Frontiers in Artificial Intelligence and Applications

PATTERN RECOGNITION AND APPLICATIONS

Edited by
M.I. Torres and A. Sanfeliu

IOS Press
Ohmsha

Pattern Recognition and Applications

Edited by

M.I. Torres

Departamento Electricidad y Electrónica, Facultad Ciencias, Universidad del País Vasco, Bilbao, Spain

and

A. Sanfeliu

Institut de Robòtica i Informàtica Industrial, Universitat Politècnica de Catalunya, Barcelona, Spain

IOS Press
Ohmsha

Amsterdam • Berlin • Oxford • Tokyo • Washington, DC

ISBN 1 58603 034 5 (IOS Press)
ISBN 4 274 90362 1 C3000 (Ohmsha)
Library of Congress Catalog Card Number: 00-102512

Publisher
IOS Press
Nieuwe Hemweg 6B
1013 BG Amsterdam
The Netherlands
fax: +31 20 620 3419
e-mail: order@iospress.nl

Distributor in the UK and Ireland
IOS Press/Lavis Marketing
73 Lime Walk
Headington
Oxford OX3 7AD
England
fax: +44 1865 75 0079

Distributor in the USA and Canada
IOS Press, Inc.
5795-G Burke Centre Parkway
Burke, VA 22015
USA
fax: +1 703 323 3668
e-mail: iosbooks@iospress.com

Distributor in Germany
IOS Press
Spandauer Strasse 2
D-10178 Berlin
Germany
fax: +49 30 242 3113

Distributor in Japan
Ohmsha, Ltd.
3-1 Kanda Nishiki-cho
Chiyoda-ku, Tokyo 101
Japan
fax: +81 3 3233 2426

PATTERN RECOGNITION AND APPLICATIONS

Frontiers in Artificial Intelligence and Applications

Series Editors: J. Breuker, R. López de Mántaras, M. Mohammadian, S. Ohsuga and W. Swartout

Volume 56

Previously published in this series:

ISSN: 0922-6389

Preface

This book deals with novel scientific and technology research in Pattern Recognition and Applications. It presents a selection of papers that summarises the main research activities in these areas developed in Spanish research centres. The book is supported by the "Asociación Española de Reconocimiento de Formas y Análisis de Imágenes" (AERFAI), associated to the International Pattern Recognition Association (IAPR). It includes thirty-one works organised into four categories reflecting the present areas of interest in the Spanish Pattern Recognition Community:

- Pattern Recognition: this Section consists of nine papers in the areas of statistical and syntactic-structural approaches. They include new approaches related to classical pattern classification problems and methodologies like multiedit algorithm, gradient-descent methods, hierarchical clustering, nearest neighbours rule, tree language compression, function described graphs, etc.
- Computer Vision: this Section includes ten papers. They present new methods in colour segmentation, visual tracking, alignment in 3D reconstruction, trademark search techniques, visual behaviours for binocular navigation and active vision systems.
- Speech Recognition and Translation: this Section consists of five papers related to continuous speech recognition and statistical translation. They include new proposals in acoustic and language models, based on Connectionist and Syntactic Pattern Recognition approaches. The word categorisation problem is also addressed in the statistical translation background.
- Applications in Computer Vision, Speech Recognition and Translation: this Section includes seven papers. They deal with application to digital TV, biomedical images, mammography, trabecular bone patterns and new calibration methods for large surface topography. A Continuous Speech Recognition system developed by a Spanish group as well as a preliminary version of a Spanish-Catalan translation system are also included in this Section.

The Scientific Committee includes nine international and twenty-two Spanish specialists in the Pattern Recognition field. Three members of this Scientific Committee have reviewed all the papers published in this book. These papers are a good summary of the Spanish research in the fields of Pattern Recognition and Image Analysis, as well as in their Applications.

We would like to thank all the authors who have submitted papers for their effort and kind collaboration. We are also grateful to the members of the Scientific Committee for their help in selecting the papers presented in this book. We also would like to thank Dr. López de Mantaras, editor of the IOS Press series *Frontiers in Artificial Intelligence and Applications*, for his support and encourage in the initial book project. Finally, we are particularly thanks to Esther Alonso for her effort in compiling the papers.

M. I. Torres and A. Sanfeliu
January 2000

Contents

Pattern Recognition

Pattern Recognition and Applications
M.I. Torres and A. Sanfeliu (Eds.)
IOS Press, 2000

An improvement of the Multiedit algorithm using metric space properties*

Pablo Aibar and Jordi Bataller[†]
Unitat Predepartamental d'Informàtica Universitat Jaume I, Castelló, Spain
[†]*Dpto. Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Spain*
`e-mail: aibar@inf.uji.es, bataller@eugan.upv.es`

Abstract. The Multiedit algorithm is currently one of the most popular editing methods in use. In this paper, we investigate how to speed it up. The improvements attempt to reduce the overall amount of distance calculations performed by the algorithm to find the nearest neighbour. By using properties of general metric spaces (based on the Triangle-Inequality), we have developed different estimators of the value of a distance. They are used to bound the search of the nearest neighbour. In this work, two new versions of Multiedit are presented and compared with the original algorithm. The results obtained in the tests show a dramatic improvement in both time and distance calculations.
Keywords: Multiedit algorithm, Fast Nearest Neighbour searching, Triangle-Inequality.

1 Introduction

The non-parametric pattern recognition approach represents a sample as a point in space and uses a set of pre-classified samples (prototypes) to classify a new one. In this approach, neighbourhood-based classifiers play a central role: given a distance measure, a sample is classified in the same class as its nearest prototype (NN rule), or in the most voted class by its k-nearest prototypes (k-NN rule). Therefore, the methods for selecting prototypes (editing methods) are central to achieving a correct recognition. These methods are intended to eliminate the prototypes that can produce an increment of the classification error (prototypes that are classified in a class to which they do not actually belong).

The Multiedit algorithm [1] is a popular editing method: it is fast compared to other editing methods and is statistically correct [2]. The algorithm is especially suited to work with a large number of prototypes (its performance considerably decreases as the number of prototypes does [2]). The algorithm randomly divides the set of prototypes in m equally sized blocks. Then, each prototype is reclassified by searching for its nearest neighbour (NN) among the prototypes in only the contiguous blocks. And finally, the prototypes which are misclassified are eliminated. This process is iterated until no prototype has been rejected in a (previously fixed) consecutive number of iterations. The lower cost of the algorithm [2] is bounded by $O(n^2 \log n)$ (with n being the number of prototypes). This implies a practical limitation to its use, especially in cases where the distance function is costly to calculate and when the number of prototypes is higher.

In this paper we investigate how to reduce the Multiedit temporal cost. We base our improvements on *fast nearest neighbour search* techniques in general metric spaces, particularly

*Work partially supported by the Generalitat Valenciana under contract GV98-14-134.

in the works related with the Approximating and Eliminating Search Algorithm (AESA) [3–5]. These works deal with a problem similar to ours in that they attempt to speed up the *process of classifying test samples* (using the NN or k-NN rule) with respect to a set of prototypes. To solve this problem, these methods need to compute and store the distances (in a previous preprocessing phase) between all (or a part of) the prototypes. Our problem is not exactly the same, because what we try to do is to speed up the *overall process of reclassifying a set of prototypes*. Thus, if we need to extract information from the prototypes in order to help the process, that time must also be added to the cost.

The remainder of the paper is organized as follows. Section 2 explains the techniques used to improve the Multiedit algorithm, and Section 3 shows the improved algorithms we are introducing. Section 4 presents the results of the tests used to evaluate our algorithms and compares them to the original one. Finally, Section 5 presents the conclusions.

2 Improving the Multiedit algorithm

2.1 *Set of base prototypes*

The main idea is to compute the distances between a set of selected prototypes (*base prototypes*) and all the prototypes (including this set) in advance [4]. Using this knowledge, a set of bound rules can be defined which will help us to decide when to stop the reclassification of a prototype in the Multiedit algorithm. The bounds use estimations of the actual distance between two points.

The size of the base prototype set is crucial. If it is too small, the computed estimations will be inaccurate. On the other hand, if the set is too large, calculating the estimations as well as the construction of the set itself will be costly. Furthermore, the topology of the set is also important in order to have really representative base prototypes, which effectively helps to save distance calculations. The selection of a uniformly sparse set of points will yield a representative set of base prototypes since there is no region in the space without such a prototype. This is the idea of the original BP-selection algorithm [4]. It uses a greedy approach which selects the prototype which is furthest from the set of the prototypes already selected. This is a general algorithm, which was proposed for a general set of prototypes. In our case, as we have *classified* prototypes, accordingly the BP-selection algorithm has been adjusted to pick at least one base prototype from each class (BP-1 algorithm). The same number of base prototypes from each class can be selected (BP-n algorithm) in order to prevent a given class from being poorly represented due to having few base prototypes. The full description of both BP-1 and BP-n new extensions of the BP-selection algorithm can be found in [6].

2.2 *Estimators*

Given a metric space (E, d), d is a distance function that satisfies:

1. $d(x, y) = d(y, x)$,
2. $d(x, y) \geq 0$,
3. $d(x, y) = 0$ iff $x = y$, and
4. $d(x, z) + d(z, y) \geq d(x, y)$ (Triangle-Inequality),

where x, y, z are points in the space E.

2.2.1 *Estimators of the distance between two prototypes*

We can use the Triangle-Inequality to define our estimations of a real distance. The inequality can be expressed as [3]:

$$d(x,y) \geq |d(x,z) - d(z,y)|,$$

where $|d(x,z) - d(z,y)|$ can be considered as an optimistic estimation of the distance $d(x,y)$ between two prototypes x and y. In fact,

$$d(x,y) \geq \max_{z \in E}\{|d(x,z) - d(z,y)|\}$$

is satisfied. To compute $\max_{z\in E}\{|d(x,z) - d(z,y)|\}$ it is necessary to know $d(x,z)$ and $d(z,y)$ for each point z. In our problem, z will be a *base prototype* (note the importance of having a representative set of base prototypes). We can calculate a first estimator of the distance $d(x,y)$ as follows [4]:

$$g_1(x,y) = \max_{u \in U}\{|d(x,u) - d(u,y)|\},$$

with U being the set of base prototypes for which we know the distances to the rest of the points. The temporal cost needed to calculate g_1 is bounded by $O(|U|)$.

In order to obtain less costly estimators, we can consider that not every prototype yields a good approximation to the real distance. In fact, only the nearby base prototypes to a given point offer a good estimation of the distance between this point and other ones. It might be interesting to use only prototypes from the same class as x or the same class as y to evaluate the estimator, since we expect points in the same class to be close in space. Therefore, we propose a second estimator,

$$g_2(x,y) = \max_{u \in (U_{c(x)} \cup U_{c(y)})}\{|d(x,u) - (d(u,y)|\},$$

where $c(x)$ returns the class of x and U_c is the set of base prototypes in class c. The cost of g_2 is also $O(|U|)$; however, in the case of an equal distribution of the base prototypes among the classes, the cost of g_2 would be $O(|U|/N)$, with N being the number of classes.

Finally, the cheapest way to evaluate the estimator of the distance between two prototypes x and y is to consider only the prototypes which are nearest to the given prototypes x and y:

$$g_3(x,y) = \max\{|d(x,x_u) - d(x_u,y)|, |d(x,y_u) - d(y_u,y)|\},$$

where x_u and y_u are the nearest prototypes to x and y, respectively. The cost of calculating g_3 is $O(1)$.

The potential trade-off between the accuracy of each estimator and its cost function is clear: g_1 is the most accurate one; however, it may be too expensive. On the other hand, g_3 is cheaper, but it may be inaccurate.

2.2.2 *Estimators of the distance between a prototype and a class*

Using the concepts of maximum and minimum radius of a class [7, 8], we can develop an estimator of the distance between a class as a whole and a given prototype outside the class. The maximum radius of class c (calculated using the base prototype $u \in U_c$) is

$$R_{c,u} = \max_{x \in W_c}\{d(u,x)\},$$

where W is the whole set of prototypes and W_c are the prototypes in class c (obviously, $U \subseteq W$ and $U_c \subseteq W_c$). Using this radius, we can estimate the distance between a prototype x and any prototype in the class c as:

$$g_R(x, c) = \max_{u \in U_c}\{d(x, u) - R_{c,u}\},$$

due to the fact [7] that $d(x, y) \geq d(x, u) - R_{c,u}$, for any prototype $y \in W_c$ and any base prototype $u \in U_c$. The cost of calculating g_R is $O(|U|)$; however, if the base prototypes are equal distributed among the N classes, the cost would only be $O(|U|/N)$.

The minimum radius of the class c calculated using the base prototype $u \in U_c$ is

$$r_{c,u} = \min_{x \in (W_c - \{u\})}\{d(u, x)\},$$

and an estimation of the distance between a prototype x and any prototype in the class c is

$$g_r(x, c) = \max_{u \in U_c}\{r_{c,u} - d(x, u)\}$$

since, for any prototype $y \in W_c$ and any base prototype $u \in U_c$, $d(x, y) \geq r_{c,u} - d(x, u)$ [8]. The cost of g_r is the same as the cost of g_R.

Finally, because g_R and g_r are always smaller than the actual distance to a given point x, we can obtain a general estimator of the distance between a prototype and a class:

$$g_c(x, c) = \max\{g_R(x, c), g_r(x, c)\}.$$

2.3 *The bound rules*

From the previous definitions, we present the following bound rules. If we are searching for the nearest neighbour to a prototype x and we have a candidate y (we know the distance $d(x, y)$ between x and y), then:

- *Prototype bound rule (g rule).* We can reject a second candidate z if $g(x, z) \geq d(x, y)$ since $d(x, z) \geq g(x, z) \geq d(x, y)$ and, therefore, z can't be nearer to x than y [3]. Function g may be any of g_1, g_2 or g_3.

- *Class bound rule (g_c rule).* We can reject every point in class c if $g_c(x, c) \geq d(x, y)$ because $d(x, z) \geq g_c(x, c) \geq d(x, y)$ for all prototype z in W_c. It should be noted that for all prototype z in $W_c : d(x, z) \geq g(x, z) \geq g_c(x, c)$. This does not mean that the estimator g_c is not useful. On the contrary, the calculation of g_c is cheaper than the calculation of g and it allows us to discard an entire class at one time.

Finally we introduce a very simple but new rule to stop the search for the nearest neighbour of a prototype:

- *Definitive classification rule.* In the Multiedit algorithm, we do not really need to know what the nearest neighbour to a prototype is, but rather whether the prototype we are considering belongs to the same class as its nearest neighbour. Therefore, if at some point in the search for the nearest neighbour, all the prototypes yet to be inspected (including the actual candidate) are from the same class as the prototype being considered, the nearest one will be of the same class. Conversely, if all the remaining prototypes (including the actual candidate) are from a different class, the prototype we are considering will be discarded because the nearest one will have a different class. In accordance with this, we can stop the search having made a decision on whether to accept or reject a given prototype.

3 New versions of the Multiedit algorithm

In this section, we present the new versions of the original Multiedit algorithm (M) which include the improvements we introduced in Section 2. Note that the new algorithms yield the same result as the original one. Here is the original Multiedit algorithm (which was described in Section 1):

```
Algorithm: Multiedit (M)

INPUT
$W \subset E$ {training set: $W = \bigcup_{c=1}^{N} W_c$. $N$ is the number of classes}
$N_B \in \mathbb{N}$ {number of blocks}
$I \in \mathbb{N}$ {number of iterations without edition}
OUTPUT
$S \subseteq W$ {set of edited prototypes}
Variables
$c_{NN}$ {nearest neighbour's class}
$d_{NN}$ {distance to the nearest neighbour}
$B \subseteq W$ {set of blocks: $B = \bigcup_{b=1}^{N_B} B_b$. $B$ is a circular structure ($B_{N_B+1} = B_1$)}
$i, b \in \mathbb{N}$
Functions
$c : W \to \{1, 2, \ldots, N\}$ {$c(x) = i \Rightarrow x \in W_i$}
$d : W \times W \to \mathbb{R}$ {distance function}
$Shuffle : \mathcal{P}(W) \times \mathbb{N} \to \mathcal{P}(W)$ {randomly put each $S$ point into some of the $N_B$ blocks of $B$}

Algorithm
$i \leftarrow 0; S \leftarrow W$
repeat
  $B \leftarrow Shuffle(S, N_B)$
  for $b \leftarrow 1$ to $N_B$ do
    for all $x \in B_b$ do
      $d_{NN} \leftarrow \infty$
      for all $y \in B_{b+1}$ do
        if $d(x, y) < d_{NN}$ then
          $d_{NN} \leftarrow d(x, y); c_{NN} \leftarrow c(y)$
      if $c(x) \neq c_{NN}$ then
        $S \leftarrow S - \{x\}$
  if $|B| \neq |S|$ then
    $i \leftarrow 0$
  else
    $i \leftarrow i + 1$
until $i = I$
```

3.1 Fast Multiedit 1 (M1)

Our first proposal, Fast Multiedit 1 algorithm (M1), uses the three bound rules presented: g, g_c and the definitive classification rule. It uses the following functions:

- *Build_U* selects the base prototypes and returns a matrix of distances between each prototype and the rest of the points.
- *Calc_g_c* computes the g_c estimator for every prototype and every class. It fills the matrix G_c.
- *NNClass* returns the class of the NN in a given block for a base prototype.
- *NN_BP* returns the nearest base prototype in a given block to a given point.
- *DelCls* applies the Class bound rule: it verifies out whether $G_c[x, c] \geq d_{NN}$ for each class c. If so, it eliminates the points of that class c from the search set.
- g stands for any of the estimators g_1, g_2, or g_3 as described in the previous section.
- *DefCls* returns *true* when the Definitive classification rule can be applied due to the fact that all the prototypes which are in the class of x (or all the prototypes which are not in the class of x) have been rejected.

Initially, *Build_U* and *Calc_g_c* are called (note that the estimator g_c does not change throughout the execution). For each prototype x to be classified, the algorithm performs the following actions:

a) if x is a base prototype, the classification is trivial because the distances to the rest of prototypes are known (*NNClass* finds out the class of the nearest neighbour).

b) if x is not a base prototype, the search is started by finding an initial NN candidate, the nearest base prototype in the contiguous block (*NN_BP*). Afterwards, we begin to inspect each point y in the contiguous block (until there are no points in the block or x is classified according to *DefCls*). We calculate $g(x, y)$ and, if the estimation is better than the current nearest candidate, the distance $d(x, y)$ is calculated and a decision is taken accordingly. Note that each time the candidate changes the *DelCls* function is tried.

```
Algorithm: Fast Multiedit 1 (M1)

INPUT
$W, N_B, I$
OUTPUT
$S$
Variables
$c_{NN}, d_{NN}, B, i, b$
$V \subseteq W$ {investigated points when looking for NN}
$U \subseteq W$ {base prototype set}
$M : [U \times W] \to \mathbb{R}$ {matrix of distances for base prototypes: $M[u, x] = d(u, x)$}
$G_c : [W \times \{1, \ldots, N\}] \to \mathbb{R}$ {matrix of $g_c$ estimations: $G_c[x, c] = g_c(x, c)$}
Functions
$c, d, Shuffle$
$Build\_U : \mathcal{P}(W) \to (\mathcal{P}(W), M)$
$Calc\_g_c : \mathcal{P}(W) \times \mathcal{P}(W) \times M \to G_c$
$NNClass : U \times M \times \{1, \ldots, N_B\} \to \{1, \ldots, N\}$
$NN\_BP : W \times M \times \mathcal{P}(W) \to U \times \mathbb{R} \times V$
$DelCls : W \times G_c \times \mathbb{R} \times \mathcal{P}(W) \times V \to V$
$g : W \times W \to \mathbb{R}$
$DefCls : \mathcal{P}(W) \times V \times \{1, \ldots, N\} \times \{1, \ldots, N\} \to \mathbb{B}$

Algorithm
$(U, M) \leftarrow Build\_U(W)$
$G_c \leftarrow Calc\_g_c(W, U, M)$
$i \leftarrow 0; S \leftarrow W$
repeat
  $B \leftarrow Shuffle(S, N_B)$
  for $b \leftarrow 1$ to $N_B$ do
    for all $x \in B_b$ do
      if $x \in U$ then
        $c_{NN} \leftarrow NNClass(x, M, B_{b+1})$
      else
        $(c_{NN}, d_{NN}, V) \leftarrow NN\_BP(x, M, B_{b+1})$
        $V \leftarrow DelCls(x, G_c, d_{NN}, B_{b+1}, V)$
        for all $y \in B_{b+1}$ and while not $DefCls(B_{b+1}, V, c(x), c_{NN})$ do
          $V \leftarrow V \cup \{y\}$
          if $g(x, y) < d_{NN}$ then
            if $d(x, y) < d_{NN}$ then
              $d_{NN} \leftarrow d(x, y); c_{NN} \leftarrow c(y)$
              $V \leftarrow DelCls(x, G_c, d_{NN}, B_{b+1}, V)$
      if $c(x) \neq c_{NN}$ then
        $S \leftarrow S - \{x\}$
  if $|B| \neq |S|$ then
    $i \leftarrow 0$
  else
    $i \leftarrow i + 1$
until $i = I$
```

3.2 Fast Multiedit 2 (M2)

The second proposal, Fast Multiedit 2 algorithm (M2), is derived from M1 and introduces and exploits all the information derived from the g estimations. Function g is calculated for each possible candidate and the candidates are ordered accordingly. In this way, the search begins with the best possible candidate. The new functions used are:

- *Updt_L* updates the sorted list of candidates, L, each time a new candidate is chosen. It rejects every candidate having an estimation which is worse than the new one found. Note that, in this case, it is unnecessary to call *DelCls*.
- *Sort* sorts the list of candidates L.

The two versions of the algorithm have complementary advantages. M1 will definitely calculate more distances than M2 since it does not use g to drive the search. On the other hand, M2 increases overhead due to the computation of g for every prototype and the sorting of the candidates.