Improved AURA k-Nearest Neighbour Approach

Michael Weeks, Vicky Hodge, Simon O'Keefe, Jim Austin, and Ken Lees

Advanced Computer Architecture Group,
Computer Science Department,
University of York,
Heslington, York, UK
{mweeks,vicky,sok,austin,lees}@cs.york.ac.uk

Abstract. The k-Nearest Neighbour (kNN) approach is a widely-used technique for pattern classification. Ranked distance measurements to a known sample set determine the classification of unknown samples. Though effective, kNN, like most classification methods does not scale well with increased sample size. This is due to their being a relationship between the unknown query and every other sample in the data space. In order to make this operation scalable, we apply AURA to the kNN problem. AURA is a highly-scalable associative-memory based binary neural-network intended for high-speed approximate search and match operations on large unstructured datasets. Previous work has seen AURA methods applied to this problem as a scalable, but approximate kNN classifier. This paper continues this work by using AURA in conjunction with kernel-based input vectors, in order to create a fast scalable kNN classifier, whilst improving recall accuracy to levels similar to standard kNN implementations.

1 Introduction

Many pattern matching techniques, whilst effective, do not scale well with increased sample data space. The *k-Nearest Neighbour* (kNN) approach is one such method that is widely-used[1]. For each applied query, a distance measurement must be calculated between the query and every sample in the data space, and the data ranked to determine the queries neighbours. With a large set of known data, this can be restrictively slow.

In order to increase performance, we apply AURA to the kNN problem[7]. AURA is a highly-scalable associative-memory based binary neural-network, intended for high-speed approximate search and match operations on large unstructured datasets. It is typically used in large pattern-matching applications that are unsuited to conventional pattern-matching algorithms. Upon presentation of a query, AURA can rapidly process a large dataset, returning a smaller subset of approximate matches. This subset can then be rapidly processed using more traditional, computationally-intensive methods such as kNN. This two-stage approach is used in the FEDAURA project for fraud detection in large data sets [2]. Initial work on AURA kNN used "blurred" input vectors (3-bits set) to obtain a recalled subset of approximate matches[3]. Though fast and

scalable, it has been shown to be inaccurate. In order for the top k neighbours to be contained in the subset recalled from AURA, the subset must be large. In [3], we showed that the AURA kNN was faster than a standard C++ kNN implementation. Here we investigate the recall accuracy of various kernel shapes applied to the AURA input vectors with different data types and specifications.

2 kNN Classifier

Several distance metrics are used to measure the similarity of a query sample and all known samples in the data space. For accurate geometrical distance measurement, the **Euclidean** metric is used (see equation 1). However, since this must be calculated for all samples, this can require intensive computation with large data sets. The **City Block** metric simplifies measurement though introduces some error (equation 2).

$$d(X,Y)_{euc} = \sqrt{\sum_{i=1}^{n}(X_i - Y_i)^2} \tag{1}$$

$$d(X,Y)_{cb} = \sum_{i=1}^{n}|X_i - Y_i| \tag{2}$$

By plotting these distance equations about a query point in 2-dimensional space (figure 1) we can see the effect of both metrics. The query vector lies at the centre of the plots with a distance score of zero. All other data points are measured from the query point, radiating out linearly with increasing distance score.

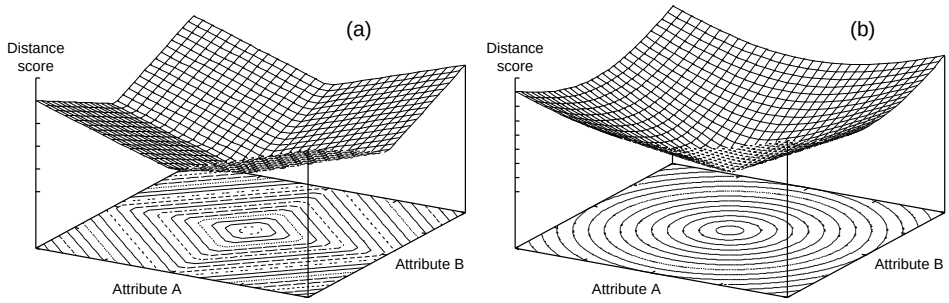

Fig. 1. Plots of (a) city block and (b) euclidean distance around a (central) point in 2D space

3 AURA

AURA (Advanced Uncertain Reasoning Architecture) is a family of techniques and implementations intended for high-speed approximate search and match operations on large unstructured datasets [4]. AURA technology is fast, scalable, economical, and offers unique advantages for finding near-matches not available with other methods.

AURA is based on a high-performance binary neural network called Correlation Matrix Memory (CMM). Typically, several CMM elements are used in combination to solve soft or fuzzy pattern-matching problems. AURA finds exact and near-matches between indexed records and a given query, where the query itself may have omissions and errors. The degree of nearness required during matching can be varied through thresholding techniques. AURA implicitly supports powerful combinatorial queries, which accept a match between, for example, any 5 from 10 fields in the query against the stored records.

A CMM consists of a binary correlation matrix neural network implemented in memory, for the storage and retrieval of vector patterns. Each column of the matrix is seen as a neuron, and each row represents an input and synapses to each neuron. The CMM concept has two main modes of operation; teach and recall. In teach mode, associated input and output binary vectors are trained into the CMM and its matrix weights (bits) are set. The training operation is expressed in equations 3 and 4, where M_0 is the initial empty CMM weights memory matrix, and M_k is the CMM matrix after k training operations. $\oplus$ denotes a logical OR operation, S_k^T is the transposed separator pattern and P_k is the input pattern. N is the number of patterns trained into the CMM.

$$M_0 = 0 \tag{3}$$

$$M_k = M_{k-1} \oplus S_k^T \cdot P_k, where k \in N \tag{4}$$

$$O = \sum M P_i^T \tag{5}$$

To perform a CMM recall, only the input binary vector is applied to the CMM matrix. Rows are selected by bits set in the input pattern, P_i, and columns in the CMM are summed to create an output integer vector, O (equation 5). The resulting summed column data can be read unprocessed, or thresholded to obtain the final binary vector containing the possible matches, S_k. When recalling, the input pattern can also be applied as a weighted (positive integer) vector. The integer input, I, is applied to a CMM row and any bits set in that row are added I times to their respective column count. Two types of thresholding can be applied (Willshaw or Lmax) dependent upon the application. Willshaw [5] thresholding compares the summed columns with a thresholded level, whilst Lmax [6] retrieves the top L or more matches from all of the summed columns. A detailed description of CMM neural networks can be found in [7].

CMM techniques lend themselves to such applications as inverted indices, whereby objects are stored in the CMM categorised by certain attributes. For attributes containing discrete, real, and categorical data, binning must be used to encode the data into the input vector. Each attribute is quantised over a range

of bins, with each bin represented as a row in the CMM. Multiple attributes can be encoded into an input vector by allocating a sub-vector to each attribute and binning each attribute within this range.

4 AURA kNN Classifier

Comparing the conventional kNN and the AURA-based approach, the latter calculates the k-nearest neighbours by traversing rows in a matrix. The standard kNN is similar to traversing columns in the same CMM with floating-point matrix entries rather than the binary entries of the AURA CMMs. The nested loop for standard on-line kNN 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)

Thus we can speed calculation using AURA as we only select specific rows (bins), so working with only a fraction of the data.

The input data is quantised and binned prior to its application to the CMM. The quantised data retrieved from the CMM is therefore an approximation of kNN. To compensate for this we retrieve a larger than k subset from AURA, then post-process this small batch using conventional kNN. Another problem with 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. A technique developed as part of the FEDAURA project [2] attempted to overcome this problem during recall by blurring the input vectors. When setting a bit in the input query to represent an attribute's bin, bits are also set for the two adjacent bins. Thus, we retrieve any values that lie just across the bin boundary and hence may be closer. This blurring of the input query proved to be fast, but inaccurate, and so for the top k-nearest neighbours to be recalled from AURA, the recalled subset must be large[3].

Figure 2(a) shows how distance scores are calculated around a central query point, for a binary-weighted block kernel (multiple-bits set) method in a 2-attribute (2-dimensional) problem. Note that the distance score at the query is at maximum, and a nearest-neighbour record is only valid if its distance score is greater than 1 (i.e. the top plateau). Records that return a score of 1 lie within one of the radial field weighting regions, and can only be said to be an approximate match. Similarly a record that scored '0' could be actually closer to the query than a record that lay at the extremities of the '1' region. Filtering out these uncertainties can be easily achieved by absolute (Willshaw) thresholding at level 2. For n-dimensional space using binary-weighted input vectors, equation 6 determines the threshold level at which valid records are returned.

Equation 6 can be expanded when input vectors are weighted, and assuming that every field's input vector shares a common maximum weight, w, then equation 7 applies.

$$threshold > n - 1 \tag{6}$$

$$threshold > (n - 1) \cdot w \tag{7}$$

In order to emulate kNN distance measurement using AURA, we decided to improve on the binary-weighted approach by applying a kernel-weighted integer input function to the query. For each attribute within the input vector query, we apply a quantised integer kernel function centred on the query's attribute bins. The summed intersection of these kernel projections contains stepped concentric patterns of equal scores. These scores, representing distance, will be at a maximum $(n \cdot w)$ at the query, and will radiate out into n-dimensional space until the limiting threshold is reached (equation 7). Two kernel shapes were selected: a triangular function that attempts to emulate the block-distance metric and a semi-circular function that attempts to emulate euclidean-distance measurement. In n-dimensional scenarios, the kernels produce an n-dimensional stepped hyper-diamond or stepped hyper-sphere respectively. Figures 2(b) and 2(c) plot score against attribute distance from a central (query) point, for both kernel methods in a two-dimensional scenario. It can be seen that when using the triangular kernel function, the intersecting scores, although stepped, compare well with the traditional kNN block-measurement approach (figure 1). Note that AURA-kNN, unlike traditional kNN, gives a maximum distance score at the query and reduces as we move away from the query. The semi-circular kernel method produces an intersection that approximates to euclidean distance, however, the distance to score stepping is non-linear. This non-linearity creates poor resolution nearest the query point. For thorough evaluation we set the value of

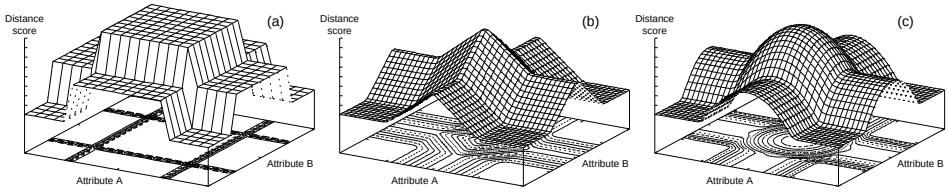

Fig. 2. Plot of AURA distance scores around a (central) point in 2D space using (a) block (multi-bits set), (b) triangular, and (c) semi-circular kernel functions

$k = 99$. We vary the number of candidate matches retrieved by AURA using a parameter we call the $ErrorMargin(EM)$, which is effectively a multiple of k. AURA reduces the L-Max threshold value until it has recalled at least $EM * k$ matches. Throughout our evaluation we use an EM of 10 which retrieves at

least 990 candidate matches. The 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 k nearest neighbours as defined by standard Euclidean kNN but increases the retrieval time as more candidate matches have to be post-processed.

4.1 Kernels

We evaluate 3 kernel shapes in this paper; Triangle, Semi-Circle, and a blurred (or 3-Bits Set) kernel. In the following, n_i is the number of bins for attribute i, $max(n)$ is the maximum number of bins across all attributes and α_i is the scale factor to ensure all attributes have the same potential maximum score so all attributes are ascribed an equal weight.

Triangle Has a linear decrement and is equivalent to quantised City Block Distance, see equation 8.

$$Triangle(x,y) = \sum_i \left[\frac{max(n)}{2} - (x_i - y_i)\alpha_i \right] \text{ where } \alpha_i = \frac{max(n)}{n_i} \quad (8)$$

Semi-Circle Has a quadratic decrement, and is equivalent to quantised Euclidean Distance, see equation 9[1].

$$SemiCircle(x,y) = \sum_i \left[\left(\frac{max(n)}{2} \right)^2 - (x_i - y_i)^2\alpha_i \right] \text{ where } \alpha_i = \frac{max(n)^2}{n_i^2} \quad (9)$$

3-Bits Set Cheapest - only activate 3 lines of CMM per attribute, no control required, tendency to retrieve too many equal-valued matches which all have to be post-processed so overall retrieval can be slow.

5 Evaluation

We implemented the standard kNN that forms the baseline comparator using C++. We employed range normalisation in the Euclidean Distance calculation (see equation 10) to ensure all attributes were ascribed equal weights as all attributes have equal weight in the AURA evaluation. We note that the weights can be varied in both approaches if desired to reflect attribute weights. We omitted the square root for speed as this does not affect the order of the nearest neighbours. $range_i$ denotes the range of data for attribute i.

$$EuclidDist(x,y) = \sum_i (\frac{x_i - y_i}{range_i})^2 \text{ for all i attributes} \quad (10)$$

[1] Derived from discussions with Bojian Liang and Garry Hollier

The k nearest neighbours for vector i are the k records with the lowest $EuclidDist(x, y)$. This equates to the inverse of the Semi-Circle. AURA retrieved the nearest neighbours as those with the highest score but conversely, standard kNN retrieves the records with the lowest score.

For our evaluation we use three data sets:

LR The Letter Recognition data set from the UCI data repository [8]. The data comprises 20,000 vectors of 14 randomly-distributed, integer-valued attributes ranging from 0-15.

REAL A data set containing 200,000 vectors of 14 real-valued attributes with values between 0 and 1. The data set was generated using the Java random number generator.

IBM A data set containing 20,000 vectors of 9 integer-valued attributes with ranges between 0-4 and 0-1,349,600. This data set was generated using the IBM data set generator [9] with standard settings. Note that we removed the final class attribute from the data.

For the purposes of our evaluation here, we are only interested in the recall accuracy of binning continuous or discrete attributes combined with the kernel shape input vector scores. Thus, we treat all attributes from all three data sets as continuous or discrete and subdivide the ranges into a series of bins.

We use 10 bins for the LR data set (LR_10) as the range of values in each attribute is small. We use a higher number of bins (25) for the REAL data set (REAL_25) as the attributes are continuously valued between 0 and 1 so a higher degree of binning accuracy is desirable. For the final IBM data set, there is a disparity in the attribute ranges so we analyse two setting of 10 and 25 bins (IBM_10 & IBM_25) to investigate which performs best when attributes ranges differ between 4 and 1,349,600.

We compare the top 99 matches retrieved by each of the kernel shapes across the four settings (LR_10, REAL_25, IBM_10, IBM_25) against the 99 top matches retrieved by a standard kNN technique implemented using C++. This will verify that the AURA pre-processing step which effectively minimises the search space is accurate. Post-processing of the top matches ensures the nearest neighbour orders will be identical. Therefore we are interested only in the number of neighbours in the top 99 as their orders will be identical.

6 Results and analysis

In table 1 we list the recall accuracy percentages for the various kernel shape and data set combinations. The Semi-circle kernel is most accurate, closely followed by the Triangle. The original 3-Bits Set kernel performs poorly. The Triangle and the Semi-Circle are sufficiently accurate (over 99% recall accuracy) for most application domains. The difference in the recall accuracy between these two is negligible, though it is expected that the Semi-Circle kernel approach will prove more accurate as we reduce the size of the recalled subset.

	LR_10	REAL_25	IBM_10	IBM_25
Triangle	99.49	99.95	100.00	100.00
Semi-Circle	99.57	100.00	100.00	100.00
3-Bits Set	89.27	17.01	63.51	52.33

Table 1. Table listing the recall accuracy (percentage) for the three kernel shapes using the three data sets. There are two settings for the IBM data set for comparison.

We note that the records missed by the Triangle and Semi-Circle kernels are always at the tail of the neighbour list. 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 [3].

7 Conclusions

Previous work [3] has shown the AURA-knn approach to be faster than the standard computational approach. This is achieved by using AURA as a coarse filter to rapidly extract a subset of records from the data space, then processing this smaller, manageable subset with traditional methods. In this paper we have improved on this approach by dramatically increasing the accuracy of the recalled AURA subset, which equates to a smaller recalled subset and reduced post processing requirements.

References

1. D.Wettscherek. A Study of Distance-Based Machine Learning Algorithms. PhD Thesis, Dept of Computer Science, Oregon State University, 1994
2. Fedaura Project, Advanced Computer Architecture Group, Computer Science Department, University of York, UK. http://www.cs.york.ac.uk/fedaura/
3. V. Hodge & J. Austin. A High Performance k-NN Approach using Binary Neural Networks. Submitted to, *Neural Networks*, Elsevier Science, 2002.
4. Jim Austin, John Kennedy, and Ken Lees. The Advanced Uncertain Reasoning Architecture. In *Weightless Neural Network Workshop*, 1995.
5. D.J.Willshaw, O.P.Buneman, H.C.Longuet-Higgins. Non-Holographic Associative Memory. Nature 222(1969), 960-962.
6. D.P.Casasent and B.A.Telfer. High Capacity Pattern Recognition Associative processors. Neural Networks 5(4), 251-261, 1992.
7. Jim Austin. Distributive Associative Memories for High Speed Symbolic Reasoning. Int. J. Fuzzy Sets Systems, 82, 223-233, 1996.
8. IBM. Quest: Data Mining Quest Synthetic Data Generation Code (2) Classification: http://www.almaden.ibm.com/cs/quest/syndata.html
9. C.L. Blake and C.J. Merz. UCI Repository of machine learning databases, Dept. of Information and Computer Sciences, University of California, Irvine, 1998 http://www.ics.uci.edu/ mlearn/MLRepository.html

Supported by EPSRC grant number GR/R55191/01.