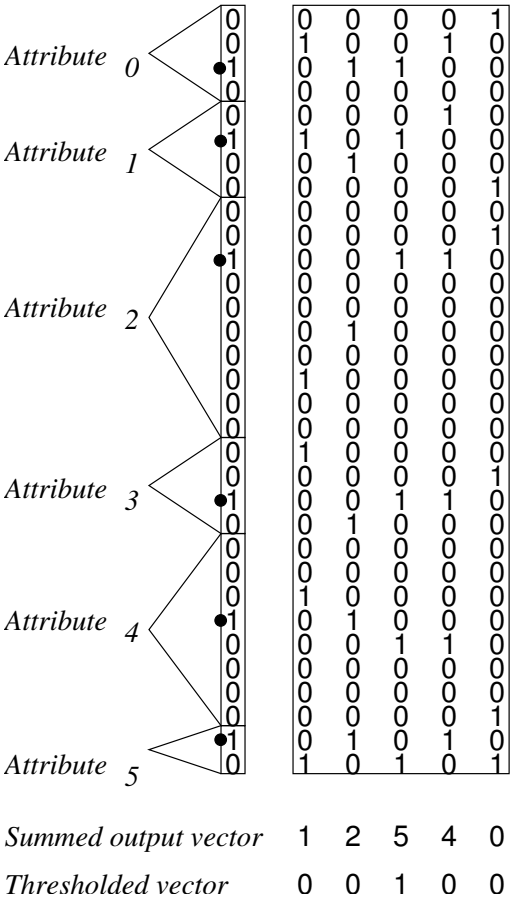

Figure 4: Diagram showing the CMM recall for a partial match. The left hand column is the input vector. The dot is the value for each attribute (a value for an unordered attribute or a bin for an ordered numeric attribute). Each column of the CMM matrix represents an equivalence set. AURA multiplies the input vector by the values in the matrix columns, using the dot product, sums each column to produce the summed output vector and then thresholds this vector at a value equivalent to the highest value in the vector (5 here) to produce the thresholded attribute vector which indicates the matching column (the middle column here).

6 Schema Training

In the decision table body CMM, the rows represented attribute values and the columns represented equivalence sets. In the schema CMM used for the first attribute selection algorithm (Mutual Information Attribute Selection, detailed in section 7.1), the rows represent attribute values and the columns represent individual records. For our second attribute selection algorithm (Probabilistic Las Vegas Algorithm, detailed in section 7.2), we use two CMMs where the first CMM indexes the second CMM (see figure 5). In the first, CMM1, the rows represent records and the columns represent attribute values. In the second, CMM2, the rows represent attribute values and the columns represent the records. This second CMM2 is therefore identical to the CMM used for the first attribute selection algorithm

During training for the first attribute selection algorithm and CMM2 of the second attribute selection algorithm, the input vectors I_j represent the attribute values in the data records. The CMM associates the input with a unique output vector O_j. Each output vector is orthogonal with a single bit set corresponding to the record's position in the data set, the first record has the first bit set in the output vector, the second and so on. Training these CMMs entails one iteration for each record in the data set. During training for CMM1, the records represent the input vectors I_j with a single bit set and the output vectors O_j represent the attribute values in the data records. The CMM training process is given in equation 1. Training this CMMs entails one training iteration for each input-output association. For the Mutual Information Attribute Selection algorithm, training is an n-iteration process. For the Probabilistic Las Vegas Algorithm, training is an n-iteration process but with two CMM loops per iteration (each $I_j {O_j}^T$ in equation 1).

7 Schema Attribute Selection

As with Kohavi, we assume that all records are to be used in the body and during attribute selection.

There are two fundamental approaches to attribute selection which are used in classification: a filter approach that selects the optimal set of attributes independently of the classifier algorithm and the wrapper approach that selects attributes to optimise classification using the algorithm. We examine two filter approaches which are more flexible than wrapper approaches as they are not directly coupled to the classification algorithm.

For a data set with N attributes there are $O(N^M)$ possible combinations of M attributes which makes exhaustive search intractable. In the following, we discuss the mapping of two approaches to the binary AURA architecture – a filter approach (mutual information attribute selection) that examines attributes on an individual basis, and a probabilistic filter approach (probabilistic Las Vegas algorithm) that examines randomly selected subsets of attributes.

7.1 Mutual Information Attribute Selection

Wettscherek [18] describes a mutual information attribute selection algorithm which calculates the mutual information between class C and each attribute F_j using the data from the training set. The mutual information between two attributes is "the reduction in uncertainty concerning the possible values of one attribute that is obtained when the value of the other attribute is determined" [18].

Using the notation from [18], for unordered attributes nFV is the number of distinct attribute values (f_i) for attribute F_j and $nClasses$ the number of classes (C):

$$I(C, F_j) = \sum_{i=1}^{nFV} \sum_{c=1}^{nClasses} p((C = c) \wedge (F_j = f_i)) \cdot \log \frac{p((C = c) \wedge (F_j = f_i))}{p(C = c) \cdot p(F_j = f_i)} \quad (7)$$

For ordered numeric and continuous attributes, the technique must compute the mutual information between a discrete random variable (class) and a continuous random variable (attribute). Wettscherek assumes that attribute F_j has density $f(x)$ and the joint density of C and F_j is $f(x, C = c)$. Then the mutual information across the data set x is:

$$I(C, F_j) = \int_x \sum_{c=1}^{nClasses} f(x, C = c) \cdot \log \frac{f(x, C = c)}{f(x) \cdot p(C = c)} dx \quad (8)$$

Equation 8 requires an estimate of the density function $f(x)$ and the joint density function $f(x, C = c)$. Wettscherek uses density estimation: using the k-th nearest neighbour density estimate to estimate the density at a certain point using the distance of the k-th nearest data point from this point. The density $\hat{f}(x)$ at point x is given by equation 9:

$$\hat{f}(x) = \frac{k - 1}{2N d_k(x)} \quad (9)$$

where N is the number of records and $d_k(x)$ is the distance of the k-th nearest point from point x. Wettscherek posits a k value of 25 as providing good density estimation. This requires the calculation of the distance from x to all other points in the data set to determine the 25^{th} nearest neighbour.

Wettscherek also approximates $\int_x$ by computing the value of f(x) for values of x from 0 to 1 in steps of 0.002. This requires 501 computations to calculate each mutual information value for each attribute, each calculation being of:

$$\sum_{c=1}^{nClasses} f(x, C = c) \cdot \log \frac{f(x, C = c)}{f(x) \cdot p(C = c)} \quad (10)$$

To approximate $f(x)$ and $f(x, C = c)$, we utilise the binning to represent the density. This is analogous to the Trapezium Rule for using the areas of slices (trapezia) to represent the area under the graph for integration. We use the bins to represent strips of the probability density function and count the number of records mapping into each bin to estimate the density.

Considering AURA, for unordered data, from figure 3 we see that each attribute is mapped to a number of (adjacent) rows in the CMM. The number of distinct values for the attribute is the number of rows required, so $n(rowsFV) = nFV$ where $n(rowsFV)$ gives the number of rows used for attribute F_j.

For training, we set only one bit in the vector O_k indicating the location of the record in the data set. This makes the length of O large, but not unfeasibly so when using AURA with data sets of millions of records.

Each row of the CMM corresponds to a particular value of a particular attribute. The number of bits set on that row corresponds to the number of training records which had that value of that attribute.

Representing the CMM row for f_i by BVf_i, and taking BVc as a binary vector of the same size constructed so that it contains bits set for all records in a particular class, then taking the dot product (implemented as a logical AND and sum over elements) of BVf_i and BVc gives the number of records in a particular class with a particular attribute value. We represent this number of bits set in the vector by $n(BVf_i \wedge BVc)$.

We can thus obtain the probabilities required by equation 7 in terms of the bit vectors created in the CMM (where N is the number of training examples stored).

$$p((C = c) \wedge (F_j = f_i)) = \frac{n(BVf_i \wedge BVc)}{N} \quad (11)$$

$$p(c) = \frac{n(BVc)}{N} \quad (12)$$

$$p(F_j = f_i) = \frac{n(BVf_i)}{N} \quad (13)$$

Substituting the above into equation 7, the mutual information is given by equation 14:

$$\begin{aligned} I(C, F_j) &= \sum_{i=1}^{nRowsFV} \sum_{c=1}^{nClasses} \frac{n(BVf_i)}{N} \cdot \frac{n(BVf_i \wedge BVc)}{n(BVf_i)} \cdot \log \left(\frac{\frac{n(BVf_i)}{N} \cdot \frac{n(BVf_i \wedge BVc)}{n(BVf_i)}}{\frac{n(BVc)}{N} \cdot \frac{n(BVf_i)}{N}} \right) \\ &= \sum_{i=1}^{nRowsFV} \sum_{c=1}^{nClasses} \frac{n(BVf_i \wedge BVc)}{N} \log \left(\frac{N \cdot n(BVf_i \wedge BVc)}{n(BVc) \cdot n(BVF_i)} \right) \end{aligned} \quad (14)$$

where $nRowsFV$ is the number of rows in the CMM for attribute F_j, N is the number of records in the data set, BVf_i is a binary vector (CMM row) for f_i, BVc is a binary vector with one bit set for each record in class c, $n(BVf_i \wedge BVc)$ is a count of the set bits when BVc is logically ANDed with BVf_i and $n(BVc)$ is the number of records in class c.

We can follow the same process for real/discrete ordered numeric attributes in AURA. In this case, the mutual information is given by equation 15:

$$\begin{aligned} I(C, F_j) &= \sum_{i=1}^{nB} \sum_{c=1}^{nClasses} \frac{n(BVb_i)}{N} \cdot \frac{n(BVb_i \wedge BVc)}{n(BVb_i)} \cdot \log \left(\frac{\frac{n(BVb_i)}{N} \cdot \frac{n(BVb_i \wedge BVc)}{n(BVb_i)}}{\frac{nClass_c}{N} \cdot \frac{n(BVb_i)}{N}} \right) \\ &= \sum_{i=1}^{nB} \sum_{c=1}^{nClasses} \frac{n(BVb_i \wedge BVc)}{N} \cdot \log \left(\frac{N \cdot n(BVb_i \wedge BVc)}{n(BVc) \cdot n(BVb_i)} \right) \end{aligned} \quad (15)$$

where nB is the number of bins in the CMM for attribute F_j, N is the number of records in the data set, BVb_i is a binary vector (CMM row) for f_i, BVc is a binary vector with one bit set for each record in class c, $n(BVb_i \wedge BVc)$ is a

count of the set bits when BVc is logically ANDed with BVb_i and $n(BVc)$ is the number of records in class c.

Having reduced the attribute selection problem to operations of rows of the CMM, implementation is straightforward. Because the AURA library is targeted at hardware support of matrix operations, the operation to count the number of bits set in a vector does not require iteration along the whole length of the vector, so it is an efficient operation for AURA and thus the selection of attributes is fast.

The technique assumes independence of attributes and ignores missing values. It is also the user's prerogative to determine the number of attributes to select.

7.2 Probabilistic Las Vegas Algorithm

Liu and Setiono [14] introduced a probabilistic Las Vegas algorithm which uses random search and inconsistency to evaluate attribute subsets. For each equivalence set of records (where the records match according to the attributes designated in the schema), consistency is defined as the number of matching records minus the largest number of matching records in any one class. The inconsistency scores are summed across all equivalence sets to produce an inconsistency score for the particular attribute selection.

The technique uses random search to select attributes as random search is less susceptible to local minima than heuristic searches such as forward search or backward search. Forward search works by greedily adding attributes to a subset of selected attributes until some termination condition is met whereby adding new attributes to the subset does not increase the discriminatory power of the subset above a pre-specified threshold value. Backward search works by greedily removing attributes from an initial set of all attributes until some termination condition is met whereby removing an attribute from the subset decreases the

discriminatory power of the subset above a pre-specified threshold. A poor attribute choice at the beginning of a forward or backward search will adversely effect the final selection whereas a random search will not rely on any initial choices.

Liu and Setiono define their algorithm as:

```
1)   Fbest = N;
2)   For j = 1 to MAX_TRIES
3)     S = randomAttributeSet(seed);
4)     nF = numberOfAttributes(S);
5)     If(nF < nFbest)
6)       If(InconCheck(S,D) < g)
7)      Sbest = S; nFbest = nF;
8) End for
```

where D is the data set, N the number of attributes and g the permissible inconsistency score. Liu and Setiono recommend setting MAX_TRIES to $77 \times N^5$.

Liu and Setiono's algorithm may be calculated simply using the AURA schema CMMs. We need to use two linked CMMs for the calculation as in figure 5. We rotate the schema CMM (CMM1) through 90 degrees, so that CMM1's rows index the records and CMM1's columns index the attribute values. When we present a vector indexing a record to CMM1, we recall the associated attribute values. If we then take the output from CMM1 (the activated attribute values) and present it to CMM2, we will recall a vector with indices for all records that match the set of attribute values. Bitwise comparison of this vector with vectors representing class membership thus allows us to calculate the inconsistency scores easily.

Line 6 of Liu and Setiono's algorithm listed above then becomes:

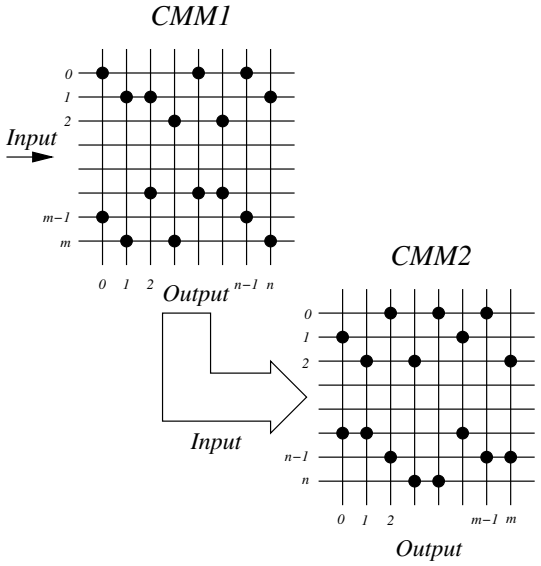

Figure 5: Showing the two CMM combination we use for Liu and Setiono's algorithm. In the first CMM (CMM1), the records index the rows (one row per record) and the attribute values index the columns. The outputs from the CMM (matching attribute values) feed straight into the second CMM(CMM2), where the attribute values index the rows and the records index the columns (one column per record).

```
Place all records in a queue Q;
While !empty(Q)
  Remove R the head record from Q;
  Activate row R in CMM1;
  Threshold CMM1 at value 1;
  Feed CMM1 output into CMM2;
  Threshold CMM2 at value Fbest;
  B = bits set in thresholded_vector;
  Max = cardinality of largest class;
  InconCheck(S,D) += B - Max;
End while
```

The queue holds the records to be processed. Given the method of storage of the data in CMM1, the queue is effectively the input vector to CMM1. Successive records are accessed by stepping through elements of the input vector. By setting the bit for the record at the top of the "queue", the row in CMM1 is accessed, and Willshaw thresholding at value 1 (denoting all active columns (i.e., all attribute values in the record)) gives us that record's attribute values. When these values are passed into CMM2, we access all records matching these values.

The search over the space of combinations of attributes is random. We can generate random subsets of attributes S_i to use by generating a mask to AND with the output from CMM1 before it is passed to CMM2.

By thresholding the output from CMM2 at the value $F_{best} = \|S_i\|$ (the number of attributes in S_i), we retrieve the equivalence set of matching records where equivalence is specified by the current attribute selection in the algorithm. It is then simply a matter of counting the number of matching records (the number of bits set in the thresholded output vector), calculating the number of these matching records in each class, identifying the largest class membership and subtracting the largest class membership from the number of records.

The algorithm has now processed all of the records in this equivalence set so it removes these records from the queue by ORing the thresholded output vector from CMM2 with another mask - the next record to be accessed in CMM1 is the next element of the input vector for which the mask bit is not set. If we repeat this process for each record at the head of the queue until the queue is empty, we will have processed all equivalence sets. We can then calculate InconCheck(S_i, D) for this attribute selection and compare it with the threshold value as in line 6 of Liu and Setiono's algorithm.

This approach to calculating inconsistency for subsets of attributes is efficient as the CMMs store all attributes and their values but we only focus on those attributes under investigation during each iteration of the algorithm. An alternative approach would be to just store those attributes selected in the random subset each time we execute line 6 of Liu and Setiono's algorithm but the CMMs would need to be retrained many times (up to $77 \times N^5$).

Once we have iterated through Liu and Setiono's algorithm MAX_TRIES times then we have selected an "optimal" attribute subset. We have not tried all combinations of all attributes as this is intractable for a large data set. However, Liu and Setiono demonstrated empirically that it was a sufficient approximation.

8 Analysis

The two attribute selection methods detailed in section 7 are just two of many possible techniques that may be used in the AURA decision table. In this section, we select one method, a standard implementation of Wettscherek's attribute selection algorithm, and compare the run times with our AURA-based implementation across a range of data sets. We note that all real-valued attributes are normalised into the range $[0, 1]$ for this investigation. Any categorical attributes or discrete numeric attributes are treated identically between the two methods under investigation so the main difference lies with the speed of

Data set	Num Atts	Num Real	Num Data
heart	13	6	270
hypo	25	7	3163
sick-eu	25	7	3163
vehicle	18	18	846
IBM	9	9	20000
REAL	14	14	200000

Table 2: Table listing the specifications of the data sets where $NumAtts$ is the number of attributes in the data set $NumReal$ is the number of continuous attributes and $NumData$ is the number of records in the data set.

the technique on real-valued attributes. Both techniques use C++ algorithms compiled with GNU g++ v2.95.3 using the Solaris8 OS and run as command-line applications on a 750MHz SPARC-based Sun Blade1000 with 4GB RAM. The AURA methodology uses the AURA C++ class library [16] which provides classes and methods for CMMs and thresholding.

We use four data sets from the UCI data repository [7] (heart, hypothyroid, sick-euthyroid and vehicle), one data set generated by the IBM data set generator [15] and a data set generated using a random number generator that we have used in previous evaluations [11].

We ran the standard selector and then the AURA-based selector five times on each data set to allow us to calculate an average attribute selection time for each technique for each data set.

The average times (across 5 runs) for the Wettscherek standard attribute selector and our AURA-based implementation run on the 6 data sets are listed in table 3

The AURA-based technique is between 14 and 111 times faster for selecting attributes compared to the standard implementation.

Data set	Standard	AURA	AURA speedup
heart	0.71	0.05	14
hypo	1.50	0.034	45
sick-eu	1.51	0.033	45
vehicle	5.80	0.05	111
IBM	37.6	0.97	39
REAL	1146.0	29.6	39

Table 3: Table listing the average run time (across 5 runs) for the standard attributes selector, for the AURA-based attribute selector and the speedup (times faster) of the AURA technique compared to the standard

9 Conclusion

In this paper we have introduced a binary neural decision table classifier. The AURA neural architecture, which underpins the classifier, has demonstrated superior training and recall speed compared to conventional indexing approaches such as hashing or inverted file lists which may be used for a decision table when all techniques are implemented in software and run on identical machines. AURA trains 20 times faster than an inverted file list and 16 times faster than a hashing algorithm. It is up to 24 times faster than the inverted file list for recall and up to 14 times faster than the hashing algorithm. In this paper, we further demonstrated the superiority of the AURA framework as the AURA-based attribute selector was up to 111 times faster than a standard implementation when both algorithms were run on identical machines.

We have shown how two quite different attribute selection approaches may be implemented within the AURA decision table framework. We described a mutual information attribute selector that examines attributes on an individual basis and scores them according to their class discrimination ability. We also demonstrated a probabilistic Las Vegas algorithm which uses random search and inconsistency to evaluate attribute subsets.

The technique is flexible and easily extended to other attribute selection algorithms.

10 Acknowledgement

This work was supported by EPSRC Grant number GR/R55101/01.

References

[1] I. Aleksander and R.C. Albrow. Pattern recognition with Adaptive Logic Elements. In *IEE Conference on Pattern Recognition*, pages 68–74, 1968.

[2] I. Aleksander, W.V. Thomas, and P.A. Bowden. Wisard: A radical step forward in image recognition. *Sensor Review*, pages 120–124, 1984.

[3] Sujeewa Alwis and Jim Austin. A Novel Architecture for Trademark Image Retrieval Systems. In *Electronic Workshops in Computing*. Springer, 1998.

[4] J. Austin. *RAM-Based Neural Networks*. Progress in Neural Processing: 9. World Scientific Pub. Co., Singapore, 1998.

[5] J. Austin, John Kennedy, and Ken Lees. A Neural Architecture for Fast Rule Matching. In *Artificial Neural Networks and Expert Systems Conference*, June 1995.

[6] Jim Austin. Distributed associative memories for high speed symbolic reasoning. In R. Sun and F. Alexandre, editors, *IJCAI '95 Working Notes of Workshop on Connectionist-Symbolic Integration: From Unified to Hybrid Approaches*, pages 87–93, Montreal, Quebec, August 1995.

[7] C.L. Blake and C.J. Merz. UCI Repository of machine learning databases - *http://www.ics.uci.edu/∼mlearn/MLRepository.html*, University of Cali-

fornia, Irvine, Dept. of Information and Computer Sciences., last accessed 23rd Sept, 2005.

[8] W.W. Bledsoe and I. Browning. Pattern recognition and Reading by Machine. In *Proceedings of Eastern Joint Computer Conference*, pages 225–231, 1959.

[9] V.J. Hodge. *Integrating Information Retrieval and Neural Networks.* PhD thesis, Department Of Computer Science, University of York, Heslington, York, UK, YO10 5DD, September 2001.

[10] V.J. Hodge and J. Austin. An Evaluation of Standard Retrieval Algorithms and a Binary Neural Approach. *Neural Networks*, 14(3), 2001.

[11] V.J. Hodge and J. Austin. A High Performance k-NN Approach using Binary Neural Networks. *Neural Networks*, 17(3):441–458, 2004.

[12] V.J. Hodge and J. Austin. Discretisation of Real-Valued Data In A Binary Neural k-Nearest Neighbour Algorithm. *Submitted to, Data Mining and Knowledge Discovery*, 2005.

[13] R. Kohavi. The Power of Decision Tables. In *Proceedings of European Conference on Machine Learning*, volume LNAI 914, pages 174–189. Springer, 1995.

[14] H. Liu and R. Setiono. A Probabilistic Approach to Feature Selection - A Filter Solution. In *Proceedings of 13th International Conference on Machine Learning (ICML'96)*, pages 319–327, 1996.

[15] IBM Quest Data Mining Project. The Quest Synthetic Data Generation Code for Classification - *http://www.almaden.ibm.com/software/quest/Resources/datasets/syndata.html#classSynData*, last accessed 28th Sept, 2005.

[16] Aaron Turner. Introduction to CMMs and AURA-Based Systems - *http://www.cs.york.ac.uk/arch/NeuralNetworks/binary.html*, last accessed 23rd Sept, 2005.

[17] Aaron Turner and Jim Austin. Performance Evaluation of a Fast Chemical Structure Matching Method using Distributed Neural Relaxation. In *Proceedings 4th International Conference on Knowledge-Based Intelligent Engineering Systems, (KES-2000)*, 2000.

[18] Dieter Wettscherek. *A study of distance-based machine learning algorithms.* PhD thesis, Department of Computer Science, Oregon State University, Corvallis, 1994.

[19] Ping Zhou, Jim Austin, and John Kennedy. A High Performance k-NN Classifier Using a Binary Correlation Matrix Memory. In *Advances in Neural Information Processing Systems*, volume 11, 1999.