Fast nearest neighbour testing algorithm for small feature sizes

O.T. Yıdız, L. Akarun and H.L. Akın

A fast nearest neighbour algorithm with a logarithmic expected testing time for small feature sizes is presented. It has been tested on a Robot Vision application for YUV space. The algorithm time requirement is better than a multilayer perceptron trained for the same purpose.

Introduction: 1 (k) nearest neighbour is the simplest and most extensively studied nonparametric learning algorithm in machine learning. Although it has a very good performance compared to its parametric opponents and has no learning phase due to its nonparametric nature, it has a serious drawback: linear testing time. To find out the class of a test case, it must be compared with N (number of instances in the training set) elements and the element (k elements) with the minimum distance to the test case must be found. To compare one instance with another, several functions exist, such as Euclidian distance (L_2 norm), Mahalonobis distance, etc. Each of these functions can be thought as a mapping from d features of the instance to a single number (distance). So, the expected testing time of the nearest neighbour algorithm is $O(dN)$. Several solutions have been proposed for reducing this linear testing time. Some of them are K-d trees [1], geometric hashing [2] and R-trees [3]. In this Letter, we propose a novel algorithm based on binary search to efficiently calculate **1** (k) nearest neighbour in $O(d!\log N)$ time. If d is small, i.e. less than or equal to 5, and N is significantly larger than d, the algorithm behaves like an $O(\log N)$ algorithm. We also show that this corresponds to estimating the reduced ordering of vectors from different conditional orderings [4].

Algorithm: The novel algorithm mainly depends on binary search on sorted instances. As a preprocessing step, $d!$ sorted arrays are prepared. If an instance has d features, then there are $d!$ possible permutations of the features. For each possible permutation, all instances are sorted in alphabetic order according to that permutation. This is defined in the literature as the conditional ordering of a set of vectors [4]. The 1-nearest neighbour which we are looking for is the nearest vector in the reduced ordering of the vectors. We will show that this can be estimated from different conditional orderings that correspond to permutations of vector components. For each permutation, N elements can be sorted in $O(N \log N)$ time using the quicksort algorithm. So for $d!$ possible permutations, the preprocessing step requires $O(d!N \log N)$ time and $O(d!N)$ memory space. Fig. 1 shows a data set with $d=3$ features, $N=10$ instances sorted in $6=3!$ different ways. The preprocessing step of the algorithm can be thought as the learning phase of the nearest neighbour algorithm.

x_1	x_2	x_3
1	1	3
1	4	2
2	1	3
2	2	2
2	2	4
3	1	2
3	1	3
3	2	1
3	3	1
4	3	1

x_1	x_3	x_2
1	2	4
1	3	1
2	2	2
2	3	1
2	4	2
3	1	2
3	1	3
3	2	1
3	3	1
4	1	3

x_2	x_1	x_3
1	1	3
1	2	3
1	3	2
1	3	3
2	2	2
2	2	4
2	3	1
2	4	2
3	1	3
4	1	2

x_2	x_3	x_1
1	2	3
1	3	1
1	3	2
1	3	3
2	1	3
2	2	2
2	2	4
2	4	2
3	1	3
4	2	1

x_3	x_1	x_2
1	3	2
1	3	3
1	4	3
2	1	4
2	2	2
2	3	1
3	1	1
3	2	1
3	3	1
4	2	2

x_3	x_2	x_1
1	2	3
1	3	3
1	3	4
2	1	3
2	2	2
2	4	1
3	1	1
3	1	2
3	1	3
4	2	2

Fig. 1 *Data set after preprocessing step*

In the testing, each test instance x^t is searched in $d!$ sorted arrays using binary search. If an exact match is found, there is no need to search other arrays for 1-nearest neighbour. Otherwise, we search the two closest instances at each of the $d!$ arrays. These two closest instances are left and right border instances in the binary search. Because searching an element in a sorted array with N elements using binary search takes $O(\log N)$ time, the algorithm has a time complexity of $O(d!\ \log N)$. After finding the two closest instances at each array, we search these $2d!$ instances for 1-nearest neighbour. Compared to the binary search this step of the algorithm takes only $O(d!)$ time, which is smaller than searching arrays.

The algorithm can be easily extended to k-nearest neighbour without affecting the time complexity. The difference occurs in the last step. We search two closest instances at each array in 1-nearest neighbour, whereas we search the closest $2k$ instances in k-nearest neighbour. As an example, assume that we want to find 1-nearest neighbour of ($x_1=2$, $x_2=3$, $x_3=4$) for the example data set in Fig. 1. The two closest instances to that instance are shown in Fig. 2. If we search these ten instances, we find that ($x_1=2$, $x_2=2$, $x_3=4$) is the nearest neighbour.

x_1	x_2	x_3
1	1	3
1	4	2
2	1	3
2	2	2
2	**2**	**4**
3	**1**	**2**
3	1	3
3	2	1
3	3	1
4	3	1

x_1	x_3	x_2
1	2	4
1	3	1
2	2	2
2	3	1
2	**4**	**2**
3	**1**	**2**
3	1	3
3	2	1
3	3	1
4	1	3

x_2	x_1	x_3
1	1	3
1	2	3
1	3	2
1	3	3
2	2	2
2	2	4
2	**3**	**1**
3	**3**	**1**
3	4	1
4	1	2

x_2	x_3	x_1
1	2	3
1	3	1
1	3	2
1	3	3
2	1	3
2	2	2
2	2	4
2	4	2
3	1	3
4	**2**	**1**

x_3	x_1	x_2
1	3	2
1	3	3
1	**4**	3
2	1	4
2	2	2
2	3	1
3	1	1
3	2	1
3	3	1
4	**2**	**2**

x_3	x_2	x_1
1	2	3
1	3	3
1	3	4
2	1	3
2	2	2
2	4	1
3	1	1
3	1	2
3	1	3
4	**2**	**2**

2	3	4

2	4	3

3	2	4

3	4	2

4	2	3

4	3	2

Fig. 2 *Two nearest instances in each array and test instance for data set in Fig. 1*

Note that one can find counter-examples where this algorithm does not find the nearest neighbour. However, we have found that when the number of samples in the training set is large, the algorithm almost always finds the nearest neighbour.

Table 1: Number of operations in MLP and fast nearest neighbour

Operation	MLP		Nearest neighbour	
	Integer	Float	Integer	Float
Multiplications	0	91	36	0
Additions	0	98	72	0
Division	0	7	0	0
Comparison	0	9	113	0
Exponentiation	7		0	

Robot application: The algorithm was tested in Robocup 2002 competitions for the Sony four-legged robot league [5]. In the four-legged robot league each team has four autonomous robots playing football against each other. Each robot has a colour built-in camera with a resolution of 176×144. Each pixel's colour is coded with YUV system. The camera takes pictures with a speed of 30 frames per second. This puts a time threshold on the algorithms running on the robot. The most time consuming part of the system is the analysis of a picture that is finding the objects in a frame. The operations in each frame:

Classify each pixel into one of the possible ten colours
Using classified pixels, find regions in the frame
Classify regions into known objects

All these operations must be performed in 1/30th of a second in the robot's processor in order to match the camera speed. To classify a pixel, first, data are collected. To achieve good accuracy on the test frames, 150 000–200 000 data points must be collected. Each data point consists of the pixel's Y, U, V values and corresponding class number (colour). We compared two classification algorithms on these data. First, MLP (multilayer-Perceptron algorithm), second, our fast nearest neighbour algorithm. The algorithm's training times are unimportant, because the weights of the neurons in MLP and the sorted arrays in nearest neighbour are prepared in advance and installed into the robot. Therefore, only testing time (classifying pixels in a frame) is important. Since there are $d=3$ features (Y, U, V) and $d!=6$ is insignificant compared to the number of data ($N=200\,000$), the novel algorithm is well suited for this problem. Nearest neighbour has two advantages over MLP due to the nature of

the problem. First, because of the lighting conditions, usually a large number of the pixels of the test frames are exactly the same as the pixels in the training frames. Assuming that the labeller correctly labels the pixels, the nearest neighbour algorithm will have 100% performance on these pixels, whereas MLP will not have the same performance. Secondly, since the colour space is smoothly distributed (small changes in *Y*, *U*, *V* values will result in no change in colour classification), nearest neighbour will behave better than MLP.

MLP has four input units, *Y*, *U*, *V*, and a bias unit, ten output units or colours and in our training seven hidden units. To test an instance with MLP (see the related column in Table 1):

3×7 multiplications for the first layer, and 7×10 multiplications for the second layer, total 91, floating point multiplications must be carried out. To sum up the multiplications to obtain hidden values and output values 91 floating point addition operations must be performed
There are also seven sigmoid operations, which are much more costly than multiplication because of the exponentiation operation. There are also seven addition and seven division operations in the sigmoid function
To find the best class, nine comparisons must be performed on the output units

The novel nearest neighbour has six sorted arrays with 200 000 instances in each. To test an instance with the novel nearest neighbour (see the related column in Table 1):

Searching an instance in one array requires $\log(200\,000) = 17$ comparison operations. Six arrays require 102 comparison operations
To find nearest neighbour in 12 instances, 36 integer multiplications, 36 integer additions, 36 integer subtractions, and 11 comparisons must be performed

One possible extension of the algorithm may be to convert *Y*, *U*, *V* sorted values into a long integer and performing all operations on these values. Then, the second step in nearest neighbour will reduce to 11 comparison operations.

Conclusion: We have developed a novel algorithm that estimates the nearest neighbour from different conditional orderings of vector components. The algorithm is computationally advantageous especially when the number of training samples is large. We have applied the algorithm to a colour quantisation problem in a robot application. We are currently working on the statistical analysis of the algorithm.

24 October 2003
Electronics Letters online no: 20040144
doi: 10.1049/el:20040144

O.T. Yıldız, L. Akarun and H.L. Akın (*Department Of Computer Engineering, Boğaziçi University, Bebek, İstanbul*)

E-mail: yildizol@cmpe.boun.edu.tr

References

1 Bentley, J.L., and Weide, B.W.: 'Optimal expected-time algorithms for closest point problems', *ACM Trans. Math. Soft.*, 1980, **6**, (4), pp. 563–580
2 Califano, A., and Mohan, R.: 'Multidimensional indexing for recognizing visual shapes'. Proc. IEEE Conf. Computer Vision and Pattern Recognition, Maui, HI, USA, 1991, pp. 28–34
3 Guttman, A.: 'R-trees: a dynamic index structure for spatial searching'. Proc. ACM SIGMOD, Boston, MA, USA, 1984, pp. 47–57
4 Hardie, R.C., and Arce, G.R.: 'Ranking in Rp and its use in multivariate image estimation', *IEEE Trans. Circuits Syst. Video Technol.*, 1991, **1**, (2), pp. 197–209
5 Akin, H.L., Pavlova, P., and Yildiz, O.T.: '"Cerberus 2002" Robocup 2002: Robot Soccer World Cup VI'. 2002 Int. Robocup Symp. Preproceedings, Fukuoka, Japan, 2002, p. 448