A High Performance k-NN Approach Using Binary Neural Networks

Victoria J. Hodge **Ken J. Lees** **James L. Austin**

Advanced Computer Architecture Group
Department of Computer Science
University of York,
York,
YO10 5DD,
UK

Abstract

This paper evaluates a novel k-Nearest Neighbour classifier built from binary neural networks. The binary neural approach uses robust encoding to map standard ordinal, categorical and numeric data sets onto a binary neural network. The binary neural network uses high speed pattern matching to recall a candidate set of matching records which are then processed by a conventional k-nearest neighbour approach to determine 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

Binary Neural Network, Associative Memory, Correlation Matrix Memory, k-Nearest Neighbour, Euclidean Distance, Robust Encoding, Quantisation, Binary Mapping.

Nomenclature

Record – a collection of attributes. The set of all records is the data set.
Attribute – a dimension within a data record.
AURA – C++ library of classes and methods for an associative memory neural network.
CMM – a matrix memory structure class within AURA.
Vector – an n-tuple to address rows or columns in a CMM.
I_j^l - input vector ***j***, vector element ***l***.
O_j^k - output vector ***j***, vector element ***k***.
► - a many-to-1 mapping.
⇨ - a 1-to-1 mapping.

1 Introduction

The approach described in this paper is an enhancement of the binary neural network k-NN classifier described in Zhou, Austin & Kennedy (1999) which outperformed neural classifiers such as Multi-Layer Perceptron (MLP) and Radial-Basis Function (RBF) networks with respect to speed and recall accuracy on a classification task. Lees, O'Keefe & Austin (2001) enhanced it for imputing missing values and outlier detection in large-scale data sets but it is generically applicable to any pattern classification task.

Standard k-nearest neighbour is a widely applicable classification technique; see Wettschereck (1994) for an analysis of k-nearest neighbour. For standard classification, k-NN examines those points in a particular data space lying 'nearest' to a query point, as identified by some suitable distance metric such as Euclidean distance, and uses the respective classifications of these nearest neighbours to determine the class of the query point. For all unclassified records, a k-NN approach must calculate the distance to **all** records in the data set to determine the nearest neighbours.

K-Nearest Neighbour is also used for outlier detection; see for example, Hodge & Austin (2002) or Barnett & Lewis (1994). Outlier detection aims to pinpoint those records in a data space which lie isolated from the bulk of the data and hence indicate 'abnormal' data records. A record with a large distance to its nearest neighbours may well indicate a system fault, instrument error or anomalous reading. Such fault detection is vital in safety critical applications. Such outlier detection, Hodge & Austin (2002), requires the calculation of the distance between each record and all other records in the data set to identify the k-nearest neighbours for each record. We can then examine the distances to the k-nearest neighbours to pinpoint records that are most distant from their neighbours and hence records which may represent outliers.

1.1 Rationale

For the classification and outlier detection techniques described above the computational growth of standard k-NN is $\boldsymbol{O(n^2)}$ with respect to the number of records in the data set, Knorr & Ng (1998), Dasarathy (1991) so even on modern high speed computers there is a practical upper limit to the number of records that may be processed dependent on the processor time available. Therefore, a method is desired to speed the identification of the nearest neighbours while maintaining the recall accuracy of the standard k-NN to allow extremely large data sets to be processed.

1.2 Motivation

The binary neural technique evaluated here uses the Advanced Uncertain Reasoning Architecture (AURA), Austin (1995), which comprises a C++ library of classes and functions. AURA has previously been used in an information retrieval system, Hodge (2001), high speed rule matching systems, Austin, Kennedy & Lees (1995), 3-D structure matching, Turner & Austin (2000) and trademark database searching, Alwis & Austin (1998). AURA techniques have previously demonstrated superior performance with respect to speed compared to conventional data indexing approaches Hodge & Austin (2001) and superior speed and accuracy compared to conventional neural classifiers Zhou, Austin & Kennedy (1999). Our technique described in this paper uses a robust quantisation and encoding method to map numeric attributes from the data set onto binary and integer-valued vectors for training and recall in an AURA k-NN classifier.

AURA belongs to a class of neural networks called Random Access Memory (RAM) networks. RAM networks were first developed by Bledsoe & Browning (1959) and Aleksander & Albrow (1968) for pattern recognition and led to the WISARD pattern recognition machine Aleksander, Thomas & Bowden (1984). See also Austin (1998) for a detailed compilation of RAM techniques. RAMs are founded on the twin principles of matrices and n-tupling to store associations between inputs $\boldsymbol{I_j}$ and outputs $\boldsymbol{O_j}$. Each matrix accepts $\boldsymbol{n}$ inputs as a vector or tuple addressing $\boldsymbol{n}$ rows or $\boldsymbol{n}$ columns of the matrix. During the training phase, the matrix weights $\boldsymbol{M^{lk}}$ are incremented if both the input row I_j^l and output column O_j^k are set. During recall, the presentation of vector $\boldsymbol{I_j}$ elicits the recall of vector $\boldsymbol{O_j}$ as vector $\boldsymbol{I_j}$ contains all of the addressing information necessary to access and retrieve vector $\boldsymbol{O_j}$ from the matrix.

In RAMs, 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”. Storage is efficient in RAMs due to their simple matrix structure. New input patterns do not require additional memory allocation as they are overlaid with existing trained patterns, see Turner & Austin (1997). Associative memory-based networks such as AURA are able to partially match records during retrieval unlike the conventional neural networks such as MLP or RBF. Thus AURA 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. Hopfield Associative Memories (HAMs), Hopfield (1982), have extended the method by feeding the outputs back into the inputs allowing the network to converge to a steady state. The stored tuples represent the stable states of the network and the convergence function allows extra error correction compared to a standard associative memory. We do not require this extra error correction for the k-NN.

The AURA methodology has introduced a thresholding technique which we describe later that can retrieve the top $\boldsymbol{n}$ matches unlike the other RAMs. We have coupled this with a quantisation technique to map numeric data on to the binary components of AURA. 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.

1.3 Aim

The first aim of the paper is to identify to what extent this quantisation is preserving the Euclidean distances between the attribute values and thus whether our binary neural methodology is recalling the correct k-nearest neighbours as identified by a conventional Euclidean distance approach. The second aim is to demonstrate the speedup produced by using AURA techniques to retrieve a small set of candidate matches and then calculate the nearest neighbours from this set compared to processing the entire data set using Euclidean Distance.

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

2 AURA

Correlation Matrix Memories (CMMs) described in Austin (1995) form the central paradigm of the AURA techniques. AURA uses binary and integer-valued input and output vectors (n-tuples) to train records in to the CMM and recall sets of matching records from the CMM. AURA is typically configured with a single CMM as in this paper or with multiple CMMs operating in parallel.

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. AURA provides this rapid first match to speedily prune the data set and produce a set of ***n*** candidate matches using CMMs, summing, thresholding and input and output vectors.
- Process these ***n*** candidate matches using a conventional Euclidean distance-based nearest neighbour algorithm which is more accurate but slower.

Essentially, we reduce the size of the search space very rapidly by retrieving a small set of ***n*** candidate matches and then perform a finer-grained match using the slower but more specific Euclidean Distance to pinpoint the ***k*** nearest neighbours from this small set. This allows us to speed up k-NN and to use k-NN on larger data sets than would be possible using the standard Euclidean Distance k-NN alone.

2.1 Correlation Matrix Memory Training

The CMM stores linear mappings between binary input vectors ***I*** (numeric-to-binary quantised inputs in our k-NN implementation) and binary output vectors ***O*** which uniquely identify each record in the data set as shown in Fig. 1 and Eq.1.

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

There is one training step for each record in the data set preserving AURA's high speed.

2.1.1 Numeric-to-binary quantisation

The CMMs in AURA require binary input vectors for training so we need to quantise and encode any numeric attributes into a binary format. For the data sets evaluated here the data ranges from $0.0 \le x \le +1.0$ for two data sets and

$0 \le x \le 15$ *and* $x \in$ *Integer* for the other. However, the robust encoding technique can quantise values across any range.

The quantisation approach developed by Zhou, Austin & Kennedy (1999) aims to distribute the records uniformly across the binary codes for each attribute i.e., the bins do not cover equal intervals but rather an equal number of records from the data set maps onto each bin. The range of values for each attribute is divided into ***b*** bins such that each bin contains an equal number of records for that attribute. The even distribution of records in the bins means an even distribution of set bits in the CMM. This permits separation of input and output patterns and prevents regions of CMM saturation that produce too many matches while other regions yield no matches. Additionally, most data has non-uniform distribution with respect to the individual attributes so this fixed cardinality quantisation approach also makes no assumptions about the distribution. Many standard classification techniques assume the data distribution follows specific models such as a Gaussian distribution model in Gaussian Mixture Models, Bishop (1995).

The quantisation procedure processes each attribute in turn. With respect to each attribute,

1. Sort all ***N*** data points into ascending order,
2. Find the maximum number of identical points N_i and thus estimate the number of distinct data values in each bin N_p as $N_p = (N-N_i)/N_b$ where N_b is the number of bins.
3. Set the uppermost boundary of each bin as the next N_p^{th} data value in the sorted order.
4. Count the number of data values either side of N_p which are equal, and either include these points in the current bin or promote them to the next bin.
5. If the number of distinct values in the final bin N_f is greater than $(N_p + N_b)$ then increase N_p by $(N_f - N_p)/N_b$ and rerun the partitioning process from step 2 for that attribute.

We note that rerunning the binning process may increase or reduce the number of bins relative to the initial number specified. In our empirical analysis in this paper, we analyse our binary neural classifier with varying number of bins. We always state the initial number of bins in our empirical evaluations detailed in sections 3 & 4 which may be changed by the binning process. However, it is the initial user-specified parameter we are analysing to examine the recall accuracy with various parameter settings so we list the bin numbers as they were originally specified in sections 3 & 4.

Once the bins have been determined, we need to map the records onto binary input vectors which input to the CMM as in Eq. 1 and Fig. 1. Each attribute maps onto a consecutive chunk in the binary vector as shown in Fig. 2. Each bin within each attribute maps on to a single position in the input vector. If there are 4 bins for a particular attribute, for example attribute$_0$ in Fig. 2, then we allocate 4 positions in the vector. To quantise each record in the data set:

- For each attribute in turn,
    1. Calculate the bin for the attribute value
    2. Set the corresponding bit in the vector.

Each concatenated binary vector (the right-hand bit vector in Fig. 2) represents a record from the data set and forms an input $\boldsymbol{I_j}$ to the CMM. The concatenated binary vector is trained in to the CMM and associated with an output vector $\boldsymbol{O_j}$ which uniquely identifies that particular record in the data set, see Eq. 1. Each output vector has 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.

2.2 Recall

The recall process retrieves the candidate matches to pass to the Euclidean Distance post-processor for a particular query record as outlined in section 2. To retrieve the candidate matches for a record, we quantise each attribute value in turn to determine the bin for that attribute value, set the corresponding bit and thus produce an input vector as in section 2.1.1 and in Fig. 2.

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 but 5.01 is much closer to 4.99 than 4.01. To overcome this problem during recall Lees, O'Keefe & Austin (2001) enhanced Zhou, Austin & Kennedy (1999) previous approach to set the 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 call this approach **3-Bits Set,** see Fig. 3.

We have enhanced 3-Bits Set further to produce an alternative technique named **Integer Pyramid**. Integer Pyramid exploits the ability of the AURA techniques to take positive integer-valued input vectors rather than binary input vectors during recall. The 3-Bits Set approach is extended by using integer inputs with integer values reflecting the distance from the target values of each attribute during CMM recall across the entire range of values as shown in Fig. 4.

For the Integer Pyramid, the bin containing the query value effectively receives the highest score with the score decreasing 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 pyramid is offset and truncated at one end as in attribute 2 of Fig. 4 where the pyramid is centred near the top and truncated at the top. If all attributes have an equivalent number of bins then the superimposed Integer Pyramids will be identical. However, if the number of bins varies across the attributes, then the width of the pyramids varies accordingly as in Fig. 4. The maximum values of the integer pyramids may also be varied to reflect the importance of the attributes as in weighted standard k-NN, Wettscherek (1994), where attributes are multiplied by weight factors to indicate their relative importance. For the data sets used in this paper, all attributes have an equivalent width and importance. Fig. 5 shows the process to create a 3-Bits Set and Integer Pyramid input vector for the first attribute of the REAL data set used in this paper (described in section 3.1) which has values in the range $0.0 \le x \le 1.0$. The diagram also shows two query records, one with the first attribute with value 0.50 and the second with attribute value 0.95.

2.2.1 CMM Recall

To retrieve the best matching patterns for a particular input using either 3-Bits Set or Integer Pyramid coding, the CMM effectively calculates the inner product of the input vector ***I*** and the CMM, computing a positive integer-valued output vector ***O*** as in Eq. 2 and Fig. 6 where a 2 attribute vector is input with 3-Bits Set to the leftmost CMM and with Integer Pyramid to the rightmost CMM and the top 2 matches retrieved from each.

$$I \bullet CMM = O^T \tag{2}$$

In Fig. 6, the binary-valued input vector used in the 3-Bits Set technique activates the respective rows of the CMM. The integer-valued input vector of the Integer Pyramid technique activates the rows of the CMM with the respective integer values.

The integer-valued activation vector ***O*** generated for both techniques is thresholded to produce a superimposed binary output vector. We use the L-Max threshold (detailed in Austin (1995)) where ***L*** is set to the value of ***k*** (number of nearest neighbours) for the 3-Bits Set and ***k***×***EM*** for the Integer Pyramid, where ***EM*** is the Error Margin described next. L-Max thresholding essentially retrieves *at least* ***L*** top matches, i.e., at least ***k*** nearest neighbours. It sets a bit in the superimposed output vector for every location in the integer activation 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. In Fig. 6, the threshold value was set to 5 and this enables two matches to be retrieved. If ***L*** were set to 3 (at least 3 matches required) then we would set the threshold value to 4 which would in fact retrieve five matches, columns 1,2,4,5,6 in Fig. 6. **All** equivalent matches are passed in to the post-processing vector distance calculation using conventional k-NN (see section 2).

The Integer Pyramid process allows finer-grained attribute differentiation than the 3-Bits Set due to its graduated scoring compared to the all-or-nothing scoring metric of 3-Bits Set approach. Hence, the Integer Pyramid permits finer-grained distance approximation and matching so we can control the number of matches retrieved more closely than for 3-Bits Set. We vary the number of candidate matches retrieved from the CMM for the Integer Pyramid using a parameter we call the **Error Margin (EM)**, which is effectively a multiple of ***k***. The AURA L-Max thresholding method reduces the threshold value until it has recalled at least ***k***×***EM*** matches. For example, an ***EM*** of 2 will retrieve at least ***k***×***2*** candidate matches so for a ***k*** value of 100, it will retrieve at least 200 candidate matches. This allows us to find an optimal ***EM*** setting in the empirical analysis detailed in section 4.2. The Error Margin setting is a trade-off between recall accuracy and retrieval speed. A higher value will retrieve more candidate matches which maximises the likelihood of recalling the true 100 nearest neighbours as defined by standard Euclidean k-NN but increases the retrieval time as more candidate matches have to be post-processed.

AURA can identify the candidate records for both the 3-Bits Set and Integer Pyramid methods by the bits set in the superimposed binary output vector produced after thresholding, see Fig. 6. The superimposed output vector is effectively split into a set of separated output vectors each with one bit set which identifies the matching record. There is one separated output vector for each bit set in the superimposed output vector. In the work here, bit-0 in the separated 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 separated output vector then the first record is a match.

Comparing the conventional k-NN and the AURA k-NN approach, 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 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)
```

We exploit the row-based approach in AURA due to the ability of the CMMs to optimise storage by packing rows; they only store the locations of the set bits as a list of integers rather than storing the binary values of all matrix locations, Hodge (2001). For a sparsely populated matrix such as those analysed here, this compacted list is more efficient than storing all of the matrix entries, see Hodge (2001) for a discussion. If we use 20 bins for each attribute, then only 5% of the entries in each row will be set (5% of the positions will be stored in the list) as the quantisation process maps 5% of the records to each bin. We could convert to the column-based approach by activating columns in the CMM during recall but we would either have to store the entire matrix to allow column access hence vastly increasing the memory overhead otherwise the recall would be far less efficient than for row-based. For column-based, we would quantise the query and produce the input vector as previously. We would then have to traverse the entire list for each column to count the number of locations that match with those in the input and place the score for each column in an accumulator. This is less efficient than the row-based as we have to examine all entries for all columns but the row-based approach only needs to examine the activated rows and can ignore all others. For example with 30 bins and the 3-Bits Set technique, only 3 out of every 30 lines (the 3 bits set for each attribute comprising 30 bins) need be processed by the row-based technique. The column-based would need to match against all entries in all columns. The row-based technique is 90% more efficient in this case as only 10% of lines are activated.

2.3 Determining the k *nearest neighbours*

The set of best matches retrieved from the CMM for both 3-Bits Set and the Integer Pyramid contains ***R*** records where ***R*** > ***k***. We use conventional Euclidean-distance k-NN on these ***R*** records from the original data set to identify the k-nearest neighbours. If the quantisation is accurate, then the ***R*** matches will always contain the ***k*** true matches with respect to Euclidean distances and will form the ***k*** top matches during the standard k-NN post-processing. In section 4, we analyse the recall percentage, i.e., the percentage of true matches recalled by the numeric-to-binary AURA methodology across a range of queries.

3 Evaluation

The classification and outlier detection techniques outlined in the Introduction require the calculation of the k-nearest neighbours for every record in the data set. In this paper, we compare the training speed, recall speed, memory overhead and recall accuracy of the AURA k-NN against a standard k-NN implementation. This will identify the speedup and memory improvement introduced by using AURA to implement k-NN and also compare the recall accuracy of the AURA k-NN against the standard technique to verify that these efficiency gains are not offset by a fall in recall accuracy. We calculate the k-nearest neighbours with k set to 100 for the techniques under evaluation using three data sets we call **REAL**, **LR** and **REAL_2M**. We chose a value of k=100 as we felt this was higher than used in current implementations of k-NN so will provide a through analysis of the techniques under investigation.

3.1 Data Sets

The first evaluation data set is synthetic (**REAL**) and is generated using a Java random number generator application. It contains 200,000 non-multivariate vectors with 14 real-valued attributes. All attributes range between $0.0 \leq x \leq +1.0$ and the distributions are uniform. This data set provides a thorough examination of the recall accuracy of the binary neural method. It empirically evaluates the accuracy of the binning procedure and the scoring metric used for matching records as real-valued attributes are most difficult to quantise and score accurately. Integer-based attributes are intrinsically discrete and amenable to further quantisation. Real attributes are continuous and necessitate a more rigorous and considered quantisation technique.

The second data set is the Letter Recognition data set (**LR**) from the UCI data set repository, Blake & Merz (1998). This consists of 20,000 integer-valued 16-attribute vectors where all attributes range between $0 \leq x \leq 15$ and $x \in \mathit{Integer}$. The distributions of the 16 attributes are shown in Fig. 7. None of the attributes follows a uniform distribution unlike the **REAL** and **REAL_2M** data sets which are both uniformly distributed. We use the data set in the same way as the **REAL** and **REAL_2M** data sets, mapping the attributes of the records onto input vectors in the same way and retrieving the nearest neighbours for each record in the data set in turn (as described in section 2). We are not explicitly using the **LR** data set for letter recognition as we are using the data set to assess the accuracy of the quantisation and scoring metrics in the AURA k-NN with non-uniformly distributed integer-valued attributes.

The third data set (**REAL_2M**) is a larger version of the **REAL** data set and contains 2 million records each with 14 real-valued attributes again generated by the java random number generator application. All attributes range between 0 and 1, $0.0 \leq x \leq +1.0$. This data set is used to test the scalability of both the conventional k-NN and the binary neural methodologies. We subdivide the data set and analyse the training and recall speeds of both approaches on various sized subsets. We also verify the recall accuracy of the binary neural approach on different sized data sets.

Vector quantisation, particularly mapping onto binary values can skew and distort numeric distributions so the first and third data sets provide a thorough evaluation of the quantisation accuracies of the two AURA k-NN techniques. Both real-valued data sets are uniformly distributed so we include the **LR** data set which has randomly distributed integer attributes to further evaluate the quantisation and recall accuracy.

3.2 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 (version 1.0.9.3), Turner (2002), that provides classes and methods for CMMs, binary and integer vectors and thresholding.

3.2.1 Standard k-nearest neighbour

For the conventional nearest neighbour method, there are two sub-techniques for outlier detection, retrospective and on-line, Hodge & Austin (2002). The retrospective technique identifies outliers in previously obtained data. Therefore, the entire data set must be processed to identify any outlying records and thus the entire distance matrix storing the distance between every pair of records must be calculated and stored. This has running time $\boldsymbol{O(n^2)}$ with respect to the number of records, Knorr & Ng, (1998). The on-line technique matches new records against a stored data set to check whether the new record is an outlier. We simply store the data set and calculate the nearest neighbours for each query record at run-time with no prior processing. This has running time $\boldsymbol{O(mn)}$ where $\boldsymbol{m}$ is the number of queries executed and $\boldsymbol{n}$ the number of records stored.

For the retrospective standard k-NN technique, we read in the vectors from a file and calculate the ranges of all attributes. We then generate the data structure by storing each record and its 100 nearest neighbours as a C++ ordered list of <record identifier, Euclidean distance> pairs. The list data structure from the C++ Standard Template Library (STL) is a trade-off. It has higher memory requirements than the vector. Each list node has a pointer to the node in front and the one behind. Each node requires only 4 bytes with GNU g++ v2.95.3 using the Solaris8 OS. List nodes require 12 bytes under MS Visual C++6 and MS Windows2000. However, inserting into the list is cheaper than vectors as only pointers have to be redirected whereas the vector class has to move sections of the array during insertions and deletions which is comparatively more expensive. Converting the data structure from a vector to a list reduced the retrieval time by 0.5 seconds for the on-line k-NN approach (53.2 from 53.7) and would have reduced the $O(n^2)$ structure calculation as there are 100 times fewer calculations involved.

To determine the $\boldsymbol{k}$ nearest neighbour structure using the list class from the STL,

- For each record $\boldsymbol{x}$ in turn calculate the $\boldsymbol{k}$ nearest neighbours using the 2 steps below and store them in an ordered list in the k-NN data structure. Each list is $\boldsymbol{k}$ elements in length and holds the indices of the $\boldsymbol{k}$ nearest records and their respective distances from $\boldsymbol{x}$**,** sorted in ascending distance order.
    1. For every other record $\boldsymbol{y}$, calculate ***EuclidDist(x,y)*** as in Eq. 3.
    2. If ***EuclidDist(x,y)*** is less than the distance stored for the last element in the ordered list (the most distant current neighbour) then remove the last element in the list and add $\boldsymbol{y}$ to the list in its correct position with respect to distance.

We use range normalisation in our Euclidean distance calculation given in Eq. 3 to ensure that all attributes are in the range 0 to 1 and hence each attribute produces an equal weight in the Euclidean Distance calculation.

$$EuclidDist(x,y) = \sum_i \left(\frac{(x_i - y_i)}{range_i} \right)^2 \textit{ for all attributes } i \qquad \textbf{(3)}$$

We provide two timings for the k-NN data structure generation. We recorded the time to 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. We then implemented a speedup by exploiting the commutativity of distance and recorded the training time. Once we have calculated ***EuclidDist(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 and we **also** add ***x*** to the nearest neighbour list of ***y*** if it also falls in the top ***k*** neighbours. Essentially we only need calculate the half distance matrix where ***x < y*** using $\boldsymbol{n^2/2}$ calculations.

For the on-line technique described above where records are matched against the data set but no structure is created, we read in the vectors from a file, storing the ranges of each attribute and then calculate the nearest neighbours for each query vector in turn again storing the 100 nearest neighbours in an ordered list. We again use range normalisation in our Euclidean distance calculation, see Eq. 3.

3.2.2 AURA k-nearest neighbour

The first step for calculating the nearest neighbours using the AURA k-NN is to read in each record from the data set and train the record into the CMM as described in section 2.1. To retrieve the k-nearest neighbour candidate matches for the 3-Bits Set, we set the bit representing the bin for the query value and the two bits either side for each attribute and concatenate the attribute vectors. We input this vector to the CMM and threshold the output vector using L-Max thresholding described earlier to retrieve at least k matches (at least 100 matches in our evaluations). Once we have retrieved our set of candidate matches from the CMM, we post-process them using a standard Euclidean distance calculation and sort the records by ascending distance to identify the ***k*** nearest neighbours.

To retrieve the k-nearest neighbour candidate matches for the Integer Pyramid, we again identify the bit representing the bin for the query value for each attribute, superimpose the pyramid across the range of the attribute and concatenate the attribute vectors. We input this vector to the CMM and threshold the output vector using L-Max thresholding coupled with the Error Margin to control the number of candidate matches. Once we have retrieved and identified our set of candidate matches from the CMM, we post-process them using a standard Euclidean distance calculation and sort the records by ascending distance to identify the k-nearest neighbours.

3.3 Speed and Memory Evaluation

In section 4.1, we list the two training times to produce the standard nearest neighbour data structure ($\boldsymbol{O(n^2)}$ and $\boldsymbol{O(n^2/2)}$) and the memory overhead of the nearest neighbour table for the REAL and LR data sets. We also include the time to reread the k-NN structure from disk. Once the structure has been generated, the entire structure may be written to a file on disk and then on future runs only the structure need be read in from file rather than reading all records and calculating the structure from first principles.

We also list the time to train the AURA k-NN and the CMM's memory overhead for both the REAL and LR data sets in section 4.1 to allow comparison of the timings. The training time includes: the time to quantise each attribute, concatenate the quantised attributes to produce a CMM input vector and the time to train the concatenated input and the unique output vector into the CMM, $I_j \times O_j^T$ for all ***j*** associations. We note that the training time for the standard k-NN actually calculates the lists of nearest neighbours for all records whereas the CMM technique generates a matrix representation of the data set and the nearest neighbour lists are calculated after. We analyse the combination of training and retrieval in section 5.

3.4 Recall Accuracy and Speed

For this evaluation, we analyse and compare various configurations of each technique. The results of the comparisons are detailed in section 4.2. We compare the nearest neighbour lists for 200 query records using the two data sets REAL and LR. We felt recalling the nearest neighbours for 200 queries would provide a representative selection of queries to compare recall accuracies and produce a recall time with minimal variance. Timing the recall for a single record is likely to suffer perturbations due to operating system overheads and timing the full retrieval for all 200,000 records is very time consuming with so many configurations and data sets to analyse. We can calculate the average time for a single record from the 200 record time and extrapolate this to 200,000 records using the analyses in section 5.

3.4.1 Recall Accuracy

We extracted the nearest neighbours for the first 200 records for the conventional k-nearest neighbour and saved the two lists to a file, one list for REAL and one list for LR. These lists provide a benchmark to compare the respective nearest neighbour lists for the 3-Bits Set and Integer Pyramid techniques.

We vary the number of bins used for both 3-Bits Set and Integer Pyramid for the two data sets using 5,10,15,20,25,30,50,75,100, and 1000 bins for REAL and 5,10,15 bins for LR due to its lower attribute ranges**.** The number of bins alters the specificity of the matching by varying the number of records with equivalent scores and thus varies the number of candidate matches retrieved from the CMM at specific threshold values. The number of bins is inversely proportional to the number of values in each bin and hence the number of records that will map to each bin per attribute. This varies the granularity of the matching. For example, on the LR data set with just 5 bins 20% of the records will map to each bin for that attribute so 20% of the records will receive the top score for that attribute during retrieval. Increasing the number of bins to 15 means 6% of the records will map to each bin so only 6% of the records will receive the top score. 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. We note that the attribute ranges are identical for all attributes in each of the data sets so the number of bins will be identical for each attribute unless altered by the re-binning procedure described in section 2.1.1.

We trained the CMM and then recalled and listed the k-nearest neighbours identified by the 3-Bits Set approach for the first 200 records from the REAL and LR data sets for the various numbers of bins listed above. From these 13 lists of neighbours for 200 queries (three settings for LR and ten settings for REAL) we counted the number of nearest neighbours common to this list and the corresponding standard k-NN top 100 list for that data set. We averaged the number of matches retrieved by the 3-Bits Set approach for the 200 queries to give the average recall percentage for each of the thirteen configurations.

We repeated this comparison for the first 200 records for the various numbers of bins and the 2 data sets for the Integer Pyramid approach. The Integer Pyramid also allows variation of the **Error Margin** as described in section 2.2.1 for thresholding (number of candidate matches retrieved) due to the conjunction of finer-grained matching and L-Max thresholding unlike the coarser matching of the 3-Bits Set technique. We vary the parameter setting between 1 and 10 to retrieve between 100 and 1000 candidate matches respectively. For each configuration, we counted the number of nearest neighbours in the Integer Pyramid list also in the list produced by the standard k-NN approach. This produces a recall figure for each of the 200 queries which we summed and divided by 200 to ascertain the average recall accuracy across 200 queries.

The recall accuracy figures are listed in Table 4 & Table 5 for the REAL and LR data sets respectively. Fig. 10 & Fig.12 show the accuracy for the Integer Pyramid against the number of bins for the REAL and LR data sets respectively. Fig. 11 is a magnified version of Fig. 10 to allow differentiation of the plots at the higher end of the graph. The Integer Pyramid graph shows the Error Margin which effectively dictates the number of candidate matches that must be post-processed for all k nearest neighbours identified by the standard approach to be retrieved and hence forms a yardstick for the quantisation accuracy and scoring accuracy of the various parameter settings. Thus, we can find an optimal trade-off to retrieve sufficient candidate matches to achieve high accuracy while not retrieving too many to make the post-processing with conventional k nearest neighbour too slow.

3.4.2 Recall Speed

For each configuration of each system, we timed the recall for 100 nearest neighbours for 200 queries with 10 repeated runs and calculated the average retrieval time for 1 run. For the timing evaluation, all three algorithms were performing identical k-nearest neighbour calculations and were outputting identical data to the output files for equality. For the retrospective technique, we timed the standard k-NN retrieving the top 100 matches for 200 queries from the data structure and outputting the matches to a file. For the on-line approach, we timed the standard k-NN calculating the top 100 matches from first principles for the 200 queries and then outputting the matches to a file. We timed the various configurations of the 3-Bits Set and Integer Pyramid techniques. The retrieval time per query includes: the time to convert the query record to a CMM input vector; the time to recall the candidate matches; the time to calculate the Euclidean distance between the query record and all candidate matches retrieved; the time to sort the respective Euclidean distances for the candidate matches into order; and, the time to output the 100 nearest neighbours to a file on disk. The results are detailed in section 4.2, Table 4 & Table 5 and Fig. 8 & Fig. 9 .

3.5 Scalability

Using subsets of the REAL_2M data set with 400K, 800K, 1200K, 1600K and 2M records, we noted the training time and overall memory usage for both the standard and Integer Pyramid k-NN techniques for each data set size. All timings and memory usage statistics are listed in Table 6. Fig. 13 is a graph illustrating the scalability of the k-NN and Integer Pyramid for memory overhead as the data set grows from 400,000 to 2 million records.

Using the same data subsets, we retrieved the 100 nearest neighbours for the first 200 records for each data set, as previously, using the on-line k-NN approach and the Integer Pyramid (20 bins with Error Margin 10), timing each set of 200 retrievals. This permits an evaluation of the scalability with respect to retrieval time of the techniques and the timings are listed in Table 6 and shown in Fig. 14. We elected to use 20 bins for the Integer Pyramid with the REAL_2M data set as 20 bins proved optimal with respect to both accuracy and retrieval time for the REAL data set. We note that in the scalability test we fix the number of bins to note the increase in memory usage, training time and recall times. 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 a constant number of records mapping to each bin and hence a constant number of records scoring equivalent values.

Finally, we compared the nearest neighbour lists output by the on-line k-NN and the Integer Pyramid k-NN for each data set size to derive the recall accuracy for the Integer Pyramid at each data size. There is a trade-off, recall time against recall accuracy. The faster we retrieve the candidate matches, the lower the recall accuracy (see Table 5) so a suitable point has to be identifies that preserves sufficient accuracy yet provides a feasible run-time. This analysis validates how much we are maintaining recall accuracy with our quantisation and recall procedure with increasing data set sizes and whether the recall times are still feasible. The recall percentages are listed in Table 6 and pictured in Fig. 15.

4 Results

4.1 Speed and Memory Evaluation

In Table 1, we list the training times in seconds for the various algorithms. We include three training times for the standard k-NN, the time to calculate the retrospective *all k-NN* structure using an ***O(n^2)*** algorithm, the comparative time using an ***O(n^2/2)*** approach and the time to read the structure in from a file where the structure has been calculated previously and written out to a file. The training procedure for the 3-Bits Set and Integer Pyramid techniques is identical as the CMM stores the associations identically for both approaches so we only include an overall CMM training time for each bin size. The number of bins affects the training time. The size of the CMM increases systematically with bin size so the size of the input vector also increases proportionally and slows training.

In Table 2, we list the training times for the AURA k-NN with various bin sizes for the two data sets REAL and LR. By varying the number of bins, we alter the sizes of the respective input vectors and CMMs as more bins entail a larger input vector and hence larger CMM so this will effect the training time.

In Table 3, we list the memory overheads of the various algorithms in kilobytes. We include the size of the k-NN structure and the sizes of the CMM for various numbers of bins. Again, as both 3-Bits Set and Integer Pyramid use identical CMMs the memory overheads are the same but varying the number of bins varies the number of bits per attribute so the CMM size varies accordingly.

4.2 Recall Accuracy and Speed

The retrieval times for the 100 nearest neighbours for 200 queries with the standard on-line k-NN were **53.2** seconds for the REAL data set from first principles and **6.6** seconds for the LR data set from first principles. To retrieve the 100 nearest neighbours for 200 queries from the retrospective structure took **0.42** seconds for the REAL data set and **0.34** seconds for the LR data set. In Table 4 & Table 5, we note that the binary neural k-NN match is faster for all configurations than calculating the standard on-line k-NN from first principles.

Fig. 8 & Fig. 9 are the recall time graphs showing the variations in recall time with varying Error Margin (candidate matches) and number of bins for the REAL and LR data sets respectively. Both figures show the retrieval time for the standard k-NN for comparison to the various Integer Pyramid configurations.

Fig. 10, Fig. 11 & Fig. 12 are the recall graphs showing the variations in recall accuracy with varying Error Margin (candidate matches) and number of bins for the REAL and LR data sets respectively. Fig. 11 is a magnification of Fig. 10 to allow the graph to be viewed at higher definition and thus the subtle variations in recall accuracy discerned. We can identify all configurations that provide sufficient recall accuracy. In conjunction with Table 4 and Fig. 8 for REAL and Table 5 and Fig. 9 for LR, we can find any setting that produces an optimal trade-off. These settings retrieve sufficient candidate matches to achieve a high recall accuracy while not retrieving too many to make the overall retrieval slow due to the post-processing of too many candidates.

4.3 Scalability

Table 6 details the scalability timings and memory usage for the various techniques evaluated using the REAL-2M data set with between 400,000 and 2,000,000 records.

Fig. 13, Fig. 14 & Fig. 15 are graphical representations of the memory usage scalability for the retrospective k-NN and Integer Pyramid, the training time scalability for the retrospective k-NN and Integer Pyramid, the recall speed scalability for the on-line k-NN and Integer Pyramid and the recall accuracy scalability for the Integer Pyramid techniques with the REAL_2M data set.

5 Analysis

5.1 Speed and Memory Evaluation

The AURA approach (incorporating both 3-Bits Set and Integer Pyramid) is the fastest to train. It requires between 0.009 and 0.0015 of the time for the standard $\boldsymbol{O(n^2)}$ k-NN for the REAL data set and 0.02 for LR data set. It also requires between 0.0015 and 0.0025 of the time for the $\boldsymbol{O(n^2/2)}$ k-NN for the REAL and 0.032 for the LR data set. The AURA methodology simply has to quantise each attribute of each vector and train the appropriate bit in the CMM so for a data set comprising 200,000 vectors of 14 attributes there will be 2,800,000 quantisations and bit sets. In comparison, the standard k-NN is $\boldsymbol{\Theta(n^2m)}$ with respect to the number of vectors and attributes so this requires 200,000×200,000×14 which is 5.6×10^{11} distance calculations $\boldsymbol{(x_i - y_i)^2}$ and additions $\sum_i$ from Eq.3. In fact, the CMM training takes between 0.94 and 1.6 of the time required to read in a pre-prepared k-NN structure stored in a file for the REAL data set and 2.2 times for LR data set. We note that we did not include the time to read the data set (read in all vectors) from the file for the standard k-NN, the $\boldsymbol{O(n^2/2)}$ k-NN or the CMM. This is constant for all methods and takes approximately 17 seconds for the REAL and 1 second for the LR data set.

The AURA approach (incorporating both 3-Bits Set and Integer Pyramid) has the lowest memory overhead. The exact requirement varies with the number of bins as more bins necessitate more CMM rows so this increases the memory usage slightly as more lists have to be stored. For the REAL data set, the CMM uses between 0.047 and 0.067 of the memory overhead of the k-NN structure for 5 and 1000 bins respectively. For the LR data set, the CMM uses between 0.054 and 0.055 of the memory overhead of the k-NN structure for 5 and 15 bins. This huge reduction is due to the efficient storage employed by CMMs where rows are compacted as a list of integers with each integer denoting the location of each set bit in the row, Hodge (2001). We note slight variations in the memory usage for the CMM. As the number of bins increases the memory usage does not increase uniformly. The compacted CMMs are stored as arrays and each array is initially pre-allocated a specific amount of memory. If the number of positions to be stored exceeds the memory pre-allocation then a new block of memory is added for the list to expand. It is more efficient to add a reasonably large block (512 bytes) rather than simply adding 4 bytes to allow 1 extra integer to be stored as the new allocation is necessitated 128 times less often. Hence, the memory usage may include allocated memory where a whole block has not been used so this causes slight variation in the figures.

5.2 Recall Accuracy

The Integer Pyramid produces faster recall than the on-line k-NN for all configurations with both data sets. It demonstrates over 97.3% accuracy with between

10 and 1000 bins and Error Margin 6 to 10 with the REAL data set and is up to 3.6 times as fast as on-line k-NN. It achieves a peek accuracy of 99.9% with 20 bins and EM 10. It also has over 98.3% accuracy for 10 or 15 bins and Error Margin 6 to 10 with the LR data set and is up to 5 times faster. The peek accuracy is 99.2 % for 15 bins and EM 6 to 10.

However, the on-line k-NN does not require any training or storage whereas the CMM-based Integer Pyramid requires between 50 and 81 seconds to train the REAL data set and 11217 Kbytes of storage (for 20 bins). However, there is a number of records *x* where the Integer Pyramid becomes faster then the on-line k-NN where $50 + 17.2 \times \frac{x}{200} < 53.2 \times \frac{x}{200}$, i.e., $\boldsymbol{x} > 278$ records. To retrospectively identify the outliers in a data set requires the calculation of the nearest neighbours of all records. The on-line k-NN would need the same time as the $\boldsymbol{O(n^2/2)}$ k-NN structure which is 30221 seconds if it were optimised to exploit the commutativity of distance. CMM training takes 49 seconds and the time to retrieve the neighbours for 200 queries is 17.2 seconds with EM 10 to achieve 99.9% recall accuracy. Extrapolating this figure to the full REAL data set, it would take approximately $50 + 17.2 \times \frac{200000}{200}$ seconds to process all 200,000 queries which is 17250 seconds. If we could exploit the commutativity of distance shortcut we employed for the $\boldsymbol{O(n^2/2)}$ k-NN structure we could reduce this to $50 + 17.2 \times \frac{200000}{200 \times 2}$ which is 8650 seconds. Therefore, we feel the benefits of the Integer Pyramid outweigh the disadvantage of training and storage compared to the on-line k-NN unless an approach is needed just to calculate the nearest neighbours of only a few records (fewer than 278). We note that the on-line k-NN cannot exploit the commutativity for only a few records as only a small portion of the matrix is calculated so the $\boldsymbol{O(n^2)}$ approach is necessary.

From the empirical analysis, the Integer Pyramid has more consistent recall accuracy and speed than the 3-Bits Set with respect to the initial settings so has higher usability as the initial parameter settings are less critical. For the REAL data set, which provides a thorough examination of the consistency and accuracy of both the quantisation and scoring of binary methods, 36 of the 60 configurations of the Integer Pyramid have over 90% recall accuracy. The 3-Bits Set achieves a peek accuracy of 61.8%. For the LR data set, 12 of the 18 configurations of the Integer Pyramid exceed 90% accuracy. In contrast, only one of the 3-Bits Set configurations achieves over 90% accuracy.

Conversely, the 3-Bit Set is generally faster then the Integer Pyramid. The 3-Bits Set is an all or nothing approach with a very narrow focus of attention from an attribute perspective so only a few rows of the CMM are activated thus minimising recall processing. It produces its highest recall with fewest bins as this ensures that it selects the true matches on each attribute. The narrow focus produces only minimal scoring differentiation so it scores relatively few records and awards equivalent zero scores to many records. The lack of differentiation is accentuated in the recall and it retrieves many candidate matches during thresholding. An approximation of Euclidean Distance needs to examine the global picture, maintaining a wide focus of attention on each attribute and then narrowing the focus globally by selecting the k-nearest neighbours across the entire spread of all attributes. A potential best matching record may map to the third bin away from the query value on each attribute and the 3-Bits Set approach will inevitably omit this match due to the narrow focus.

The Integer Pyramid maps across the range of the attribute with a wide focus and then narrows the focus at the global level during the thresholding stage of retrieval. Integer Pyramid also allows finer-grained matching, as the attribute score is decremented proportional to the distance between the query value and the stored value. We can thus vary both the number of bins and the number of matches to achieve finer retrieval, allowing for the speed versus accuracy trade-off dependent on the processing time available. There is inevitably a speed-accuracy trade-off for quantisation. Increasing the speed necessitates post-processing less candidate matches which can serve to reduce the recall accuracy as less of the true k nearest neighbours according to the standard Euclidean-based k-NN are likely to be recalled in a smaller candidate set.

We can see from Fig. 10 that the recall accuracy is much lower for the Integer Pyramid with just 5 bins compared to the recall accuracies achieved with the other bin configurations for the REAL data set. This is due to the lack of differentiation between attribute values and records. 20% of the records will achieve the same score for each attribute so many records will have equivalent scores and there is insufficient spread to provide recall accuracy. We need enough bins to enable the distance scoring inherent in the Integer Pyramid to be effective and differentiate near neighbours. We ran the recall accuracy analysis with the REAL data set and 7 and 9 bins to verify this and the recall accuracy was higher for 7 than 5 bins and higher again for 9 bins with the plot for 9 bins lying just below the plot for 10 bins. These are not shown in Fig. 10.

Both the Integer Pyramid and 3-Bits Set need to retrieve a higher number of candidate matches than ***k*** to ensure a high accuracy. Quantisation inevitably loses some accuracy with respect to Euclidean distances when mapping numeric attributes into binned vectors. For example, the Integer Pyramid needs to recall approximately 400 matches for the REAL data set to achieve 95% recall accuracy with 20 bins. However, even though it recalls and post-processes four times the required number of matches, it is still faster than conventional on-line k-NN for all configurations. The Integer Pyramid needs to recall approximately 200 matches for the LR data set to achieve 97% recall accuracy with 15 bins.

For a fixed Error Margin, the exact number of candidates retrieved using the Integer Pyramid methodology is approximately inversely proportional to the number of bins. Increasing the number of bins increases the granularity of matching and decreases the number of records with equivalent scores thus decreasing the number of records recalled at each threshold value. For example, for Error Margin 6 and the REAL data set, the average number retrieved were 794 for 10 bins, 814 for 15 bins, 733 for 20 bins, 756 for 25 bins, 706 for 30 bins, 696 for 50 bins, 701 for 75 bins, 684 for 100 bins and 677 for 1000 bins. For Error Margin 6 and the LR data set, the average number retrieved were 680 for 10 bins and 675 for 15 bins.

The 3-BitsSet achieves highest accuracy with just 5 bins for both data sets due to the narrow focus of attention and recalls 4203 and 6644 candidate matches for the REAL and LR data sets respectively which is far higher than the Integer Pyramid even with the Error Margin set to 10. While the recall is high, conversely the speed of recall is slower than with the other 3-Bit Set configurations due to the large numbers of candidate matches processed.

If we take the Integer Pyramid results for the REAL data set with Error margin 10 and 20 bins the recall accuracy is 99.9%. For the 200 queries each recalling 100 neighbours, the Integer Pyramid omits 29 neighbours in total that should be retrieved according to the standard k-NN. However, these omissions are always at the tail of the neighbour list as can be seen in Fig. 16. For most applications, we feel this slight inaccuracy at the least critical end of the neighbour list is easily mitigated by the performance enhancements facilitated by the AURA techniques.

5.3 Scalability

The Integer Pyramid memory growth is slightly below linear ***<O(n)*** and thus the memory usage is progressively lower than the k-NN which has exactly linear growth ***Θ(n)***. The Integer Pyramid sub-linearity is due to the efficiencies exploited by the AURA library with rows stored as compacted lists. For 400K records, the k-NN requires 21.3 times the memory usage compared to the CMM. For 2 million records this has increased to 21.4 times.

The training time for the Integer Pyramid grows linearly ***O(n)***. The training time for the retrospective k-NN suffers quadratic growth $\boldsymbol{O(n^2/2)}$. The training time thus becomes virtually infeasible with 800,000 records requiring 564228 seconds or 6.53 days. However, we note that the training for the Integer Pyramid quantises the data whereas the training for the retrospective k-NN prepares the full nearest neighbour structure. The combined training and recall time for 800,000 records with the Integer Pyramid would extrapolate to $236 + (\frac{72.7}{200}) \times 800000 = 291036$ seconds. This is approximately half the time. If we extrapolate to 2 million records then the retrospective k-NN would need $(\frac{200000^2}{2}) \times 564228 \times \frac{2}{800000^2}$ which is 3,526,425 seconds in comparison to the Integer Pyramid preparation time of $757 + (\frac{193.8}{200}) \times 2000000$ which is 1,938,757 seconds. To process the full data set to determine outliers, the Integer Pyramid is twice as fast as the retrospective $\boldsymbol{O(n^2/2)}$ k-NN.

We note that the Integer Pyramid k-NN recall time is dependent on the number of neighbours (***k***) as this dictates the number of candidate matches to both retrieve and post-process. A lower value of ***k*** will speed the Integer Pyramid recall as fewer matches are recalled and post-processed. In contrast, this will not affect the standard k-NN as all records still have to be compared against all others to determine the ***k*** neighbours irrespective of the value of ***k***.

The Integer Pyramid recall speed growth is linear and the k-NN also has linear growth. This is due to the recall for both techniques depending purely on the number of records when the value of ***k*** is fixed and thus the recall time increases directly proportional to the number of records. The number of records in each bin increases in proportion to the number of records so the number of list locations stored in each CMM row will increase linearly thus recall will increase linearly. For the on-line k-NN the number of records to compare the query record increases directly proportional to the data size. However from Fig. 14, the gradient for the CMM growth is much lower than the k-NN growth as the recall time increments are much smaller as the data size grows for the Integer Pyramid recall.

The Integer Pyramid maintains the recall accuracy as the data size grows as the quantisation in conjunction with scoring is differentiating the records sufficiently.

6 Conclusion

We feel that using the AURA Integer Pyramid methodology to rapidly pre-select a small subset of candidate matches and then simply performing the standard Euclidean Distance calculation on this small subset is the most favourable approach for k-nearest neighbour classification and outlier detection. It outperforms the other methodologies evaluated for memory use coupled with training speed, retrieval speed and recall accuracy and is scalable with respect to both memory and recall time.

The Integer Pyramid methodology retrieves the nearest neighbours faster than conventional on-line k-NN across a wide range of configurations and its recall speed scales with ***O(n)*** growth. Even after factoring in the CMM training time, it outperforms the on-line k-NN recall for the REAL data set if more than 278 queries are performed (see discussion in section 5.2). It is much faster to train and has much lower storage overhead compared to the retrospective k-NN and again scales ***O(n)*** for memory usage. The training time for 800,000 records for $\boldsymbol{O(n^2/2)}$ retrospective k-NN is 564,288 seconds. The combined training time and recall time for Integer Pyramid is half of this value. For the large data sets, the Integer Pyramid is twice as fast as the standard k-NN when calculating the 100 nearest neighbours of all records in the data set. Any method employed will have $\boldsymbol{O(n^2)}$ growth as the number of records to compare and the number of records to compare each record against both grow at a rate of ***O(n)*** giving $\boldsymbol{O(n^2)}$ running time growth.

The Integer Pyramid displays a high consistency of recall accuracy and recall speed unlike the 3-Bits Set which is only accurate for low bin numbers when it is paradoxically slower. For example, any combination of bins between 10 and 1000 and Error Margin between 6 and 10 for the REAL data set will produce at least 97.3% accuracy and is faster than a standard on-line k-NN. This consistency improves usability as these two initial parameter settings are less critical. The user will not have to repeatedly run the Integer Pyramid with different parameter settings to find a suitable configuration. They can be relatively confident that a sensible selection will be nearly optimal.

The CMM-based approach is more flexible than the k-NN structure. If the user wished to change to retrieving 200 nearest neighbours, then the CMM would not need re-training. The CMM would remain identical for 200 neighbours so the memory usage would also not alter. In a similar vein, if a vector is added to or deleted from the data set or if a vector changes then only one column in the CMM has to be adjusted. It is a relatively simple matter to add entries to the row lists, to delete entries or to amend the list entries to reflect the change in the data set. In contrast, the k-NN structure would have to be re-calculated from first principles if the number of neighbours changes or the data set alters which would be very time consuming, requiring at least 33000 seconds for the ***$O(n^2/2)$*** technique and a 200,000 vector data set. If the number of neighbours increased from 100 to 200 then this would also double the memory overhead as the length of the neighbour lists in the structure would double.

We intend to use the AURA k-NN for future work involving classification and outlier detection in the fraud detection domain.

7 Acknowledgement

We would like to thank the reviewers for their helpful and insightful comments which allowed us to improve this paper.

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

8 References

Aleksander, I., & Albrow, R.C. (1968). Pattern recognition with Adaptive Logic Elements. *IEE Conference on Pattern Recognition*, pp 68-74.

Aleksander, I., Thomas, W.V., & Bowden, P.A. (1984). Wisard: A radical step forward in image recognition. *Sensor Review*, pp 120-124.

Alwis, S., & Austin, J. (1998). A Novel Architecture for Trademark Image Retrieval Systems. In, *Electronic Workshops in Computing*.

Austin, J. (1995). Distributed Associative Memories for High Speed Symbolic Reasoning. In, *IJCAI Working Notes of Workshop on Connectionist-Symbolic Integration: From Unified to Hybrid Approaches*, pp. 87-93.

Austin, J. (1998). RAM-based Neural Networks, *Progress in Neural Processing 9,* Singapore: World Scientific Pub. Co., ISBN: 9810232535

Austin, J., Kennedy, J., & Lees, K. (1995). A Neural Architecture for Fast Rule Matching. In, Artificial Neural Networks and Expert Systems Conference (ANNES'95), Dunedin, New Zealand.

Barnett, V., & Lewis, T. (1994). Outliers in Statistical Data, New York: J. Wiley & Sons, 3rd Edition, ISBN 0471930946.

Bishop, C. (1995). *Neural Networks for Pattern Recognition,* ISBN 0198538642.

Blake, C.L., & Merz, C.J. (1998). UCI Repository of machine learning databases, Dept. of Information and Computer Sciences, University of California, Irvine. Retrieved 8th August 2002 from *http://www.ics.uci.edu/~mlearn/MLRepository.html*.

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

Dasarathy, B. V. (1991). Nearest neighbor (NN) norms: NN pattern classification techniques; [ed. by] Belur V. Dasarathy. IEEE Computer Society tutorial, ISBN 0818689307.

Hodge, V. (2001). Integrating Information Retrieval and Neural Networks. PhD Thesis, Department of Computer Science, University of York, York, UK.

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

Hodge, V., & Austin, J. (2002). A Survey of Outlier Detection Techniques. Submitted to, *AI Review*: Kluwer.

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

Knorr, E., & Ng, R. (1998). Algorithms for Mining Distance-Based Outliers in Large Datasets. In, Proceedings 24th International Conference on Very Large Data Bases, {VLDB), pp. 392-403.

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

Turner, A. (2002.). Introduction to CMMs and AURA-Based Systems. Retrieved 8th August, 2002, from *http://www.cs.york.ac.uk/arch/NeuralNetworks/binary.html*

Turner, A., & Austin, J. (2000). Performance Evaluation of a fast Chemical Structure Matching Method using Distributed Neural Relaxation. In, *4th International conference on Knowledge Based Intelligent Engineering Systems*.

Turner, M., & Austin, J. (1997). Matching Performance of Binary Correlation Matrix Memories. *Neural Networks*, **10**(9), 1637-1648.

Wettscherek, D. (1994). A Study of Distance-Based Machine Learning Algorithms. PhD Thesis, Dept of Computer Science, Oregon State University.

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

9 Vitae

Dr. Victoria Hodge is a Research Associate in the Department of Computer Science, University of York. She is a member of the Advanced Computer Architecture Group investigating both the integration of neural networks and information retrieval focussing particularly on document retrieval from large corpora and also detecting anomalous records in large datasets.

Ken Lees is a Senior Research Associate in the Department of Computer Science, University of York. He is a member of the Advanced Computer Architecture Group, undertaking research in pattern matching and comparing methods for performing statistical edit and imputation tasks. He has contributed to the development of AURA and CMM methods since 1994.

Professor 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.

Figure Legends

Fig. 1 Diagram of a CMM with binary input vector *I* which indexes the rows of the correlation matrix and binary output vector *O* which indexes the columns of the matrix. Both vectors are applied to the CMM during training and matrix elements are set to 1 (binary) where the input vector (row) and output vector (column) are both set to 1.

Fig. 2 Diagram of the attribute→input vector bit mapping prior to setting any bits (left). Each attribute of the records in the data set maps to a set of bit positions within the input vector. The set of bit positions corresponds to the quantisation bins for that attribute; each bin maps to a separate bit position in the input vector. For a given record in the data set, the quantisation algorithm calculates the bin for each attribute value in the record in turn and sets the corresponding bit in the input vector (right).

Fig. 3 Diagram of the quantised input vector (left) with one bit set for each attribute value in the record. For the 3-Bits Set technique, the quantisation algorithm calculates the bin for each attribute value in the record in turn and sets the corresponding bit in the input vector as previously *but also sets the bit either side* (right). The spot indicates the bit corresponding to the bin for each attribute value.

Fig. 4 Diagram of the quantised input vector (left) with one bit set for each attribute value in the record. The quantisation algorithm calculates the bin for each attribute value in the record in turn and superimposes the Integer Pyramid on to the set of bit positions for that attribute (right). The Integer Pyramid has the highest value for the input vector position corresponding to the bin for each attribute value. The spot indicates this position for each attribute and the integers in the column indicate the Integer Pyramid scores.

Fig. 5. a) Diagram showing 15 sample attribute values for the first attribute of the REAL data set used in this paper divided into 5 bins using the quantisation technique described in section 2.1.1. 3 records map to each bin in a many-to-1 mapping shown as ►. Each bin maps to a particular bit in the input vector in a 1-to-1 mapping shown as ➪. b) Diagram showing the 3-Bits Set and Integer Pyramid input vectors for the first attribute of the query record with value 0.50 and then for query value 0.95.

Fig. 6 Diagram showing recall from a CMM for 3-Bit Set (a) and Integer Pyramid (b). For both methods, AURA applies the input vector to the CMM and activates the CMM rows where the input vector is non-zero with the input value (binary 1 for (a) 3-Bits Set and an integer value for (b) Integer Pyramid). AURA sums the CMM columns and retrieves an integer activation vector which effectively holds the sums of the columns. This integer output vector is thresholded with L-Max (L=2) to retrieve the top 2 matches and AURA can then identify the matching output vectors from the superimposed binary output vector. There is one bit set in the superimposed binary output vector for each matching record so by producing a separated binary vector for each bit set in the superimposed binary output vector, we can identify the matching records. In the diagram, the 4th and 6th records match for both (a) 3-Bit Set and (b) Integer Pyramid recall.

Fig. 7 Graph showing the number of records with particular attribute values (0-15) for each attribute (0-15) in the LR data set. Each attribute is clearly non-uniformly distributed.

Fig. 8 Graph of the recall time for the REAL data set with the Integer Pyramid technique showing the recall time variation against the number of candidate matches recalled (IP 100 to IP 1000) with the number of bins varying between 5 and 1000. The on-line k-NN retrieval time is shown for comparison as a flat surface at 53.2 seconds.

Fig. 9 Graph of the recall time for the LR data set with the Integer Pyramid technique showing the recall time variation against the number of candidate matches recalled (IP 100 to IP 1000) with the number of bins varying between 5 and 1000. The on-line k-NN retrieval time is shown for comparison as a flat surface at 6.6 seconds.

Fig. 10 Graph of the recall accuracy for the REAL data set with the Integer Pyramid technique showing the recall accuracy variation against the number of candidate matches recalled with the number of bins varying between 5 and 1000.

Fig. 11 Magnified version of Fig. 10 showing the recall accuracy variation against the number of candidate matches recalled with the number of bins varying between 10 and 1000.

Fig. 12 Graph of the recall accuracy for the LR data set with the Integer Pyramid technique showing the recall accuracy variation against the number of candidate matches recalled with the number of bins varying between 5 and 15.

Fig. 13 Graph showing the increase in memory usage of the data structure in the standard k-NN and the CMM in the Integer Pyramid techniques as the number of records in the data set increases from 400,000 to 2 million.

Fig. 14 Graph showing the increase in recall time of the standard k-NN and the Integer Pyramid technique as the number of records in the data set increases from 400,000 to 2 million.

Fig. 15 Graph showing the recall accuracy of the Integer Pyramid technique when the nearest neighbour lists are compared to those of the standard k-NN as the number of records in the data set increases from 400,000 to 2 million.

Fig. 16 Graph showing the positions in the list of 100 nearest neighbours returned by the standard k-NN approach of those neighbours omitted by the Integer Pyramid approach.

Tables

Data Set	k-NN O(n^2)	k-NN O($n^2/2$)	k-NN (read in)
REAL	52303	32966	51
LR	548	346	5

Table 1. Table showing the training times (seconds) for the three standard k-NN approaches (standard O(n^2), optimised O($n^2/2$) and reading a pre-prepared data structure in from a file) using the two data sets REAL and LR.

Data Set	AURA 5 Bins	AURA 10 Bins	AURA 15 Bins	AURA 20 Bins	AURA 25 Bins	AURA 30 Bins	AURA 50 Bins	AURA 75 Bins	AURA 100 Bins	AURA 1000 Bins
REAL	56	52	52	51	51	51	50	52	52	81
LR	11	11	11							

Table 2. Table showing the training times (seconds) for the binary neural k-NN approach for both the REAL and LR data sets with the number of bins ranging between 5 and 1000 and 5 and 15 for the REAL and LR data sets respectively.

Data Set	K-NN	AURA 5 Bins	AURA 10 Bins	AURA 15 Bins	AURA 20 Bins	AURA 25 Bins	AURA 30 Bins	AURA 50 Bins	AURA 75 Bins	AURA 100 Bins	AURA 1000 Bins
REAL	240000	11227	11271	11308	11217	11333	11234	11556	11420	11642	16072
LR	24000	1284	1296	1320							

Table 3. Table showing the memory requirements (Kilobytes) for the conventional k-NN data structure for both the REAL and LR data sets and the binary neural k-NN approach with the number of bins varied between 5 and 1000 and 5 and 15 for the two data sets respectively.

	3 BS		IP 100		IP 200		IP 400		IP 600		IP 800		IP 1000	
Bins	Recall Accuracy %	Recall Time Secs	Recall Accuracy %	Recall Time Secs	Recall Accuracy %	Recall Time Secs	Recall Accuracy %	Recall Time Secs	Recall Accuracy %	Recall Time Secs	Recall Accuracy %	Recall Time Secs	Recall Accuracy %	Recall Time Secs
5	61.3	10.1	44.8	9.6	59.8	9.8	74.4	10.2	81.4	10.5	86.2	10.9	88.8	11.3
10	23.5	6.9	61.8	13.9	80.6	14.2	93.8	14.4	97.6	14.8	99.0	15.1	99.5	15.5
15	18.0	5.9	57.5	14.6	79.1	14.7	93.3	14.9	97.5	15.1	98.9	15.6	99.5	15.8
20	12.7	5.1	59.9	15.9	81.0	16.0	95.3	16.4	98.5	16.6	99.5	16.9	99.9	17.2
25	9.5	4.7	56.8	16.0	79.1	16.2	93.8	16.6	97.9	16.7	99.3	17.1	99.7	17.4
30	7.3	4.2	57.8	16.6	80.1	16.8	94.7	17.2	98.4	17.4	99.5	17.6	99.8	17.9
50	4.0	3.4	55.7	17.5	78.7	17.6	94.2	18.1	98.2	18.2	99.4	18.5	99.8	18.7
75	4.2	3.3	63.1	17.7	79.2	17.8	93.4	18.0	97.8	18.3	99.3	18.6	99.8	18.9
100	2.2	2.9	53.9	17.9	77.1	18.1	93.3	18.5	97.9	18.6	99.3	18.9	99.7	19.3
1000	1.4	2.9	52.1	21.2	75.1	21.2	92.4	21.5	97.3	21.7	99.0	22.3	99.7	22.8

Table 4. Table listing the recall accuracy (percentage) and retrieval times (seconds) for the REAL data set k-NN recall from the 3-Bits Set (3BS) and the Integer Pyramid (IP) with Error Margin set to retrieve between 100 (IP 100) and 1000 (IP 1000) candidate matches. The figures shaded grey indicate where the Integer Pyramid is as least as accurate as 3-Bits Set with 5 bins (where 3-Bits Set has high accuracy). All timings for both 3-Bits Set and Integer Pyramid are faster than the on-line k-NN retrieval timing.

Bins	3 BS		IP 100		IP 200		IP 400		IP 600		IP 800		IP1000	
	Recall Accuracy %	Recall Time Secs	Recall Accuracy %	Recall Time Secs	Recall Accuracy %	Recall Time Secs	Recall Accuracy %	Recall Time Secs	Recall Accuracy %	Recall Time Secs	Recall Accuracy %	Recall Time Secs	Recall Accuracy %	Recall Time Secs
5	98.1	1.36	60.2	1.03	76.2	1.14	88.1	1.38	92.9	1.69	94.9	1.97	96.3	2.46
10	87.5	0.92	69.5	1.09	87.8	1.21	96.4	1.46	98.3	1.69	98.9	1.96	99.1	2.26
15	85.4	0.87	83.2	1.02	97.4	1.12	99.0	1.33	99.2	1.57	99.2	1.88	99.2	2.11

Table 5. Table listing the recall accuracy (percentage) and retrieval times (seconds) for the LR data set k-NN recall from the 3-Bits Set (3BS) and the Integer Pyramid (IP) with Error Margin set to retrieve between 100 (IP 100) and 1000 (IP 1000) candidate matches. The figures shaded grey indicate where the Integer Pyramid is as least as accurate as 3-Bits Set with 5 bins (where 3-Bits Set has high accuracy). All timings for both 3-Bits Set and Integer Pyramid are faster than the on-line k-NN retrieval timing.

Algorithm	Analysis	0.4M	0.8M	1.2M	1.6M	2.0M
Retro k-NN O(n^2/2)	Memory (kB)	480000	960000	1440000	1920000	2400000
	Training Time (secs)	137205	564228	*Too slow*	*Too slow*	*Too Slow*
On-line k-NN	Recall Time(secs)	60.6	213.0	319.3	425.4	531.3
Integer Pyramid 20 bins	Memory (kB)	22543	44909	67274	89638	112146
	Training Time (Secs)	106	236	389	561	757
	Recall Time(secs)	34.9	72.7	111.1	152.8	193.7
	Recall Accuracy %	99.7	99.8	99.8	99.8	99.9

Table 6. Table listing the memory overhead in kilobytes for the retrospective k-NN and the Integer Pyramid with Error Margin 10 and 20 bins, the training time in seconds for the retrospective k-NN and the Integer Pyramid, the recall time in seconds for the on-line k-NN and the Integer Pyramid and the recall accuracy (percentage) for the Integer Pyramid. We used the REAL_2M data set with 400,000 (0.4M), 800,000 (0.8M), 1,200,000 (1.2M), 1,600,000 (1.6M) and 2,000,000 (2.0M) records.

Figures

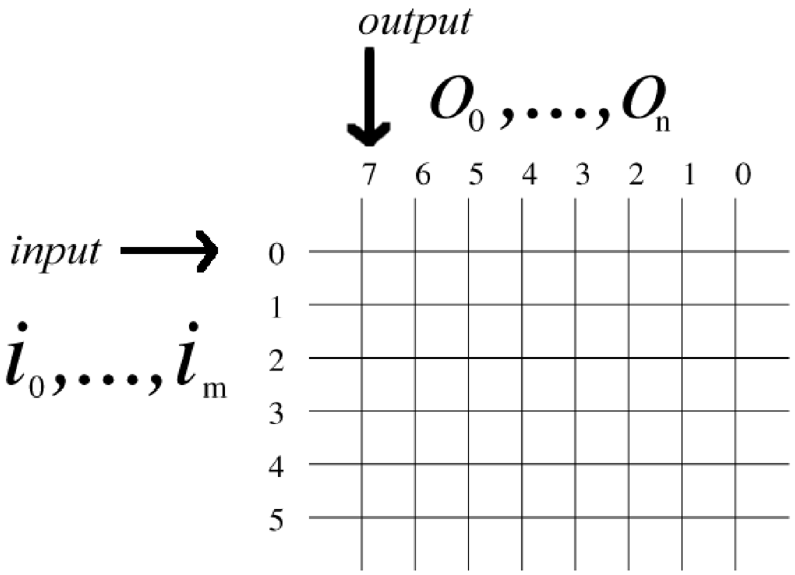

Fig. 1.

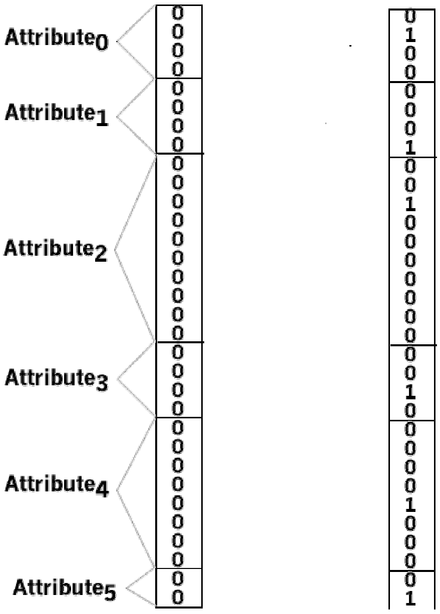

Fig. 2.

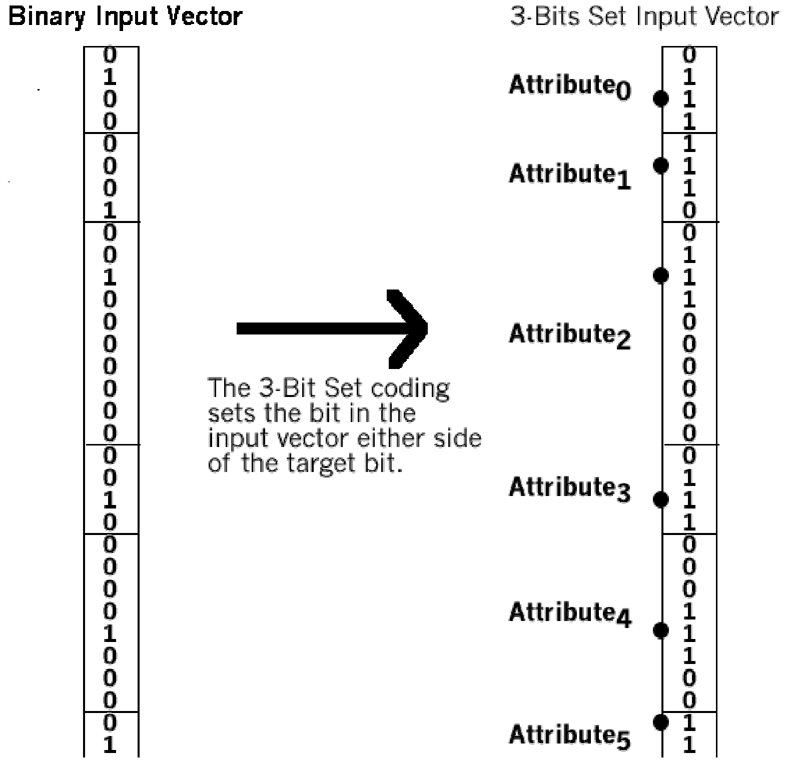
Binary Input Vector
0 1 0 0
0 0 0 1
0 0 1 0 0 0 0 0 0 0
0 0 1 0
0 0 0 0 1 0 0 0
0 1
3-Bits Set Input Vector
Attribute$_0$
0 1 1 1
Attribute$_1$
1 1 1 0
Attribute$_2$
0 1 1 1 0 0 0 0 0 0
Attribute$_3$
0 1 1 1
Attribute$_4$
0 0 0 1 1 1 0 0
Attribute$_5$
1 1
The 3-Bit Set coding sets the bit in the input vector either side of the target bit.

Fig. 3.

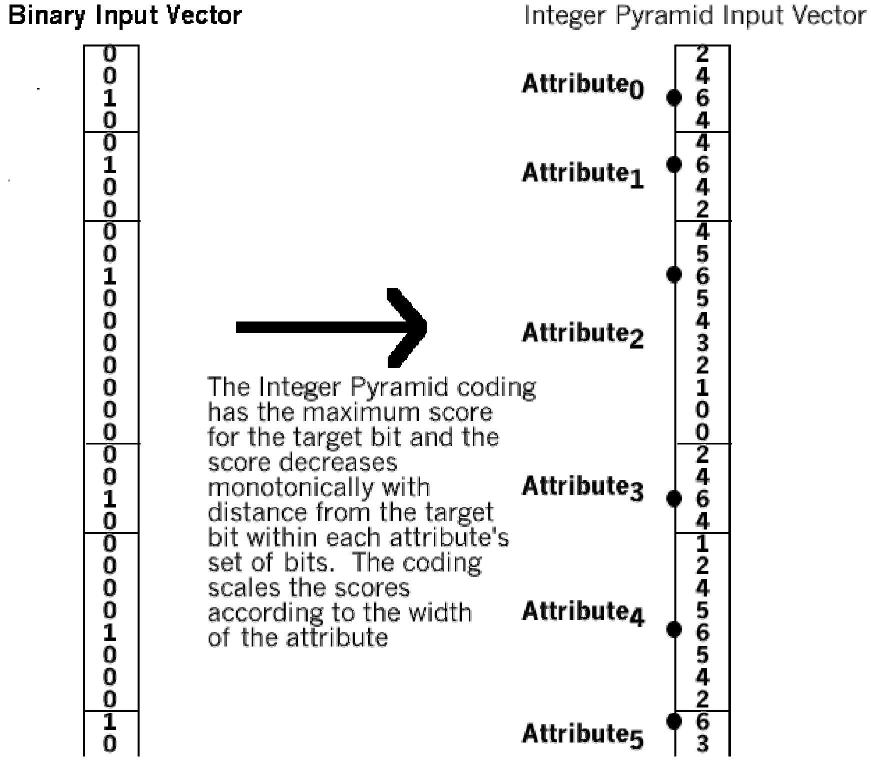
Binary Input Vector
0 0 1 0
0 1 0 0
0 0 1 0 0 0 0 0 0 0
0 0 1 0
0 0 0 0 1 0 0 0
1 0
Integer Pyramid Input Vector
Attribute$_0$
2 4 6 4
Attribute$_1$
4 6 4 2
Attribute$_2$
4 5 6 5 4 3 2 1 0 0
Attribute$_3$
2 4 6 4
Attribute$_4$
1 2 4 5 6 5 4 2
Attribute$_5$
6 3
The Integer Pyramid coding has the maximum score for the target bit and the score decreases monotonically with distance from the target bit within each attribute's set of bits. The coding scales the scores according to the width of the attribute

Fig. 4.

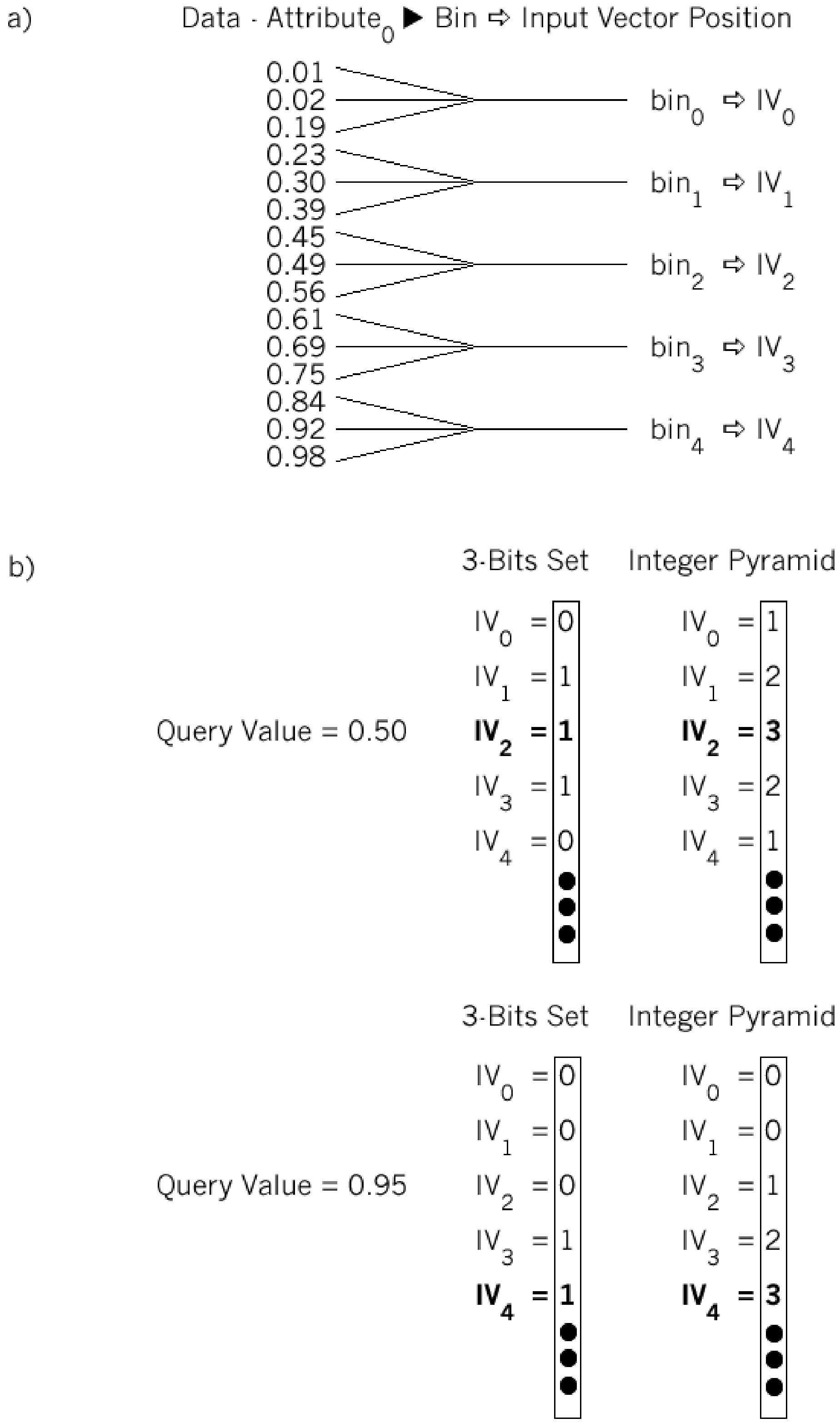
a)
Data - Attribute$_0$ ▶ Bin ⇨ Input Vector Position
0.01
0.02
0.19
0.23
0.30
0.39
0.45
0.49
0.56
0.61
0.69
0.75
0.84
0.92
0.98
bin$_0$ ⇨ IV$_0$
bin$_1$ ⇨ IV$_1$
bin$_2$ ⇨ IV$_2$
bin$_3$ ⇨ IV$_3$
bin$_4$ ⇨ IV$_4$
b)
3-Bits Set
Integer Pyramid
Query Value = 0.50
IV$_0$ = 0
IV$_1$ = 1
IV$_2$ = 1
IV$_3$ = 1
IV$_4$ = 0
IV$_0$ = 1
IV$_1$ = 2
IV$_2$ = 3
IV$_3$ = 2
IV$_4$ = 1
3-Bits Set
Integer Pyramid
Query Value = 0.95
IV$_0$ = 0
IV$_1$ = 0
IV$_2$ = 0
IV$_3$ = 1
IV$_4$ = 1
IV$_0$ = 0
IV$_1$ = 0
IV$_2$ = 1
IV$_3$ = 2
IV$_4$ = 3

Fig. 5

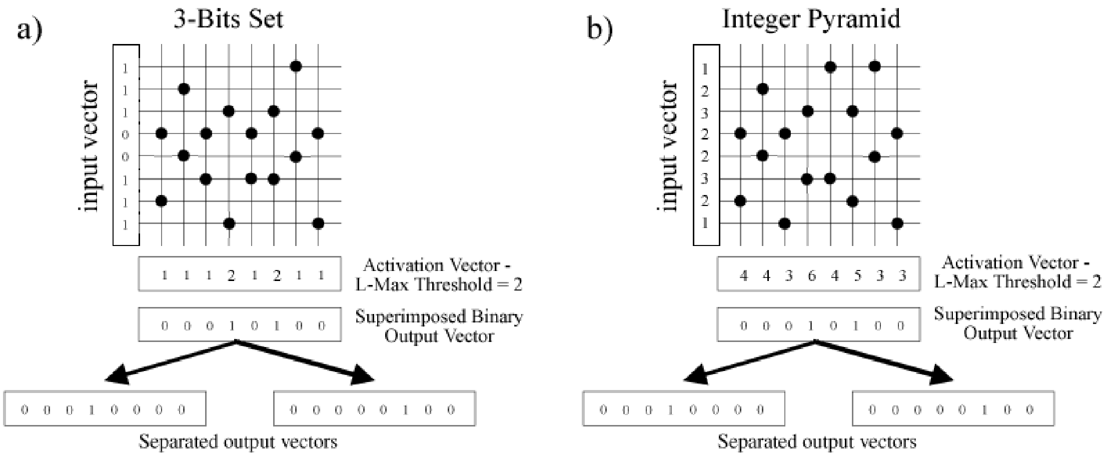
a)
3-Bits Set
input vector
1
1
1
0
0
1
1
1
1 1 1 2 1 2 1 1
Activation Vector - L-Max Threshold = 2
0 0 0 1 0 1 0 0
Superimposed Binary Output Vector
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
Separated output vectors
b)
Integer Pyramid
input vector
1
2
3
2
2
3
2
1
4 4 3 6 4 5 3 3
Activation Vector - L-Max Threshold = 2
0 0 0 1 0 1 0 0
Superimposed Binary Output Vector
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
Separated output vectors

Fig. 6.

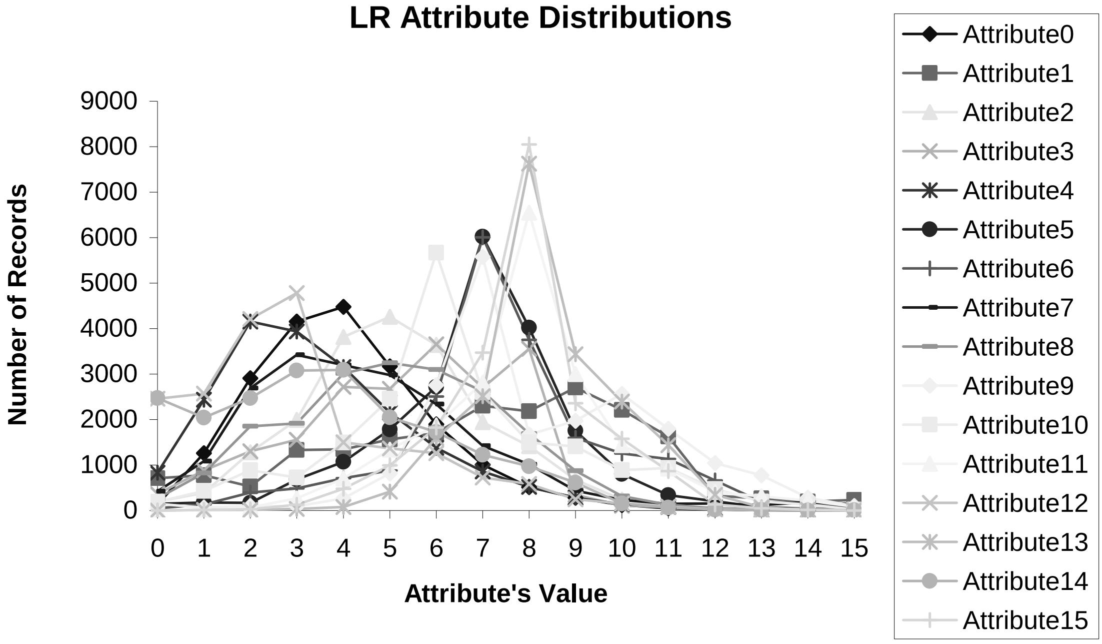
LR Attribute Distributions
Number of Records
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Attribute's Value
Attribute0
Attribute1
Attribute2
Attribute3
Attribute4
Attribute5
Attribute6
Attribute7
Attribute8
Attribute9
Attribute10
Attribute11
Attribute12
Attribute13
Attribute14
Attribute15

Fig. 7.

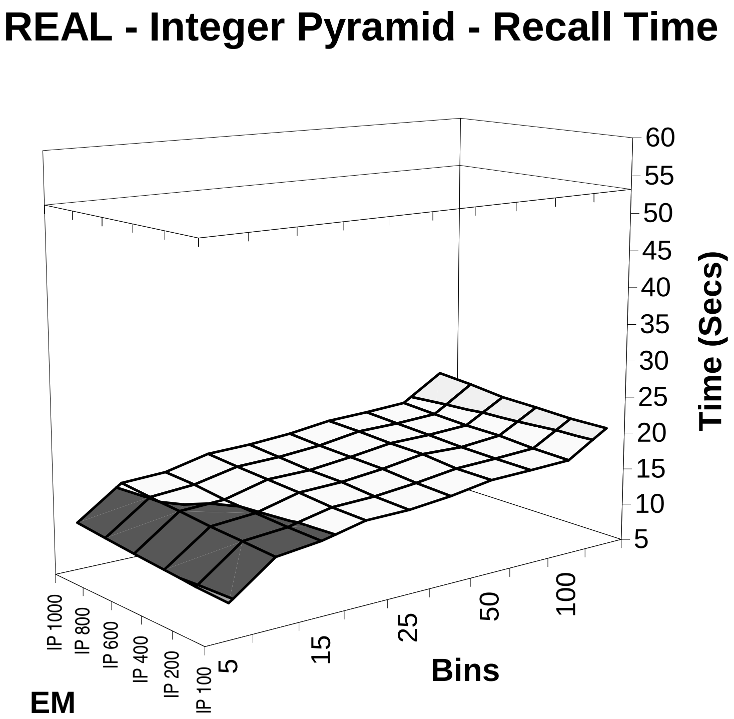
REAL - Integer Pyramid - Recall Time
60
55
50
45
40
35
30
25
20
15
10
5
Time (Secs)
IP 1000
IP 800
IP 600
IP 400
IP 200
IP 100
EM
5
15
25
50
100
Bins

Fig. 8.

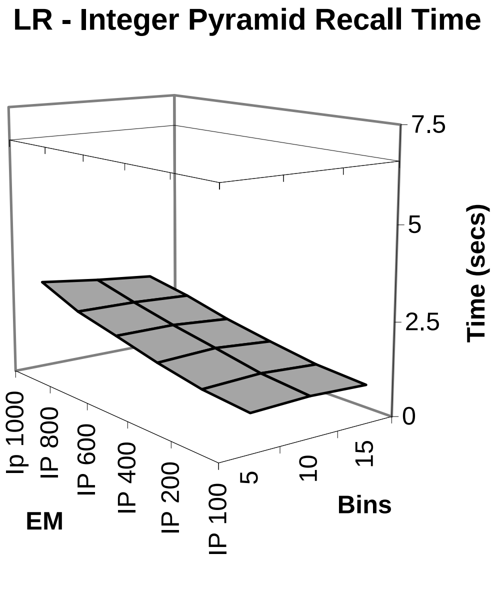
LR - Integer Pyramid Recall Time
7.5
5
2.5
0
Time (secs)
Ip 1000
IP 800
IP 600
IP 400
IP 200
IP 100
EM
5
10
15
Bins

Fig. 9.

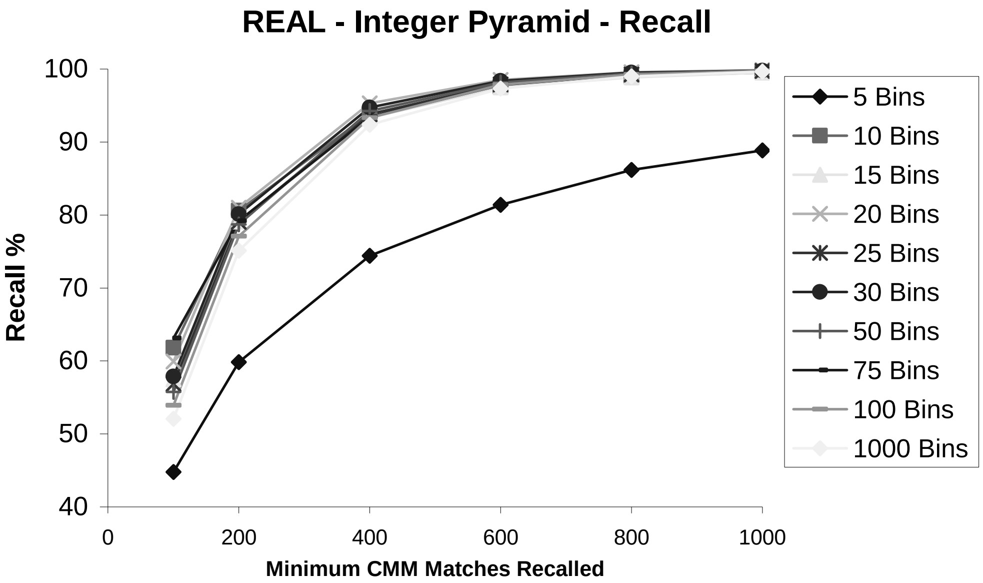
REAL - Integer Pyramid - Recall
Recall %
100
90
80
70
60
50
40
0
200
400
600
800
1000
Minimum CMM Matches Recalled
5 Bins
10 Bins
15 Bins
20 Bins
25 Bins
30 Bins
50 Bins
75 Bins
100 Bins
1000 Bins

Fig. 10.

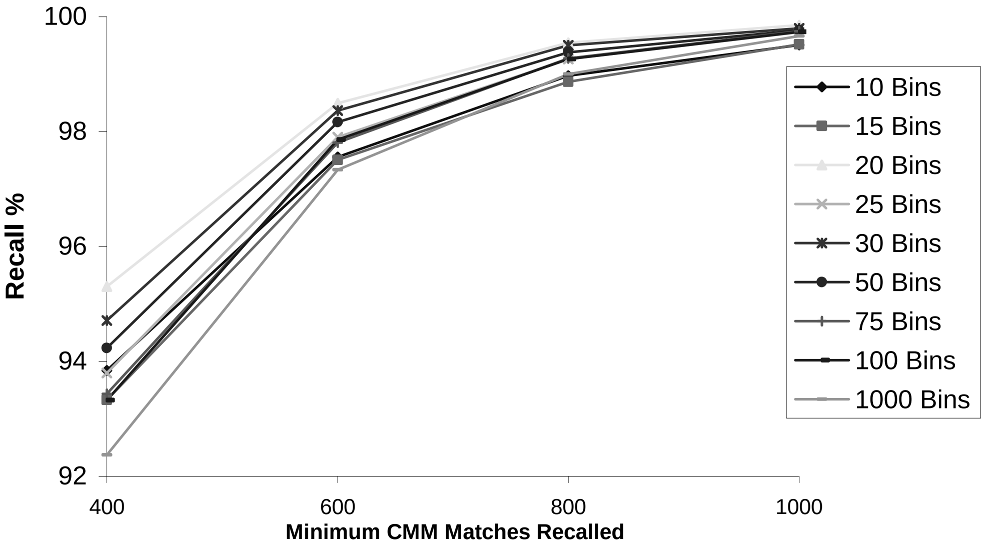
Recall %
100
98
96
94
92
400
600
800
1000
Minimum CMM Matches Recalled
10 Bins
15 Bins
20 Bins
25 Bins
30 Bins
50 Bins
75 Bins
100 Bins
1000 Bins

Fig. 11.

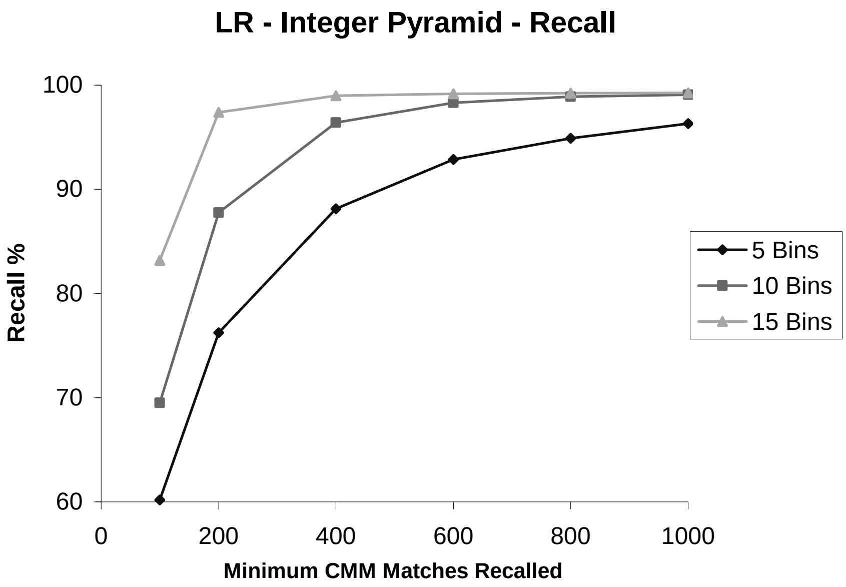
LR - Integer Pyramid - Recall
Recall %
100
90
80
70
60
0
200
400
600
800
1000
Minimum CMM Matches Recalled
5 Bins
10 Bins
15 Bins

Fig. 12.

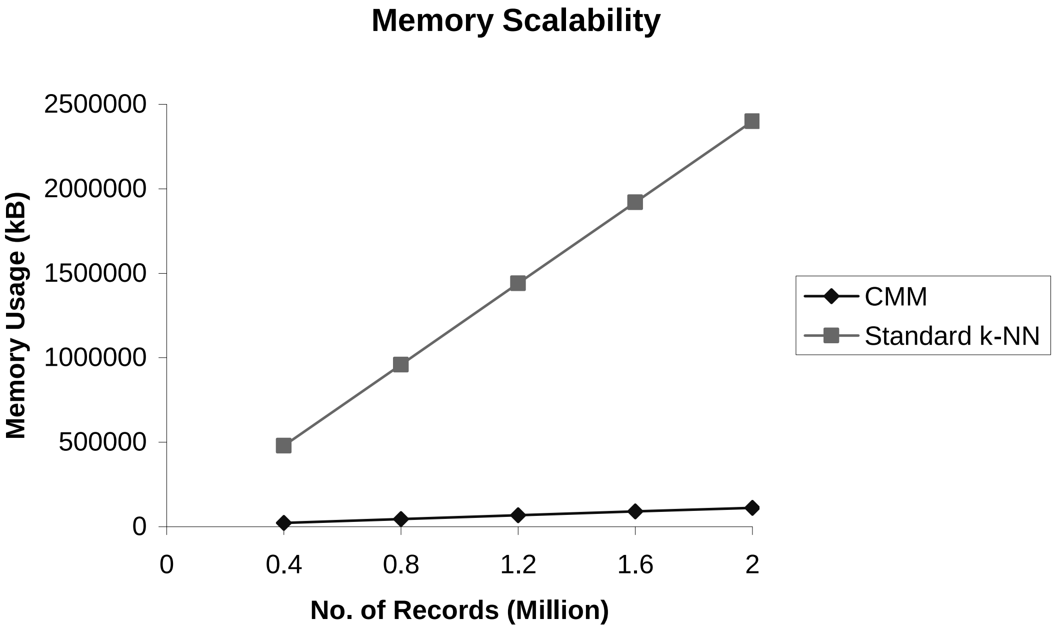
Memory Scalability
Memory Usage (kB)
2500000
2000000
1500000
1000000
500000
0
0
0.4
0.8
1.2
1.6
2
No. of Records (Million)
CMM
Standard k-NN

Fig. 13.

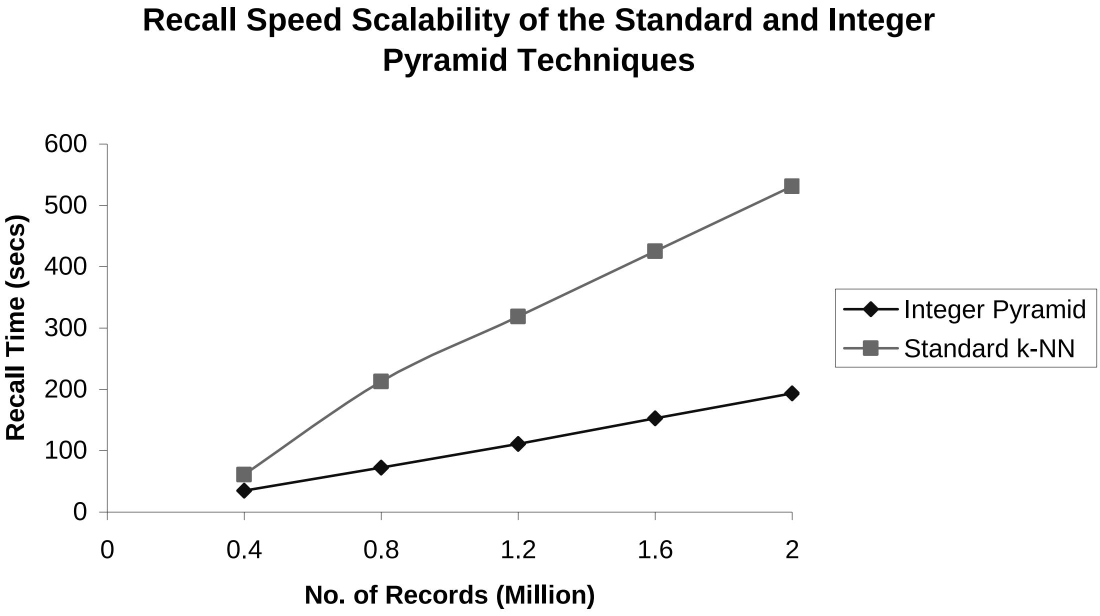
Recall Speed Scalability of the Standard and Integer Pyramid Techniques
Recall Time (secs)
600
500
400
300
200
100
0
0
0.4
0.8
1.2
1.6
2
No. of Records (Million)
Integer Pyramid
Standard k-NN

Fig. 14.

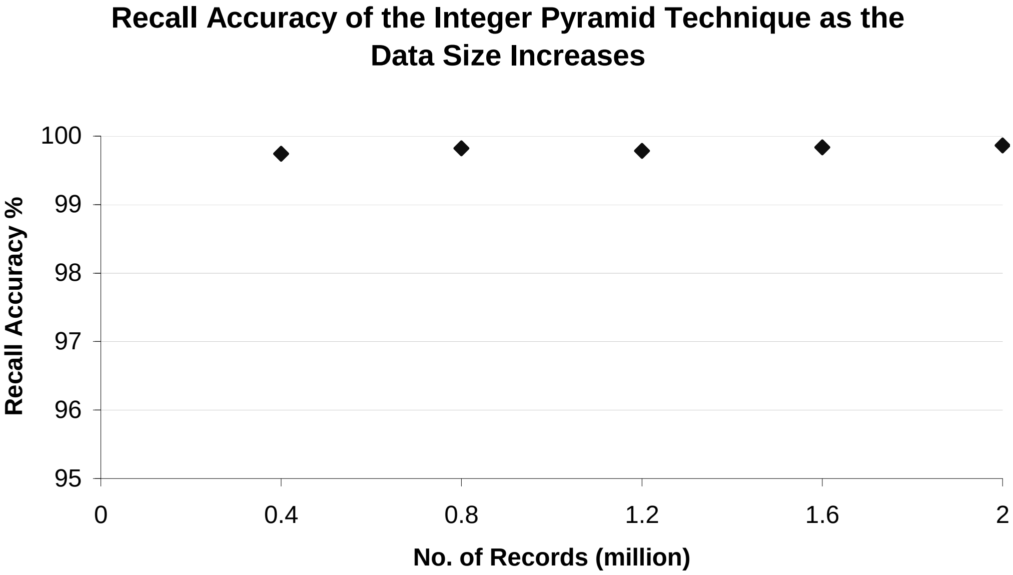
Recall Accuracy of the Integer Pyramid Technique as the Data Size Increases
Recall Accuracy %
100
99
98
97
96
95
0
0.4
0.8
1.2
1.6
2
No. of Records (million)

Fig. 15.

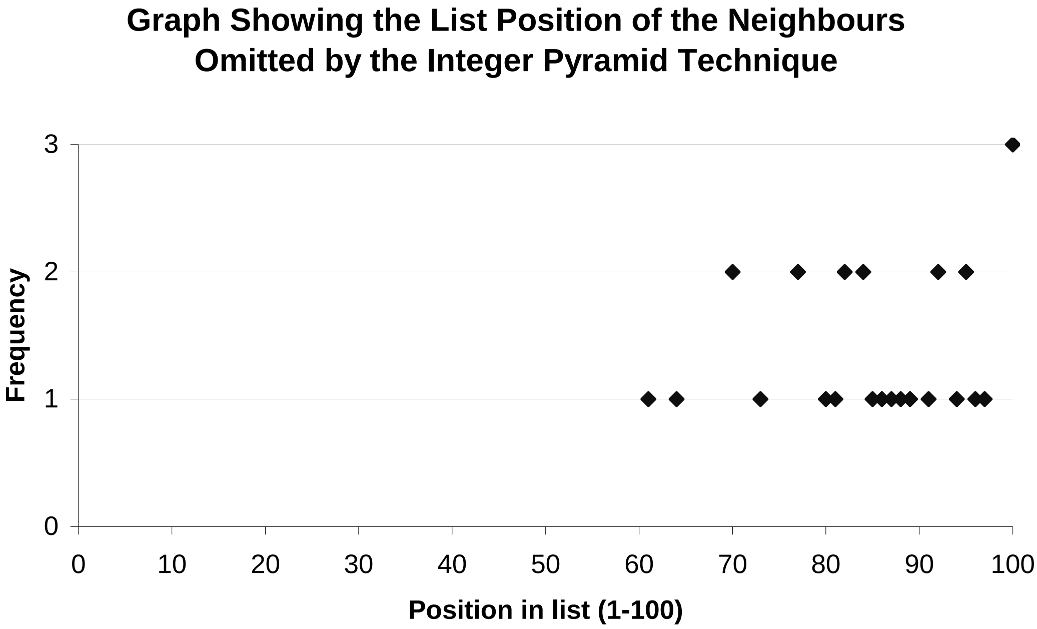
Graph Showing the List Position of the Neighbours Omitted by the Integer Pyramid Technique
Frequency
0
1
2
3
Position in list (1-100)
0
10
20
30
40
50
60
70
80
90
100

Fig. 16.