CI & A

ANNs FOR CLASSIFICATION AND RECOGNITION

BASIC ALGORITHMS

General context

What is classification ?

... learning the similarities and differences between instances of objects from a population of non-identical objects.

General context

Two stages:

Classification - the system learns the general characteristics of classes based on specific characteristics of instances.

Recognition - the system overlap the characteristics of an instance over the characteristics of known classes and identifies the class to which that instance belongs.

General context

Two learning types:

Supervized learning – applies whenever the class to which each input pattern (an instance of classified objects) is associated to is known in advance.

Unsupervized learning - applies when the class to which each input pattern (an instance of classified objects) is associated to is not known in advance.

Similarity / Dissimilarity measures

APPROACH

Beginning with a set of input vectors $X = \{x^{(1)}, x^{(2)}, \dots, x^{(M)}\}$, with a common structure $x^{(m)} = (x_1^{(m)}, x_2^{(m)}, \dots, x_N^{(m)})$.

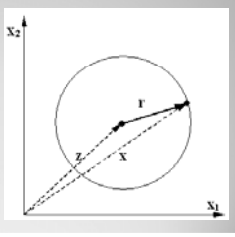
The aim: to separate K classes – denoted $X_1, X_2, \dots, X_K$ – each one being characterized by a prototype z^k $(k=1,\dots,K)$.

Finally, based on the K prototypes z^k each vector $x^{(m)}$ is associated to one of the K classes.

Similarity / Dissimilarity measures

Euclidiana distance, 2 - norm

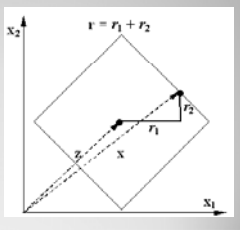
$$\| \mathbf{x} - \mathbf{z}^k \|_2 = \sqrt{\sum_{n=1}^{N} (x_n - z_n^k)^2}$$



Similarity / Dissimilarity measures

Distance by 1 - norm

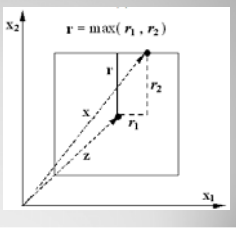
$$\| \mathbf{x} - \mathbf{z}^k \|_1 = \sum_{n=1}^{N} \left| x_n - z_n^k \right|$$



Similarity / Dissimilarity measures

Distance by ∞ - norm

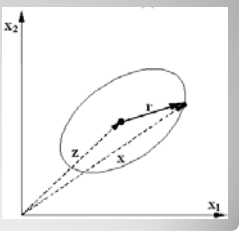
$$\| \mathbf{x} - \mathbf{z}^k \|_\infty = \max_{n=1,\ldots,N} \left(\left| x_n - z_n^k \right| \right)$$



Similarity / Dissimilarity measures

Mahalanobis distance

$$\| \mathbf{x} - \boldsymbol{\mu} \|_M = [(\mathbf{x} - \boldsymbol{\mu})^T \cdot \mathbf{C}^{-1} \cdot (\mathbf{x} - \boldsymbol{\mu})]^{-1/2}$$



Similarity / Dissimilarity measures

Distance by Hamming - norm

$$\| \mathbf{x} - \mathbf{z} \|_H = \sum_{n=1}^{N} |x_n - z_n|$$

For binary representation, the **Hamming distance** measures the number of different components of vectors **x** and **z**.

For : **x** = (1101) and **z** = (0110) the Hamming distance is:

$||\mathbf{x}-\mathbf{z}||_H = |1-0| + |1-1| + |0-1| + |1-0| = 3$

Classification algorithms

1. K nearest neighbor algorithm
2. C-mean algorithm
3. ISODATA Algorithm
4. Kohonen networks (self-organization)
   - 4.1. VQ networks – Vector Quantization
   - 4.2. LVQ networks – Learning Vector Quantization
   - 4.3. SOFM networks – Self Organizing Feature Maps

K nearest neighbor algorithm

Principle

The algorithm selects the first **K** vectors already classified (the value of **K** is specified in advance), that are closest to the current vector, and the latter is classified into the dominant class associated with the **K** reference vectors.

K nearest neighbor algorithm

Initialization

The algorithm starts with the learning data set consisting of vectors $\{x^{(1)}, x^{(2)}, \dots, x^{(M)}\}$ and the number of classes C that must be separated. When one of these vectors $x^{(m)}$ is associated to a class c, the notation :

$$Class(m) = c$$

is used. After randomly rearranging vectors $x^{(m)}$ from the learning data set, the first C vectors are associated each one to the C classes.

K nearest neighbor algorithm

The actual classification

For each of the remaining vectors $x^{(C+1)}, x^{(C+2)}, \dots, x^{(M)}$ the following steps are done:

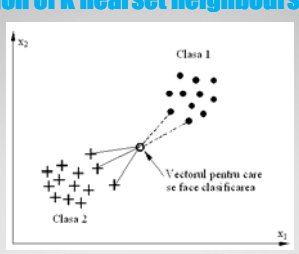
(a)calculate distances to the already classified vectors;

(b) vectors are listed in ascending order of their distances and the first K vectors in this list are considered;

(c) determine the dominant class of these K vectors and associate the current vector to this class.

K nearest neighbor algorithm

Selection of K nearset neighbours (K=5)



K nearest neighbor algorithm

The algorithm

1. Input data: training patterns, as input vectors $\mathbf{x}^{(m)}$, m = 1,..., M; parameter k (number of neighbors); number of classes C.
2. Randomly rearrange training patterns $\{\mathbf{x}^{(m)}\}$ m = 1,...,M.
3. Associated first C patterns to the C classes:
   for c = 1 to C do $Clasa(c) = c$
4. Associate classes for the rest of training patterns:
   for $m = C+1$ to M do
   // Calculate distances to vectors already classified
   for q = 1 to m - 1 do
   $d(q) = \| \mathbf{x}^{(m)} - \mathbf{x}^{(q)} \|$

K nearest neighbor algorithm

The algorithm – ctd.

```
// List in ascending order distances d(q) by rearranging
// q indices
for i = 1 to m – 1 do Ix(i) = i
  for i = 1 to m – 2 do
    for j = i + 1 to m – 1 do
      if d(Ix( j)) < d(Ix(i)) then Ix( j) ←→ Ix(i)
```

K nearest neighbor algorithm

The algorithm – ctd.

```
// Calculate number of vectors associated to each class
// for the first K distances d(q), coresponding to
// rearranged indices
vmax = 0;
for c = 1 to C do
  v = 0
  for j = 1 to k do
    if Clasa(Ix( j)) = c then v = v + 1
  if v > vmax then
    Clasa(m) = c
    vmax = v
END.
```

K nearest neighbor algorithm

Remarks

(a) requires in advance specification of the number of classes to be separated;

(b) classification results are influenced by the order of vectors presentation;

(c) the very principle of nearest neighbor compares the current vector with vectors at the limit of the areas associated to each class.

(d) the algorithm does not determine characteristic vectors or prototypes.

C – mean algorithm

Principle

C-means algorithm introduces for the first time the concept of **prototype** or **center-vector** or **encoding vector**, which describes globally a class. Thus, for a class **c** , the prototype $z^{(c)}$ is calculated as the mean of the ***n(c)*** vectors that were associated to that class:

$$\mathbf{z}^{(c)} = \frac{1}{n(c)} \sum_{m=1}^{M} \mathbf{x}^{(m)}$$

∀ **m**, with property **Class (m) = c**

C – mean algorithm

Initialization

The algorithm starts from the training data set $\{x^{(1)}, x^{(2)}, \dots, x^{(M)}\}$ and the number of classes **C < M** to be separated. After a random rearranging of the training data set, first **C** patterns are assigned arbitrarily to the **C** classes, and vectors $x^{(1)}, \dots, x^{(C)}$ become prototypes $z^{(1)}, \dots, z^{(C)}$.

C – mean algorithm

Actual classification

Further, each of the remaining ***M - C*** training patterns is associated to a class based on minimum distances from the ***C*** prototypes.

Next, class-prototypes are recalculated and new class-association are done.

The process repeats until - in two successive iterations - prototypes of classes does not change or changes in a measure considered insignificant.

C – mean algorithm

Precision quantification

The assessment of the size of the space area covered by each class - standard deviation of the class prototype from vectors in the training data set associated to this class:

$$\sigma_c^2 = \frac{1}{n(c)} \sum_{m=1}^{M} \|\mathbf{x}^{(m)} - \mathbf{z}^{(c)}\|^2 \quad \forall\, m, \text{ with property } Class(m)=c$$

The total square deviation:

$$\sigma_T^2 = \sum_{c=1}^{C} \sigma_c^2$$

is a measure of classification accuracy .

C – mean algorithm

The algorithm

1. Input data: training patterns, as input vectors $\mathbf{x}^{(m)}$, $m = 1,\ldots, M$; number of classes C.
2. Randomly rearrange training patterns $\{\mathbf{x}^{(m)}\}$ $m = 1,\ldots,M$.
3. Associated first C patterns to the C classes :

   for $c = 1$ to C do

   $\mathbf{z}^{(c)} = \mathbf{x}^{(c)}$; $n(c) = 0$

C – mean algorithm

The algorithm – ctd.

4. Each training pattern is associated to the closest prototype:

```
for m = 1 to M do
    Dmin = 10^6                    // Initialize minimum distance
    for c = 1 to C do
        if ||x^(m) – z^(c)|| < Dmin then
            Dmin = ||x^(m) – z^(c)||
            Cmin = c
    Class(m) = Cmin
    n(Cmin) = n(Cmin) + 1
```

C – mean algorithm

The algorithm – ctd.

5. Compute new prototypes for C classes:

```
for c = 1 to C do
    w^(c) = 0
    for m =1 to M do
        if Class(m) = c then
            w^(c) = w^(c) + x^(m)
    w^(c) = w^(c) / n(c)           // Temporary prototypes
```

C – mean algorithm

The algorithm – ctd.

6. Compute square standard deviations:

```
σ_T^2 = 0
for c = 1 to C do
    σ_c^2 = 0
    for m = 1 to M do
        if Class(m) = c then
            σ_c^2 = σ_c^2 + ||x^(m) – w^(c)||^2
    σ_c^2 = σ_c^2 / n(c)
    σ_T^2 = σ_T^2 + σ_c^2
```

C – mean algorithm

The algorithm – ctd.

7. Stopping condition:

If prototypes have insignificantly changed, i.e.:

$$||\mathbf{z}^{(c)} - \mathbf{w}^{(c)}|| < \varepsilon \text{ for all classes } c = 1,\ldots, C$$

then the algorithm ends. Elsewhere, new prototypes are stored:

for $c = 1$ to C do $\mathbf{z}^{(c)} = \mathbf{w}^{(c)}$

and the algorithm returns to step 4.

C – mean algorithm

Remarks

(a) requires in advance specification of the number of classes to be separated;

(b) convergence is not guaranteed;

(c) better results than the K nearest neighbor algorithm

(d) produces good results, especially for clearly separable classes.