A Binary Correlation Matrix Memory $k$-NN Classifier

Ping Zhou and Jim Austin

Department of Computer Science, University of York, York Y01 5DD, UK
Email: zhoup@cs.york.ac.uk, austin@cs.york.ac.uk, Fax: +44 (0) 1904 432767

Abstract

In this work we investigate the use of a CMM (Correlation Matrix Memory) neural network for pattern classification. It is known that the $k$-NN rule is applicable to a wide range of classification problems but it is slow, and that CMM is simple and quick to train, and has highly flexible and fast search ability. We combine the two techniques to obtain a generic and fast classifier which uses a CMM for storing and matching a large amount of patterns efficiently, and the $k$-NN rule for classification. To meet requirements of the CMM, a robust encoder has been developed to convert numerical inputs into binary ones with the maximally achievable uniformity. Experiment results on several benchmarks show our method can be over 4 times faster than a simple $k$-NN classifier with less than 1% lower classification accuracy.

1 Introduction

Desirable characteristics of Correlation Matrix Memory (CMM) neural networks include simple and quick training, and highly flexible and fast search ability [2]. Whereas most neural networks need a long iterative training times, a CMM is trained using an one-shot storage mechanism and simple binary operations. This leads to software implementation and hardware solutions for fast search [2]. CMM has been used as a match engine in a number of successful applications, e.g., symbolic reasoning [3], chemical structure match [4] and post code matching. This work investigates its use for pattern classification tasks. It is known that the $k$-NN rule [5] is applicable to a wide range of classification problems. However, this method is too slow to use for many applications with large amounts of data. To speed up, previous researchers have considered reducing training data [6] and improving computational efficiency via complex pre-processing of training data [7]. In contrast to these, a CMM is a simple, general and powerful approach which can be used to store a large number of training patterns efficiently, and to retrieve both exact and near matches quickly for a test pattern. Therefore, the combination of CMM and $k$-NN techniques may result in a generic and fast classifier.

For most classification problems, patterns are in the form of multi-dimensional real numbers, and appropriate quantisation and encoding are needed to convert them into binary inputs to CMM. A robust quantisation and encoding method is developed to meet requirements for CMM input codes, such as uniformity, orthogonality and sparseness [1], and to overcome the problem of identical data points in data, e.g.,
background of images or normal features in a diagnostic problem.

The next section discusses the CMM technique for pattern classification and the robust uniform (RU) encoding. Four benchmarks and experiment results are given in Section 3, followed by concluding remarks in the last section.

2 CMM classifier

Figure 1 shows the architecture of the CMM classifier. The RU encoder (as detailed in 2.2) quantises numerical inputs and generates binary codes; the CMM engine stores training patterns and selects stored patterns close to a test pattern to supply to a conventional k-NN module for classification.

Figure 1: Architecture of the CMM classifier

2.1 Pattern match and classification with CMM

In the CMM there is a binary matrix $M$ and, prior to any learning, all of its elements are set to ‘0’. In a training process an unique binary vector (or separator as often called) $s_i$ is generated to label an unseen input binary vector $p_i$; the CMM learns through the association of the two vectors by performing the following logical ORing operation,

$$M = \bigcup_i s_i p_i$$  \hspace{1cm} (1)

In a recall process, for a given test input vector $p_k$, the CMM performs,

$$v_k = M p_k^T = \bigcup_i s_i p_i p_k^T$$  \hspace{1cm} (2)

followed by a threshold $n_f$ applied to $v_k$, that is some proportion of the bit density of $v_k$ [1]. A $L$-Max thresholding method [8] is often used for the thresholding operation, in which the $L$ maximal values in $v_k$ are set to ‘1’ and the others to ‘0’. This may be repeated until the required number of matches is obtained. For speed, it is appropriate to threshold $v_k$ only once at a certain level. Individual separators are recovered from threshold $v_k$ using the MBI (Mid Bit Index) method [2].
When $p_k$ is an 'seen' or stored pattern, Equation 2 can be written as,

$$v_k = \sum_{i \neq k} p_i F_i + \|p_k F_k\|^2 = n_p F_k + a_k$$  \hspace{1cm} (3)

$a_k$ in the above will disappear to give a perfect recall of $s_k$ (scaled by a constant $n_p$) if two different patterns $p_i$ and $p_k$ for $i \neq k$ are orthogonal. The orthogonality requirement is often met by an encoder. In practice 'partially orthogonal' codes may be used to increase the storage capacity of CMM, and $s_k$ is obtained by appropriately thresholding the summed vector $v_k$. Sparse codes are usually used, i.e., only a few bits in separators and input vectors being set to ‘1’, as this maximises the number of codes and minimises the computation time [1].

CMM exhibits an interesting 'partial match' property when the data dimensionality $d$ is larger than one and input vector $p$ consists of $d$ concatenated components. We have $p_i F_k = 0$ in Equation 3 for $i \neq k$ if the two different patterns $p_i$ and $p_k$ have some common components. Therefore, $v_k$ also contains separators for partially matched patterns, and these separators can be obtained at lower threshold levels, $n_p$. This partial or near match property is useful for pattern classification as stored patterns close to the test pattern in the Hamming distance can be retrieved.

The $k$-$NN$ module classifies a test pattern from matched training patterns using the $k$-$NN$ rule. Distances are computed in the original input space to minimise the information loss due to quantisation and noises in the above match process. As the number of matches is much smaller than the training data size, the distance computation and comparison are dramatically reduced comparing with the simple $k$-$NN$ method. Therefore, the CMM classifier can be faster.

### 2.2 Robust uniform encoder

In addition to the above sparseness and orthogonality, another primary requirement for CMM input codes is that they should be distributed as uniformly as possible in order to avoid some parts of CMM being used heavily while the others rarely used. Figure 2 shows three stages of the encoding process, that is quantising $d$-dimensional real numbers, $x_i$, generating sparse and orthogonal binary vectors, $c_i$, and concatenating them to form a CMM input vector.

![Figure 2 Quantisation, code generation and concatenation](image)

The code uniformity is met at the quantisation stage. For a given set of $N$ training samples in some dimension (or axis), it is required to divide the axis into $N_b$ small intervals, called bins, such that they contain uniform numbers of data points. As the
data often have a non-uniform distribution, the sizes of these bins should be different. It is also quite common for real world problems that many data points are identical, for instance, there are 11%-99.9% identical ones in benchmarks used in this work. Our robust quantisation (RQ) method described below is designed to cope with the above problems and to achieve a maximal uniformity.

In our method data points are first sorted in ascending order, $N_i$ identical points are then identified, and the number of non-identical data points in each bin is estimated as $N_p = (N - N_i)/N_b$. Bin boundaries or partitions are determined as follows. The right boundary of a bin is initially set to the next $N_p$ \textsuperscript{th} data point in the ordered data sequence; the number of identical points on both sides of the boundary is identified; these are either included in the current or next bin. If the number of non-identical data points in the last bin is $N_f$ and $N_f \geq N_p + N_b$, $N_p$ may be increased by $(N_f - N_p)/N_b$ and the above partition process may be repeated to increase the uniformity. Bins and their boundaries obtained become parameters of the encoder in Figure 2.

3 Experiment results on benchmarks

Performance of the robust quantisation method and the CMM classifier have been evaluated on four benchmarks consisting of large sets of real world problems from Statlog project [9], including a satellite image database, letter image recognition database, shuttle date set and image Segmentation data set.

To visualise the result of quantisation, Figure 3a shows the distribution of numbers of data points of the $8^\text{th}$ feature of the image segment data for equal-size bins. The distribution represents the inherent characteristics of the data. Figure 3b shows our robust quantisation (RQ) has resulted in the uniform distribution desired.

![Figure 3: Distributions of the image segment data for (a) equal bins, (b) RQ bins](image)

We compared the CMM classifier with the simple $k$-NN method, a multi-layer perceptron (MLP) and radial basis function (RBF) networks [10]. The performance of interest are classification rate (c-rate) on test data sets and relative speed (r-speed), measured against the $k$-NN method. The r-speed of the CMM includes the encoding, training and recall time; the r-speed of MLP and RBF nets includes the training and recall time. In the evaluation we used the CMM software libraries.
developed in the project AURA (Advanced Uncertain Reasoning Architecture) at
the University of York. It is appropriate to set 1-3 ‘1’ bits in input vectors and
separators. Experiments were conducted to study influences of a CMM’s size on

c-rate and r-speed as shown in Figure 4, and effects of the number of bins  \(N_b\) on the

performance.

![Figure 4: Effects of the CMM size on (a) c-rate and (b) r-speed on the satellite image data](image)

Choices of the CMM size and the number of bins \(N_b\) may be application dependent,
for instance, in favour of the speed or accuracy. In the experiment it was required
that the r-speed is not 4 times less and c-rate is not 1% lower than that of the \(k\)-NN

method. The results are give in Table 2. We also used dedicated CMM hardware
[2] for the classification of the four benchmarks and the r-speed was further
increased over 3 times.

<table>
<thead>
<tr>
<th>method</th>
<th>satellite image</th>
<th>image segment</th>
<th>shuttle</th>
<th>letter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>c-rate</td>
<td>r-speed</td>
<td>c-rate</td>
<td>r-speed</td>
</tr>
<tr>
<td>MLPN</td>
<td>0.914</td>
<td>0.16</td>
<td>0.950</td>
<td>0.03</td>
</tr>
<tr>
<td>RBFN</td>
<td>0.914</td>
<td>0.08</td>
<td>0.939</td>
<td>0.01</td>
</tr>
<tr>
<td>(k)-NN</td>
<td>0.906</td>
<td>1</td>
<td>0.956</td>
<td>1</td>
</tr>
<tr>
<td>CMM</td>
<td>0.901</td>
<td>4.1</td>
<td>0.948</td>
<td>6</td>
</tr>
</tbody>
</table>

The ‘two-spirals’ benchmark in Figure 5 is interesting as this highly non-linear
problem is extremely hard for back-propagation networks and relatively easy for a
Cascade-Correlation net [11]. We found that this task was extremely easy for
CMM. Figure 5b shows that a CMM correctly discriminated all data points,
including training and unseen ones.

![Figure 5: (a) Two spirals, and (b) classification by the CMM](image)
4 Conclusions

In this paper we have presented the CMM classifier, which uses a CMM for storing and matching a large amount of patterns efficiently, and the \( k \)-NN rule for classification. The RU encoder converts numerical inputs into binary ones with the maximally achievable uniformity to meet requirements of the CMM. Experimental results on the four benchmarks shows that the CMM classifier gave slightly lower classification accuracy, less than 1\%, and over 4 times speed in software and 12 times speed in hardware. Therefore our method has resulted in a generic and fast classifier.

Comparing with MLP and RBF networks, CMM needs a very short training time. When new training data arrive in an incremental way, MLP and RBF nets needs to be retrained, but with the CMM, the new samples can be simply added to the memory.

References


