FAST k-NN CLASSIFICATION WITH AN OPTIMAL k-DISTANCE TRANSFORMATION ALGORITHM

Olivier Cuisenaire [1,2] *and Benoît Macq* [2]
[1] Signal Processing Laboratory, EPFL, Swiss Federal Institute of Technology, CH-1015 Lausanne, Switzerland. Olivier.Cuisenaire@epfl.ch
[2] Communications and Remote Sensing Laboratory, Université catholique de Louvain, Place du Levant 2, B-1348 Louvain-la-Neuve, Belgium. Macq@tele.ucl.ac.be

ABSTRACT

The k-NN classification rule uses information from the k nearest prototypes in order to classify a pattern. In this paper, we improve Warfield's lookup table approach, where the classification problem is reformulated in terms of distance transformations. We propose a new k-distance transformation algorithm using ordered propagation. We show that - using this algorithm - the k-NN classification of F possible patterns in a D-dimensional space has a $O(k.D.F)$ complexity.

1 INTRODUCTION

The k-Nearest Neighbors (k-NN) rule is a non-parametric supervised pattern classification technique. Given the knowledge of N prototype patterns (vectors of dimension D) and their correct classification into several classes, it assigns an unclassified pattern to the class that is most heavily represented among the k closest prototypes in the pattern space.

The first formulation of this rule was made by Fix and Hodges [4]. They established the consistency of the rule for sequences such that $k \to \infty$ and $k/N \to 0$. The probability of error R of the k-NN rule is of course at least as large as the Bayes probability of error R^*, resulting from the overlap of the probabilistic distribution of the classes in the pattern space. Cover [2] shows that R is bounded by $(1 + 1/k)R^*$. Thus, when $k \to \infty$, $R \to R^*$, which is remarkable considering that no assumptions are made on the probabilistic distributions involved.

Implementing the k-NN rule with a brute-force method in order to classify F patterns using N prototypes requires $F.N$ distance computations and $o(F.N.\log(N))$ comparisons. For large data sets, this is often unpractical, which has triggered the search of efficient algorithms.

For instance, several authors such as Jiang and Zhang [6] propose a branch and bound approach where the prototypes are hierarchically decomposed into disjoint subsets. A powerful tree-search algorithm, the branch and bound method, is then applied to the resulting groups. Alternatively, Friedman [5] orders the training data along the axis with the maximal sparsity for each pattern. He can then restrict the computations to a band around the projection of the test data onto this axis. The expected number of distance computations is reduced to $O(F.k^{1/D}.N^{1-1/D})$ with D dimensional patterns. A more exhaustive discussion of these and other techniques can be found in [3].

Finally, **Warfield** [10] considers a particular type of applications where the number of possible patterns is much smaller than the number of patterns to classify. One such application is the classification of MRI data, where patterns consists of 2-3 channels (D) of data quantified over a small (0-255) range of values, for a 3D volume including typically $1 - 6 \times 10^6$ voxels. Then, it becomes efficient to precompute a lookup table for every possible pattern, then to classify the voxels by accessing the location of their values in the lookup table.

In image processing, distance transformations [1, 9, 7, 8, 3] are algorithms that compute, for every pixel of an image, the distance to the nearest pixel of a given object. If one considers the pattern space as an image and the prototype patterns as object pixels, the computation of the above lookup table and distance transformations are obviously similar concepts. Warfield's k-distance transformation (k-DT) algorithm is based on Borgefors' chamfer DT [1] in 2D and on Ragnelmam's quasi-Euclidean corner EDT [8] in higher dimensions. The difference with those methods is that the k nearest patterns (object pixels) are considered, instead of 1. It goes as follow:

Algorithm 1 *Warfield's k-distance transformation*

Insert training data patterns identifiers into the map.
for all distance transform mask scans **do**
  for all pixels p in the map **do**
    Propagate the k-NN identifiers from each mask edge pixel to the center pixel p,

Compute distance from p to each of the training patterns,
Sort in order of increasing distance,
Select the identifiers of the k nearest patterns

The method requires $O(2^D F(D+1)k)$ distance computations and $O(F(D+1)k\log((D+1)k))$ comparisons. Warfield shows that - for this type of applications - it is an order of magnitude faster than the above k-NN methods.

Finally, let us notice that Warfield's method and Ragnelmam's Euclidean DT by raster scanning (on which it is based) are both prone to a small amount of errors due to the discontinuity of the discrete Voronoi polygons around patterns in the pattern space. The following method is also prone to such errors, but they do not appear to be of consequence for practical applications. An in-depth analysis of those errors can be found in [3].

2 THE k-DISTANCE TRANSFORMATION.

2.1 Notations

In terms of k-NN classification, the k-DT problem can be formulated as follows: given a set of N prototype patterns $q(l)$ with labels l $(1 \leq l \leq N)$, determine - for every possible pattern p in the pattern space I - $\mathrm{kNN}(p) = \{\mathrm{NN}_i(p)\ ,\ 1 \leq i \leq k\}$ where $\mathrm{NN}_i(p)$ is the label of the i^{th} nearest prototype pattern to p. For instance, $\mathrm{NN}_0(p) = l$ such that $dist(p, q(l)) \leq dist(p, q(k)) \forall 1 \leq k \leq N$. Ties are broken arbitrarily. The metric $dist(,)$ is application dependent. It will often be the square of the Euclidean metric.

Similarly, the k-DT problem can be described in familiar image processing terms: given a set of N object pixels $q(l) \in O$, determine the k nearest object pixels $\mathrm{kNN}(p)$ for every pixel p in the image I.

Although both formulations sound similar, there is a minor difference in the fact that the "image" formulation supposes that prototypes are unique, i.e. that $q(l_1) = q(l_2) \Rightarrow l_1 = l_2$. The "pattern space" formulation does not make this assumption. There can be several identical patterns among the prototypes.

2.2 Our approach

In algorithm 1, computational power is wasted in two ways. First, as pointed out by Ragnelmam in [7], the raster scanning procedure propagates the information further than needed. This is especially true for high-dimensional pattern spaces, where 2^D scans are performed. Secondly, a large part of the computational power is used by the sorting procedure, especially for large values of k.

We propose to generate the k-DT using ordered propagation to scan the pattern space, starting from the prototype patterns, then to their neighbors, their neighbors' neighbors, ... by order of increasing distance. The ordered propagation is achieved by bucket sorting the patterns in the propagation front, as first suggested by Verwer [9] for simple metrics. The benefits of the method are twofold. First the propagation of every label is restricted to the zone of influence of the pattern it represents. Secondly, it is possible, by delaying the updates of the propagated patterns, to avoid any sorting beside the bucket sorting.

For every pixel p in the propagation front, we store both its coordinates and the propagating label l. The propagation front is implemented as an array of buckets $bucket(i)$. A propagating label l at pixel p is stored in $bucket(dist(p, q(l)))$ [1]. Buckets are emptied by increasing values of i.

In addition to the k label maps $\mathrm{NN}_i(p)$ that are computed, we store three additional temporary information for each pixel. First, $i_{cur}(p)$ indicates how many labels have reached p at any step of the algorithm. Secondly, $d_{cur}(p)$ is the value of the distance from p to the prototype of the last label to have reached p, i.e $d_{cur} = dist(p, q(\mathrm{NN}_{icur}(p)))$. Thirdly, if more than one label in $\mathrm{kNN}(p)$ corresponds to a prototype at distance d_{cur}, then $i_{dcur}(p)$ stores the smallest i for which $dist(p, q(\mathrm{NN}_i(p))) = d_{cur}(p)$.

Let us now consider that distance d has been reached, i.e all $buckets(d')$, $d' < d$ are emptied and that (p, l) in $bucket(d)$ is being processed. The processing includes two steps. First we check if l should be added to $k\mathrm{NN}(p)$. If so, label l is then propagated to p's neighbors.

In the first step, l should be added to $\mathrm{kNN}(p)$ if two conditions are fulfilled. First, there should be less than k labels in $\mathrm{kNN}(p)$ already, i.e. $i_{cur}(p) < k$. Secondly, label l should not belong to $\mathrm{kNN}(p)$ yet. If $d_{cur}(p) < d$, it obviously does not. Otherwise, i.e when $d_{cur}(p) = d$, all labels $\mathrm{NN}_i(p)$ with $i_{dcur}(p) \leq i \leq i_{cur}(p)$ should be checked.

In the second step, label l is propagated to p's direct neighbors, i.e. pixels $p' = p + n$ with $n \in N = \{(0,1), (0,-1), (1,0), (-1,0)\}$. Practically, only those neighbors to lead to a larger distance $dist(p', q(l))$ need to be considered. That is those in the same direction as vector $p - q(l)$.

[1] This requires that the metric $dist(,)$ only takes integer values

2.3 The algorithm

Algorithm 2 *k-DT algorithm by bucket-sorting propagation.*

Input: N prototypes $q(l)$ with labels l, $1 \leq l \leq N$
Output: the sets $\text{kNN}(p) = \{\text{NN}_i(p),\ 1 \leq i \leq k\}$, $\forall p \in I$

for all $p \in I$ **do** {Initialization}
  $i_{cur}(p) \leftarrow 0$
  $d_{cur}(p) \leftarrow 0$
for $l = 1$ to N **do**
  put $(q(l), l)$ in $bucket(0)$
$d \leftarrow 0$

repeat {Main loop}
  while $bucket(d)$ is not empty **do**
    get (p, l) from $bucket(d)$
    if $i_{cur}(p) < k$ **then**
      if $d_{cur}(p) < d$ **then**
        process(p, l)
      else if $\text{NN}_j(p) \neq l \ \ \forall j, i_{dcur}(p) \leq j \leq i_{cur}(p)$ **then**
        process(p, l)
  $d \leftarrow d + 1$
until all buckets are empty

procedure process(p, l)
  $i_{cur}(p) \leftarrow i_{cur}(p) + 1$
  $\text{NN}_{i_{cur}(p)}(p) \leftarrow l$
  if $d_{cur}(p) \neq d$ **then**
    $i_{dcur}(p) \leftarrow i_{cur}(p)$
  $d_{cur}(p) \leftarrow d$
  for all $n \in N$ **do** {Propagation}
    if $dist(p + n, q(l)) > d$ **then**
      put $(p + n, l)$ in $bucket(dist(p + n, q(l)))$

Let us note that the implementation of this algorithm requires a special attention. In particular, the dynamic data structure used to implement the buckets should allocate memory in chunks and not element by element. On the other hand, the k NN_i(p) label maps and the additional temporary information can be stored statically.

3 COMPUTATIONAL COMPLEXITY

3.1 Theoretical analysis

In [10], it is shown that using a k-DT to compute a lookup table is the most efficient method to perform the k-NN classification of a large data set where the number of possible different patterns is comparable with or lower than the size of the data set. In this paper, we show that algorithm 2 has an optimal computational complexity for a k-DT. To make comparisons easier, we use Warfield's notations, i.e. we consider the problem of classifying F patterns in a D dimensional space, using the k-NN rule.

The output of the algorithm is made of k maps covering the F patterns. The complexity of any k-DT algorithm is then of course at least that of its output, i.e. $O(F.k)$. More realistically, a k-DT algorithm should at least consider the direct-neighbors of a pixel to compute its value, which means a $O(F.D.k)$ complexity in D dimensions.

In our algorithm, procedure $process(p, l)$ is called exactly $F.k$ times, since it increments $\text{NN}_{icur(p)}(p)$ and since the propagation stops at soon as $\text{NN}_i(p) = k\ \forall p$. In that procedure, the neighbors of p are entered in the buckets structure. Using 2D-direct neighborhoods, restricted to those in the same direction as $p - q(l)$, there are between D and $2D$ neighbors propagated for each of the $F.k$ pixels that enter $process(p, l)$. Thus, the total amount of elements passing through the buckets is $O(F.D.k)$.

The distance $dist(p, q(l)$ is only computed inside the $process(p, l)$ procedure, in order to determine in which bucket (p, l) should go. Thus, the total amount of distance computations is exactly the same as the number of times $process(p, l)$ is called, i.e $O(F.D.k)$.

Finally, the number of comparisons performed inside the main loop for an element (p, l) taken from $bucket(d)$ is fixed, unless $d_{cur}(p) = d$. In this case, it is compared $i_{cur}(p) - i_{dcur}(p)$ times. This number is in average very low when prototypes are unique. The total number of comparisons is then also $O(F.D.k)$. In the worst case, with k identical prototypes at every prototype location, the average number of comparisons is close to k. It raises the number of comparisons to $O(F.D.k^2)$. This could be avoided by replacing prototypes (represented by a label l) by prototype locations (represented by a label l and a number m of occurrences in that location). Nevertheless, for practical applications, this does not appear to be needed.

In table 1, extended from the original in [10], the complexity of our algorithm is compared to the brute force algorithm and those of Friedman [5] and Warfield [10].

3.2 Experiments

In order to confirm the theoretical analysis, we ran 3 experiments on synthetical 2D data, varying the number k of nearest neighbors, the size $(n \times n)$ of the pattern space and the number N of prototypes. In experiment 1, k varies from 1 to 10, 3 values of n are considered and N is fixed to 1000. In experiment 2, n varies from 128 to 1024, 3 values of k are considered and $N = 1000$. In experiment 3, N varies from 100 to 1000, n is fixed to 512 and 3 values of k are considered. In

	Distance computations	Comparison operations
Brute force	$F.N$	$O(F.N.\log(N))$
Friedman	$O(F.k^{1/D}.N^{1-1/D})$	$D.N.\log(N)$+$O(F.k^{1/D}.N^{1-1/D})$
Warfield	$O(2^D.F.(D+1).k)$	$O(2^D.F.(D+1).k.\log((D+1).k)$
Our algorithm	$O(F.D.k)$	$O(F.D.k)$ to $O(F.D.k^2)$

Table 1: Complexity of k-NN classification algorithms, with F the number of patterns to classify, N the number of training prototypes, k the number of nearest neighbors and D the dimension of the pattern space.

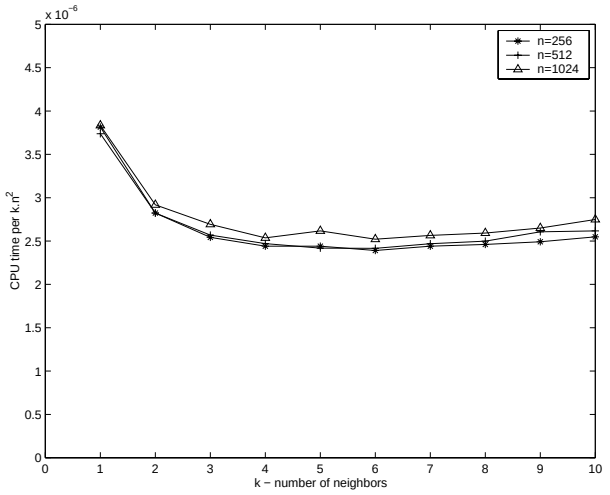

Figure 1: k-DT algorithmic complexity: dependence from the number of neighbors k for several image sizes

all experiments, the prototypes are randomly generated.

Theoretically, the computational complexity is $O(F.D.k) = o(n^2.2.k)$, so that the ratio of the CPU time by $k.n^2$ should be a constant. The 3 experiments were performed on a Pentium II computer running at 233 MHz. The CPU time per $k.n^2$ for experiment 1 is illustrated at figure 1. Figures related to experiments 2 and 3 can be found in [3].

In experiment 1 (Figure 1), the CPU time per k.pixel is constant for $k > 3$. For $k \leq 3$, the fixed cost of handling the additional information in i_{cur},d_{cur} and i_{dcur} is a non-negligible factor, so that the CPU time per k.pixel is slightly higher. In experiments 2 and 3, the image size and number of prototypes have no influence at all on the CPU time per k.pixel. In both cases, the times with $k = 1$ are significantly higher than the other two, which is explained by the results of experiment 1.

4 CONCLUSION

We show that it is possible to implement the k-distance transformation with an optimal complexity, i.e. in a time proportional to the size of the output. For applications where a lookup table approach is sensible - i.e. when the number of possible different patterns is smaller than or comparable with the amount of data to classify, this is the fastest implementation of the multi-dimensional classification using the k-NN rule.

References

[1] G. Borgefors. Distance transformation in arbitrary dimensions. *Computer Vision, Graphics, and Image Processing*, 27:321–145, 1984.

[2] T.M. Cover. Estimation by the nearest neighbor rule. *IEEE Transactions on Information Theory*, 14:50–55, 1968.

[3] O. Cuisenaire. *Distance transformations: fast algorithms and applications to medical image processing*. PhD thesis, Université catholique de Louvain, October 1999.

[4] E. Fix and J.L. Hodges. Discriminatory analysis, non-parametric discrimination. Technical report, USAF School of Aviation Medicine, Randolf Field, Tex. Project 21-49-004, Rept. 4, Contract AF41(128)-31, February 1951.

[5] J.H. Friedman, F. Baskett, and L.J. Shustek. An algorithm for finding nearest neighbors. *IEEE Transactions on Computers*, 24:1000–1006, 1975.

[6] Q. Jiang and W. Zhang. An improved method for finding nearest neighbors. *Pattern Recognition Letters*, 14:531–535, 1993.

[7] I. Ragnelmam. Neighborhoods for distance transformations using ordered propagation. *CVGIP, Image Understanding*, 56(3):399–409, 1992.

[8] I. Ragnelmam. The euclidean distance transformation in arbitrary dimensions. *Pattern Recognition Letters*, 14:883–888, 1993.

[9] B.H. Verwer, P.W. Verbeek, and S.T. Dekker. An efficient uniform cost algorithm applied to distance transforms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(4):425–429, 1989.

[10] S. Warfield. Fast k-NN classification for multichannel image data. *Pattern Recognition Letters*, 17:713–721, 1996.