Under consideration for publication in Knowledge and Information
Systems

A Binary Neural k-Nearest Neighbour Technique

Victoria J. Hodge and Jim Austin

Dept. of Computer Science,
University of York, YO10 5DD, UK,
{vicky,austin}@cs.york.ac.uk

Corresponding Author: Dr. Victoria Hodge,
Email: vicky@cs.york.ac.uk
Tel: +44 1904 433067
Fax: +44 1904 432767

Abstract. K-Nearest Neighbour (k-NN) is a widely used technique for classifying and clustering data. k-NN is effective but is often criticised for its polynomial run-time growth as k-NN calculates the distance to every other record in the data set for each record in turn. This paper evaluates a novel k-NN classifier with linear growth and faster run-time built from binary neural networks. The binary neural approach uses robust encoding to map standard ordinal, categorical and real-valued data sets onto a binary neural network. The binary neural network uses high speed pattern matching to recall the k-best matches. We compare various configurations of the binary approach to a conventional approach for memory overheads, training speed, retrieval speed and retrieval accuracy. We demonstrate the superior performance with respect to speed and memory requirements of the binary approach compared to the standard approach and we pinpoint the optimal configurations.

Keywords: k-Nearest Neighbour, binary neural network, Correlation Matrix Memory, parabolic kernel

1. Introduction

Standard k-NN is a widely applicable clustering, outlier detection and classification technique that demonstrates high recall accuracy; see (Dasarathy, 1991; Wettschereck, 1994; Hodge and Austin, 2004) for an overview of k-nearest neighbour techniques. k-NN uses similar procedures for clustering, outlier detection and classification: examining the distances to the nearest neighbours. To cluster data, the k-NN determines the distance to a record's neighbours using some suitable distance metric such as Euclidean Distance so the cluster bounds may be identified. During outlier detection (Knorr and Ng, 1998), the k-NN calculates the distance between a record and its nearest neighbours. Any points with a high distance to its nearest neighbours is designated an outlier. For classification, k-NN examines those points in a particular data space lying nearest to a query point. K-NN then uses the respective classifications of these nearest neighbours to determine the class of the query point. There are various approaches described in (Dasarathy, 1991; Wettschereck, 1994) for determining the class of a point from the set of classes of its nearest neighbours such as majority vote or weighted majority vote.

The computational growth of standard k-NN is $\mathcal{O}(n^2)$ (Dasarathy, 1991; Knorr and Ng, 1998) with respect to the number of records n in the data set as the approach calculates the distance to each record for every record in the data set. The computational complexity is also directly proportional to the dimensionality of the data m so k-NN has $\mathcal{O}(n^2m)$ runtime. As a result, there is a practical upper limit to both the number of records and the data dimensionality that may be processed even on modern high speed computers dependent on the processor time available.

A method is desired to speed the identification of the k-nearest neighbours while maintaining the recall accuracy of a standard k-NN procedure to allow extremely large data sets to be processed. Two methods for speeding k-NN are: prototype selection and attribute selection (Skalak, 1994). A large data set can be stored as a few lower dimensional prototype vectors thus decreasing the k-NN processing time exponentially (Skalak, 1994). However, prototyping must be applied carefully and selectively as it will increase the sparsity of the distribution and the density of the nearest neighbours. Attribute selection demonstrates high accuracy when coupled with k-NN (Aha and Bankert, 1994) but is computationally expensive due to the combinatorial problem of attribute subset selection. In this paper we focus on speeding the underlying k-NN process and we note that both prototyping and attribute selection will speed our k-NN further.

The aim of this paper is not to evaluate the clustering, outlier detection or classification accuracy of k-NN as that has been well documented elsewhere (for example (Wettschereck, 1994)) but to show the speedup achieved using the Advanced Uncertain Reasoning Architecture (AURA (Austin, 1995)) to implement k-NN. The approach described in this paper is an enhancement of the binary neural network k-NN classifier described in (Zhou, Austin and Kennedy, 1999; Hodge and Austin, 2003; Weeks et al., 2003) which is generically applicable to any pattern classification or clustering task. It quantises numeric-values to form binary vectors which are matched in the binary neural network. (Zhou, Austin and Kennedy, 1999) demonstrated that the approach outperformed Multi-Layer Perceptron (Bishop, 1995) and Radial Basis Function (Bishop, 1995) networks with respect to speed and recall accuracy on a classification task. We demonstrated that a previous version of the technique (Hodge and Austin, 2003) was faster than standard k-NN with much lower computational growth while achieving comparable accuracy. We showed in (Weeks et al., 2003) over 99% accuracy for our AURA k-NN compared with standard k-NN. We used the AURA k-NN to select a set of candidate matches and then post-processed the candidate matches using standard k-NN. This paper augments the technique to further speed the recall by a factor of 1.8 using a more accurate Squared Euclidean Distance approximation within the binary neural network so that no post-processing using standard k-NN is necessary.

The first aim of the paper is to identify to what extent this numeric-to-binary quantisation preserves the Euclidean distances between the attribute values and thus whether our binary neural approximation recalls the correct nearest neighbours as identified by a conventional Euclidean distance approach. The second aim is to demonstrate the speedup achieved by using AURA compared to processing the data set using standard Euclidean Distance k-NN. We also identify the ideal configuration for the two data sets in this paper and provide heuristics for determining the ideal configuration for other data sets.

In the remainder of this paper we provide: a detailed overview of binary neural networks in section 2; AURA and our numeric-to-binary quantisation method in section 3; a description of the evaluation methodology and the results in section 4; a detailed analysis and comparison of the methods evaluated in section 5; and the conclusions we have drawn from our analyses in section 6.

2. RAMs

The AURA C++ library provides a range of classes and methods for rapid partial matching of large data sets (Austin, 1995). 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, 1959; Aleksander and Albrow, 1968) for pattern recognition and led to the WISARD pattern recognition machine (Aleksander, Thomas and Bowden, 1984). They include Hopfield Associative Memories (Hopfield, 1982). See also (Austin, 1998) 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 to store associations between inputs I_j and outputs O_j as shown in figure 1. 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, the matrix weights M_{lk} are incremented if both the input row I_{jl} and output column O_{jk} are set. 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 from the matrix.

In RAM-based networks, training is thus a single epoch process with one training step for each input-output association preserving their high speed. This also makes associative memories computationally simple and transparent with well-understood properties. In contrast, in most conventional neural networks used for classification such as MLP or RBF, (Bishop, 1995), training takes time and the resultant network is effectively a *black box*. RAM-based networks are able to partially match records during retrieval. Therefore, they can rapidly match records that are close to the input but do not match exactly. This partial matching is a central concept for our binary k-NN described in the following paragraphs.

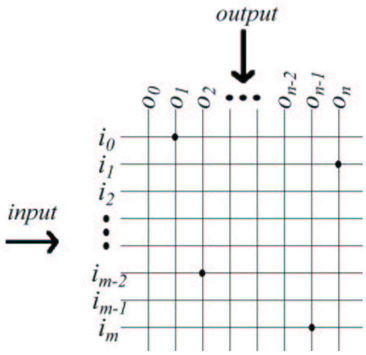


Fig. 1. Diagram of a Correlation Memory Matrix with input vector $i_0, i_1, i_2, ..., i_{m-1}, i_m$ and output vector $o_0, o_1, o_2, ..., o_{n-1}, o_n$. The input vector addresses the rows and the output vector the columns. The CMM is trained by associating input and output vectors which set the elements in the CMM to 1. All elements are initialised as 0. In the diagram elements $i_0 o_1$, $i_1 o_n$, $i_{m-2} o_2$ and $i_m o_{n-1}$ are set.

3. AURA

The AURA methodology has introduced a thresholding technique which we describe later that can retrieve the top n matches unlike the other RAM-based networks. During recall, AURA correlates the inputs to the stored matrix weights, sums the matrix columns and thresholds the summed column totals to retrieve sets of the best matching records. We have coupled this with a quantisation technique to map numeric data on to the binary inputs needed by the CMM. This rapid training, computational simplicity, network transparency, partial match capability and thresholding coupled with our quantisation technique make AURA ideal to use as the basis of an efficient k-NN implementation. A more formal definition of AURA, its components and methods now follows.

CMMs, shown in figure 1, are the building blocks for AURA systems. AURA uses binary and integer-valued input I and output O vectors to train records in to the CMM and recall sets of matching records from the CMM. For the methodology described in this paper, we:

- Train the data set into the CMM which indexes all records in the data set and allows them to be matched.
- Apply query records to the CMM in turn and retrieve a set of the best matching records, i.e., the nearest neighbours.

3.1. Training

In our k-NN implementation, input vectors represent quantised records during CMM training and output vectors uniquely identify each record in the data set. The training process is given in equation 1.

$$CMM = \bigvee_{\text{all j}} I_j \times O_j^T \text{ where} \bigvee \text{ is logical OR} \qquad (1)$$

Training is a single epoch process with one training step for each input-output association (each $I_j \times O_j^T$ in equation 1) which equates to one step for each record in the data set.

3.1.1. Quantisation

The CMMs in AURA require binary input vectors for training so we need to quantise (bin) and encode any numeric attributes. Our approach is to map the attribute values onto bins each of which indexes a specific row in the CMM. Each individual bin thus maps onto an integer as in equations 2 and 3 which identifies the bit to set within the CMM input vector and thus corresponds to a row in the CMM. Equation 2 represents the quantisation algorithm where a set of input values for attribute f map onto each bin which in turn maps to a unique integer to index the CMM row. We then set the appropriate bit in the input vector as in equation 3. The range of attribute values mapping to each bin is equal.

$$\mathbb{R}_{fi} \rightarrow bins_{fk} \rightarrowtail (\mathbb{Z}_{fk} + offset_f) \tag{2}$$

$$I_j^{'} = I_j \oplus (\mathbb{Z}_{fk} + offset_f) \tag{3}$$

where $i \in AttributeValue_f, cardinality(\mathbb{Z}_f) \equiv cardinality(bins_f)$ $offset_f$ is a cumulative integer offset within the binary vector I_j for each attribute f where $offset_{f+1} = offset_f + numberOfBins(f)$, $\rightarrow$ is a many-to-one mapping and $\rightarrowtail$ a one-to-one mapping and $|oplus$ sets the appropriate bit in the vector.

If a new record is evaluated with an attribute value that lies beyond those previously seen, it is placed in an extreme value bin at the appropriate end of the data range for values either larger than or smaller than those previously processed. Once sufficient records have values mapping to either of these bins, we recalculate all bin boundaries for that attribute to ensure that the set of bins covers the range of values. We do not show the extreme value bins in the figures in this paper for simplicity.

The data sets used in our evaluation are both unsupervised (no class attributes) and comprise solely numeric attributes so we require an unsupervised binning method (Dougherty, Kohavi and Sahami, 1995; Witten and Frank, 1999). There are two unsupervised binning methods: equi-width binning and equi-frequency binning and we use the former. Equi-width binning 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 and the number of records E mapping to a particular bin is proportional to the number of records n and inversely proportional to the number of bins b.

$$Width(bins_{fk}) = \frac{max(x_f) - min(x_f)}{b} \text{ and } E_{bins_{fk}} \propto \frac{n}{b} \text{ for all data.} \tag{4}$$

The even widths of the bins prevents distortion of the approximation of the Squared Euclidean Distances in the CMM compared to, for example, equi-frequency binning used previously in AURA (Zhou, Austin and Kennedy, 1999) which aligns the bin boundaries so each bin contains an approximately equal number of records. This then prevents regions of CMM saturation and evens the recall speed as all rows have an equivalent number of set bits. However, variable width binning distorts the Euclidean Distance approximation as there are a larger number of bins where the attribute values are clustered and relatively few bins representing the outlying values so the spread of attribute values represented by the bin varies markedly. This is particularly important for normally distributed attribute values where there will be many bins near the mode value and very few near the extremes. This distortion of distances is particularly germane for distance-based machine learning techniques such as k-NN. It lowers the recall accuracy of the AURA k-NN compared to fixed-width binning when compared to the results from a standard k-NN by up to 3% from an evaluation we performed previously (Hodge, Weeks & Austin, Unpublished). See (Dougherty, Kohavi and Sahami, 1995) for an overview of supervised and unsupervised binning techniques.

We vary the number of bins (b) in our evaluations to pinpoint the ideal configuration with sufficiently high recall yet acceptable run-time as more bins slows the recall as more CMM rows have to be processed. Choosing a suitable value for b is more difficult for unsupervised data compared to unsupervised data so a heuristic approach is necessitated, (Witten and Frank, 1999) recommend a cross-validation approach. We provide heuristics for selecting the number of bins in sections 5 and 6.

3.1.2. Input Vectors

Once the bins and integer mappings have been determined, we need to map each record D onto a binary input vector I_j. Each attribute D_f maps onto a consecutive section of bits in the binary vector as in equations 5, 6 where D_{fv} is the value of attribute f in record D.

$$D_{fv} \rightarrow bins_{fk} \rightarrowtail (\mathbb{Z}_{fk} + \mathit{offset}_f) \tag{5}$$

$$I_j^{'} = I_j \oplus (\mathbb{Z}_{fk} + \mathit{offset}_f) \tag{6}$$

Each concatenated binary vector represents a record from the data set and forms an input I_j to the CMM. The CMM associates the input with a unique output vector O_j^T during training, see equation 1. Each output vector is orthogonal with a single bit set corresponding to the records position in the data set, the first record has the first bit set in the output vector, the second and so on.

3.2. Recall

To recall the nearest 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 equations 5 and 6 and section 3.1. One problem with this quantisation is the boundary effect. The bins have hard boundaries so records lie within one bin only. Hence, for a particular value the distance to other points in the same bin may be greater than the distance to a point in a neighbouring bin. For example, if the binning boundaries lie on whole numbers and the query value is 4.99 then 4.01 will lie in the same bin yet 5.01 will be in the adjacent bin. However, 5.01 is much closer to 4.99 than 4.01.

To overcome this problem during recall Lees, OKeefe & Austin (Lees, O'Keefe & Austin, Unpublished) enhanced (Zhou, Austin and Kennedy, 1999) to set the

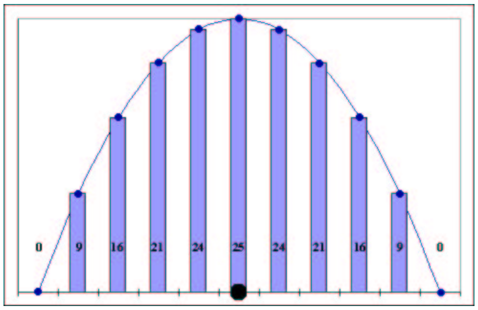


Fig. 2. The input values (shown as bars) of the CMM rows are set to emulate the parabola (line) which represents the Euclidean Distance from the central value (shown as large dot on central bar). The row input values (bar graph) are thus a discrete approximation of Squared Euclidean Distance.

bit representing the bin for the query record attribute and also set the bits for the two adjacent bins to retrieve any values that lie just across the bin boundary and hence may be closer. We improved this previously by exploiting AURA's ability to handle integer-valued input vectors (Hodge and Austin, 2003; Weeks et al., 2003). The technique called Integer Pyramid (or Triangular), uses one triangular kernel per attribute. Each kernel is superimposed onto the CMM input vector to form a concatenated kernel input vector for recall from the CMM. During recall, the summed intersection of the kernels contains discrete concentric patterns of equivalent CMM score. These scores represent the quantised City Block Distance (see equation 7) and are at a maximum where the target values for all F attributes coincide and decrease with distance from the target values.

$$CityBlockDist = \sum_{\text{all } f} |x_f - y_f| \text{ for all } f \text{ attributes.} \tag{7}$$

The Integer Pyramid achieved 99.49% accuracy and 100% accuracy for the two data sets used in this paper (Weeks et al., 2003) compared to 17.01% and 63.51% respectively for the 3-Bits Set. In this paper, we improve the technique further using a parabolic kernel which is analogous to quantised Squared Euclidean distance (see equation 8)

$$SquaredEuclideanDist = \sum_{\text{all } f} (x_f - y_f)^2 \text{ for all } f \text{ attributes.} \tag{8}$$

rather than City Block Distance as in figures 2 and 3. It has shown to be more accurate than Integer Pyramid (Weeks et al., 2003) where the Parabolic kernel (called SemiCircle in the paper) achieved 99.57% accuracy and 100% accuracy compared to 99.49% accuracy and 100% accuracy respectively with the Integer Pyramid (Triangular) kernel for the two data sets used in this paper.

The Parabolic kernel value for each bin ($bins_{fk}$) in attribute f is given in equation 9 where the target value's bin is $bins_{ft}$, $max(n)$ is the maximum number of bins across all attributes and n_f is the number of bins for attribute f. All kernels have the same maximum value $\left(\frac{max(n)}{2}\right)^2$ to ensure no bias across the attributes. We scale the kernel using α_f to spread the kernel across the range

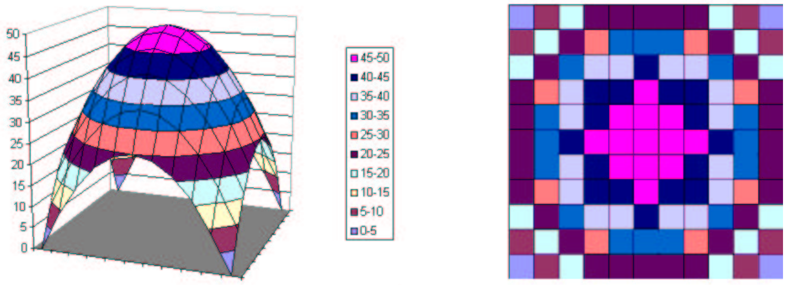


Fig. 3. Figure a) shows the smoothed Parabolic kernel intersection for a two attribute data with scores divided into ten discrete concentric regions. Figure b) shows the cumulative CMM column scores (representing the summed kernel intersections) for the AURA k-NN for the same two attribute data set with 11 bins per attribute and identical parabolas to figure 2 on both input attributes. The colours (scores) in the squares in b) match the banded colours on a) and represent the discrete concentric regions of equivalent score.

of each attribute in turn within the CMM input vector. The parabolic kernel is then superimposed onto the input vector as in equation 10.

$$Parabolic_{bins_{fk}} = \left[\left(\frac{max(n)}{2} \right)^2 - (bins_{ft} - bins_{fk})^2 \alpha_f \right] \text{ where } \alpha_f = \frac{(max(n))^2}{n_f^2} \tag{9}$$

$$I_j^{'} = I_j \oplus (Parabolic_{bins_{fk}} + \text{offset}_f) \text{ for all bins } (bins_{fk}) \text{ in all attributes } f \tag{10}$$

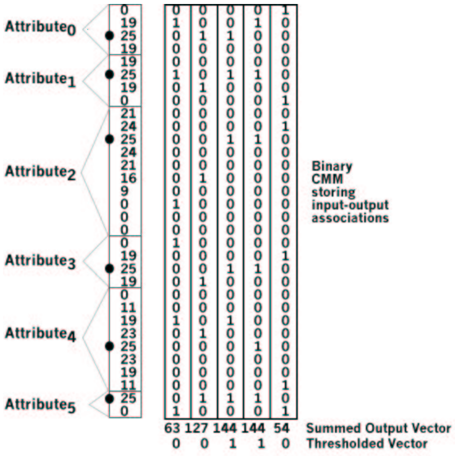
We move the kernels to match the input values unlike RBF (Bishop, 1995) where the kernels are fixed. The bin containing the query (target) value effectively receives the highest score with the score decreasing monotonically as the distance between the query value and a bin increases. If the bin of the target value is offset, i.e. not the median bin, then the Parabola is offset and truncated at one end as in $attribute_2$ of figure 4 where the Parabola is centred near the top and truncated at the top. If all attributes have an equivalent number of bins then the superimposed Parabolas will be identical. However, if the number of bins varies across the attributes, then the width of the Parabolas varies accordingly due to α_f in equation 9 spreading the kernel across the attribute width as shown in figure 4.

3.2.1. CMM Recall

To retrieve the best matching records for a particular query record (represented by integer-valued input I_k) using Parabolic kernels, the AURA k-NN effectively calculates the dot product of the input vector I_k and the CMM, computing a positive integer-valued output vector O_k (the summed output vector) as in equation 11 and figure 4.

$$O_k^T = I_k \bullet CMM \tag{11}$$

The summed output O_k is thresholded to produce a binary output vector as in figure 4. We use the L-max threshold (Austin, 1995). L-Max thresholding



0	0	0	0	0	1
19	1	0	0	1	0
25	0	1	1	0	0
19	0	0	0	0	0
19	0	0	0	0	0
25	1	0	1	1	0
19	0	1	0	0	0
0	0	0	0	0	1
21	0	0	0	0	0
24	0	0	0	0	1
25	0	0	1	1	0
24	0	0	0	0	0
21	0	0	0	0	0
16	0	1	0	0	0
9	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	1	0	0	0	0
19	0	0	0	0	1
25	0	0	1	1	0
19	0	1	0	0	0
0	0	0	0	0	0
11	0	0	0	0	0
19	1	0	1	0	0
23	0	1	0	0	0
25	0	0	0	1	0
23	0	0	0	0	0
19	0	0	0	0	0
11	0	0	0	0	1
25	0	1	1	1	0
0	1	0	0	0	1
Summed Output Vector	63	127	144	144	54
Thresholded Vector	0	0	1	1	0

Fig. 4. Diagram of the CMM recall with parabolic kernels. The left-hand column is the input vector. The dot indicates the target value for each attribute. The integer values in the column indicate the scores and represent the parabolic kernels superimposed onto each attribute centred about the target value. The AURA k-NN multiplies the input vector (using the dot product) by the binary matrix (set (1) and unset (0) matrix elements) and sums each column to produce the summed output vector which it thresholds to produce the thresholded vector. The thresholded vector effectively lists the matching records. The summed output vector is thresholded with value 2 here to retrieve the top 2 matches.

essentially retrieves *at least* L top matches, i.e., *at least* L nearest neighbours. L-max thresholding 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 threshold value is set to the highest integer value that will retrieve at least L matches. For k-NN, L is set to the value of k, where k is the number of nearest neighbours required.

3.2.2. Retrieving the k-Nearest Neighbours

AURA can identify the k-nearest matching records by the bits set in the thresholded output vector. In the work here, bit_0 in the output vector corresponds to the first record in the data, bit_1 to the second record and so on. Therefore, if bit_0 is set in the thresholded output vector then the first record is a match.

3.3. AURA k-NN versus Standard k-NN

Comparing the conventional k-NN and the AURA-based approach, the nearest neighbour in the standard k-NN is the record with the *lowest* Squared Euclidean Distance. Paradoxically, in the AURA k-NN, the best matching record (nearest neighbour) is the record with the *highest* score in the summed output vector. Also, the CMM-based approach calculates the k-nearest neighbours by traversing rows in the matrix, it is row-based. The standard k-NN is similar to traversing columns in the same CMM with floating-point matrix entries rather than the binary entries of the AURA CMMs so it may be considered column-based. The nested loop for standard on-line k-NN for a single query record is:

```
For all records (columns)
  For all attributes (rows)
```

In contrast the loop for the CMM for a single query record is:

```
For all attributes (rows)
  For all records (columns)
```

4. Evaluation

In this section, we analyse the processing time of the standard k-NN versus the AURA k-NN, the recall accuracy achieved by AURA and the scalability with respect to data size of the two techniques.

All 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 (AURA Web Page, 2003) which provides classes and methods for CMMs and thresholding.

4.1. Data Sets

We use two data sets to assess the techniques, chosen to contain large numbers of records with numeric attributes. The REAL data set contains 200,000 records with 14 continuous-valued attributes $0.0 \leq x \leq 1.0$ generated using a Java random number generator. The IBM data set contains 20,000 records with 9 integer-valued attributes where the attribute ranges vary from $0 \leq x \leq 4$ to $50000 \leq x \leq 1,350,000$ generated using the (IBM Data Generator, 2003). The first data set analyses the recall accuracy and recall precision of our quantisation and indexing due to the fine grained differences between attribute values. The second data set analyses the recall accuracy of our quantisation and binning compared to the standard Squared Euclidean Distance when the data ranges vary widely.

The standard k-NN, calculates the entire distance matrix (the distance between every pair of records) storing each records k nearest neighbours in an ordered linked list k elements in length. The list holds the indices of the k nearest records and their respective distances from x, sorted in ascending distance order. It has computational growth $\mathcal{O}(n^2)$ with respect to the number of records. The algorithm is:

```
For each record $x$
  For each record $y$
    If $SquaredEuclidDist(x, y)$ in equation 12 is less then the distance of the last
    record in the list (the most distant current neighbour), add $y$ to the list
    (in the correct position to maintain the sorted ascending distance order)
    and remove the last record from the list.
```

We use range normalisation in our Squared Euclidean distance calculation to ensure that all attributes are in the range 0 to 1 and hence each attribute produces an equal weight in the Squared Euclidean Distance calculation.

$$SquaredEuclidDist(x, y) = \sum_{\text{all } f} \left(\frac{(x_f - y_f)}{range_f} \right)^2 \text{ for all } f \text{ attributes.} \tag{12}$$

4.2. Timing

We provide two timings for the standard k-NN data structure generation. We recorded the time to read in the entire data set from a file on disk, generate the k-NN data structure with n^2 distance calculations where each vector is compared to all other except itself to calculate its nearest neighbours and output the 100 nearest neighbours to a file on disk. We then implemented a speedup by exploiting the commutativity of distance ($SquaredEuclidDist(x, y) \equiv SquaredEuclidDist(y, x)$) and again recorded the training time for this approach. Once we have calculated the distance between records x and y, ($SquaredEuclidDist(x, y)$ where $x < y$), we add y to the nearest neighbour list of x if it is closer than the most distant neighbour in the current list as previously but we *also* add x to the nearest neighbour list of y if it falls in the top k neighbours. Essentially we only need calculate the half distance matrix where $x < y$ using $\frac{n^2}{2}$ calculations.

The AURA k-NN processing time includes: the time to read in the entire data set from a file on disk and the time to train the data in to the CMM. For each record in the data set in turn, the time to convert the record to a CMM input vector and superimpose the parabolic kernels onto the vectors; the time to sum and threshold all columns of the CMM; the time to sort the matches into order; and, the time to output the 100 nearest neighbours to a file on disk.

For the AURA k-NN, we vary the number of bins used: 49,99,149,249,499 and 999 bins. The number of bins alters the specificity of the matching by varying the number of records with equivalent scores. The number of bins is inversely proportional to the number of values in each bin from equation 4 and hence the number of records that will map to each bin per attribute. We aim to identify the optimum number of bins across various data sets which is a trade off between retrieval time and granularity. Too many bins will take longer to process but too few bins will mean too many equivalently scored matches during recall thus slowing recall as all matches are processed and also lowering accuracy by reducing the scoring granularity.

Table 1 lists the processing time (training time plus recall time) for the n^2 and $\frac{n^2}{2}$ standard techniques and the AURA k-NN with 49,99,149,249,499 and 999 bins for the REAL (200,000 records with 14 continuous-valued attributes) and IBM (20,000 records with 9 integer-valued attributes) data sets. We ensured the process was as similar as possible for all algorithms under investigation to allow an unbiased comparison.

	Standard k-NN		AURA k-NN					
Dataset	n^2	$\frac{n^2}{2}$	49Bins	99Bins	149Bins	249Bins	499Bins	999Bins
REAL	52322	32985	13124	13316	13332	13538	13763	14526
IBM	337	229	85	86	88	90	93	100

Table 1. Table listing the processing time (seconds) for the REAL and IBM data sets for the standard k-NN (n^2 and $\frac{n^2}{2}$) and the AURA k-NN with 49,99,149,249,499 and 999 bins calculating 100 nearest neighbours.

	AURA k-NN					
Dataset	49Bins	99Bins	149Bins	249Bins	499Bins	999Bins
REAL-100	84.5	84.5	84.3	84.1	83.9	83.8
REAL-50	90.3	91.0	91.3	91.4	91.3	91.3
REAL-25	94.7	96.5	97.2	97.6	97.8	97.9
IBM-100	94.6	96.0	96.6	96.9	97.0	97.1
IBM-50	95.5	97.4	98.3	98.7	98.0	99.1
IBM-25	95.8	98.0	99.1	99.4	99.7	99.8

Table 2. Table listing the recall accuracy (%) for the REAL and IBM data sets for the AURA k-NN with 49,99,149,249,499 and 999 bins when the first 100 nearest neighbours (REAL-100 and IBM-100) retrieved by the AURA k-NN, the first 50 nearest neighbours (REAL-50 and IBM-50) and then the first 25 nearest neighbours (REAL-25 and IBM-25) are compared to the nearest neighbours calculated by the standard k-NN.

4.3. Accuracy

We saved the two nearest neighbour lists produced by the standard k-NN (with k=100) to a file, one list for REAL and one list for IBM. These lists provide a benchmark to compare the nearest neighbour list for the AURA technique. We recalled and listed the 100 nearest neighbours identified by the AURA technique for the first 20,000 records from the REAL and all 20,000 records from the IBM data set for the various numbers of bins listed above. The REAL data set is randomly ordered so the 20,000 records provide a statistically sound evaluation. From these 20,000 lists of 100 nearest neighbours, we counted the number of nearest neighbours common to this list and the corresponding standard k-NN list for that record. We calculated the average number of common records for each of the 20,000 comparisons to give the average recall percentage for each of the six configurations of AURA. We repeated this evaluation for the first 50 nearest neighbours and then the first 25 nearest neighbours in the lists.

Table 2 lists the percentage recall accuracy when the nearest neighbour lists retrieved by the AURA k-NN with 49,99,149,249,499 and 999 bins are compared with the corresponding list for the standard k-NN using both the REAL and IBM data sets for 100 nearest neighbours, 50 nearest neighbours and for 25 nearest neighbours.

	Data Set Size				
Algorithm	40,000	80,000	120,000	160,000	200,000
149Bins	506	2068	4708	8381	13332
n^2	1930	7634	17191	32320	52322
$\frac{n^2}{2}$	1307	5236	11829	20842	32985

Table 3. Table listing the scalability (processing time (seconds)) for the standard k-NN (n^2 and $\frac{n^2}{2}$) and the AURA k-NN with 149 bins for the REAL data set with between 40,000 and 200,000 records.

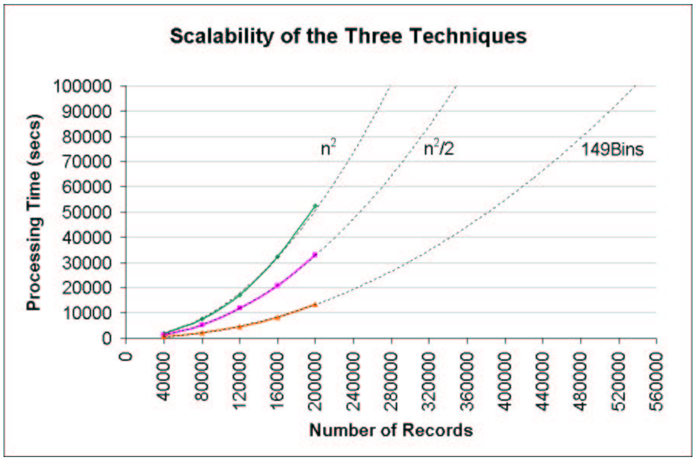


Fig. 5. Diagram showing the processing time for each technique up to 200,000 records and the trend-line for each of the three plots extrapolated to show the expected growth rate.

4.4. Scalability

Using subsets of the REAL data set with 40K, 80K, 120K, 160K and 200K records, we noted the processing time for the n^2 and $\frac{n^2}{2}$ k-NN techniques and the AURA k-NN with 149 bins and calculating 100 nearest neighbours. We elected to use 149 bins for the AURA k-NN as this performed well for both the REAL and IBM data sets (see tables 1 & 2). We note that in the scalability test we fix the number of bins to note the increase in processing time. In a real-world system, the user may elect to increase the number of bins proportionally as the size of the data set grows to maintain separability of the data values.

Table 3 lists the computational growth of the standard k-NN (n^2 and $\frac{n^2}{2}$) versus the AURA k-NN with 149 bins when the REAL data set size increases from 40,000 to 200,000 records. Figure 5 plots the results from table 3 and extrapolates the trend-line of the plots to allow the reader to view the computational growth (plot gradient).

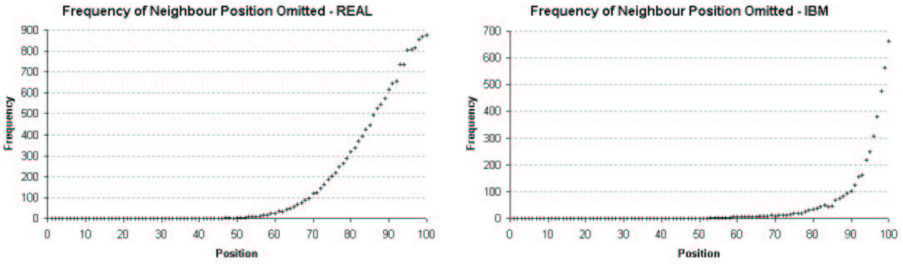


Fig. 6. Diagram showing the position (1-100) of the neighbours in the standard k-NN list which are omitted by the AURA k-NN with 149 bins for the first 1000 records of the IBM (right plot) and REAL (left plot) data sets.

5. Analysis

For the REAL data set, the AURA k-NN ranges from 4.0 times faster with 49 bins to 3.6 faster with 999 bins compared with n^2 standard k-NN and ranges from 2.5 to 2.3 faster compared with $\frac{n^2}{2}$ standard k-NN. For the IBM data set, the AURA k-NN ranges from 4.0 times faster with 49 bins to 3.3 faster with 999 bins compared with n^2 standard k-NN and ranges from 2.7 to 2.3 faster compared with $\frac{n^2}{2}$ standard k-NN.

The method in this paper is 1.8 times faster than the AURA k-NN detailed in (Weeks et al., 2003) which takes 156 seconds to process the IBM data set using 149 bins and 23249 seconds for the REAL data set with 149 bins compared to 88 seconds and 13332 seconds respectively for the method in this paper.

The recall accuracy for the AURA k-NN is 97% for the REAL data set and 99% for the IBM data set when the first 25 nearest neighbours are compared. It is 91% for the REAL and 99% for the IBM when the first 50 nearest neighbours are compared and 84% for the REAL data set and 97% for the IBM data set when the 100 nearest neighbours are compared. This compares to 99.59% for the REAL and 100% for the IBM with the AURA k-NN in (Weeks et al., 2003) when 100 nearest neighbours are compared. We note that the largest percentage of the nearest neighbours omitted by the AURA k-NN are towards the end of the nearest neighbour list (i.e. the 90th-100th nearest neighbours) from figure 6.

Even though we are using the kernels to overcome the bin boundary problem alluded to in section 3.2, for the continuous-valued REAL data the bin boundaries are more of an issue than for the integer valued IBM data set. The integer values are discrete so the minimum distance between any two integers is 1. As we increase the number of bins, we are tending towards one integer value per bin and hence no bin boundary problem. The real-values are accurate to 16 decimal places so there may be only 0.0000000000000001 between two data points separated by a bin boundary but from equation 4 the maximum intra-bin distance is $width(bins_{fk}$. Thus, the recall accuracy achieved for the REAL data set is just below the recall accuracy achieved for the IBM data set for all configurations. We can see from table 2 that the recall accuracy increase with the number of bins for REAL-25 but conversely decreases for REAL-100. For REAL-50, it increases then decreases with respect to the number of bins. The recall accuracy increases consistently as the number of bins increases for all configurations of the IBM data set. For a real-valued data set, we posit the rule-of-thumb more bins for fewer neighbours but conversely fewer bins for more neighbours. For a discrete data set we posit the rule-of-thumb more bins is better. However, we note that there is a trade-off between the number of bins and the processing time. Even though increasing the number of bins for REAL-25 and all IBM configurations increases the accuracy it also increases the processing time. We would recommend 149 bins as the optimum value for these two data sets with respect to this trade-off although there is little difference between the recall accuracy between 99 bins and 999 bins within each configuration. We would recommend a cross-validation to ensure maximum accuracy if time permits starting with 149 bins.

The AURA k-NN scales better that the standard techniques. It is up to four times faster than the standard techniques with 200,000 records. Although growth is still near $\mathcal{O}(n^2)$ due to the nature of k-NN, the faster initial processing time means we can process up to approximately 540,000 records in 100,000 seconds whereas n^2 k-NN can only process 270,000 records in 100,000 seconds at which point the computational growth becomes steep for n^2. If we increase the number of records by 20,000, we increase the processing time for n^2 by 20,000 seconds. K-NN for classification or outlier detection as described in this paper is inherently $\mathcal{O}(n^2)$ due to the following loop embedded in the algorithm:

```
For all $n$ records
  Calculate distance to all other $(n-1)$ records
```

6. Conclusion

This paper has demonstrated that AURA k-NN has faster recall speed than standard techniques even when optimisations which effectively halve the number of calculations are included within the standard technique. AURA scales better than the standard technique and the difference between the recall times (AURA time/standard k-NN time) increases as the data size increases.

There is a trade-off with the AURA k-NN of speed versus accuracy. We have previously post-processed with a standard k-NN (Hodge and Austin, 2003; Weeks et al., 2003) a set of candidate matches extracted from the CMM (usually of size $10 \times k$). We effectively use AURA to minimise the search space for the k-NN and then use the more accurate but slower Euclidean technique to process the candidate matches. This improves recall accuracy with 100 nearest neighbours to 99.59% and 100% for the REAL and IBM data sets respectively (Weeks et al., 2003) to the slight detriment of the recall speed but (Weeks et al., 2003) is still faster than standard k-NN. Alternatively, if speed is the issue, the kernel approximation detailed in this paper emulates the distances sufficiently accurately for 50 nearest neighbours or very accurately with 25 nearest neighbours while speeding recall by a factor of 1.8 compared to the AURA k-NN in (Weeks et al., 2003). For large data sets we posit the method detailed in this paper due to the faster speed preferably with 25 nearest neighbours and recommend a k value of fewer than 50. From the evaluation here, we would recommend 149 bins as this provides the best trade-off between recall accuracy versus recall speed for both data sets. We would recommend a cross-validation to determine the optimum number of bins if time permits starting with 149 bins. For a real-valued data set, we posit the rule-of-thumb more bins for fewer neighbours but conversely fewer bins for more neighbours. For a discrete data set we posit the rule-of-thumb more bins is better. For smaller data sets we posit the method detailed in (Weeks et al., 2003) due to the higher accuracy.

The recommended AURA k-NN configurations emulate the standard k-NN by 99%+ with respect to neighbours retrieved. For a classification task, the class counts of the neighbours retrieved for each query record would be 99%+ similar for the recommended configurations compared to the standard technique.

For the AURA k-NN technique, there are two user-specified parameters: the number of neighbours and the number of bins per attribute. Thus the use of AURA to speed the k-NN has introduced only one additional parameter compared to standard k-NN: the number of bins per attribute. There are four variables which effect the execution speed in the AURA k-NN: the number of neighbours, the number of attributes, the number of records and the number of bins per attribute. The use of AURA to speed the k-NN has introduced only one additional variable compared to standard k-NN: the number of bins per attribute.

We propose to use the AURA k-NN developed here, within a fraud detection system and within an outlier detection system where the data sets are large with approximately 400,000 records and 360 attributes.

Acknowledgements. This work was supported by Engineering & Physical Sciences Research Council (EPSRC) Grant GR/R55191/01.

References

Aha, D. W. and Bankert, R. B. (1994). Feature Selection for Case-Based Classification of Cloud Types: An Empirical Comparison. In *Proceedings of the AAAI-94 Workshop on Case-Based Reasoning.*

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

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

Austin, J. (1995). 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.

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

Bishop, C.M. (1995). *Neural networks for pattern recognition.* Oxford, Clarendon P.

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

Dasarathy, B. (editor) (1991). *Nearest Neighbor (NN) norms: NN pattern classification techniques.* IEEE Computer Society.

Dougherty, J., Kohavi, R. and Sahami, M. (1995) *Supervised and Unsupervised Discretization of Continuous Features..* In *Proceedings of 12th International Conference on Machine Learning*, pages 194–202, San Francisco, CA: Morgan Kaufmann.

Hodge, V., Lees, K. and Austin, J. (2003). A High Performance k-NN Approach Using Binary Neural Networks. *Neural Networks*, 17(3) pp 441–458, Elsevier Science.

Hodge, V. and Austin, J. (2004). A Survey of Outlier Detection Methodologies. *In press, Artificial Intelligence Review, Kluwer.*

Hodge, V., Weeks, M., and Austin, J. (2003). AURA k-Nearest Neighbour Approach. *Unpublished.*

Hopfield, J. (1982). Neural networks and physical systems with emergent collective computation abilities. *Proc.Nat.Acad.Sci.USA*, 79:2554–2558.

IBM Quest Data Mining Project (2003). The Quest Synthetic Data Generation Code for Classification - *http://www.almaden.ibm.com/software/quest/Resources/datasets/syndata.html#classSynData*, last accessed 16th October, 2003.

Lees, K., O'Keefe, S. and Austin, J. (2001). Imputation Using a Binary Neural Network. *Unpublished*.

Knorr, E. and Ng, R. (1998). Algorithms for Mining Distance-Based Outliers in Large Datasets . In *Proceedings of the VLDB Conference*, pages 392–403, New York, USA.

Skalak, D. (1994). Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Machine Learning: Proceedings of the Eleventh International Conference*. pages 293–301.

Turner, A. (2003). Introduction to CMMs and AURA-Based Systems - *http://www.cs.york.ac.uk/arch/NeuralNetworks/binary.html*, last accessed 8th August, 2003.

Weeks, M., Hodge, V., O'Keefe, S., Austin, J., and Lees, K. (2003). Improved AURA k-Nearest Neighbour Approach. In *Proceedings of IWANN-2003, International Work-conference on Artificial and Natural Neural Networks, Mahon, Menorca, Balearic Islands, Spain, June 3-6.*.

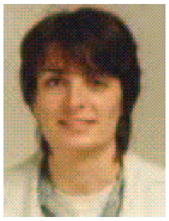
Wettschereck, D. (1994). *A study of distance-based machine learning algorithms.* PhD thesis, Department of Computer Science, Oregon State University, Corvallis.

Witten, I. and Frank, E. (1999). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, October 1999, ISBN 1-55860-552-5.

Zhou, P., Austin, J. and Kennedy, J. (1999). A High Performance k-NN Classifier Using a Binary Correlation Matrix Memory. In *Advances in Neural Information Processing Systems 11.*

Author Biographies

Prof. Jim Austin has the Chair of Neural Computation in the Department of Computer Science, University of York, where he is the leader of the Advanced Computer Architecture Group. He has extensive expertise in neural networks as well as computer architecture and vision. Jim Austin has published extensively in this field, including a book on RAM based neural networks.



Dr. Victoria Hodge received a B.Sc. degree from the University of York, UK, in 1997 and a Ph.D. from the University of York, in 2001. She is a member of the Advanced Computer Architecture Group in the Department of Computer Science, University of York, investigating the integration of neural networks and information retrieval, focussing particularly on document retrieval from large corpora and detecting anomalous records in large datasets.