Deliverable R3–B4–P
Task B4 : Benchmarks

Covering the period from: July, 1st, 1994 to: June, 30th, 1995

ESPRIT Basic Research Project Number 6891

Enhanced Learning for Evolutive Neural Architecture

Project Coordinator
C. Jutten (INPG)

Task leader
A. Guérin-Dugué (INPG)

Authors
F. Blayo (EERIE)
Y. Cheneval (EPFL)
A. Guérin-Dugué, R. Chentouf, C. Aviles-Cruz (INPG)
J. Madrenas, M. Moreno (UPC)
J.L. Voz (UCL)

June 30, 1995

Contents

Chapter 1

Introduction

1.1 Overview of Task B4

Important advances have been made in the field of classifier learning from examples (also called *supervised learning*). Many learning algorithms have been developed and applied in a large variety of fields such as biology, medical science, character recognition, target identification, finance, system failure prediction, ... The principle of classifier learning from examples may be expressed as follows.

> *Given a training set, a set of examples in the form of vectors of attribute values and known classes, induce a theory that can predict the classes of unseen cases coming from the same problem.*

If the search for informative features and the design of an effective classifier are essential steps in pattern recognition system design, the prediction of future classification performance and the comparisons to the performances of other methods is by no means a less important one.

Task B4 of the ELENA project is mainly dedicated to the benchmarking studies of neural network classification algorithms either developed in the framework of the project, either representative of the state-of-the-art in this domain. This report is the result of a strong cooperation between all partners of the project. The different topics of the benchmarking methodology have been the subject of several useful discussions during the partners meetings held in Lausanne, Barcelona and Louvain-La-Neuve (on the occasion of the Elena Industrial Workshop) during the last year of the project. Moreover the coordination of the benchmark task was subject to a constructive bilateral cooperation between the INPG and UCL partners (two one-week visits).

The main objectives of task B4 may be summarized as follows:

- to provide an overall comprehensive view of the general problem of comparative benchmarking studies and to propose a useful common test basis for existing and further classification methods,
- to obtain objective comparisons of the different chosen classifiers on a set of databases as more as possible representative of the most common real case prob-

lems (each classifier being used with its optimal configuration for each particular database),

- to study the possible links between the data structures of the databases viewed by some parameters, and the behavior of the studied classifiers (mainly the evolution of their the optimal configuration parameters).

- to study the links between the preprocessing methods and the classification algorithms from the performances and hardware constraints point of view (especially the computation times and memory requirements).

Most of the classifiers which were used for the benchmark comparative studies are are well known by the neural network and machine learning community. These are the k-Nearest Neighbour **KNN** classifier, selected for its powerful probability density estimation properties; the Gaussian Quadratic Classifier **GQC**, the most classical statistical parametric simple classification method; the Learning Vector Quantizer **LVQ**, a powerful non-linear iterative learning algorithm proposed by Kohonen [1]; the Reduced Coulomb Energy **RCE** algorithm, an incremental Region Of Influence algorithm [2]; the Inertia Rated Vector Quantizer **IRVQ** and the Piecewise Linear Separation **PLS** classifiers, developed in the framework of the Elena project.

In order to obtain comparable and easy to replicate results, all benchmarks of the ELENA project where performed in the *PACKLIB* programming environment (see [3] or [4]) under exactly the same test conditions. For this, the BENCH module has been the reference tool for the benchmarking studies of the classifiers modules. This module, extensively described in the R3-B3 report has been developed in order to provide a common test basis to researchers involved in machine learning and unsupervised classification problems.

For example, the test setup (machine used, database, type of test, parameters used for the classifier,..) and performance statistics (computed confusion matrix with 95% confidence intervals, averaged error, standard deviation on the error,...) are automatically saved in an ASCII easy-to-read *log* file from which it is always possible to repeat the same test in the same conditions. The interest of this kind of tool, developed in the Packlib modular environment spirit, is that users could always compare existing machine learning algorithms on their own datasets according to the Elena benchmarking methodology (and even setup their own new methodology); and researchers who could wish to experiment new algorithms or make modifications to existing ones could always process benchmarks of their new classifier modules in order to compare their results with the available one in a reliable way [1].

In the next section, we will focus on the general problem of classifier design in order to recall the design methodology, introduce the terminology used in this report and position the different topics of task B4 in this general context. The next chapters treats of databases preprocessing and data structure analysis methods selected by the partners (chapter 2); of the performance evaluation and the benchmarking methodology setup in order to obtain useful and comparable results (chapter 3). Chapter 4 is devoted to a recall of the ELENA databases used for the benchmarks (general information on

[1] the only new code to write being the one of the classification algorithm

these databases being available in the B1 report) and integrate new parameters to try to characterize the databases before the classification tasks (inertia, local dimension, overlapping,...). Chapter 5 provides for each selected classification algorithm, a complete individual study relative to its parameters configuration, hardware constraints and to guidelines for the choice of optimal parameters configuration according to the database characteristics. The performance comparisons of the different classifiers on the ELENA databases is provided in chapter 6 and the last chapter is dedicated to conclusions and guidelines resulting from these benchmarks.

1.2 The design of a classifier

1.2.1 Classification

To decide if a machine is in a normal or abnormal operation mode by observing its behavior over a period of time we have to collect different "useful" measures about this behavior and to apply to these measures some kind of discriminant function. This corresponds to a classification problem [5]. Another example is the recognition of printed characters which corresponds to the classification of geometric figures characterized by some measurable parameters.

So we have first to collect a set of d observable characteristics of the sample. These d measures form a d-dimensional vector (or sample) which is a *random vector* U since the measures are different each time an observation is made. Many observations u corresponding to a given class ω_i (the normal or the abnormal machine behavior, a given character, ...) form this class distribution in the d-dimensional space, each class having its own probability density function $p(u|\omega_i)$.

If we know or learn these distributions from past experience, we can set up boundaries between the different class distributions and define a discriminant function f which divides the d-dimensional space into as many regions as there are classes. This discriminant function defines a classifier or categorizer. Thus, in order to design a classifier, we have to find a proper discriminant function from the study of the characteristics of the distribution of U in each class. This is the *learning process* and the samples used to design the classifier are called the *learning or training samples*, picked up in the *learning or design set*.

The theoretically best classifier, assuming that the distributions of the random vector U are known , is the *Bayesian classifier*, which minimizes the probability of misclassification and thus has the smallest possible error (the *Bayes error*) according to the distributions. These distributions are never known in practice and the only thing we have at our disposal is a set $D_N = \{x(n), \omega_{x(n)}, 1 \leq n \leq N\}$ of vectors $x(n)$ and their associated known classes $\omega_{x(n)}$ (subsequently called the *database*).

1.2.2 Number of samples, data space dimension

With a finite number of samples, the parameters used to build the classifier become random variables and subsequently, the classification error becomes also a random variable, biased with a variance. The number of available samples in the learning set is thus very important.

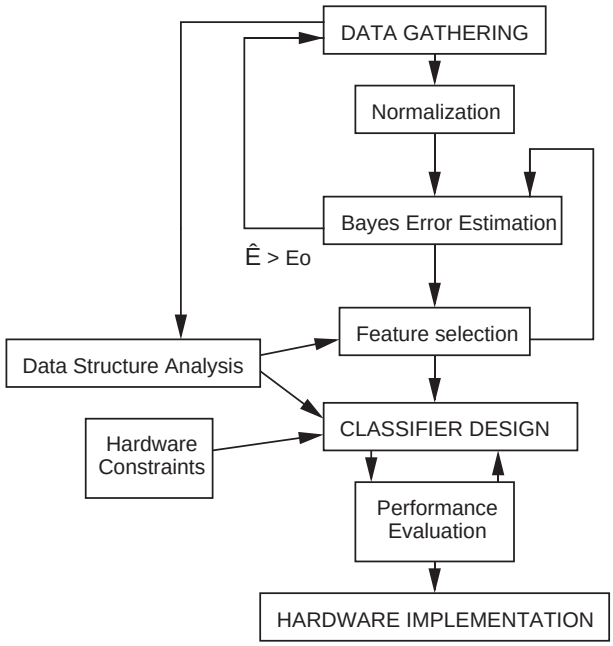

Figure 1.1: The process of a classifier design.

From distributions which are only known from a set of representatives, it is possible to find an upper bound of the Bayes error by the use of non-parametric techniques such as the *kernel estimator* or *k-nearest neighbor* approaches for the estimation of the density functions. These methods require a computational burden and a memory use that is unrealistic in practical situations where the design set is large and the classification decision must be taken in real time or in a relatively short time; they could thus be used in order to estimate the upper bound of the Bayes error to ensure that the gathered data carry enough classification information to meet the specification, but never to build a practical classifier according to restrictive implementation constraints (mainly hardware constraints).

Usually, in order to ensure that the measures carry the whole classification information contained in the original data, the data space dimension becomes rapidly high. This high-dimensionality makes the classification problem very difficult and the necessary size of design set rises exponentially with the dimension for equivalent performances (this is the *empty space* phenomenon presented in [6] and [7]). We have thus to select or extract important features from the observed samples. This is the *feature selection or extraction process*. It can be considered as a mapping from the original d-dimensional space to a lower-dimensional feature space, which should be carried out without severely reducing the class separability.

1.2.3 The classifier design chain

Figure 1.1 presents the general method used in the classifier design process. After data is gathered, a normalization process takes place (note that different data require different kinds of normalization). The data structure analysis may be processed and the class-separability of the data is then measured by estimating the Bayes error $\hat{E}$. If

this error is larger than the final classifier error we wish to achieve (Eo) the data does not carry enough classification information and we must go back to data gathering and find better measures. For the ELENA project, this estimate of the Bayes error $\hat{E}$ has been computed using the *k-NN* classifier with a Leave-One-One test either on the original databases, either on the *CR* databases. results are provided in the R3-B1 report.

When the estimate of the Bayes error is acceptable the data structure analysis provides information for the feature selection. The Bayes error in the feature space is estimated to measure the classification information lost in the feature selection process. The best classification algorithms candidates are then chosen according to the data structure analysis and the hardware plan for the classifier implementation.

After having designed the classifier, its performances must be estimated by some useful evaluation methods. There must be a compromise between the amount of data from the database used for the classifier design (learning set) and for its performances evaluation (see chapter 3 for more details). The resulting errors are compared together with the Bayes estimate and with the results of other classifiers. If the performances are acceptable, the best candidate for the hardware implementation is selected.

1.3 The Elena databases

1.3.1 General presentation

Testing existing and new developed algorithms requires to have databases at disposal, on which tests and benchmarks of the methods can be realized.

Some database sets where recently proposed as standard benchmark sets for classifier learning. Let us cite for example [8] and also the Proben1 set [9] which is a collection of problems for neural network learning in the realm of pattern classification and function approximation plus a set of rules and conventions for carrying out benchmark tests with these or similar problems. Since these sets where not available at the beginning of the project, the ELENA partners set up their own database set, the choice having been guided by various parameters, such as availability of published results concerning conventional classification algorithms, size of the database, number of attributes, number of classes, overlapping between classes and non-linearities of the borders,...

The databases selected for the ELENA benchmarks are described in detail in the R3-B1 report while chapter 4 of of this report provides a more technical study of their data structure analysis (new parameters to try to characterize the databases before the classification tasks: inertia, local dimension, overlapping,...). Here is a brief reminder about these databases. They are splitted into two parts: the *artificial* ones, being generated in order to obtain some defined characteristics, and for which the theoretical Bayes error can be computed, and the *real* ones, collected in existing applications, mainly coming from the UCI repository of machine learning databases [10].

- The *artificial* databases were generated according to the following requirements:
  - heavy intersection of the class distributions,
  - high degree of nonlinearity of the class boundaries,

- various dimensions of the vectors,
- already published results on these databases.

They are restricted to two-class problems, since we believe it yield answers to the most essential questions [11]. The artificial databases are mainly used for rapid test purposes on newly developed algorithms.

- The *real* databases were selected according to the following requirements:
  - classical databases in the field of classification (example: *Iris*),
  - already published results on these databases (example: *Phoneme*, from the ROARS ESPRIT project [12], *Satimage* from the STATLOG ESPRIT project [13] [14]),
  - various dimensions of the vectors,
  - sufficient number of vectors (to avoid the "empty space phenomenon").

1.3.2 Terminology

In order to avoid repetitive definitions and/or some misunderstanding problems we briefly present here the terminology used for the databases and their preprocessings.

Each database is referenced by its name, either in italic either in quotes. We will thus talk of the *Clouds*, *Concentric*, *Gaussian*, *Iris*, *Satimage*, *Phoneme* and *Texture* databases. The artificial "Gaussian" database is made up of seven databases corresponding to the same problem, but with dimensionality ranging from 2 to 8. These seven databases are named "gauss_iD", where i corresponds to the dimensionality of the data.

The names of datasets resulting of a preprocessing applied to original databases is always obtained by adding a suffix to the database name. Preprocessings only concerns the *real* databases. The data are centered and reduced for normalization . The resulting datasets are called the "CR" databases and their names are thus *Iris_CR*, *Satimage_CR*, *Phoneme_CR* and *Texture_CR*. For the "Satimage" and "Texture" databases the Principal Component Analysis (PCA) and Discriminant Factorial Analysis (DFA) were used as preprocessing on the "CR" datasets. The resulting datasets are called the *Satimage_PCAi*, *Satimage_DFA* , *Texture_PCAi* and *Texture_DFA* databases, where *i* indicate the dataset dimension (for example, *Satimage_PCA18* is a dataset of 18 dimensions with the 18 first principal components of the "Satimage_CR" database as attributes).

Chapter 2

General preprocessing and data structure analysis

2.1 Normalization and feature selection

The goal of feature selection is to find the minimum number of features that retains all information needed for an accurate classification. Adding more and more features can increase the recognition rate if they are carefully chosen, but also increase the complexity of the application.
Some preprocessing realizes a feature selection but unfortunately does not always extract the "optimal" features for classification. Let us consider here three classical linear preprocessing methods applicable before classification : normalization by reduction, Principal Components Analysis (PCA) and Discriminant Factorial Analysis (DFA).

By the way of a normalization process, each original feature will have the same importance in a subsequent classification process. A typical method is first to center each feature separately an than to reduce it to a unit variance; this process is recommended when the original features have very different scales. It has been applied on all the *real* Elena databases in order to build the "CR" databases.

The Principal Components Analysis is a very classical method in pattern recognition [15]. PCA reduces the sample dimension in a linear way for the best representation in lower dimensions keeping the maximum of inertia. The best axe for the representation is however not necessary the best axe for the discrimination. After PCA, features are selected according to the percentage of initial inertia which is covered by the different axes and the number of features is determined according to the percentage of initial inertia to keep for the classification process. This selection method has been applied on the *satimage_CR* and *texture_CR* databases in order to build the "PCAi" databases. When quasi-linear correlations exists between some initial features, these redundant dimensions are removed by PCA and this preprocessing is then recommended. In this case, before a PCA, the determinant of the data covariance matrix is near zero; this database is thus badly conditioned for all the process which use this information (the quadratic classifier for example).

The Discriminant Factorial Analysis can be applied to a learning database where each learning sample belongs to a particular class [15]. The number of discriminant features selected by DFA is fixed in function of the number of classes (c) and of the

number of input dimensions (d); this number is equal to the minimum between d and $c-1$. In the usual case where d is greater than c, the output dimension is fixed equal to the number of classes minus one and the discriminant axes are selected in order to maximize the between-variance and to minimize the within-variance of the classes. The discrimination power (ratio of the projected between-variance over the projected within-variance) is not the same for each discriminant axis: this ratio decreases for each axis. So for a problem with many classes, this preprocessing will not be always efficient as the last output features will not be so discriminant. This analysis uses the information of the inverse of the global covariance matrix, so the covariance matrix must be well conditioned (for example, a preliminary PCA must be applied to remove the linearly correlated dimensions). The DFA preprocessing method has been applied on the *satimage_PCA18* and *texture_PCA18* databases in order to build the *satimage_DFA* and *texture_DFA* databases, having respectively 5 and 10 dimensions [1].

2.2 Vector Quantization as preprocessing

Vector quantization techniques are used to reduce the number of samples of a dataset. The underlying distribution of the codebook is roughly the same as the original one. These techniques are very often used in many application areas as for example image compression and classification. For classification purpose, the nature of the probability density functions of the prototypes after a vector quantization is more important because vector quantization produces the learning samples on which the discriminant function will be estimated.
From the computational point of view, vector quantization has two advantages :

- It reduces the computing time and
- It reduces the memory requirement.

From the theoretical point of view, the most important questions are :

- *How to choose the appropriate vector quantization algorithm for preprocessing before classification ?* For example, the widely used LBG methods [16] and its adaptive version (the Competitive Learning [17]) minimize the same quadratic criterion, but converge in practice towards slightly different solutions.
- *How to choose the number of prototypes ?* If the number of prototypes is too large, the process losts its advantages and if it is too small, the selected prototypes don't correctly match the input distribution and the error rates will be too important. The number of prototypes to select depends on the number of input samples and on the structure of the database (number of modes, ...). Moreover, it is necessary that the number of input samples is already sufficient for a "good estimate" of the *pdf* inside each class.

To determine experimentally the number of prototypes, some practical approaches can be used :

[1] The *satimage* database having 6 classes and *texture* 11

- To make several vector quantizations with a growing number of prototypes. If the curve of the mean distortion between the input samples and the prototypes doesn't decrease from a particular number of prototypes, this number can be viewed as sufficient.
- To make several vector quantizations and classification with a growing number of prototypes. If the curve of the error recognition doesn't decrease from a particular number of prototypes, this number can be viewed as sufficient (see figures 5.34, 5.35 and 5.28).

The aim of the VQ preprocessing and the chosen test method are explained in more details in the last section of chapter 6 together with the comparative studies of the classifiers performances for this preprocessing.

2.3 Data structure analysis

Database, preprocessing and classifier are a whole for an application in pattern recognition. To understand which is the best classifier and which is the more suitable preprocessing technique for a given data base, it is important to "understand" the underlying structure of the input samples. When the input dimension is greater than three, our usual ways of representation are unusable. The input databases have to be characterized by a set of relevant structural parameters.

In [14] Michie and al. describe three types of measures to characterize a dataset: (i) very simple measures (number of samples, of attributes and of classes); (ii) statistical measures (homogeneity of covariances, departure from normality, canonical discriminant correlations (related to the DFA), ...) and (iii) information theory measures (entropy of attributes, entropy of classes, noisiness of attributes, ...). Other useful characterization parameters may also be found in [8].

Let us consider here four particular useful parameters :

- the fractal dimension,
- the relative number of available samples,
- the inertia (within, between, global) and
- the dispersion and the overlapping rate between classes.

2.3.1 Fractal dimension

Introduction

In real applications, the data dimension (or freedom degrees) of the problem is in general a parameter not a priori known. Data are obtained in order to characterize the problem and a number of characteristics or features large enough to be able to extract the relevant information is required. Each feature adds a new dimension in the input space.

A classifier is designed for input samples in dimension d. This samples distribution can have intrinsically less than d degrees of freedom. The intrinsic dimension lower or

equal to d is the dimension of the submanifold structure of the data [18]. A knowledge about this local dimension of the collected data would be highly desirable because this information can be used by preprocessing methods that simplify the problem by the way of database dimension reduction. Nervertheless, the choice of a suitable preprocessing which will produce the adequate projection remains an important problem because this projection can be a non-linear one.

The fractal dimension give a local measure of the local data dimension [19]. Numerous definitions of the fractal dimension have been proposed [20]. We consider here the similarity dimension.

The dimension of an object inside a n-dimensional space can be calculated as follows. Let us divide the space containing the data in hypercubes of size r, and let us increase the number of hypercubes by decreasing r. The number of hypercubes in the data space increases as r^{-n}, while the number of hypercubes containing samples of the object grows proportionally to r^{-d}. The intrinsic dimension of the object is then said to be d. For instance, if the object is a line, the growth is linear because the number of hypercubes containing the line expand in one dimension. For a plane it is a power of two. For a fractal object, that can better represent the complex nature of real objects, this number has not to be an integer. So, the calculated figure is called *fractal dimension* (FD).

The calculation of the FD of an object can be performed in different ways [21]. The practical measurement of the FD is performed by counting the number of hypercubes $N(r)$ that contain samples in a space divided in hypercubes of side r ("Box counting method"). The similarity dimension (or FD) is then obtained as

$$d = \lim_{r \to 0} \left[\frac{\log N(r)}{\log(1/r)} \right] \tag{2.1}$$

This represents the slope of $\log N(r)$ as a function of $\log(1/r)$. In the case of a self-similar object, that is, an object which remains (statistically) similar with independence of the scale of observation, the FD keeps constant, regardless of the scale. On the other hand, if this property does not hold, the FD depends on the scale. For instance, a bounded object has FD zero for values of r large enough (the object is included in just one hypercube), but as r decreases the FD varies. What is seen as a point at a very large scale, could appear as a sphere (3-D) at some smaller scale or as a strongly intertwined line (1-D).

Several problems get difficult the interpretation of this fractal dimension :

- On noisy data, the data dispersion changes the estimation of the intrinsic dimension.
- The number of samples may be insufficient for a reliable estimation of the fractal dimension.
- Which is the scale of the size r on which the slope of the regression line must be computed ?
- A database can held different modes with different dimensions (various structures for the separate classes). In that case, these dimensions overlap in their calculation.

Practical measurement of FD

As stated, the method of hypercube counting is not suitable to obtain the FD if r is too large, since the dimension value tends to zero. On the other hand, real databases are not continuous. The clouds of samples that build the database become isolated as r tends to zero, and the FD for r too small becomes zero again. The noise dimension may also overlap the true data dimension for small r. The information of the database FD has thus to be found for intermediate values of r, and in some cases the exact value is rather difficult to obtain because it depends much on the interpretation. In order to provide an aid to the decision, the curve of $\log(N(r))$ against $\log(1/r)$ of the databases is plotted. Then, from the observation of the curve slopes a reasonable guess of the FD can be done.

In order to calculate the similarity dimension of equation 2.1, an efficient algorithm developed in [22] has been used. The basic idea is the generation of a unique code for each elementary hypercube, and then counting the number of different codes generated by the distribution samples. The algorithm may be described by the following steps:

1. All the distribution points are set non-negative by translation. The initial hypercube size r_0 that covers completely the database is calculated.

2. A new r is obtained in a logarithmic scale.

3. For each sample vector ξ_i, a code $c_i = \{ci1, ci2, ..., cin\}$ is built as the integer division of each component by r:

$$c_{ik} = [\frac{\xi_{ik}}{r}] \tag{2.2}$$

   This indicates the position of the hypercube to which sample ξ_i belongs.

4. The number $N(r)$ of different c_i codes is counted.

5. For $N(r) < M/2$, go to step 2. Since M is the number of samples, it seems a reasonable condition to stop the algorithm when a mean value of 2 samples is inside a hypercube.

6. Finally, a linear regression is performed to obtain the mean slope of the log-log curve of the pairs $(r, N(r))$, that is, the mean slope of $\log N(\log r_0/r)$.

This algorithm has been implemented as a Packlib module. It has been applied to the most significant real databases used in the ELENA project, that is: Phoneme, Satimage and Texture. The practical results will be discussed at section 4.2.

2.3.2 Relative number of available data

In [23] and [6], the minimum number of samples necessary for the estimate a *pdf* of a Gaussian underlying density by kernel functions within 10% of error have been evaluated by :

$$log_{10}N \approx 0.6(d - \frac{1}{4}) \tag{2.3}$$

For classification methods based on kernel estimates (as RBF networks), it is interesting to know how the number of available learning samples is near or not to this theoretical minimum number of samples. In the case where the number of available samples is too small, preprocessing for dimension reduction should be used. Figure 4.6 shows an example for the class 4 of the Satimage database.

2.3.3 Inertia (within, between, global)

Inertia is a classical measure for the variance of high dimensional data.
We distinguish :

- the global inertia, which is computed over the whole database -I_G-,
- the within-class inertia, which is the weighted sum of the inertia computed on each class (the weight is the a priori probability of each class) -I_W-,
- the between-class inertia, which is the inertia computed on the centers of gravity of each class -I_B-.

Let us consider a database of N samples, with c classes, each feature having been centered and normalized. For the class ω_i, the number of samples is N_i and the center of gravity is $\vec{g}_i$. We have :

$$
\begin{aligned}
I_G &= \frac{1}{N}\sum_{i=1}^{N} ||\vec{x}(i)||^2 \\
I_{\omega_i} &= \frac{1}{N_i}\sum_{l=1}^{l=N_i} ||\vec{x}(l)-\vec{g}_i||^2;\ \forall \vec{x}(l)\,|\,class(\vec{x}(l))=\omega_i \\
I_W &= \frac{1}{N}\sum_{i=1}^{i=c} N_i I_{\omega_i} \\
I_B &= \frac{1}{N}\sum_{i=1}^{i=c} N_i ||\vec{g}_i||^2
\end{aligned}
$$

where $||.||^2$ is the square of the Euclidean norm: $||\vec{x}||^2 = \vec{x}.\vec{x}^t$

2.3.4 Dispersion and overlapping

The classification performances depend on the discrimination power of the features. For the databases, the overlapping between the classes is more or less oimportant. We can measure discrimination or overlapping index. Classical measures for discrimination or separability are the Fisher criteria (FC) and the divergence or the Bhattaharyya distance [24].

We have used here the Fisher criteria and simple measures for the dispersion as the mean dispersion of class ω_i in class ω_j by (the notations are the same as here above) :

$$FC = \frac{I_B}{I_W} \tag{2.4}$$

$$Dispersion(i,j) = \frac{||\vec{g}_i - \vec{g}_j||}{\sqrt{I_{\omega_j}}} \tag{2.5}$$

Class	*1*	*2*	*3*	*4*	*5*	*7*
1	0.0	1.7673	2.1275	1.5894	1.6166	2.2870
2	2.4216	0.0	4.9302	4.1477	2.3935	4.2002
3	1.4792	2.5017	0.0	1.6006	2.3436	2.8327
4	1.1845	2.2559	1.7157	0.0	1.3250	1.2788
5	1.7910	1.9352	3.7342	1.9697	0.0	1.0246
7	1.7495	2.3449	3.1166	1.3127	0.7075	0.0

Table 2.1: Dispersion matrix of the *satimage_CR* database

Class	*1*	*2*	*3*	*4*	*5*	*7*
1	98.1	0.2	1.1	0.1	0.5	0.0
2	0.0	96.5	0.1	0.7	2.0	0.7
3	0.5	0.1	93.4	4.6	0.0	1.4
4	0.0	0.8	13.7	70.6	0.8	14.1
5	3.1	0.8	0.1	0.8	89.7	5.5
7	0.0	0.1	1.9	7.3	2.0	88.7

Table 2.2: Best confusion matrix obtained by the KNN classifier ($K = 3$) with a Leave One Out method on the *satimage_CR* database. Averaged error: 8.89%

The discrimination is better if the Fisher (FC) criterion is large. If the dispersion measure between two classes is large, these classes are well separated, the between-class distance is more larger than the mean dispersion of the classes. If this measure is close to or lower than one, the classes are very overlapped, but, that doesn't necessary mean an important confusion between these two classes from the classification point of view. For example, it will be the case for multimodal classes. It is thus necessary to complete this measure with a reliable estimate of the Bayesian confusion matrix [2]. For this, a simple classification test with the KNN classifier as best practical reference for the Bayes error can be used in order to evalute the confusion between the classes.

Let us take an example with the "Satimage" database. From the dispersion matrix computed from equation 2.5 (table 2.1), we see that class 2 has few overlapping with classes 3, 4 and 7. This is confirmed by the confusion matrix (table 2.2). Moreover classes 5 and 7 seems to be overlapped, but this is not confirmed by the confusion matrix, these classes can thus be multimodal or very elongated.

[2] See chapter 3 for a definition of the confusion

Chapter 3

Performance evaluation

3.1 Introduction

A classifier is always defined by its discriminant function f which divides the d-dimensional space into as many regions as there are classes. If there are c classes ω_i, $1 \leq i \leq c$, the discriminant function may also be expressed by the c *class indicating functions* of the classifier f_i, where $f_i(u) = 1$ if $f(u) = \omega_i$ and $f_i(u) = 0$ otherwise.

The most useful way to illustrate the performances of a classifier for a given problem is to provide its confusion matrix $C(f)$. Each line of this confusion matrix is dedicated to a particular class; for a given line each element provides the probability (expressed here in percent) for patterns of this class to be attributed to any other or to the original class. We thus have $C_{ij} = 100\, Pr\{\omega_j \text{ assigned when } \omega_i \text{ is true}\}$:

$$C_{ij} = 100 \int p(u|\omega_i) f_j(u) du \tag{3.1}$$

where $p(u|\omega_i)$ is the probability density function of the points belonging to class ω_i. The best confusion matrix is the one corresponding to the Bayesian classifier (minimal attainable classification error Eo). Table 3.1 presents the confusion matrix of an hypothetic classifier on a 3-class problem. The classifier performance may also be expressed by the averaged classification error

$$E(f) = \sum_{i=1}^{c} P_i (\sum_{j \neq i} C_{ij}(f)) \tag{3.2}$$

where P_i is the a priori probability of class ω_i.

In the field of classifier learning from examples, the statistics of the problem ($p(u|\omega_i)$ and P_i) are never known and the exact confusion matrix of a classifier $C(f)$ can only be estimated over a set of patterns picked up in the database of available samples (the *test set*). This estimate is called the "apparent" confusion $\hat{C}(f)$ in opposition to the

Class	*0*	*1*	*2*
0	96.6	3.4	0.0
1	2.5	97.5	0.0
2	0.0	0.0	100.0

Table 3.1: A confusion matrix.

exact confusion $C(f)$ of the classifier which is never known. We speak in this case of *error counting methods* and the apparent confusion matrix is obtained by:

$$\hat{C}_{ij} = \frac{100}{N_i} \sum_{k=1}^{N_i} f_j(x(k)) \tag{3.3}$$

where $x(k), 1 \leq k \leq N_i$ are the point of the testset belonging to class ω_i.

From this matrix, the apparent mean classification error $\hat{E}(f)$ is computed by the same method as here above, using for P_i the values estimated from the testset: $\hat{P}_i = N_i(test)/N(test)$.

In practice, the amount of available samples is always finite and is frequently smaller than one would like. It is thus important to utilize at best the amount of available samples for a good design of the classifier and in order to have sufficient confidence in the performance prediction. In [5] Fukunaga shown that the size of the learnset alone account for the degradation in a classifier performance, while the size of the testset dominate the variance of the error estimate.

The most important error-counting methods used for performances estimation are detailed in the following section.

3.2 Error counting methods

3.2.1 The Resubstitution method

The procedure which consists in estimating the classifier and testing its performances from the same data set is called Resubstitution in statistics. It is now quite well known that performances computed with this method are optimistically biased: the nice performance figures on the design set do not extend to independent test sets: it is the overfitting problem.

3.2.2 The Holdout method

Since the bias of the resubstitution method has been discovered, cross-validation methods have been proposed. They are all based on the same principle: there should be no common data in the learning and in the test sets. In this family is the Holdout, which builds one partition in the set of available patterns into two mutually exclusive subsets (the *learnset* to build the classifier and the other one to test it : the *testset*). In order to make the result less dependent on the partition, one can average several Holdout results, by building several partitions (randomly, or exhaustively drawn); this gives the Averaged Holdout method.

Since a classifier designed on the entire data set will, on average, perform better than a classifier designed on only a part of the data; this method has a definite tendency to over-estimate the actual error-rate.

For the averaged Holdout method the mean value of the error on the different testset may be computed with its standard deviation. For the comparison of different classification algorithms on the same problem, this standard deviation may be seen an image of the algorithm robustness: if the error on the different testsets with the classifiers build on the different learnsets is very different from one test to another the

classification algorithm is not robust to a change of the learnset taken from the same database.

3.2.3 The Leave-One-Out method

The Totally Averaged Leave-One-Out method may be seen as an application of the powerful Jackknife principle to confusion evaluation. In this approach, if N samples are available, N partitions are formed by leaving one single pattern for testing, and using the remaining $N-1$ to build the classifier. The N performance results obtained this way are then averaged. It has been proven that the Leave-One-Out method, like the Holdout, gives in fact an upper bound of the error probabilities, but the estimate is much better, since the learnset sizes are the size of the databases less one. Since the resubstitution method gives a lower bound of these probabilities, the true performance lies in principle in between. The Totally Averaged Leave-One-Out method is a particular case of the Leave-k-Out method explained below.

3.2.4 The Leave-k-Out (rotation or m-fold cross validation) method

This method is a compromise between the Holdout and the Leave-One-Out estimate. In this case, $m = N/k$ different partitions are formed by leaving k patterns for testing, and using the remaining $N-k$ to build the classifier. The m performance results obtained this way are then averaged. It is clear that when $k = 1$, this method reduces to the Leave-One-Out, whereas when $k = N/2$ it reduces essentially to the Holdout method where the roles of the learn and test sets are interchanged. The rotation method reduces both the bias inherent to the Holdout method and the computational burden of the Leave-One-Out.

3.2.5 The bootstrap method

The bootstrap method proposed by Efron [25] is another non-parametric procedure for estimating parameters and error-rates in particular. For the bootstrap method, the basic idea is to repeat several times the whole classification experiment a large number of time and to estimate quantities like bias from these replicate experiments, each experiment being proceed on the basis of a new learning set obtained by resampling with replacement in the original database. Since this method is not very easy to implement while its estimate of the error has been shown to be statistically equivalent to the Leave-One-Out error estimate [5], it has not been used in the ELENA project.

3.3 Performances comparisons and confidence intervals

Many supervised learning and classification related papers give comparative experiment results by simply ranking p preselected classifiers (designed on a single learning set) on the basis of their respective mean prediction errors on a Holdout testset. This is not very reliable on the statistical point of view, because we have no idea of the variability of these performance estimation (nothing ensures that the same ranking will be obtained for a different Holdout partition of the database). The neural network community begins now to be more interested by statistics and some recent papers have considered

this problem: the statistical analysis of comparative experiments has been addressed in [26] and the problem of ranking methods by significance testing in [27].

The most easy way to give some idea of the variability of a performance measure is by the mean of confidence intervals (this method has also been presented in [6] and [28]).

Suppose we have a given classifier build on a given learning set. Let us first define a random variable A: for a pattern u to classify, if the answer of the classifier is class ω_i, A takes the value 1; otherwise, if the answer of the classifier is not class ω_i, A takes the value 0. We can say that for a given classifier the probability for A to take the value 1 is p_i and the probability for A to take the value 0 is $1 - p_i$; with p_i unknown. A is thus a discrete random variable with a *Bernouilli* distribution.

Suppose now that we have a set of N patterns to classify; if the probability to attribute class i is obtained by counting $T_i = \sum A$ succes in a testset of size N, then T_i is a $binomial(T_i, N)$ distribution. If N is large enough (greater than 30) the probability density function of B, the succes rate of class i ($B = T_i/N$), may be approximated by a normal distribution of mean p_i and of standard deviation $\sqrt{(p_i(1-p_i))/N}$.

The real value of p_i is never known, but we may use as first approximation, the succes rate of the classifier itself computed from its classification results: $\hat{p}_i = N_i/N$ where N_i is the number of the N points to classify attributed by the classifier to class i. From this estimate $\hat{p}_i$, we can approximate the probability density function of B by a normal distribution of mean N_i/N and of standard deviation $\sqrt{(N_i(N-N_i))/N^3}$. For this type of distribution, we know that the 95% confidence interval for the value of B is:

$$\frac{N_i}{N} - 1.96\sqrt{\frac{N_i(N-N_i)}{N^3}} \leq B \leq \frac{N_i}{N} + 1.96\sqrt{\frac{N_i(N-N_i)}{N^3}} \tag{3.4}$$

So, for a given apparent confusion matrix provided in percent, each element may be seen as a random variable C_{lk} having the following 95% confidence interval:

$$\hat{C}_{lk} - 1.96\sqrt{\frac{\hat{C}_{lk}(100-\hat{C}_{lk})}{N_l}} \leq C_{lk} \leq \hat{C}_{lk} + 1.96\sqrt{\frac{\hat{C}_{lk}(100-\hat{C}_{lk})}{N_l}} \tag{3.5}$$

where $\hat{C}_{lk}$ is the value (in percent) of the considered element of the confusion matrix (line l, column k) and N_l is the number of points of the testset belonging to class ω_l.

The same argument may be applied to obtain confidence intervals on the apparent mean classification error $\hat{E}(f)$. We have in this case:

$$\hat{E} - 1.96\sqrt{\frac{\hat{E}(100-\hat{E})}{N}} \leq E \leq \hat{E} + 1.96\sqrt{\frac{\hat{E}(100-\hat{E})}{N}} \tag{3.6}$$

There exists three types of test for which a confidence interval may be computed by this way on the confusion matrix:

- the resubstitution.
- a single Holdout method without averaging (one learning on a given learnset and one recognition on a testset).

- the Leave-k-Out method with k lower than the database size (N) divided by a large value, say 30 for example. For these values of k, the different classifiers build with the different learnset of size $N-k$ containing the entire database less the k vectors to use for the testset may be considered as identical classifiers, the learning sets being nearly identical. So for this case ($k \leq N/30$) we can proceed as the confusion matrix resulting of the leave-k-out test was the confusion matrix obtained after the recognition of the entire database by a classifier build on a same learning set.

We must keep in mind that if the same test method is used for each classifier, using always the same testset, the comparisons will be more accurate than the error bars suggest. This is the case for the ELENA benchmarks.

More accurate paired comparisons between classifiers results may also be realized by using statistical tests like the *McNemar's test* [29] [30], the *t-test* [27] [31] or the *risk-averaging approach* [28]. These methods were not applied in the framework of ELENA , since they are more complicated and sometimes dedicated to particular test methods.

3.4 Benchmarking studies in the framework of Elena

All benchmarks of the ELENA project where performed in the *PACKLIB* programming environment (see [3] or [4]) under exactly the same test conditions in order to ensure the reliability of the results obtained for the different classification methods.

The general benchmarking method for the "real" databases was the following:

1. All the *artificial* databases ("Clouds", "Concentric" and "Gaussian") were taken "as it" for the benchmarks without any preprocessing. For the *real* databases, the data of each set are first centered and reduced in order to obtain the "CR" databases; no test was performed on the rough *real* databases.

2. The estimate of the Bayes error $\hat{E}$ is then computed by the use of an averaged Leave-One-Out test of the k_NN classifier using the value of the K parameter giving the best result (see the R2-B3 report or chapter 5 for the description of the k_NN classifier). This provides us an upper bound of the lowest attainable error on each dataset.

3. The data structure analysis is performed to extract some useful measures for the dataset characteristics (see chapter 4) and a PCA is then performed on the "Satimage_CR" and "Texture_CR" databases in order to obtain three different "PCAnn" dataset with reduced dimension.

4. Each classifier is then examined in details (by means of averaged Holdout tests on each available database) to study the relations between measures extracted from the data structure analysis and the classifier critical parameter optimum values. From these studies deterministic methods are extracted in order to set the classifier parameters to their "optimal value" in function of the database characteristics (size, dimension, number of classes, fractal dimension, ...).

5. Finally, the test used for the comparisons of the classifier results is performed, using for each classifier the optimal parameters for each particular database. It is the Holdout method averaged over five different partitions of the original database in two independent learnset and testset, each containing half of the total amount of available patterns (patterns used in each partition of the original database in a learnset and a testset being always the same for each particular holdout trial from one classifier to another). The confusion matrix corresponding to each independent Holdout partition is calculated with its 95 % confidence intervals and the average of each mean error is computed.

   This kind of test enables several types of analysis. For each classifier the difference between the minimum and maximum value of the error obtained by several Holdout tests may be taken as a good image of the the classifier robustness to learnset modifications. On the other hand, it is always possible to compute the 95 % confidence interval for the maximum and minimum errors enabling to decide if the difference between different classifiers performances are reliable or not.

Chapter 4

Structure analysis of the Elena databases

In this chapter, information concerning the databases used for the benchmarks will be completed by additional parameters (see section 2.3). These parameters will be analyzed to help the user to characterize the selected features before the classification task. For the basis information about these databases, see the report R2-B1-P.

4.1 Artificial databases

All these databases are in 2 dimensions. Moreover, the "Gaussian" databases are available up to dimension 8. Figure 4.1 summarizes the number of available samples for the artificial databases, compared to the minimum number of samples for the estimation of the *pdf* with kernel functions. The graph concerns the classes with the smallest number of samples.

4.1.1 "Gaussian" databases

This database is used for this benchmark to study the effect of the input dimension on the error rates. The distribution is gaussian for the two classes, and there is a fully overlapping between these classes, the center of gravity is the same for the two classes.

As the dimension increases (from 2 to 8), the number of available samples remains the same, 2500 patterns for each class. Figure 4.1 shows that this number of points is sufficient up to dimension 5.

The best estimate of the error rate by a KNN classifier (LOO test) is given at table 4.1, these errors are compared to the theoretical Bayes error (figure 5.1 plots these results).

Dimension	*2*	*4*	*6*	*8*
Bayes	26.37	17.64	12.44	09.00
KNN	26.7	19.58	16.88	16.12
K	25	21	7	3

Table 4.1: Theoretical Bayes error and experimental results with the best configuration of the KNN classifier (Leave-One-Out estimation) for the "Gaussian" database.

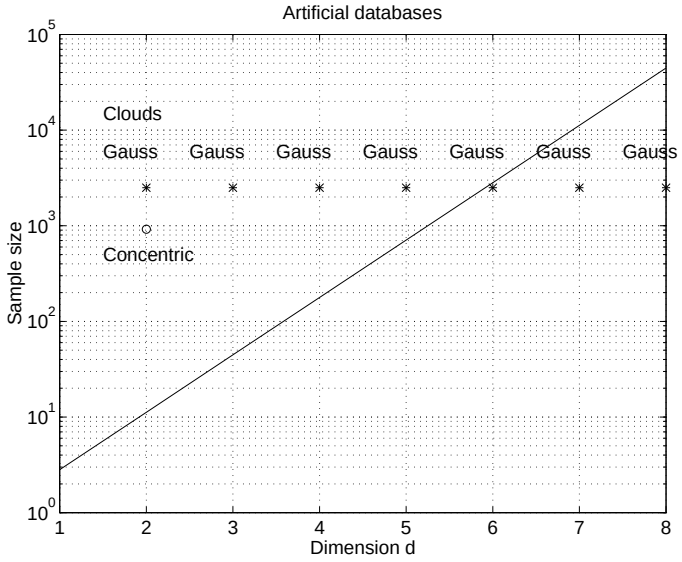

Figure 4.1: Minimum number of points to estimate a *pdf* by kernel functions, and the location of one class for the artificial databases.

Class	*0*	*1*
0	99.1	0.9
1	1.0	99.0

Table 4.2: Confusion matrix with a KNN classifier estimated by a Leave-One-Out method, for the "Concentric" database. Averaged error: 0.96%

4.1.2 "Concentric" database

The samples distribution for this database (dimension 2) is uniform, there is no overlapping, the boundaries are non linear. The theoretical error is 0%.

The best estimate of the error rate by a KNN classifier ($K = 7$) is given at table 4.2.

4.1.3 "Clouds" database

The samples distribution for this database (dimension 2) is gaussian. Class 0 is the sum of three different gaussian distributions, class 1 is a single gaussian distribution. There is an important overlapping between the two classes. The theoretical error is 9.66%.

The best estimate of the error rate by a KNN classifier ($K = 5$) is given at table 4.3.

Class	*0*	*1*
0	92.8	7.2
1	14.7	85.3

Table 4.3: Confusion matrix with a KNN classifier estimated by a Leave-One-Out method. Averaged error: 10.94%

4.2 Real databases

4.3 Real databases

In this project, we have chosen four real databases:

1. Fisher's "Iris". This is a very famous database in Pattern Recognition, and numerous references are available.
2. "Phoneme". This database was in use in the European ROARS Esprit project and presents difficulties for classification [12].
3. "Satimage" is a database in high dimensions (36) already used in the European Statlog Esprit project [13] [14].
4. "Texture" is a database with 11 classes in high dimensions (40) generated by INPG.

For the "Satimage" and "Texture" databases, the different preprocessing will be the Principal Component Analysis, the Discriminant Component Analysis and the Vector Quantization. For the "Phoneme" database, the only preprocessing studied here is the Vector Quantization. The "Iris" database has been only centered and reduced.

From the PCA preprocessing, three dimensions were chosen according to the criterion of inertia and the error of recognition versus the number of dimensions. Let us explain the approach on the "Satimage" database. Figure 4.2 shows the evolution of the inertia and the recognition error, versus the number of features. The inertia increases quickly with the dimensions. With 18 features, the percentage of resulted inertia is just greater 99%. This number of dimensions corresponds also to a beginning of a weak increase of the error rate. We have decided to choose a second configuration with a number of dimensions close to the fractal dimension and in this case, this configuration (8 features) is located almost at the greatest curvature of the two curves. For a third configuration, we have chosen a dimension (4) compatible with the available number of points (figure 4.6).

For the Vector Quantization preprocessing, we have chosen three different codebook sizes according to the hardware constraints point of view (and not according to the best number of prototypes for a given distribution). The selected mean values of the selected number of prototypes per class are respectively 10, 25, 40 and 80 for each particular "VQ" preprocessing. That means that the total numbers of prototypes are 10, 25, 40 and 80 multiplied by the number of classes. A complete description of this procedure and the obtained results will be discussed at section 6.4.

4.3.1 "Iris" database

This database is composed of three classes in 4 dimensions. There are 50 patterns per class. This is a very small database and there are not enough samples. With a representation in two dimensions (after a PCA), we see that class zero is well identified and is linearly separable.

The between-class inertia is 2.86, the within-class inertia is 1.13, and the Fischer's coefficient is 2.52. The dispersion matrix is given at table 4.4. In this table, the fact that class zero is more separated from the others is confirmed.

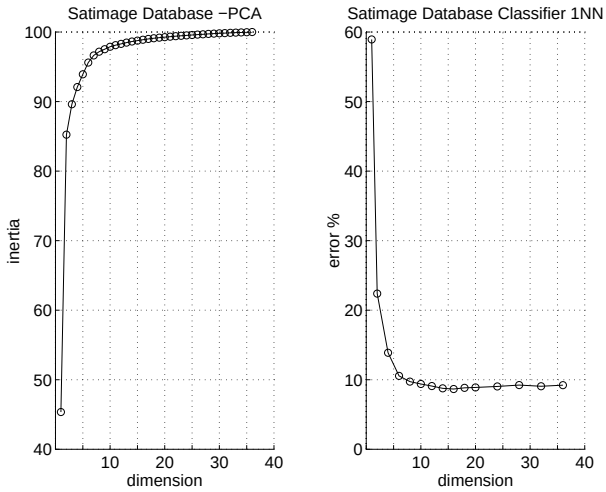

Figure 4.2: Percentage of resulted inertia (left), recognition error with a rule of the first nearest neighbor (right) versus the number of features.

Class	*0*	*1*	*2*
0	0.	2.76	3.36
1	2.85	0.0	1.27
2	4.0	1.45	0.0

Table 4.4: Dispersion matrix computed on the "Iris" database.

The best estimate of the error rate by a KNN classifier ($K = 5$) is given at table 4.5. Class one is recognized without error. The confidence intervals computed with this Leave-One-Out method are important, it confirmes that the number of available samples for this database is too small.

4.3.2 "Phoneme" database

This database is composed of two classes in 5 dimensions. There are 5404 patterns, 3818 for class zero and 1586 for class one. Figure 4.3 shows that the number of samples is just sufficient for this dimension (5).

The between-class inertia is 0.35, the within-class inertia is 4.64, and the Fischer's coefficient is 0.0756. The dispersion matrix is given at table 4.6. We see that with these parameters, there is dispersion between the two classes, and these two classes are overlapped.

The best estimate of the error rate by a KNN classifier ($K = 1$) is given at table 4.7. There is confusion for the recognition of the class one.

The computation of the fractal dimension (figure 4.4) indicates that this distribution

Class	*0*	*1*	*2*
0	100.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
1	0.0 ± 0.0	96.0 ± 5.4	4.0 ± 5.4
2	0.0 ± 0.0	6.0 ± 6.6	94.0 ± 6.6

Table 4.5: Confusion matrix with a KNN classifier estimated by a Leave-One-Out method for the "Iris" database. Averaged error: $3.33 \pm 4.0\%$

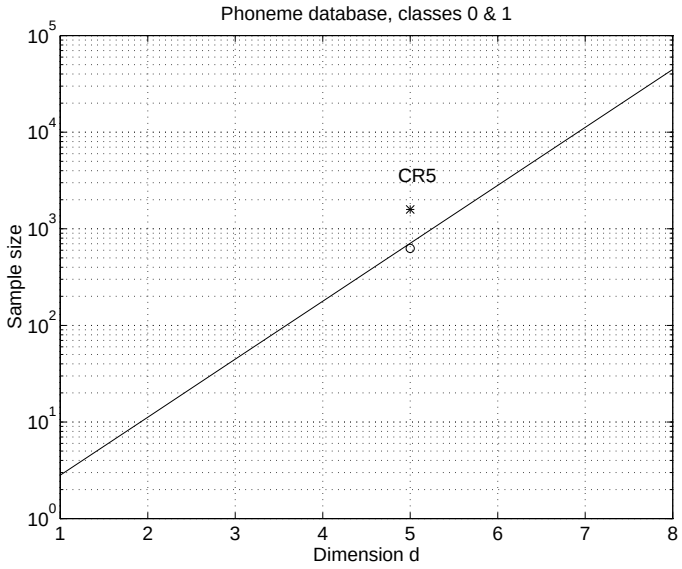

Figure 4.3: Minimum number of points to estimate a *pdf* by kernel functions, and the location of the two classes for the "Phoneme" database.

Class	*0*	*1*
0	0.0	0.62
1	0.60	0.0

Table 4.6: Dispersion matrix computed on the "Phoneme" database.

Class	*0*	*1*
0	95.0 ± 0.7	5.0 ± 0.7
1	18.5 ± 1.9	81.5 ± 1.9

Table 4.7: Confusion matrix with a KNN classifier estimated by a Leave-One-Out method for the "Phoneme" database. Averaged error: $8.97 \pm 1.1\%$

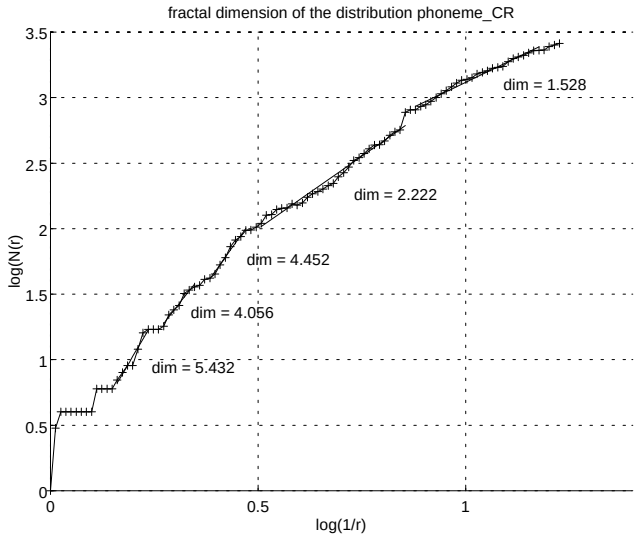

Figure 4.4: Plot of $N(r)$ for the " Phoneme" database.

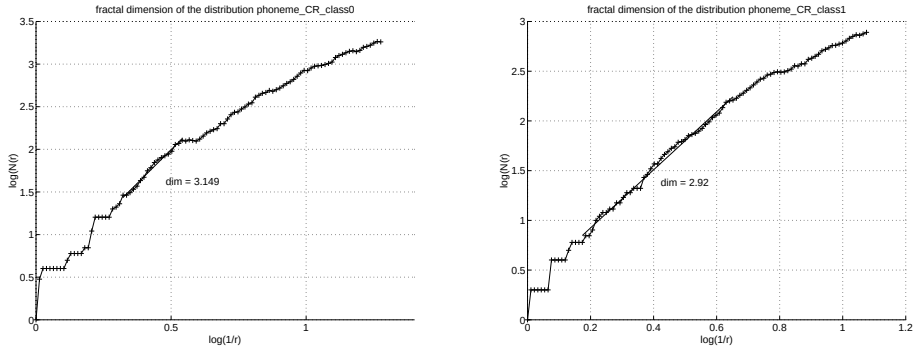

Figure 4.5: Plot of $N(r)$ for class 0 and class 1 of the " Phoneme" database.

of samples would be in a volume in 5 dimensions even if each class is viewed in space with a lower number of dimensions (figure 4.5).

For the vector quantization preprocessing, table 4.8 shows the number of prototypes for the various rates (10, 25, 40 and 80).

4.3.3 "Satimage" database

This database is composed of six classes in 36 dimensions. There are 6435 patterns. Figure 4.6 shows that the number of samples is just sufficient for a base in dimension 5. For greater dimensions, it is a small database. After the various preprocessing, we have databases in 4 (PCA), 5 (DFA), 8 (PCA) and 18 (PCA) dimensions.

For the initial database (after centering and reduction), the between-class inertia is 24.03, the within-class inertia is 11.97, and the Fischer's coefficient is 2.01. This coefficient isn't very modified by the preprocessing. The dispersion matrix is given at table 2.1.

Rate	*Class 0*	*Class 1*	*%*
10	14	6	0.37
25	35	15	0.93
40	57	6	1.48
80	113	47	2.96

Table 4.8: Number of prototypes per class after vector quantization for the "Phoneme" database.

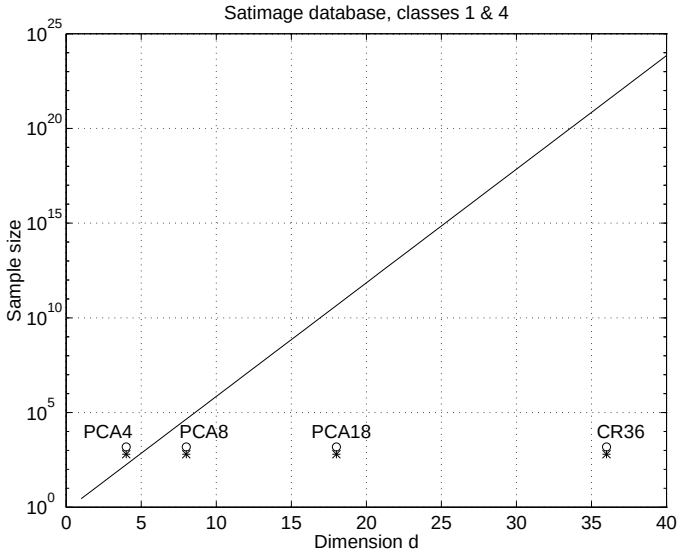

Figure 4.6: Minimum number of points to estimate a *pdf* by kernel functions, and the location of the two classes for the "Phoneme" database.

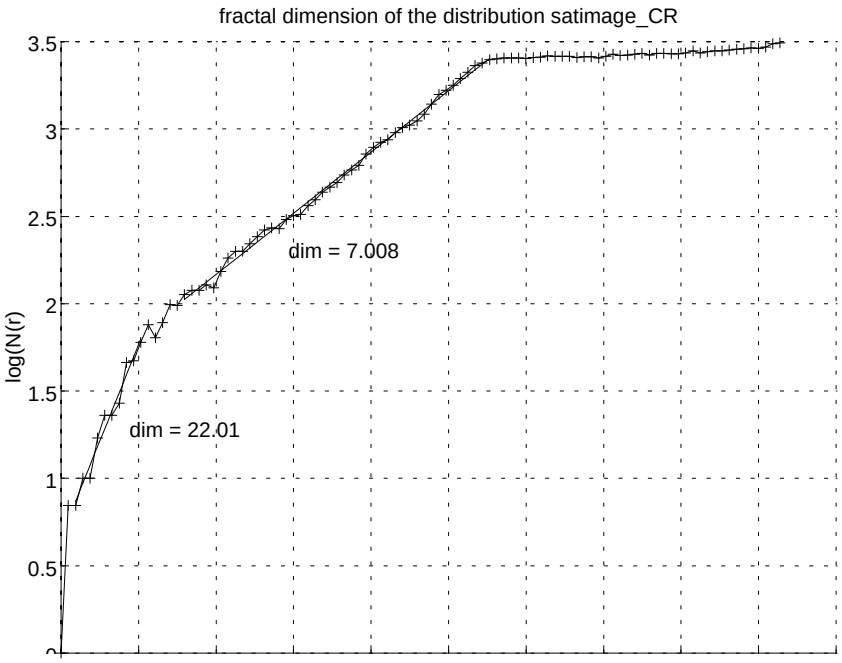

Figure 4.7: Plot of $N(r)$ for the "Satimage" database.

The best estimate of the error rate by a KNN classifier ($K = 3$) is given at table 2.2.

The computation of the fractal dimension (figure 4.7) indicates that this distribution of samples would be in a volume of about 8 dimensions. For class one (maximum number of samples), the slope of the curve seems to be greater than 10 (figure 4.8 left) and for class four, the number of points isn't sufficient to identify a straight line on the curve (figure 4.8 right)

Table 4.9 summarizes the different fractal dimenstion estimates for each preprocessing. For the vector quantization preprocessing, table 4.10 shows the number of

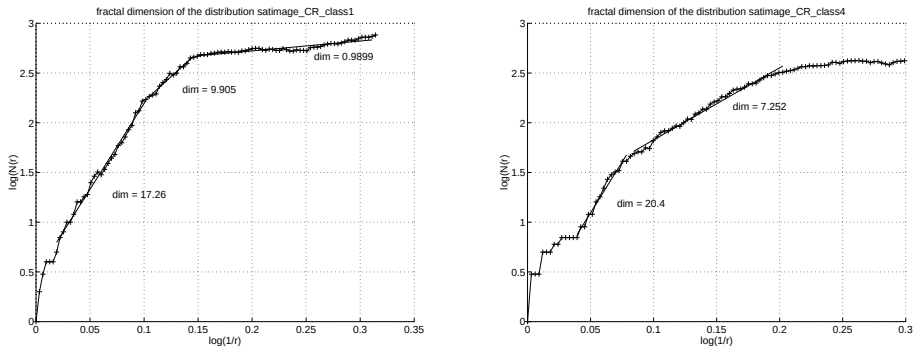

Figure 4.8: Plot of $N(r)$ for class 1 and class 4 of the " Satimage" database.

Preprocessing	*Database FD*	*Class 1 FD*	*Class 4 FD*
Satimage_CR	8	10	8
Satimage_PCA18	10	4	4
Satimage_PCA8	5	3.0	4
Satimage_PCA4	3	2.0	2
Satimage_DFA	3	3	3

Table 4.9: Estimation of the fractal dimension for the complete "Satimage" database and for classes 1 and 4, as a function of the selected number of dimensions after preprocessing.

Rate	*Class 1*	*Class 2*	*Class 3*	*Class 4*	*Class 5*	*Class 7*	*%*
10	14	7	13	6	6	14	0.93
25	36	17	32	14	16	35	2.33
40	57	26	51	23	27	56	3.73
80	114	52	101	47	53	113	7.46

Table 4.10: Number of prototypes per class after vector quantization for the "Satimage" database.

prototypes for the various VQ rates (10, 25, 40 and 80).

4.3.4 "Texture" database

This database is composed of eleven classes in 40 dimensions. There are 5500 patterns, and each class contains 500 samples. Figure 4.9 shows that the number of samples is just sufficient for a base in dimension 4. For greater dimensions, it is a "small" database. After the various preprocessing, we have databases in 3 (PCA), 6 (PCA), 10 (DFA) and 18 (PCA) dimensions.

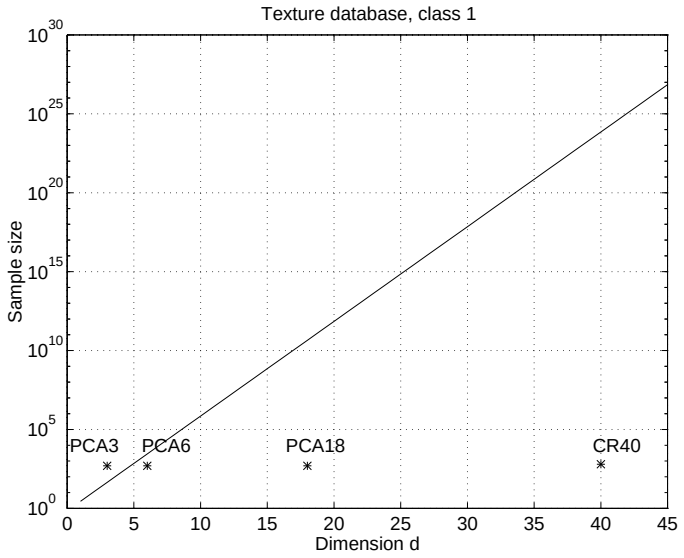

Figure 4.9: Minimum number of points to estimate a *pdf* by kernel functions, and the location of one class for the "Texture" database.

For the initial database (after centering and reduction), the between-class inertia is 29.62, the within-class inertia is 10.39, and the Fischer's coefficient is 2.85. This

Class	*2*	*3*	*4*	*6*	*7*	*8*	*9*	*10*	*12*	*13*	*14*
2	0	1.10	2.46	3.54	4.38	1.68	0.76	1.73	2.74	3.45	1.58
3	0.94	0	3.13	3.03	3.69	1.60	0.79	1.61	3.17	3.32	1.22
4	1.77	2.66	0	6.31	6.88	2.92	2.05	2.83	2.15	4.96	3.19
6	2.07	2.08	5.10	0	2.70	0.85	1.46	0.74	6.03	1.89	0.95
7	2.16	2.15	4.73	2.30	0	1.28	1.50	1.14	5.72	2.61	1.49
8	2.20	2.49	5.33	1.93	3.41	0	1.46	0.35	6.26	1.63	1.10
9	0.86	1.05	3.20	2.81	3.42	1.25	0	1.32	3.73	2.91	1.15
10	2.52	2.78	5.75	1.86	3.36	0.39	1.72	0	6.87	1.37	1.32
12	1.54	2.10	1.68	5.82	6.50	2.68	1.87	2.65	0	4.79	2.75
13	3.48	3.95	6.96	3.28	5.33	1.25	2.62	0.95	8.60	0	2.25
14	1.59	1.45	4.44	1.64	3.02	0.84	1.03	0.91	4.90	2.23	0

Table 4.11: Dispersion matrix computed on the "Texture" database.

Class	*2*	*3*	*4*	*6*	*7*	*8*	*9*	*10*	*12*	*13*	*14*
2	97.0	1.0	0.4	0.0	0.0	0.0	1.6	0.0	0.0	0.0	0.0
3	0.2	99.0	0.0	0.0	0.0	0.0	0.4	0.0	0.0	0.0	0.4
4	1.0	0.0	98.8	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	99.4	0.0	0.0	0.0	0.6	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	98.6	0.0	1.4	0.0	0.0	0.0
9	0.4	0.0	0.2	0.0	0.0	0.2	98.8	0.4	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	1.4	0.0	98.6	0.0	0.0	0.0
12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0	0.0
13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	99.8	0.2
14	0.0	0.4	0.0	0.0	0.0	0.4	0.0	0.0	0.2	0.0	99.0

Table 4.12: Confusion matrix with a KNN classifier estimated by a Leave-One-Out method for the "Texture" database. Averaged error: 1.00%

coefficient isn't very modified by the preprocessing.

For this database, there is a good matching between the indications given by the dispersion matrix (table 4.11) evaluated according to equation 2.5 and the KNN LOO confusion matrix (table 4.12). That means that when the coefficient is small in the dispersion matrix, a confusion can exist between the two classes. We can thus deduce that this database is composed of several well identified clusters (one cluster per class). The best estimate of the error rate by a KNN classifier ($K = 1$) is given at table 4.12.

The computation of the fractal dimension (figure 4.10) indicates that this distribution of samples would be in a volume of about 8 dimensions. The same method applied to class 2 and class 10 gives respectively a dimension about to 7 and 3 (figure 4.11). With this database, it seems that the number of freedom degrees is quite small relatively to the input dimension. This is consistent with the characteristics of the statistical texture features used here. Indeed, more or less complex relations exist between the features according to the samples distribution for a given texture. For two different textures, it isn't necessary the same features which are discriminant and the relations between the featuresare not necessarly the same.

Table 4.13 summarizes the different experimentations for the evaluation of the fractal dimension, for each preprocessing.

For the vector quantization preprocessing, table 4.14 shows the number of prototypes for the various rates (10, 25, 40 and 80).

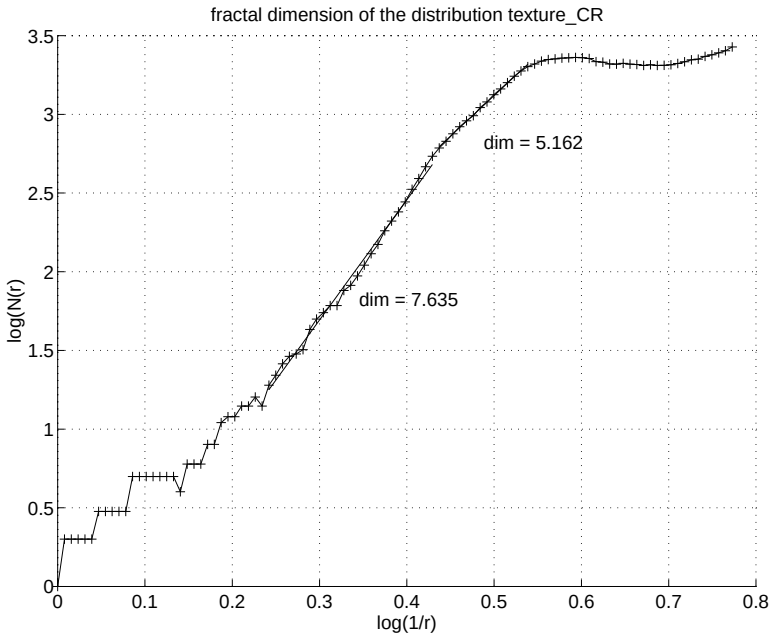

Figure 4.10: Plot of $N(r)$ for the "Texture" database.

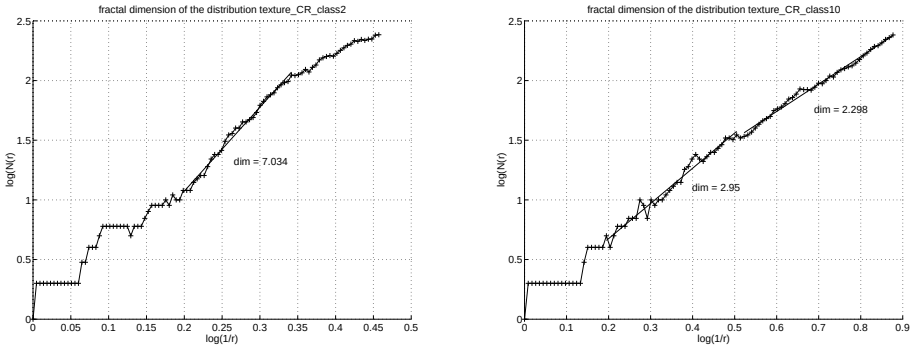

Figure 4.11: Plot of $N(r)$ for class 2 and class 10 of the "Texture" database.

Preprocessing	*Database FD*	*Class 2 FD*	*Class 10 FD*
Texture_CR	8	7	3
Texture_PCA18	4	3	3
Texture_PCA6	4	3	3
Texture_PCA3	3	2	2
Texture_DFA	7	7	5

Table 4.13: Estimation of the fractal dimension for the different preprocessing of the "Texture" database and for class 2 and 10, as a function of the selected number of dimensions after preprocessing.

Rate	*Class X*	*%*
10	10	2
25	25	5
40	40	8
80	80	16

Table 4.14: Number of prototypes per class after vector quantization for the "Texture" database.

Chapter 5

Studied methods of classification

This chapter is dedicated to the benchmarking study of each classifier individually. The documentation provided for each classifier contains

- **A brief description**
  The fundamental description of the algorithm with the technical description of the different useful parameters. More information about the algorithms and the use of their corresponding modules inside packlib may be found in the R3-B3 and R2-B3 reports.

- **The parameters configuration study**
  The influence of the different parameters on the algorithm behavior is studied in this section. Some typical curves of the classification error in function of the parameter value are provided and the critical parameters are extracted.

- **The benchmarking results**
  This section is dedicated to the extensive study of the classifier on the ELENA databases. It mainly concerns

  - the influence of the databases characteristics on the classifier results and the parameters optimal values
  - the correspondance between the theoretically estimated learning and recognition times and the equivalent values computed during the benchmarks
  - the memory requirements.

  A synthesis of the classifier properties for different classification tasks (databases characteristics, hardware constraints) may be extracted from these studies for each particular classifier.

5.1 KNN classifier

5.1.1 Brief description

The "K nearest neighbor" classifier is a very classical one [15]. It is used here as a reference for the best estimator of the theoretical Bayes error.
The a posteriori probability is locally estimated at each point x to be tested, from a database with N samples. Let us consider a volume V around the sample x. This volume captures K samples, K_i samples belong to the class ω_i. Then the joint probability $p(x, \omega_i)$ is estimated by :

$$p(x, \omega_i) = \frac{K_i}{N \times V} \tag{5.1}$$

Thus, applied to the Bayes rule, the a posteriori probability $P(\omega_i|x)$ is estimated by :

$$P(\omega_i|x) = \frac{p(x, \omega_i)}{\sum_{c=1}^{c} p(x, \omega_c)} = \frac{K_i}{K} \tag{5.2}$$

with c, the number of classes. So, the decision is taken considering the class which have the largest number of samples inside a volume with a radius given by the distance to the K-nearest neighbor. This usual rule is the *pooled* form. Another decision rule is the *grouped* form. Here, c volumes are considered. Each volume V_i captures K samples belonging to the class ω_i. The decision rule compares the c volumes and decides the class ω_i whose the volume V_i is the smallest [24].

It can be shown [15] that the nearest-neighbor rule will give an error rate greater than the minimum possible such as the Bayes error, and with an infinite number of samples, the error rate is never worse than twice the Bayes error. Under this assumption, the difference between the error with a k-nearest-neighbor rule and the Bayes error will decrease as the number K of neighbors increases. With a finite number of samples, the errors obtained will depend on "accidental characteristics" of the samples, so this parameter must be adapted for each case. In [5], the optimal parameter K comes from a compromise between a small variance of the estimator of the *pdf* and a limitate bias for this same estimator. With a larger value of K, the volume V is large and the variance is reduced. A large volume introduced a bias which decreases with the volume V and then with K. A formula is deduced for gaussian distributions where the optimal value of K decreases when the dimension increases (a rate superior to $N^{4/d+4}$).

5.1.2 Parameters configuration and benchmarking results

The parameter K must be set for each database as the processing mode ("pooled" or "grouped"). At the optimal configuration of the parameter K for each mode, we have obtained similar results. Then, in the following, the results for the "pooled" mode will be mainly discussed. If we don't take into account the choice of the mode ("pooled" or "grouped"), the configuration of this classifier is only set by the parameter K. The problem consists on finding the best value of K for a learning database with a finite number of samples. The asymptotic properties of the classifier give only guidelines, while in most of the practical cases, the real conditions are far from the asymptotic conditions (very large number of sample or infinite number of samples). So, in the practical cases, extensive computations must be done to choose the parameter K (research

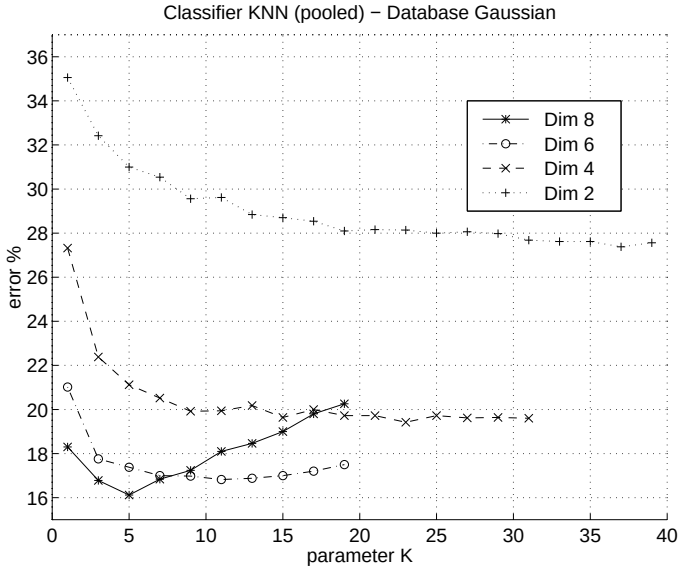

Figure 5.1: Research of the optimal value of K with the "Gaussian" databases versus the number of dimensions.

of the minimum recognition error function of different value of K). Some methods have been proposed to speed up the computation [5] but the principle remains the same.
With the benchmarks on the "Gaussian" databases, the research of the optimal parameter K is well described on figure 5.1. We see that :

- The shape of the minimum is very large as the number of dimension decreases. In other words, the value of the parameter K is not critical as the number of samples gets large (towards the asymptotic condition).
- The optimal value of K decreases as the number of dimensions increases. In other words, when the learning database is very small reletively to the data space dimension, local estimation of the class conditional *pdf* is obtained with a small value of K.

These two remarks are very well outlined at figure 5.1 and were also observed on the benchmarks with the others databases ("Satimage" and "Texture") versus the number of dimensions (figure 5.2). Figure 5.3 shows for two examples that for a number of dimension greater than 4 on a learning database with 2500 samples, the error rate is estimated with a significant bias, and this trend goes on as the dimension increases. As the dimension increases, the number of available points is not sufficient. If we compute the necessary number of samples according equation 2.3, we remark that 2500 samples are sufficient up to 5 dimensions. For the learning databases after a Vector Quantization, it has been observed that the optimal value of K is one, so in this case the optimal rule is the nearest neighbor rule. Figure 5.4 shows the evolution of the error rate versus the parameter K, each class of the learning database has been quantified with 40 codewords. So one codeword represents the contribution of about 12% of the learning samples (500 samples per class for 40 codewords). One neighbor has a smooth contribution to the *pdf* and is equivalent to the accumulation of more

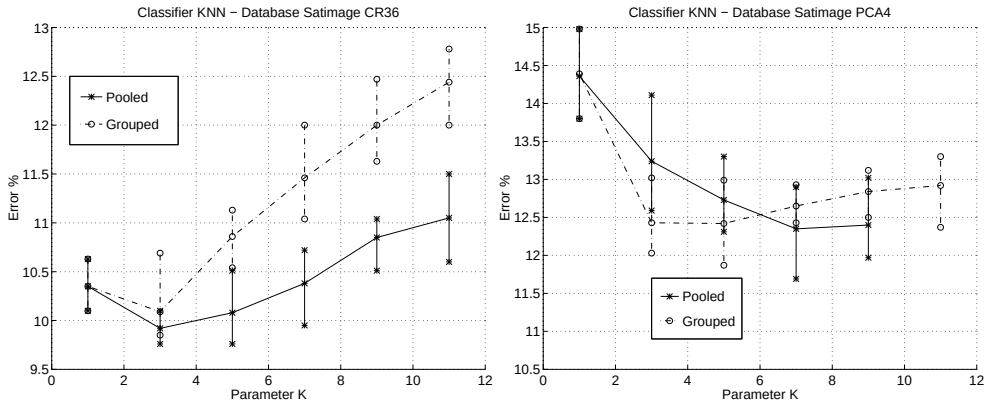

Figure 5.2: Research of the optimal value of K for the "Satimage" with 36 dimensions (left) and 4 dimensions (right).

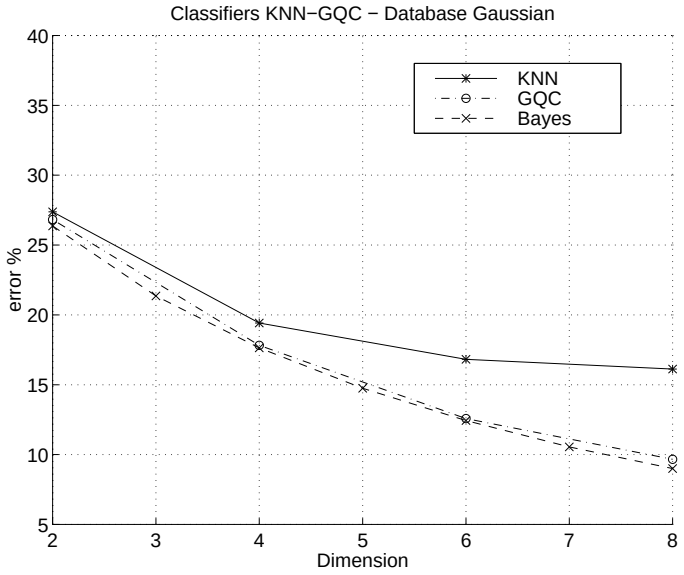

Figure 5.3: Evolution of the theoretical and experimental error rates with the "KNN" classifier, versus the dimension of the "Gaussian" databases. For each dimension, the parameter K have been set to its optimal value.

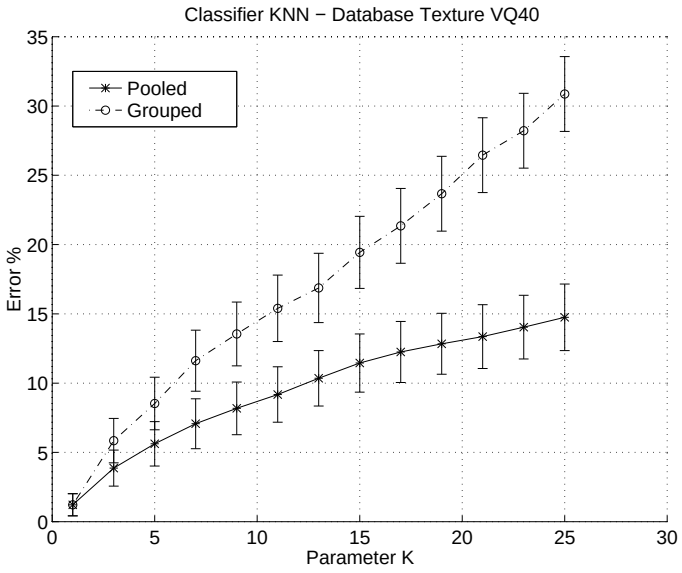

Figure 5.4: Research of the optimal value of K for a learning database ("Texture") after a Vector Quantization.

contributions (Dirac) in the original database. As it is remarked at section 3.2.2, error rates computed by a Leave-One-Out method give a better estimation than by an averaged Holdout (figure 5.5 left). With the averaged Holdout method, the difference between the minimum error and the maximum error give indication on the robustness of the classifier. In figure 5.5 (right), the confidence intervals computed for the Holdout and Leave-One-Out methods are indicated ; we remark overlapping between them. For the real databases "Satimage" and "Texture", we remark that efficient information reduction may be obtained by a linearly dimension reduction with a PCA. For these two databases, the number of features can be divided by a factor two without any significant increase of the recognition rate (figure 5.6). For the "Texture" database, the preprocessing by discriminant factorial analysis seems to be very suitable.

5.1.3 Hardware constraints

The classifier "KNN" requires a lot of memory and is very time consuming for the test. The learning time is only due to the memorization of the whole learning database. To classify a sample, the following operations must be achieved (with the classical algorithm) :

- computation of the distances between the test sample and all the learning samples,
- sort the distances to extract the k smallest distances,
- research the class which have the largest number of samples among the k-nearest neighbors.

In section 6.3, we show that the KNN classifier has clearly the most important test time and memory requirement.

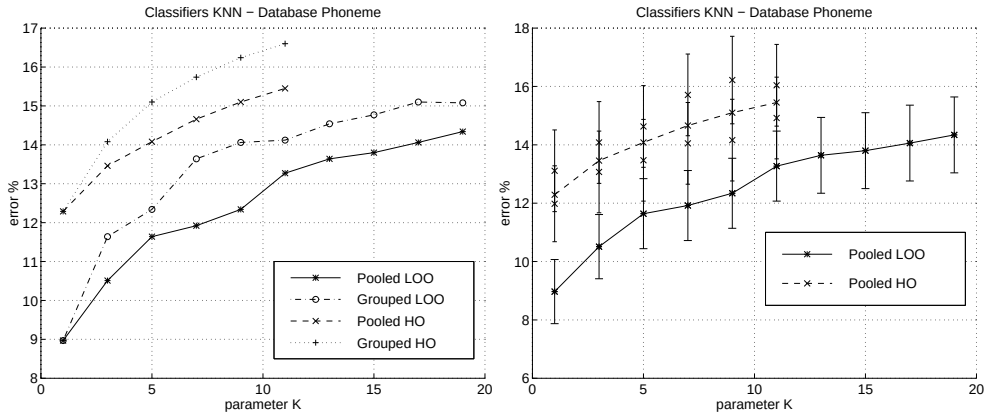

Figure 5.5: Error rate estimated by a Leave-One-Out method or an averaged Holdout method. Comparison between the grouped mode and the pooled mode (left). Overlapping of the confidence intervals computed from the errors estimated with the Leave-One-Out method and the averaged Holdout method (right).

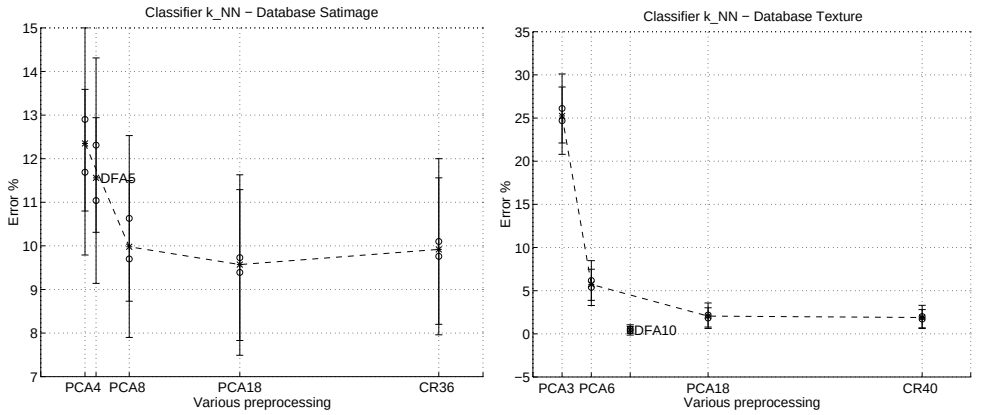

Figure 5.6: Error rates versus the number of dimensions for the database "Satimage" (left) and the database "Texture" (right). For each case, the classifier has been set with the optimal parameter K in the "pooled" mode.

5.2 GQC classifier

5.2.1 Brief description

The Gaussian quadratic classifier is a very classical parametric classifier [15]. The main assumption is to consider that the underlying distribution of the learning samples follows a gaussian distribution. So this classifier is optimal when the actual distribution is a gaussian one.
For each class ω_k the mean m_k and the covariance matrix Σ_k are estimated. The probability density function for a sample x in d dimensions, following this distribution is estimated by :

$$p_k(x) = \frac{1}{(2\pi)^{d/2} \times det(\Sigma_k)^{1/2}} exp - \frac{1}{2}(x - m_k)^T \Sigma_k^{-1}(x - m_k) \tag{5.3}$$

To decide the class, the Bayes rule is applied, the class is assigned to the maximum of the discriminant function $Pr(\omega_k) \times p_k$, with $Pr(\omega_k)$ the prior probability of the class ω_k. To simplify the computation, and eliminate the computation of the exponential function, the considered discriminant function corresponds to the logarithm of the previous one, and the class for a sample x is assigned to the class with a minimum value of $g_k(x)$, such as :

$$g_k(x) = (x - m_k)^T \Sigma_k^{-1}(x - m_k) - 2 \times ln(Pr(\omega_k)) + ln(det(\Sigma_k)) \tag{5.4}$$

5.2.2 Parameters configuration

The quadratic classifier only uses the estimation of the covariance matrix and the mean vector for each class. Then, to be able to compute the discriminant function, the determinant of the covariance matrices must be different to zero. Unfortunately, this is not usually the case when a too important number of dimensions are used. The dataset must then preprocessed to avoid this problem. For example, here, it was impossible to use this classifier with the rough databases "Satimage" and "Texture". Preprocessing by PCA was used to remove the linearly correlated dimensions.

5.2.3 Benchmarking results

On Gaussian databases, the results obtained with the GQC classifier are very close to the theoretical ones. Figure 5.7 shows these results where the error is computed by a Leave One Out procedure. It follows the evolution of the theoretical error rate. In the same graph, the dispersion on the error rates computed by an averaged Holdout (averaged over 5 tests) method is due to the dispersion in the estimation of the mean vectors and the covariance matrices. Moreover, we observe, as the dimension increases, a larger difference between the experimental estimation and the theoretical value. The experimental estimation will be excessive as the dimension increases. This is due to the same limited number of available samples even with the largest dimension (here 8). On the database "Clouds" which is a multi-modal gaussian database (class 0 with 3 gaussian modes, class 1 with one gaussian mode), the results obtained with this classifier are far from the theoretical errors, it is due to the bad estimation of the *pdf* for the class 0 with 3 modes. We obtained in practice with a "LOO" procedure an error

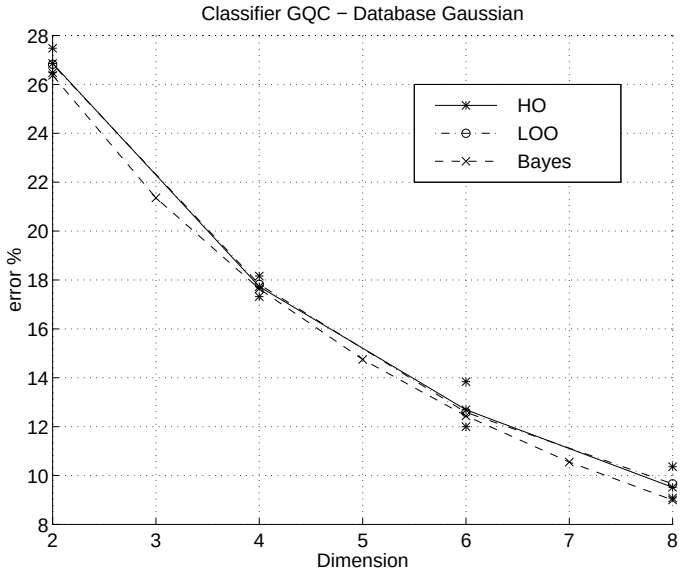

Figure 5.7: Evolution of the theoretical and experimental error rates versus the dimension of the Gaussian databases.

of 24.88%, while the theoretical Bayes error is 9.66% and the best error estimated by a "KNN" classifier is 10.9%

With the database "Concentric", the GQC classifier has also poor results. Indeed here, there is no overlapping between the classes and the distribution is uniform. This distribution doesn't match with the assumption of a Gaussian distribution. The error is estimated to 17,20% (LOO), compared to the theoretical error (0%)and the best error with a KNN classifier (1%).

With the base Phoneme, figure 5.8 shows the results both with the learning database after a vector quantization and with the initial complete database. As it will be remarked at the section 6.4, when the number of codewords for the learning base increases, the error computed with the full database decreases towards the minimal resubstitution error. For the database "Phoneme", with about 80 codewords, the error reaches the error obtained with the full learning database. With more codewords, the results obtained are no more comparable.

Figure 5.9 shows the results obtained with these two databases according to the different preprocessing by PCA or DFA. For the two databases, with an equal number of features, the DFA preprocessing is better than the PCA, this is more true for the "Texture" database.

5.2.4 Hardware constraints

From the memory point of view, the GQC classifier requires little memory capacities. Per class, the mean vector (d components), the covariance matrix ($d \times d$ components), the prior probability and the determinant of the covariance matrix must be stored. Thus, storage for $C \times (d + 2 + d \times d)$ components is necessary (C is the number of classes).

For the computation time, a distinction is made between the learning time and the test

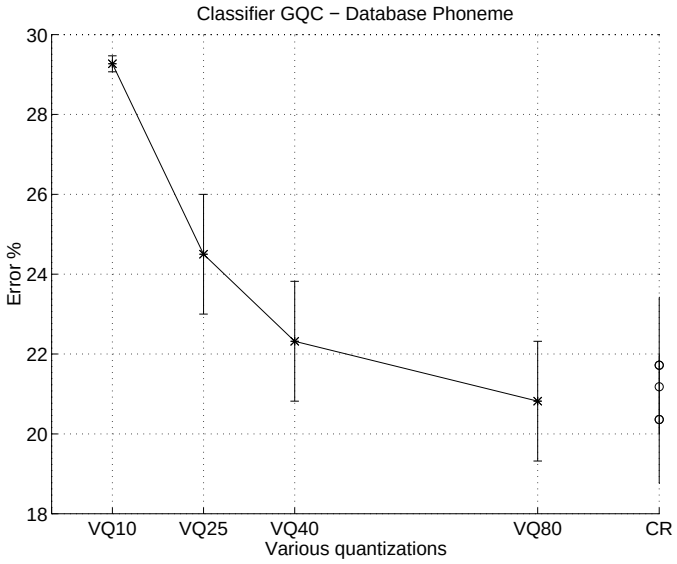

Figure 5.8: Evolution of the experimental error rates versus the number of codewords for the learning "Phoneme" database. The number of codewords are 10, 25, 40 and 80 multiply by 2 (2 classes).

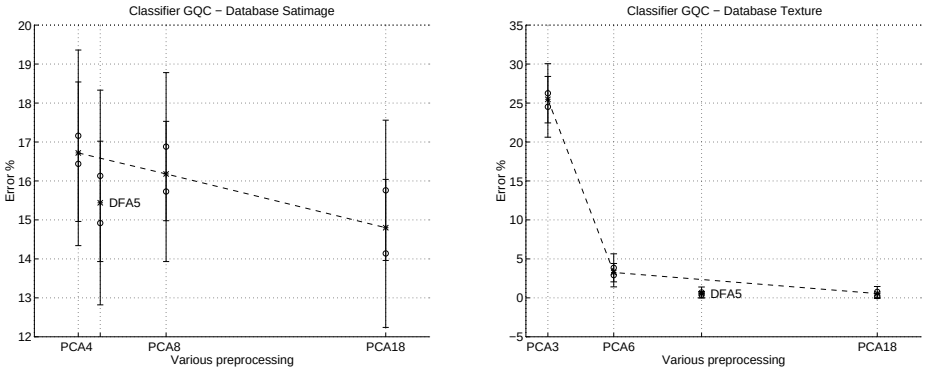

Figure 5.9: Evolution of the experimental error rates versus the number of dimensions selected after preprocessing, (right) Texture, (left) Satimage.

time. The learning time includes the computation times for the following processing per class ω_k:

- vector mean m_k of N_k samples in d dimensions,
- covariance matrix Σ_k,
- inversion of the covariance matrix,
- determinant of the covariance matrix,
- prepare the computation of the discriminant function by the computation of an offset term $(ln(det(\sigma_k)) - 2 \times ln(Pr(\omega_k))$.

For the test time, for each test sample and for each class, it is necessary to compute the Mahalanobis distance between the mean vector and the test sample, and add the offset term :

- difference of the test sample and the mean vector,
- transposition of this vector,
- multiplication of the line vector by the inverse of the covariance matrix and by the column vector, item add the offset term,

The decision is realized after the computation of the minimum value of this discriminant function for each class.

5.2.5 Conclusions

This classifier is very easy to use, it has no parameters to optimally configure. The memory requirement is not very important and it is not time consuming. At the section 6.3, these two characteristics are compared with the other classifiers studied in this framework. This classifier is optimal when the classes distributions are unimodal normal distributions.

5.3 MLP classifier

5.3.1 Algorithm description

The MLP acronym stands for Multi-Layer Perceptrons. This network, combined with the backpropagation algorithm, is the most widely known inside the neural networks community. It is useless to present it here. The interested reader can, for instance, refer to [32]. A description of the implementation of the MLP classifier for the PACKLIB environment can be found in the report of task B3. In this implementation, the standard backpropagation algorithm is used.

Learning process

Suppose we have a learning set A_N of observations $x(n)$, $1 \leq n \leq N$ with their associated known classes $\omega_{x(n)}$: $A_N = \cup\{A_{N_i}\}$ with $A_{N_i} = \{x(n), \omega_{x(n)} = \omega_i, 1 \leq n \leq N_i\}$ and $N = \sum_{i=1}^{c} N_i$. Here, the number of classes is c, and each class possesses N_i prototypes.

Although the MLP network and the backpropagation algorithm are widely known, there are many ways to learn a specific task. The coding of the output can be performed in many different ways. Because we know the number of classes, c, we have chosen to set the number of output units in the last layer to c. The output for class i is coded as a vector with all the component set to 0 except the i^{th} set to 1. Formally, the output vector for class i is $[\delta_{i0}, \delta_{i1}, ..., \delta_{ic}]$, where δ_{ij} is the Kronecker symbol. This coding scheme has been justified in [33].

Before learning, we must transform the desired output vector into this format. For the recognition phase, we must make the inverse transformation to get the class number.

Remark: Benchmarks performed on various databases have proven that this encoding/decoding scheme works correctly only if the classes are consecutive (for instance, if we have 5 classes labeled {0, 1, 2, 3, 4}). If the classes are labeled {12, 23, 78, 15, 45} however, the MLP network is not able to learn correctly, because there is a lot of "holes" between "valid" classes. It is then necessary to map them to consecutive number (e.g. {12, 23, 78, 15, 45} $\rightarrow$ {0, 1, 2, 3, 4}) before learning. Such a mapping can be performed using the "dbtl3" Packlib module.

5.3.2 Parameters configuration

In this section, we will discuss briefly the parameters that have been taken into account in the MLP benchmarking study. Before beginning, a word of warning about the performances of the MLP network:

- Is it quite difficult (and time-consuming) to find the best parameters of the MLP for a given database
- The MLP network benchmarking studies have been performed for a comparison purpose; the goal was to seek the best classification rates, but not at the expense of spending too much time in the process

As a result, we have restricted ourselves in the choice of varying parameters. For instance, the MLP structure has been kept constant for nearly all the databases, with

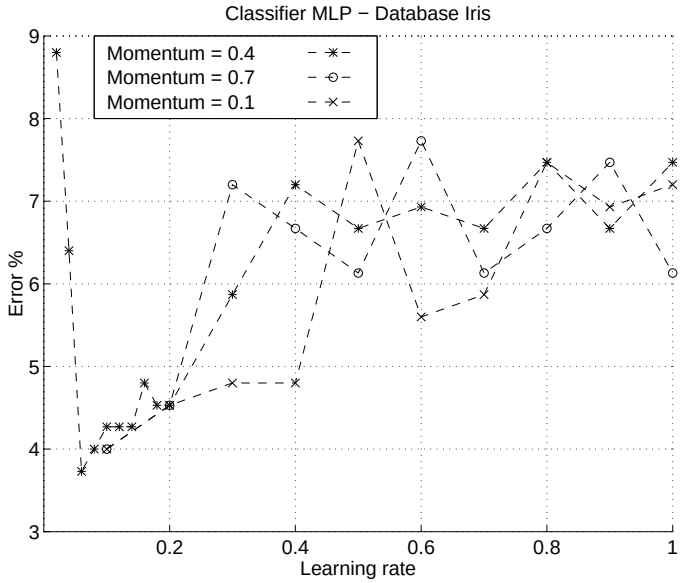

Figure 5.10: Classification errors for different values of α and γ with 100 epochs.

two hidden layers of respectively 20 and 10 units[1]. Experience has shown[2] that these values gave good results. Of course, it is certainly possible to achieve better results with a very careful setup of the parameters of the MLP. We don't pretend to have tried all the combinations of parameters.

Regarding the number of epochs needed, we have performed various tests with each database by selecting several number of epochs and train the network accordingly. Then, we have selected the smallest number of epochs that gave "good" results[3]. Usually, the curve of learning epochs vs performance looks like a $\frac{1}{x}$. We have selected a point where the derivative of this function was near 0. [4]

Learning rate α and momentum γ

Those parameters correspond respectively to the 'RATE' and 'MOMENTUM' resources of the MLP module in Packlib [34]. These parameters are generally regarded as the most sensitive parameters in the learning process. As it can be seen in fig. 5.10 for the iris database, the learning rate α must be small (minimum is attained here for $\alpha = 0.06$) to obtain the best classification errors. This result has been verified for all the databases, although the exact minimal value may slightly vary.

In the fig 5.11, you can see a systematic exploration of the rate - momentum plane for the "phoneme VQ25" database. It is interesting to note that even large variations of α and γ lead to similar classification errors - around 20%. This has been verified for other databases as well ("iris" for instance).

[1] except for the "Iris" and "Gaussian2D" database

[2] see fig. 5.12

[3] Which is, of course, not the same as selecting the smallest error classification rate

[4] This is an empirical but simple rule that performed well in all the cases

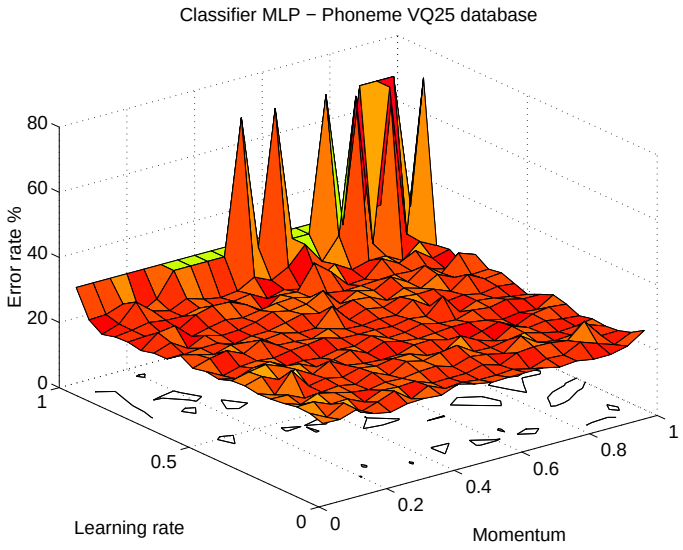

Figure 5.11: Averaged Holdout errors in the plane defined by α and γ with 500 epochs

Structure

This parameter corresponds to the 'STRUCTURE' resource of the MLP module in Packlib. This parameter is the most difficult to setup correctly. How many layers are needed for a given task ? How many units per layer ? When can we expect a network to generalize ? The answer is *at most two* hidden layers, with arbitrary accuracy being obtainable given enough units per layer [35]. It has also been proved that only one hidden layer is enough to approximate any continuous function [35]. However, these results do not tell how many units there must be.

In this benchmarking study, we have used in most cases two hidden layers. The number of units in first and second hidden layer have been set to respectively. 20 and 10 units for most databases. In fig. 5.12, you can see two examples of classification errors for different numbers of hidden unit. As shown by these curves, there is not much change in the results of the network. However, the larger the network, the longer the learning time. Therefore, we have chosen a medium-sized network.

5.3.3 Benchmarking results

Input dimension

Figure 5.13 represents the classification error for the different "Gaussian" databases. These databases are composed of patterns which follow the same distribution law, but with different dimensions (the number of attributes of each prototype) and therefore is a good candidate to study the classifier behavior for different dimensionalities of the input vectors, for heavy overlapped distributions and for non-linear separability. One study on this database [36] has been performed by Kohonen for different algorithms (MLP, Boltzmann machine and LVQ). The Kohonen curve is on fig. 5.13, along with the theoretical Bayes classification rates as computed in [36]. The performance of our MLP is better than Kohonen's MLP who used a network with only one hidden layer

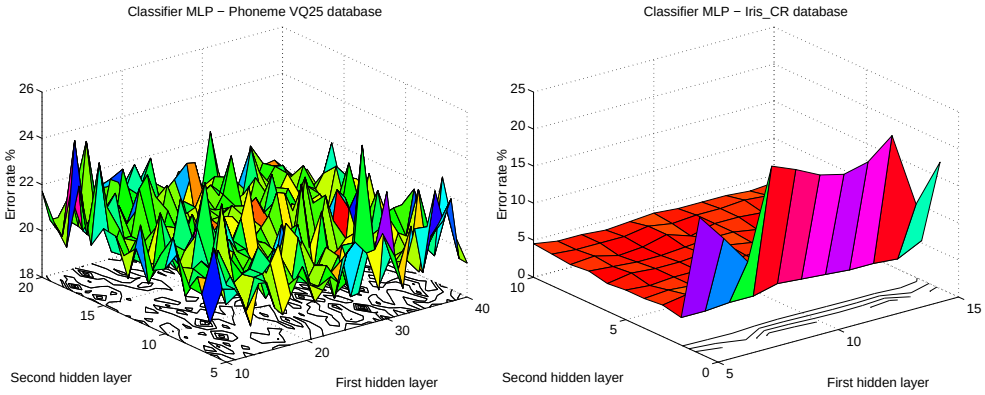

Figure 5.12: Single Holdout error for "Phoneme" (left) and Averaged Holdout for "Iris" (right) databases with different number of hidden units in the first and second hidden layer. The left figure, being only a single Holdout, has more variations.

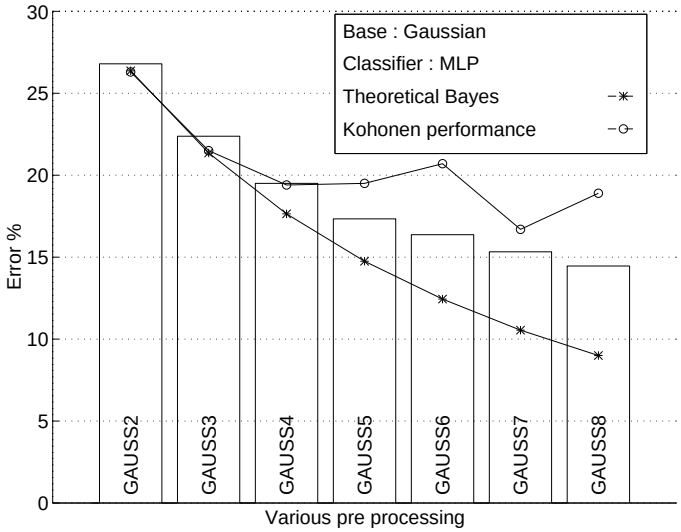

Figure 5.13: Classification error for 5 Averaged Holdout. Epochs = 200, Rate = 0.1, Momentum = 0.2

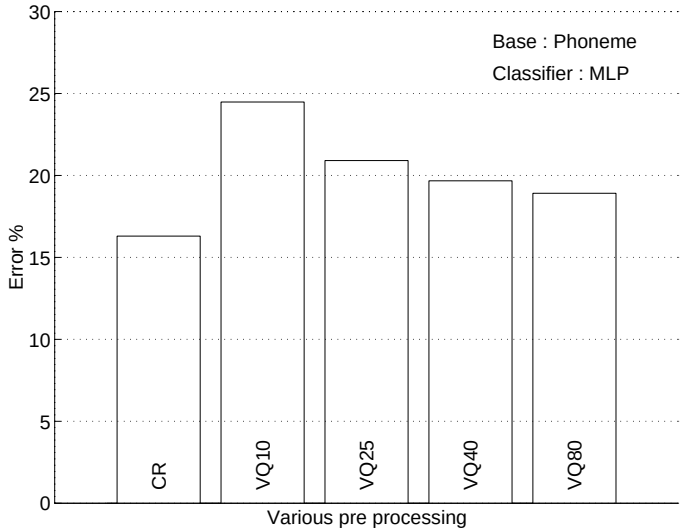

Figure 5.14: Error in function of the preprocessing for "Phoneme" database

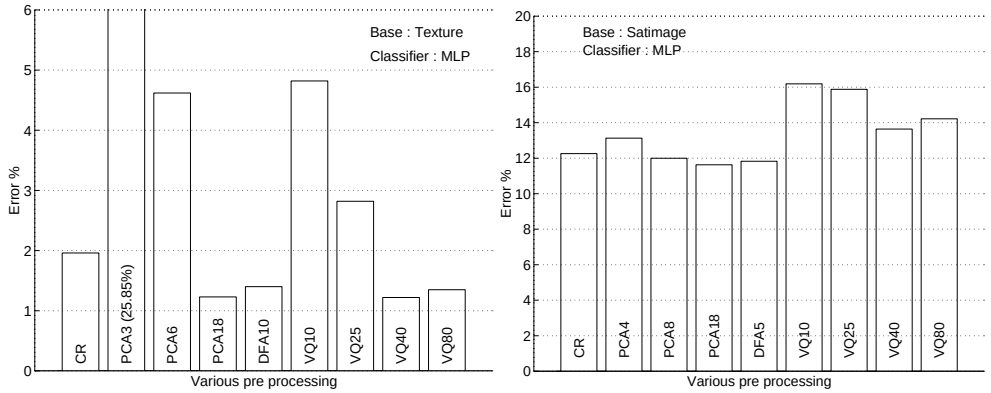

Figure 5.15: Error in function of the preprocessing for "Texture" (left) and "Satimage" (right) databases

containing 8 units. In our MLP, a second hidden layer has been added[5]. This layer allows to obtain better results.

You can also see on fig. 5.14 and 5.15 the classification error obtained with the "Phoneme", "Texture" and "Satimage" databases which have been processed using several techniques. From these figures, it can be deduced that the performance of the classifier degrades as the input dimension is reduced. By reducing the number of attributes, we are in a certain sense removing relevant information. It is therefore normal that the performance degrades. In the other hand, by diminishing the input dimension, we are also improving the generalization capabilities of the network, because we need less training examples as dimension decreases [37]. A compromise must be found between these two antagonist features (for instance, use the PCA to find the number of eigenvalues that reach a certain inertia - see the previous chapters for more explanation).

We can also see the comportment of the algorithm as the database is quantized (fig. 5.14 and 5.15 left and right, bars VQX, where X stands for the number of vector kept). Here, the number of attributes does not change, but only a reduced set of prototype vectors (i.e. X vectors) has been selected. Only those vectors will be used for the training, and the recognition will be done on the whole database. Generally speaking, it can be noted that even with small X, the performance of the classifier does not degrade significantly compared to the CR database [6] (half the database as learnset). It can also be noted that the performance generally increases with X, which is normal since we have a more accurate representation of the database. With the "satimage" and "texture" databases however, we have the best results for VQ40, for an unknown reason. But error bars given is fig. 5.18 and 5.18 show that these are probably due to a good initial configuration of the MLP network. All the tests with the VQ databases have used the same network structure, namely 2 hidden layers with 20 and 10 neurons. With 40 prototypes (VQ40), for all the databases, the result can be regarded as satisfactory. Experience shows, however, that these prototype vectors must be chosen carefully (see next chapter to know how the prototype vectors have been chosen).

Figure 5.16 shows the learning time and performance of different preprocessors for "Satimage" and "Texture" databases. According to these figures, the PCA18 preprocessing seems to be one of the best compromise, whereas the VQ databases seems to be the worst. As a result, it is more useful to do a PCA than a VQ for these databases.

Figures 5.17, 5.18 and 5.19 show the classification errors in function of the preprocessing. In abscissa is the number of attributes (dimension of input vectors). The very large error bars show that we must be very cautious about the interpretation of these results.

Results for various DFA indicates that this preprocessing method gives always good results, with a small number of dimensions (10 for "Texture" and 5 for "Satimage").

[5] For the "Gauss_2D" database, we used a network with only one hidden layer with 5 units. For all the other databases, we used our standard two hidden layers network with 20 and 10 units respectively

[6] for the "Texture" database, the performance of VQ40 and VQ80 is even better (by less than 1% though) than CR's

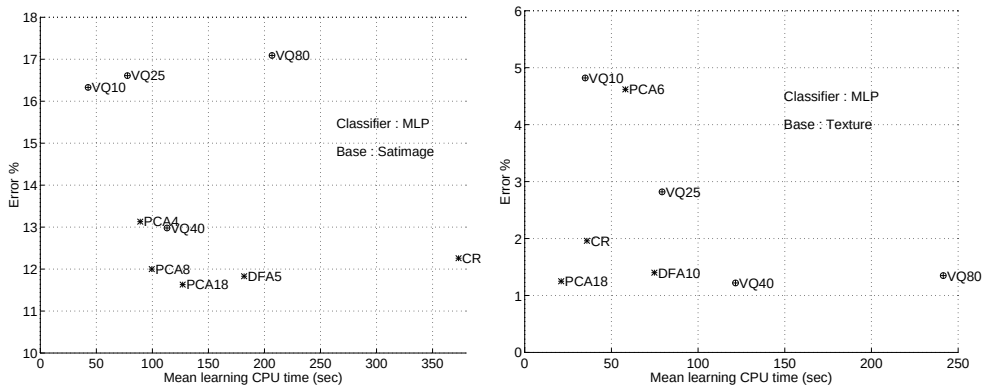

Figure 5.16: Learning time and classification errors of "Satimage" and "Texture" databases

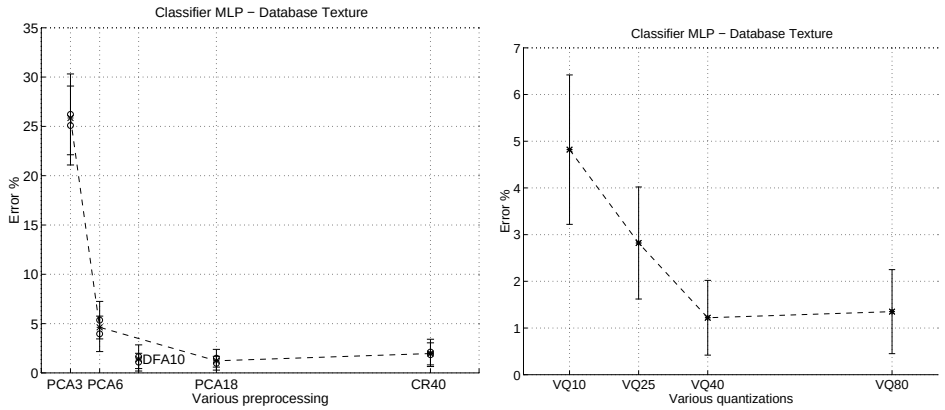

Figure 5.17: Error in function of the input dimension for "Texture" database

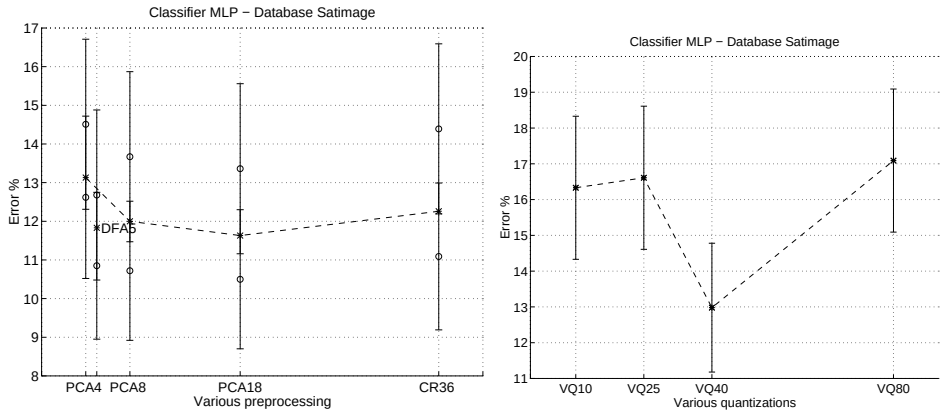

Figure 5.18: Error in function of the input dimension for "Satimage" database

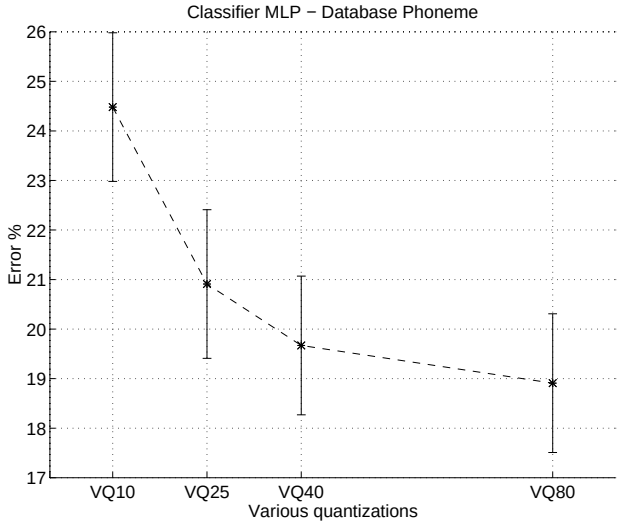

Figure 5.19: Error in function of the VQ codebook size for "Phoneme" database

5.3.4 Hardware constraints and Learning time

Suppose that the time for an ADD operation is approximatively the same as the time for a MUL (t seconds per simple operation).

- The memory requirement of the MLP network depends on its structure. Formally, if $layers$ is the number of layers and $L(j)$ is the number of units in layer j, the needed memory size in words can be computed as:

$$size = \sum_{j=1}^{layers-1} (L(j)+1)L(j+1)$$

- The recognition time may be estimated by:

$$T_R = N_e t \sum_{j=1}^{layers-1} 2(L(j)+1)L(j+1)$$

  with N_e, the number of learning epochs.

- The learning time is the sum of the recognition time plus backpropagation time. It is difficult to formally calculate the backpropagation time since it depends on the calculus of the derivative of output function σ. However, fig. 5.20 shows that the learning time is proportional to the memory size:

$$T_L \propto size$$

  We didn't investigate further to find out the exact relation (the proportionality coefficient for left and right of fig. 5.20 is not the same, meaning that the relation, although correct, is not complete).

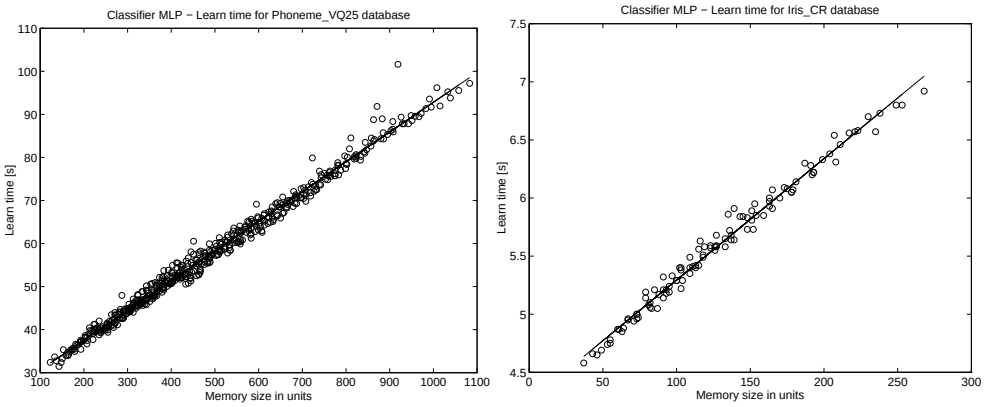

Figure 5.20: Learn time for 1000 epochs of "Phoneme VQ25" database (left) and 100 epochs of "Iris" database (right) on a Sparc20 workstation

5.3.5 Conclusions

The MLP classifier has proven to obtain good results for most databases. This explains why it is widely used in the neural network community. Although its defaults (difficulty to find the correct structure, fix the parameters, etc;), the simulation results show that the MLP is quite tolerant to mistakes in the parameters. In fact, this is not the performances that will suffer, but rather the learning time that will be more important. We have not remarked generalization problems for too large networks[7].

[7] This is maybe due to the fact that we didn't modify the number of units in hidden layers by more than a factor 2 (see fig. 5.12 for instance)

5.4 LVQ classifier

5.4.1 Brief description

The Learning Vector Quantization model, proposed by Kohonen, is a simple adaptive method of vector quantization. A finite number of prototypes, each one being labeled with a class identifier, are chosen in the input space. The learning process only modifies the location of the prototypes : each input sample is compared to all prototypes and the nearest one is selected. If the class identifier of the selected prototype is the same than the class identifier of the input sample, then the selected prototype is moved in the direction of the input sample, or moved in the opposite direction if it is not the case. This procedures defines the learning phase of the LVQ1 model. The test phase is identical for the prototype selection. There is no modification of its location. Only the class label of the selected prototype is taken into account, giving the class of the input sample. By this way, the LVQ1 performs both a quantization and a classification tasks, but no one of them is optimal. In particular, the vector quantization is better if the classification task is not achieved. The boundaries between classes do not approximate the optimal Bayes boundary, but improved versions (LVQ2 and LVQ3) have been proposed to overcome this limit. A deeper description is available in [7].

5.4.2 Parameters configuration

The two most important parameters for the LVQ algorithm are the gain parameter (α) and the number of codebooks. A test is done for different values of α, $0.1 \leq \alpha \leq 0.8$, with an increment step of 0.1. The result shows that the α parameter has not significant influence on the error (figure 5.21), for a given number of epochs (here 10). The number of prototypes is a variable which is studied for each different database. For these benchmarks, the limit number of prototypes was chosen to about 10% or 20% of the size of the learning database.

5.4.3 Benchmarking results

Generally speaking, the optimal configuration of the LVQ cannot be theoretically defined. First of all, the influence of α is negligible, but it must not be too large (not over 0.5). The codebook (set of prototypes) choice requires the knowledge of the relative number of prototypes per class label, and the whole size of the codebook.

The relative number of prototypes per class is chosen in order to respect the ratio between the different a priori probabilities of classes. The whole size of the database admits an higher bound, which is the number of available data. The lower bound cannot be easily chosen but is strongly related to the structure of the database (number of modes, ...). The definition of a criterion for the optimization of the number and the location of prototypes could improve the quality of the vector quantization, but it remains today a research challenge.

For the "Gaussian" database, the main observations concerns the gap between the minimum error rate obtained and the theoretical Bayes error rate. When the dimension increases, the gap also increases : about 3% for dimension 2 and about 6% for dimension 8 (figure 5.22). This is due to the empty space phenomenon. The optimal ratio between

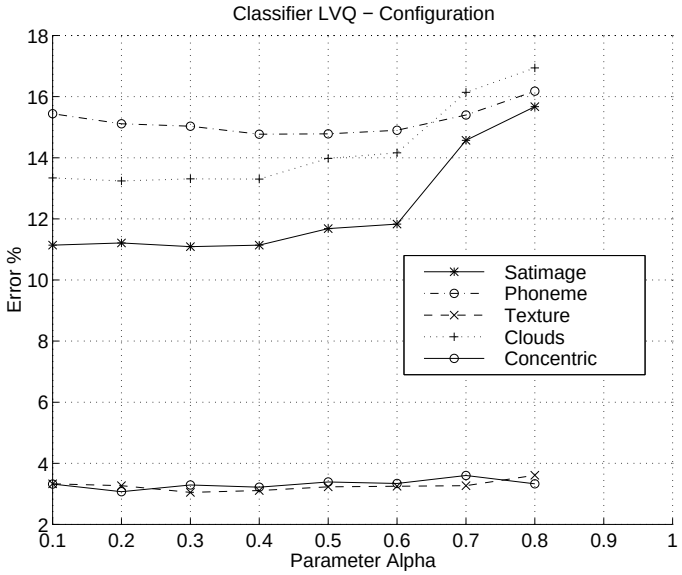

Figure 5.21: Evolution of the error rate versus the parameter α for the different databases.

the number of prototypes and the number of samples for learning is small, and is located in the interval between 0.8% and 2%. In fact, each class forms an unimodal cluster, the structure of the database is quite simple, few prototypes are sufficient.

For the "Concentric" database (figure 5.23), the minimum mean (about 2.5%) error rate is reached for a small number of codebooks (about 75). The robustness of the algorithm is good, as shown by the small size of the two error bars , about 1% (each error bar is the confidence interval on the minimum error and the maximum error obtained during the five trials). The behavior of the algorithm in that case comes from the sharp limits between the two classes : the vector quantization creates a tessellation (for the Euclidean distance measure) around the prototypes, and a prototype located at the border of one class distribution can cover points belonging to the other class. The theoretical error rate (0%) could be reached only with a number of prototypes equal to the number of available samples. This is not realistic nor useful in practice.

The "Clouds" database (figure 5.24) is a difficult problem, with overlapping classes and a non-linear border. The theoretical Bayes error rate is 9.66%. With the LVQ, the best codebook size is about 110. It has given the best error rate, but the mean error rate is about 12%. Increasing the codebook size does not improve the performance of the classifier. In that case, the optimal ratio between the number of prototypes and the number of samples for learning is about 4%.

For the "Iris" problem (figure 5.25), two classes are linearly separable, and the third one is not linearly separable. The best error rate obtained with the k-NN classifier is about 3.3% with k=7. The LVQ classifier gives a good result for a codebook size of 10 prototypes. The performance is not increased if the number of prototypes is increased.

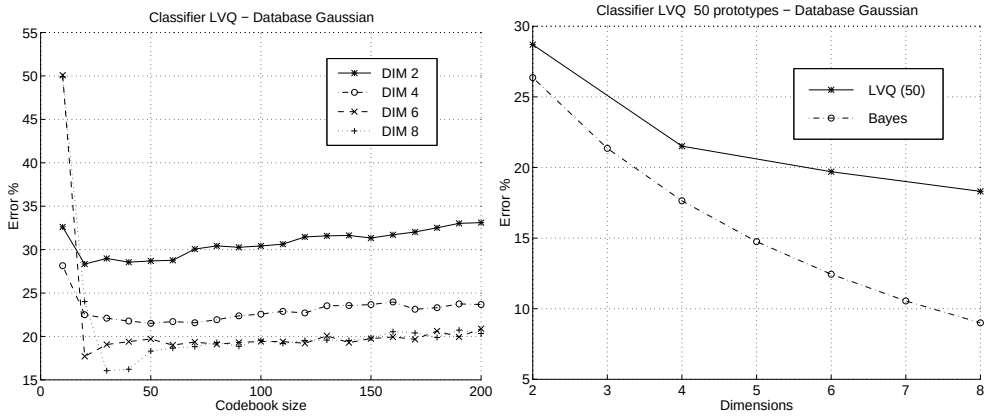

Figure 5.22: Error rates versus the number of number of prototypes for different dimensions of the "Gaussian" database (left). Comparison of the experimental results obtained by the classifier LVQ with 50 prototypes and the theoretical Bayes' error. (right).

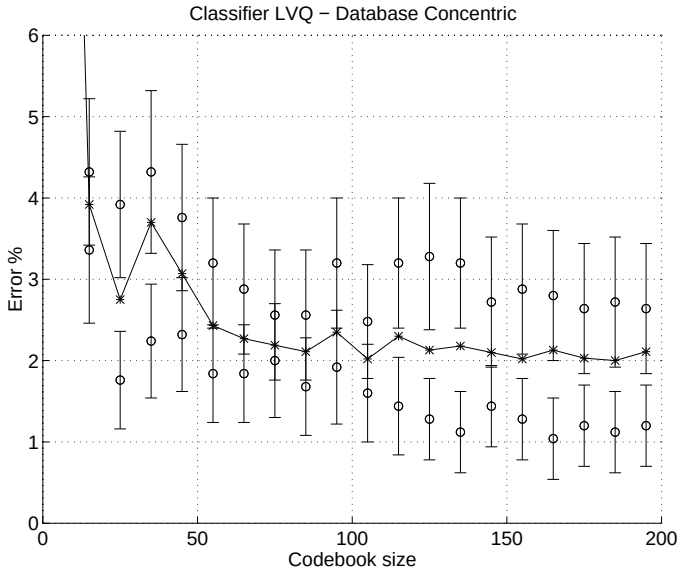

Figure 5.23: The error for "Concentric" as a function of the number of prototypes

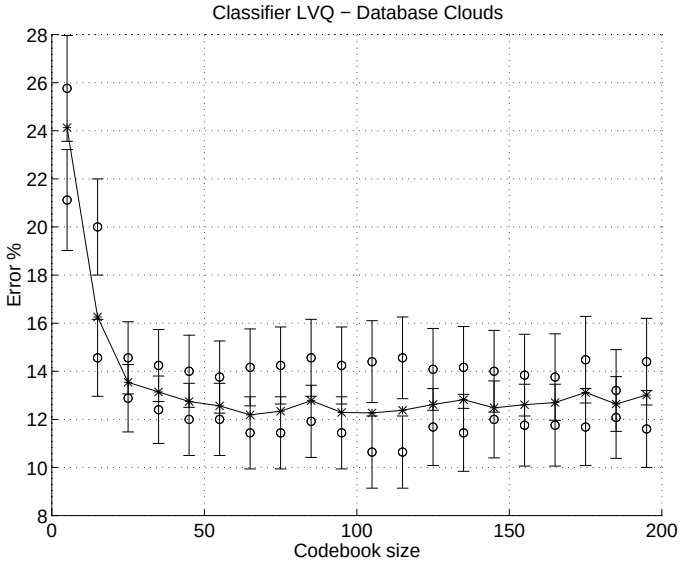

Figure 5.24: The error for "Clouds" database as a function of the number of prototypes

For the "Phoneme" problem, we observe that the error is monotonically decreasing. From about 350 prototypes, the error doesn't decrease any more, but we cannot determine if the minimum given by the k-NN classifier will be reached (which is about 8.8%, with k=1).

For the "Satimage" problem (figure 5.27), various preprocessing methods have been applied (CR36, DFA5, PCA18, PCA8, PCA4). The best error has been obtained with the PCA18. The information is reduced by a factor two, and the error is not affected by this reduction. With only 8 features, almost the same results are obtained (less than 0.5% of difference). The k-NN classifier, for this example, and with k=5, gives a minimum value of 8.8% for the error. An additional test have been done on this "Satimage" database on the comparison of the evolution of the error rate versus the number of prototypes, with the averaged Holdout (5 trials) method and the Resubstitution method. Same tests have been realized with the IRVQ classifier (figures 5.34 and 5.35). The conclusions on this test are the same. The resubstitution error is clearly identified (small and continuously decreasing with the number of prototypes). From 200 prototypes, the decrease of the error (Holdout method) becomes very small. This total number of prototypes is sufficient.

For the "Texture" problem (figure 5.29), the same preprocessing methods have been applied. We can observe that the best results are obtained for the DFA10, CR40 and PCA18. The minimum error obtained is about 1%, directly reached with the DFA10 preprocessing. The k-NN classifier gives the minimum which is 1.2% with $k = 1$. It is clear in that case that the preprocessing method is crucial for the classifier performance, which is excellent for this database in the case of the DFA preprocessing.

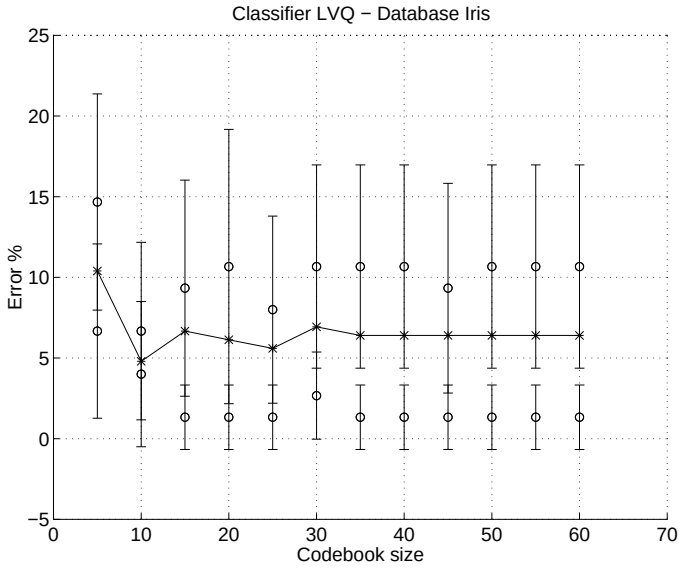

Figure 5.25: The error for "Iris" as a function of the number of prototypes

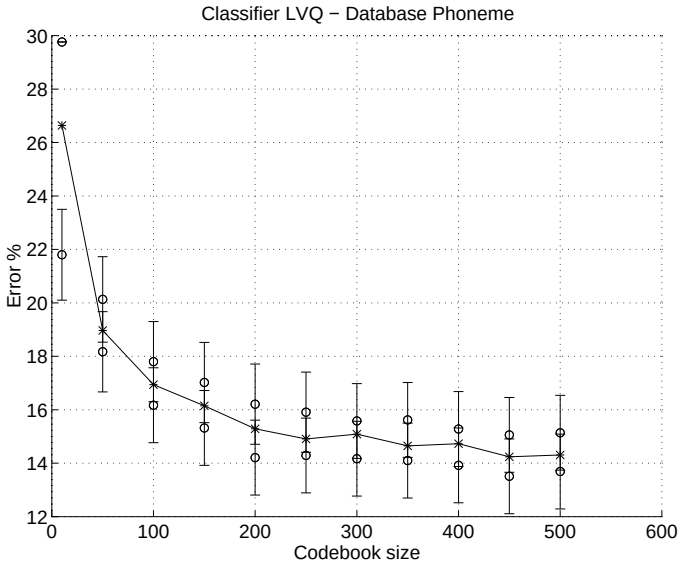

Figure 5.26: The error for "Phoneme" as a function of the number of prototypes

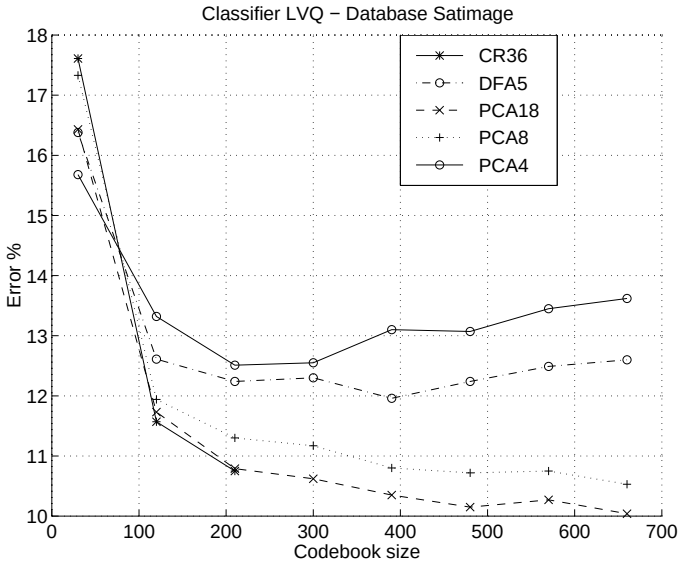

Figure 5.27: The error for "Satimage" as a function of the number of prototypes

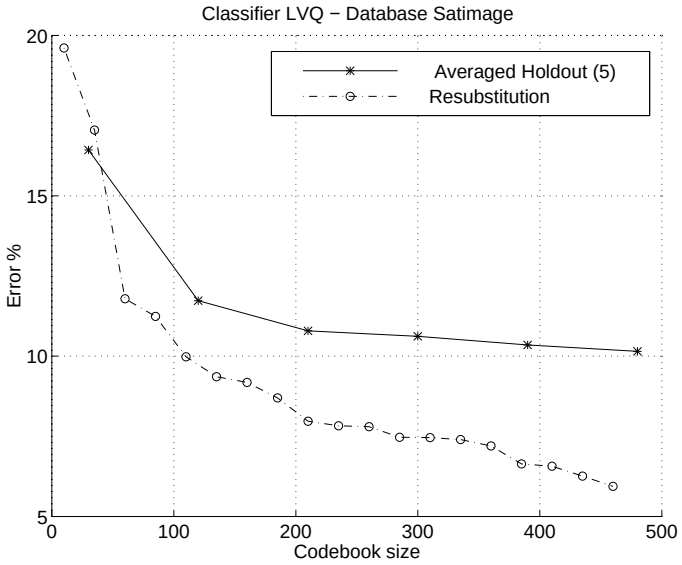

Figure 5.28: Resubstitution and Holdout errors as a function of the number of prototypes for the database "Satimage"

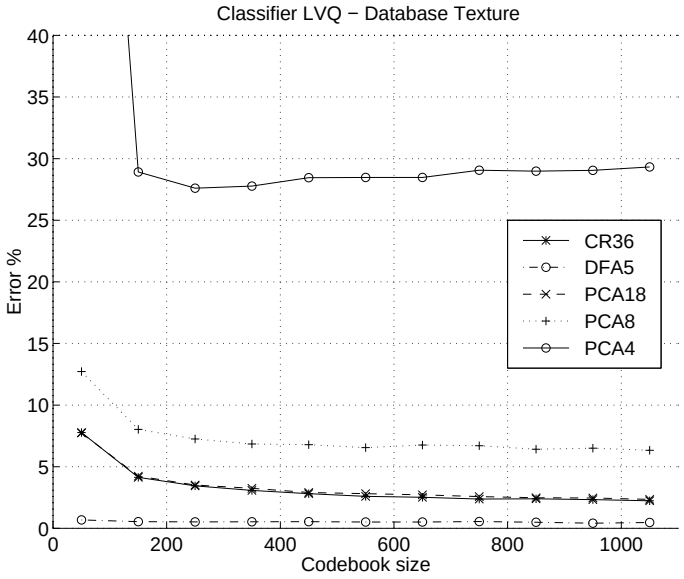

Figure 5.29: The error for "Texture" as a function of the number of prototypes

5.4.4 Hardware constraints

For the LVQ algorithm, the necessary operations for the recognition phase are very simple (mainly distance computations and search for the smallest one) and good performances are already reached for small codebook sizes. The LVQ classifier is thus a good candidate for application with strong hardware constraints (memory and time). This classifier needs the storage of $M.d$ memory words, where M is the total number of codewords and d the number of attributes per prototypes).

To evaluate the computation time, we use here the same notations as in section 5.5.3. For the learning phase, the following operations are repeated for each learning sample (N_L learning samples) and during the chosen number of epochs (N_e epochs) :

- computation of the distances with every prototype
- serach of the minimum distance
- update of the codeword location

The learning time can then be estimated by :

$$\begin{aligned} T_L &\approx (M \cdot (3d+1)t + (3d+1)t) \cdot N_L N_e \\ T_L &\approx (M+1) \cdot (3d+1) \cdot t N_L N_e \end{aligned} \tag{5.5}$$

$$\tag{5.6}$$

For the test time, the same principle is applied (only distances computation and research of a minimum). The test time can thus be estimated by :

$$T_L \approx M \cdot (3d+1) \cdot t N_T \tag{5.7}$$

The measured computation time follows correctly the theoretical estimation (relative

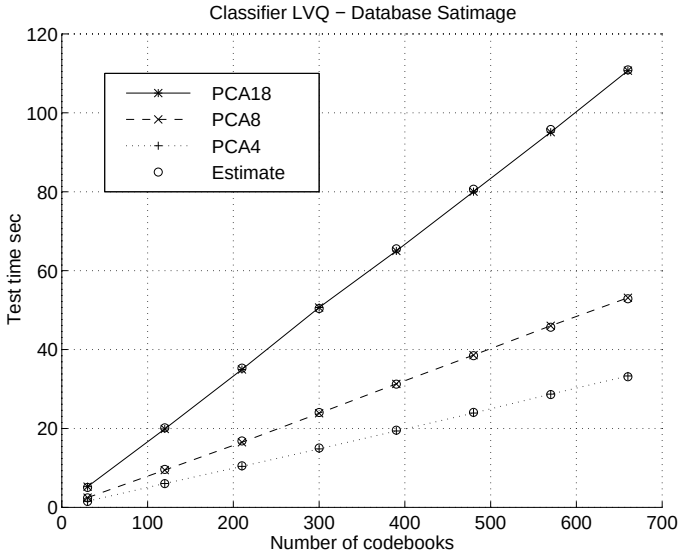

Figure 5.30: Evolution of the test time with the number of prototypes for various dimensions; comparison with the theoretical estimation.

error inferior to 10%. Figure 5.30 shows the evolution of the test time in function of the number of prototypes, for different dimensions (18, 8 and 4). Figure 5.31 precises the evolution of the learning and test times in comparison to the theoretical estimations. The estimations of the computation time have been established on the "Satimage" database, with the Holdout test method.

5.4.5 Conclusions

The LVQ algorithm is a good candidate for hardware realizations. The memory size can be chosen and remains fixed. It is directly related to the number of prototypes. It influences also the classification error, but does not forbids an implementation on a dedicated hardware. The necessary operations for the learning and recognition phases are simple: addition, subtraction, comparison and multiplication by a real value α, eventually decreasing (only for learning). No division required for the LVQ1 algorithm. The computation time grows linearly with the number of prototypes, and the ratio between the learning and the test times is about the number of epochs. This classification algorithm has been implemented on the ANKC analog classification chip developed in the framework of Elena.

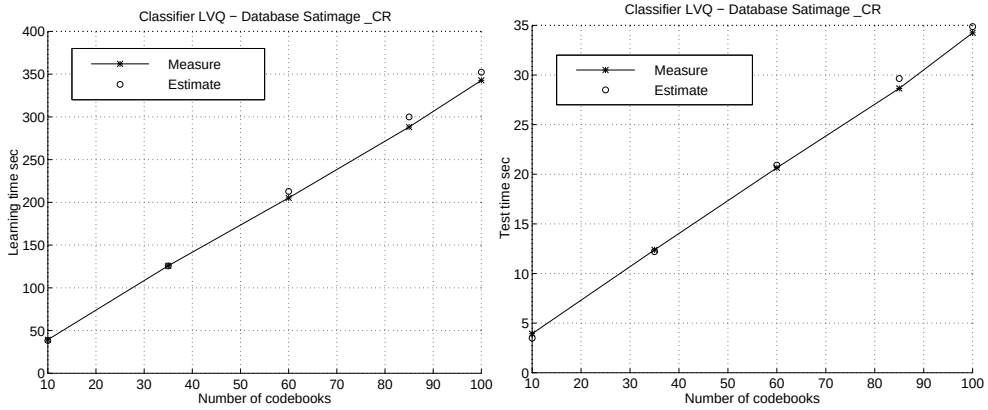

Figure 5.31: Evolution of the learning time (left) and the test time (right) with the number of prototypes. Comparison with the theoretical estimation. The number of epochs has been fixed to 10.

5.5 IRVQ classifier

5.5.1 Algorithm description

The IRVQ classifier has been developed in the framework of ELENA . It is a sub-optimal Bayesian classifier based on radial Gaussian kernels which uses an iterative unsupervised learning method based on vector quantization to obtain a low-memory kernel density estimator, while keeping sufficiently accurate estimations of probability densities. After the vector quantization process, variable radial Gaussian kernels are put on each codeword of the resulting codebook in order to obtain an estimation of the probability densities in each class $\hat{p}_x(u|\omega_i)$.

The main trick of this algorithm is the computation of the appropriate variable widths factors for the kernels used in the estimation of probability densities. These widths factors are computed from the inertia of the points of the learning set associated to each codeword resulting from the vector quantization.

Let us now examine the different operations of the learning and recognition phases in order to study the influence of different parameters of the IRVQ algorithm.

Learning process

Suppose we have a learning set A_N of observations $x(n)$, $1 \leq n \leq N$ with their associated known classes $\omega_{x(n)}$: $A_N = \cup\{A_{N_i}\}$ with $A_{N_i} = \{x(n), \omega_{x(n)} = \omega_i, 1 \leq n \leq N_i\}$ and $N = \sum_{i=1}^{c} N_i$. The principle of the IRVQ algorithm is to use a vector quantization technique to split into clusters the portion of the space where vectors can be found. The aim is thus to approximate the sets of patterns A_{N_i} by sets of so-called centroids $B_{M_i} = \{c(m), \omega_{c(m)} = \omega_i, 1 \leq m \leq M_i\}$, where $M_i << N_i$, roughly keeping the same probability density of vectors for sets A_{N_i} and B_{M_i}. The entire set of centroids $B_M = \cup\{B_{M_i}\}$ is the codebook. The vector quantization used is an iterative version of the "Generalized Lloyd Algorithm" [16], the "Competitive Learning" (CL); the iterative character of this rule is used to set the position of the centroids and to evaluate the inertia of each cluster in order to obtain an approximation of the optimal variable width factors associated to each cluster. The principle of this method is the following in each class ω_i.

First, the M_i centroids $c(m)$ are randomly initialized to any of the N_i patterns, keeping the same a priori probabilities of classes for both sets A_N and B_M. The inertia coefficients $i(m)$ associated to each cluster are initialized to zero. Then, each of the N_i patterns $x(n)$ is presented to the set B_{M_i}, and the centroid $c(a)$ closest from $x(n)$ is selected and moved in the direction of the presented pattern while its inertia coefficient is updated:

$$c(a) = c(a) + \alpha(x(n) - c(a)) \tag{5.8}$$

$$i(a) = i(a) + \alpha(\|x(n) - c(a)\|^2 - i(a)) \tag{5.9}$$

where a is the index of the closest centroid to a learning vector $x(n)$ and α is an adaptation factor ($0 \leq \alpha \leq 1$) which must decrease with time during the learning to ensure the convergence of the algorithm.

After several presentations of the whole set of patterns A_{N_i}, the distribution of centroids $c(m)$ in B_{M_i} is supposed to reflect this of the training set A_{N_i}, and the inertia coefficients $i(m)$, $1 \leq m \leq M_i$, converge to the average inertia of points in the

clusters associated to $c(m)$ (5.9) being a kind of convex combination at each iteration between the previously estimated value of $i(a)$ and a new contribution $\|x(n) - c(a)\|^2$ due to the input vector $x(n)$):

$$i(m) \simeq \frac{1}{n(m)} \sum_{v \in C(m)} \|v - c(m)\|^2 \tag{5.10}$$

where the sum goes on every point v of the original training set belonging to $C(m)$, the cluster associated to the centroid $c(m)$ in the Voronoi tessellation obtained after the vector quantization, and $n(m)$ is the number of these points.

Recognition

The classification decision is obtained by applying the Bayesian approach using the classifier estimates:

$$Decide\ u \in \omega_s \Leftrightarrow s = Arg \max_{1 \leq i \leq c} \{\hat{P}_i\, \hat{p}_x(u|\omega_i)\}. \tag{5.11}$$

$\hat{P}_i$ are the a priori probabilities of the classes estimated on the learning set ($\hat{P}(\omega_i) = N_i/N$) and the probability densities in each class $\hat{p}_x(u|\omega_i)$ are computed for each point u to classify by using a variable kernel estimate with the centroids resulting from the learning process:

$$\hat{p}(M_i, u|\omega_i) = \frac{1}{N_i} \sum_{m=1}^{M_i} n(m)\, K\left(\frac{u - c(m)}{h(m)}\right) \tag{5.12}$$

With

- $n(m)$ the number of points belonging to $C(m)$, the cluster associated to centroid $c(m)$ in the Voronoi tessellation obtained after the vector quantization;
- $K(\cdot)$ the variable radial Gaussian kernel function:

$$K\left(\frac{u - c(m)}{h(m)}\right) = \frac{1}{\left(h(m)\sqrt{2\pi}\right)^d}\, exp\,(-\frac{1}{2}\left(\frac{\|u - c(m)\|}{h(m)}\right)^2), \tag{5.13}$$

- $h(m)$ the *width factor* of the kernel function associated to centroid $c(m)$. According to [38], the optimal value of $h(m)$ is given by:

$$h(m)_{opt} = \gamma \sqrt{\frac{3}{2\, ln2}\, \frac{i(m)}{d}} \tag{5.14}$$

  where $i(m)$ is the inertia coefficient of centroid $c(m)$ computed during the learning process and γ is a multiplying factor which, as we will see in the following, depends on the codebook size, on the database characteristics and on the data space dimension.

Figure 5.32 illustrate the resulting codebook configuration after the use of the IRVQ algorithm on a two-dimension two-class problem with gaussian distributions. The total number of centroids was set to 60. The circles are the width factors $h(m)$ of the gaussian kernel functions associated to each centroids after the learning process (with $\gamma = 1$).

For a theoretical justification and more details about the IRVQ classifier see [39], [38] or [37].

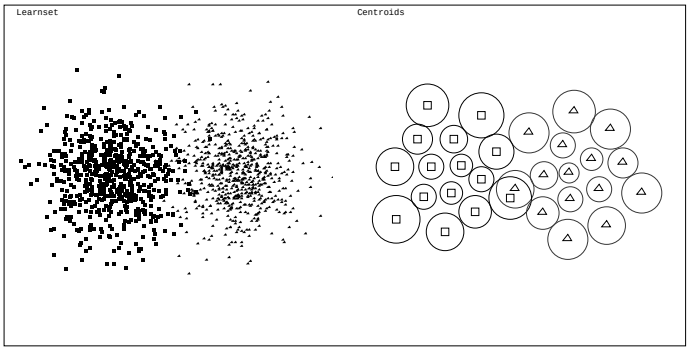

Figure 5.32: Illustration of the use of the IRVQ algorithm on a two-dimension two-class problem with gaussian distributions.

5.5.2 Parameters configuration and benchmarking results

Number of learning steps and initial learning factor

These parameters correspond respectively to the "TOTAL_NB_LEARNING_STEPS" and "LEARNING_FACTOR" resources of the IRVQ module in packlib [34]. They are the two main parameters of the Competitive Learning algorithm setting the positions and width factors of the codebook. Extensive tests of these two parameters on the different databases of the ELENA project shown that these two parameters were not really critical. Their optimal values corresponds to the usual values rending the best convergence of the Competitive Learning algorithm. In order to converge to a good codebook configuration the number of learning steps simply needs to be sufficient (more than 5 epochs [8]) and the initial value of the α learning factor (which linearly decreases during the learning) may be set between 0.2 and 0.7. After the tests of these parameters, it appears that the values corresponding most of the time to the best results were a learning on 10 epochs associated to $\alpha_{init} = 0.3$. Since these parameters where not critical, these values where chosen for all the benchmarks of the IRVQ algorithm on all the ELENA databases (all the results reported here below were thus obtained using these particular values).

Figures 5.33 show the evolution of the classification error in function of the initial value of the α learning factor for various number of learning epochs on the "Phoneme_CR" database. The confidence intervals values on the errors are relatively large (more or less 1%) and the curves are thus a bit chaotic. It is clear that a number of epochs larger than 5 becomes sufficient and that an acceptable initial value for the learning factor will be between 0.2 and 0.8.

Codebook size (M)

This parameter corresponds to the "NB_CODEBOOK" resource of the IRVQ module in packlib [34]. It sets the number of centroids M to use by the IRVQ for the approximation of the probability densities. We have $M = \sum_{i=1}^{c} M_i$, with the M_i chosen in order to keep the same a priori probabilities of classes for both sets A_N and B_M. On the hardware point of view, the complexity of the computations (memory usage, learning

[8] An epoch corresponding to one presentation of the entire learning set

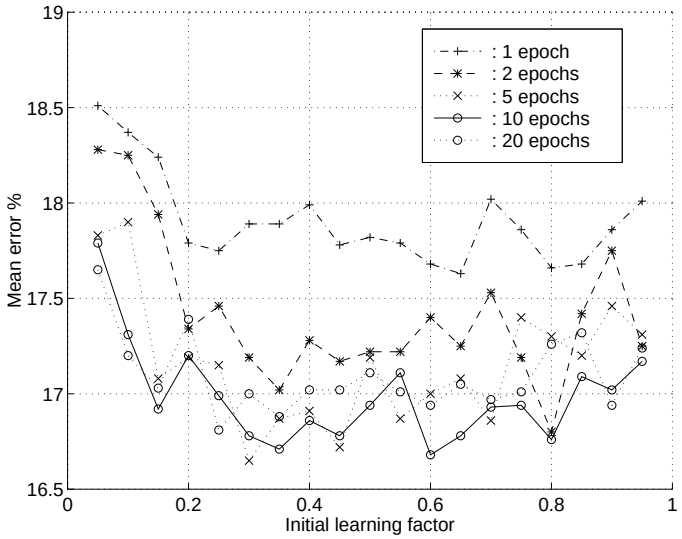

Figure 5.33: Averaged Holdout errors on the "Phoneme_CR" database in function of the initial learning factor for various sizes of the number of learning steps (in epochs)

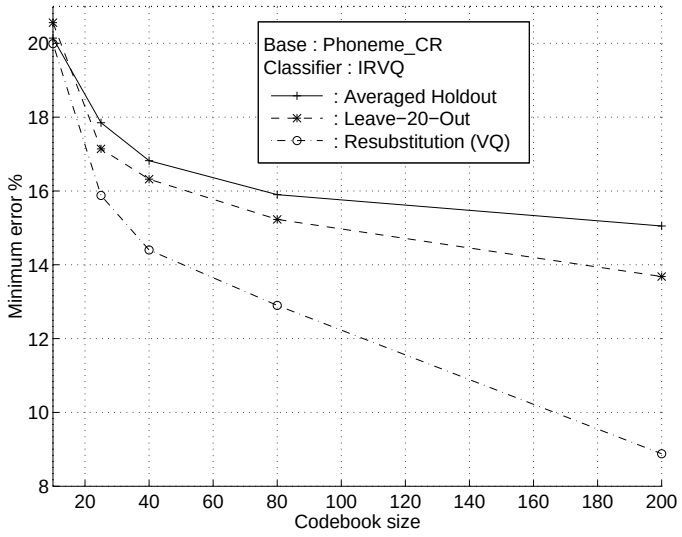

Figure 5.34: Averaged Holdout, Leave-20-Out and resubstitution error corresponding to gamma optimum in function of the mean value of the codebook size per class

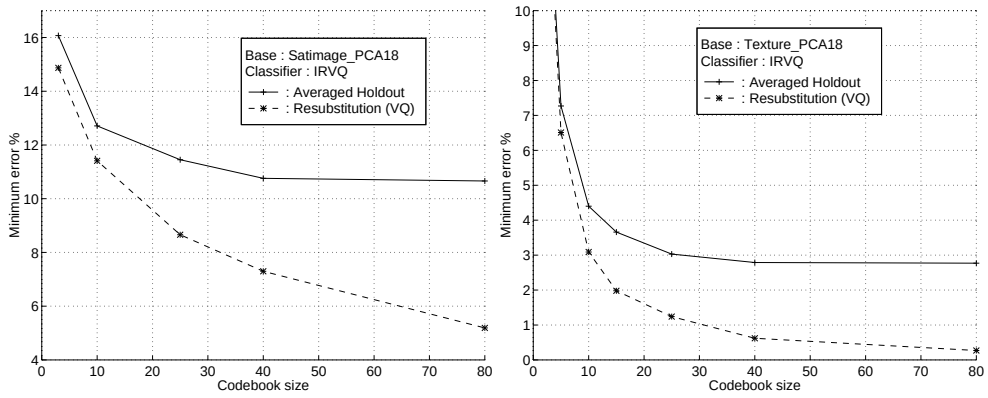

Figure 5.35: Averaged Holdout and resubstitution error corresponding to gamma optimum in function of the mean value of the codebook size per class for the "Satimage_PCA18" (left) and "Texture_PCA18" (right) databases

and recognition times) is directly linked to the codebook size. On the other hand, it is logical that the classification error decreases as M increases. The chosen value for the codebook size must thus be a compromise between the hardware constraints, fixed by the available hardware and the needs of the problem (on-line recognition, portability of the recognition system,...), and the desired classification performances. Figures 5.34 and 5.35 show the evolution of the classification error in function of the mean value of the codebook size per class ($M_{mean} = M/c$) for different real databases.

The Holdout error curves rapidly decrease with the codebook size for small values of M and then tend to a constant value as M continue to increase. There is thus a value of M from which the decrease of the classification error will not be significant and the kernel density approximation of the IRVQ can't really perform better. For the "Satimage_PCA18" and "Texture_PCA18", this value corresponds respectively to 40 and 25 codewords per class, but for the "Phoneme" database this threshold value seems to be much larger; this is due to the available number of points in the database: for the "Satimage" database, we have 6735 points in 6 classes; 5500 point in 11 classes for "Texture" and 5404 points in 2 classes for "Phoneme". 25 codewords per class for the "Texture" database corresponds thus to 5% of the total amount of available points in each class; 40 codewords per class for the 'Satimage" database corresponds to 3.73% while the same codebook size corresponds to 1.5% of the total amount of available points in each class of the "Phoneme" database, this value being not sufficient for a good approximation of the densities. Moreover, if the densities inside each class have a lot of different modes, it will be necessary to increase the codebook size in order to obtain acceptable approximations of the densities.

These figures also illustrate the properties of the different error-counting methods (see chapter 3 for more details). On each of these figures, we can clearly observe the optimistic behavior of the resubstitution method. The points of the curves corresponding to this method are always lower than the other, and the difference between the curves increases with the codebook size. On figures 5.34 we can see that the Leave-k-Out curve is very close to the Holdout one, showing the same evolution. The only difference

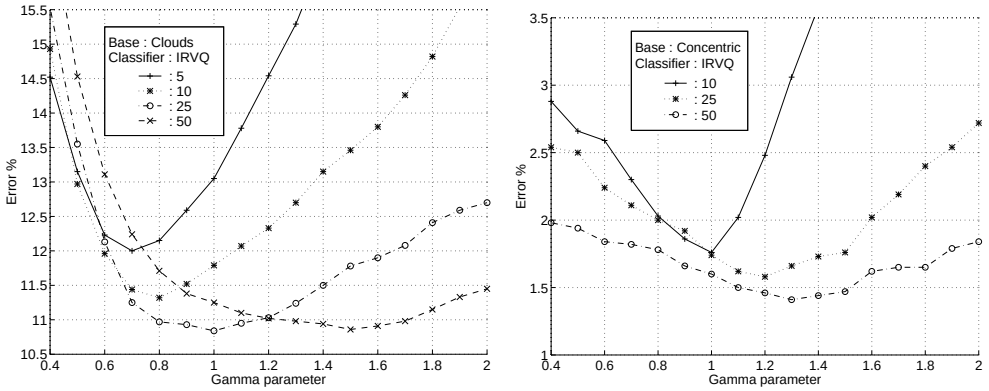

Figure 5.36: Averaged Holdout error function of the gamma parameter for the "Clouds" (left) and "Concentric" (right) artificial databases

is due to the reduced size of the learning set for the Holdout method, which is thus a bit pessimistic.

To conclude, we can say that the value of the codebook size will be chosen according to the hardware constraints, the desired classification performances and the type of distribution inside each class. Apparently, a codebook size corresponding to 4% of the total amount of available points seems to be sufficient for the IRVQ algorithm if the number of points in each class of the database is large enough to well represents the probability density functions of the distributions. The fact that this number is sufficient can be deduced from Holdout curves like in figure 5.35: if the curve tends to become constant from a certain codebook size, this means that this codebook size is sufficient for the IRVQ and that there was enough points in the initial database to well represent the data. If the curve continue to decrease, even for large codebook size, the size of the original database is not sufficient and more data should be collected.

Kernel width multiplicative factor (γ)

This parameter corresponds to the "SCALING_FACTOR" resource of the IRVQ module in packlib [34]. It is the more critical parameter of the IRVQ algorithm and has thus been studied extensively.

The γ parameter was introduced in [38] in order to take into account the dependence of the optimal width factor $h(m)$ on the codebook size, the database characteristics and the data space dimension. The underlying theory leading to the computation of $h(m)$ in function of the inertia coefficients $i(m)$ associated to the codewords during the learning of the IRVQ algorithm was presented in [39]; it stated that the optimal width factor will corresponds to $\gamma = 1$ if the number of centroids is sufficiently large so that the learning leads to clusters small enough in order to allow to approximate the true probability density inside each cluster by a constant. In [38], extensive studies of the IRVQ algorithm showed that this hypothesis is not always respected and that γ should vary between 0.68 and 1.5 for some particular problems.

According to this study, as the codebook size decreases, the vector quantization will lead to larger clusters which do no more have the above mentioned property of being

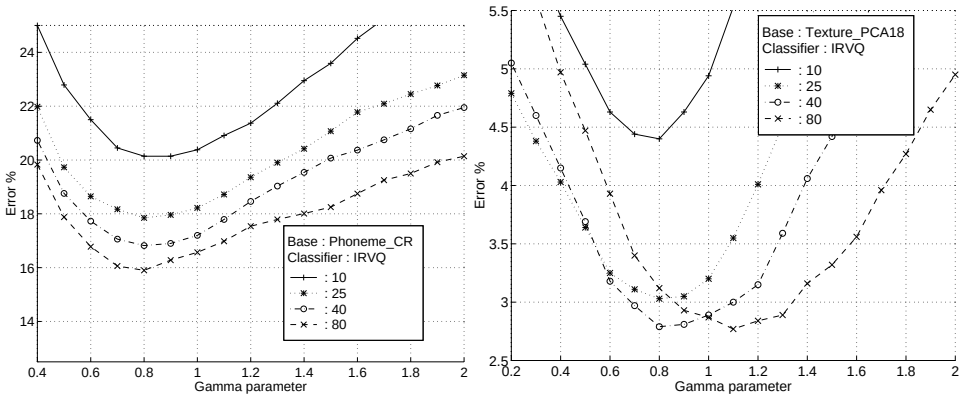

Figure 5.37: Averaged Holdout error function of the gamma parameter for various codebook sizes for the "Phoneme_CR" (left) and "Texture_PCA18" (right) databases

"small'. In fact, if the codebook size exactly corresponds to the number of modes in the learning distribution the optimal value of $h(m)$ will corresponds to the maximum likelihood estimate of the standard deviation of an isotropic Gaussian centered on centroid $c(m)$ and modeling the mode of the distribution centered on $c(m)$ [40, 6]. This minimum value of the optimal $h(m)$ is linked to the averaged inertia coefficient $i(m)$ by:

$$h(m)_{min}{}^2 = \hat{\sigma}^2 = \frac{i(m)}{d} \tag{5.15}$$

It corresponds to $\gamma = 0.6798$ in equation 5.14.

This phenomenon is well illustrated on figure 5.36, showing the averaged Holdout error function of the gamma parameter for the "Clouds" and "Concentric" artificial databases. The curves of the Holdout error in function of γ for real databases (figure 5.37 for "Phoneme" and "Texture_PCA18" and the left part of figure 5.41 for "Satimage_PCA18') show the same behavior.

The dimension dependence of the γ optimum value is illustrated by the test of the IRVQ classifier on the set of the seven artificial databases corresponding to the "Gaussian" problem, the only difference between these databases being their dimensionality, ranging from 2 to 8. For the different dimensions, the γ multiplying factor corresponding to the minimum misclassification error was computed for a codebook size of 20, 50, 100 and 200 centroids (averaged Holdout test over five partitions of the database in a learnset and a testset of the same size). The results are reported in figure 5.38 while figure 5.39 shows the averaged Holdout classification errors in function of the γ parameter for two particular codebook sizes. The importance of the data space dimension on the value of the γ parameter corresponding to the optimal width factor is well illustrated on figure 5.38. This phenomenon may be explained as follows: for a given codebook size, when the dimension increases, the coverage of the initial distribution by the codebook will be worst and the optimal width factor will tend to reach the value corresponding to the maximum likelihood estimate of the standard deviation (equation 5.15). This is due to the "empty space phenomenon" problem appearing for kernel estimators built on finite datasets in large dimension [23].

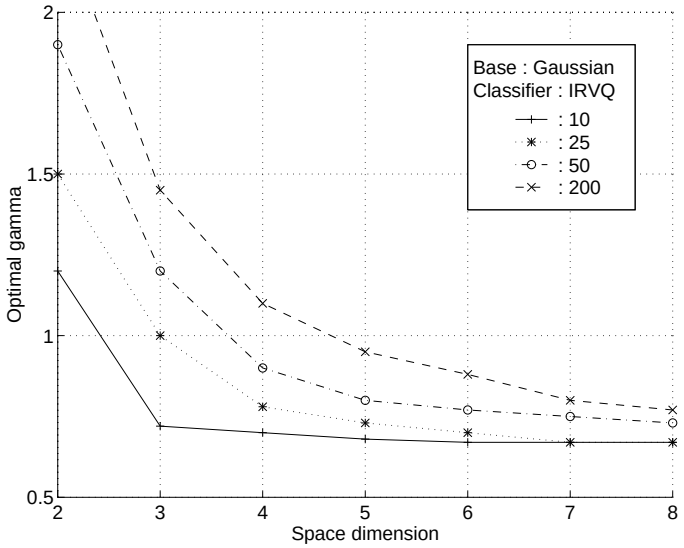

Figure 5.38: Gamma optimum in function of the data space dimension for various codebook sizes per class

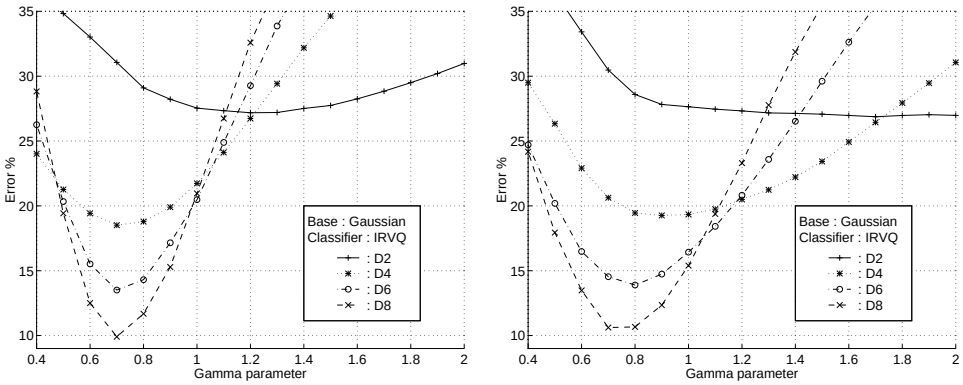

Figure 5.39: Averaged Holdout error function of the gamma parameter in function of the data space dimension for 10 centroids per class (left) and for 50 centroids per class (right)

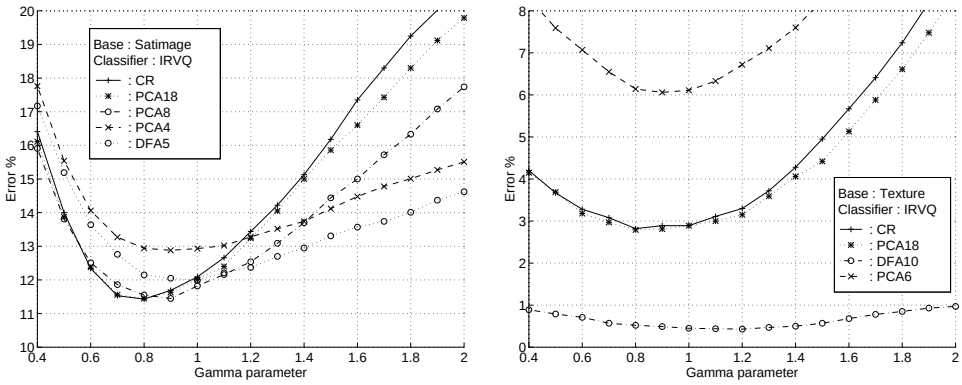

Figure 5.40: Averaged Holdout error function of the gamma parameter for various preprocessings of "Satimage" with 25 centroids per class (left) and of "Texture" with 40 centroids per class (right)

The use of the Principal Component Analysis (PCA) and of the Discriminant Factorial Analysis (DFA) as dimension reduction preprocessing seems to be useful for the IRVQ algorithm. This is illustrated in figure 5.40 showing the Holdout error in function of γ for various preprocessings for the "Satimage" and "Texture" databases. For the "Satimage" database, the performances of the IRVQ are almost identical if it uses the "CR" database or the "PCA18" or "PCA8" databases. As "Satimage_PCA8" corresponds to a memory need 5 times smaller than "Satimage_CR', this preprocessing is very useful. To keep less principal dimension than in "Satimage_PCA8" will decrease the performances of the classifier. This may be explained by the fact that dimension 8 corresponds for the "Satimage" database to the estimate of its fractal dimension and also to the breakpoint of the cumulated PCA inertia curve. Again, we can see that the γ value corresponding to the minimum of the "Satimage_CR" curve is a bit smaller than its equivalent on the "Satimage_PCA8" curve.

The same observations about PCA preprocessing may be done for the "Texture" database; but for this it will be necessary to keep more than the 8 first principal dimensions. For this database, the performances of the IRVQ with the DFA preprocessing outperform any other preprocessing, and the gain in performance from the "CR" database is very important, with a well less marked minimum value (rendering the IRVQ classifier less sensitive the to γ parameter value). This is not the case for the "Satimage" database for which the "DFA5" preprocessing error curve corresponds more or less to a "PCA5" one.

Another observation may be done on all the figures showing the Holdout error in function of the γ value (in particular in figures 5.36, 5.39 and 5.40): the minimum of the error-γ curves become less and less sharp as the codebook size increases or as the database dimension decreases (which is equivalent for the "empty space phenomenon" point of view). A dimension reduction will thus not only be interesting on the hardware point of view, but also for the classifier robustness in function of it's optimum parameters values. So, for a given memory usage, the dimension reduction will enable to be less precise for the computation of the optimal value of the γ parameter (the minimum being less sharp); and it will enable to increase the codebook size rendering

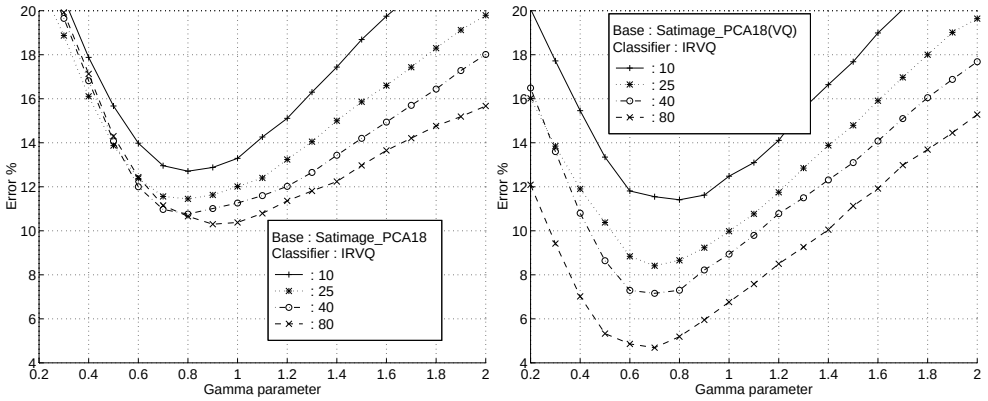

Figure 5.41: Averaged Holdout error (left) and resubstitution error (right) function of the gamma parameter for various codebook size per class for the "Satimage_PCA18" database

a more precise classifier also with less dependence on the gamma parameter value.

We have still to insist on the importance of the use of a good test method for the IRVQ classifier (as the Holdout or the Leave-One-Out one). The problems of the resubstitution method are illustrated in figure 5.41: the two part of the figure shows the error curves of the IRVQ classifier on the "Satimage_PCA18" database in function of the γ parameter value for various codebook sizes; the left curves coming from an averaged Holdout test while the right one were computed using the resubstitution method. From this figure, it is clear that the resubstitution method has too optimistic results. But another phenomenon also appears: for the resubstitution method, the apparent optimal value of the γ parameter tends to decrease while the codebook size increase. So, while averaged Holdout tests enable to find the optimal parameters configuration of the IRVQ classifier for the best generalization properties (the γ optimum being the same as the γ optimum for Leave-k-Out tests), the resubstitution method will work at the opposite: as the same data are used for the learning and the test of the classifier, the error on the test data will be lower for a smaller γ value as the codebook size increase. This phenomenon may be explained as follows. It becomes more probable with the resubstitution method to find a test pattern in the vicinity of a codeword as the codebook size increase and it becomes thus better to take into account this increasing neighborhood probability by reducing the value of γ at the expense of the generalization properties.

Has we have seen in this section, the γ width multiplicative factor is the most sensible parameter of the IRVQ algorithm. Its usual optimum value is always in the neighborhood of 0.8 [9]; but the exact optimal value depends on the codebook size, on the dimension and on the database characteristics which are never precisely known. Due to this fact, the optimal value will never have any analytical representation but could be deduced from experimental curves like the one shown in this section obtained by the mean of Holdout tests proceed with the learnset taken as database (in order to

[9] $\gamma = 0.8$ has been taken for all the comparative tests with other classifiers in order to have an IRVQ classifier totally independent of the databases characteristics

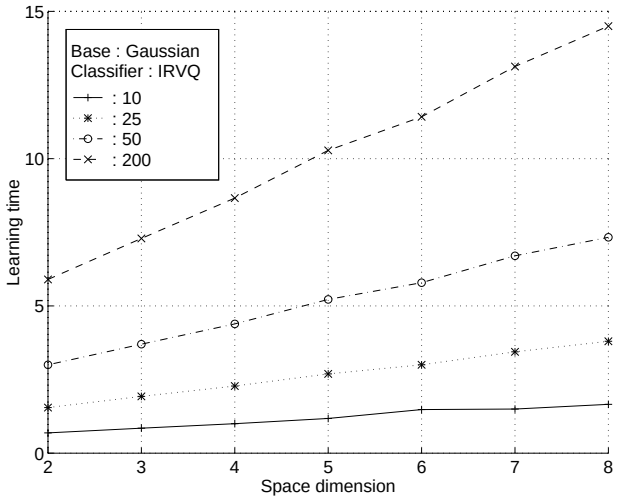

Figure 5.42: Averaged learning time (in seconds) of 2500 vectors of the "Gaussian" database in function of the data space dimension for different codebook size per class

keep a totally independent testset).

In what concerns the further developments relatives to the IRVQ algorithm two ideas could be developed. The first one concerns the following; as the optimum γ value depends on the distribution characteristics and as these characteristics may vary from one class to another, it could be useful to set one γ parameter per class. In this case the optimum gamma vector (of dimension c) could be obtained using numerical minimization methods associated with Holdout tests (like the well known "simplex" method for example). The second idea would be to use a more performant vector quantization method than the Competitive Learning in order to build better codebook.

5.5.3 Hardware constraints

Given:
- c the number of classes
- d the number of dimensions (or attributes)
- N_L the number of points in the learning set
- N_i the number of points in class i of the learning set
- M the codebook size (number of memorized centroids)
- P_i the a priory probability of class i ($Pi = N_i/N_L$)
- N_R the number of points in the testset (to recognize)

Suppose that the time for an ADD operation is approximatively the same as the time for a MUL (t seconds per simple operation).

- The memory requirement of the IRVQ classifier will be $M(d+2)$ memory words: d attributes per centroid plus the kernel radius and the number of points associated to each kernel.

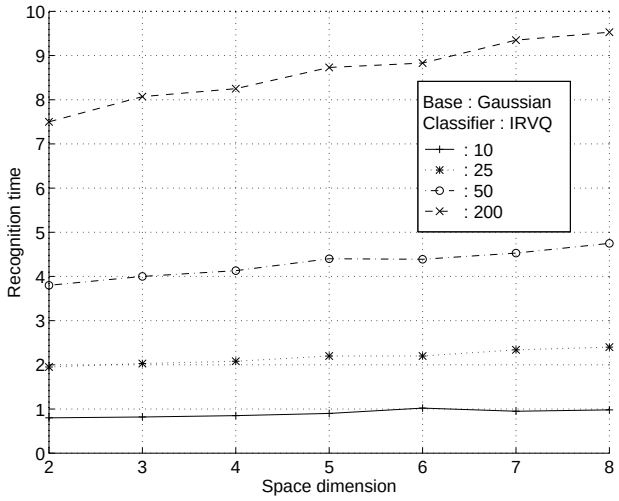

Figure 5.43: Averaged recognition time (in seconds) of 2500 vectors of the "Gaussian" database in function of the data space dimension for different codebook size per class

- The learning time may be estimated by:

$$T_L = N_e \sum_{1 \leq i \leq c} N_i (T_{nn} + T_{upd}) \tag{5.16}$$

with N_e, the number of learning epochs; T_{nn}, the time for the distance computation to every centroid plus the time to find the nearest one ($T_{nn} = 3tdM_i + tM_i$) and T_{upd}, the taken to update the centroid position, its inertia value and the learning factor value ($T_{upd} = 3td + 3t$), which is negligible. Using $M_i = MP_i$, we have:

$$T_L = tN_e((3d+1)M \sum_{1 \leq i \leq c} N_i P_i) \tag{5.17}$$

With a rough approximation T_L is of order $O(N_e M N_L d)$.

- The recognition time may be estimated by:

$$T_R = N_R \sum_{1 \leq i \leq c} T_{pdf} + T_{clas} \tag{5.18}$$

with T_{class}, the time used to compare the densities and attribute the class label, which is negligible and T_{pdf}, the time for the computation of the probability density inside each class ($T_{pdf} = M_i(3dt + 5t + 100t)$) [10].

Thus, using $\sum_{1 \leq i \leq c} Mi = M$, we have:

$$T_R = tN_R M(3d + 100) \tag{5.19}$$

Figures 5.43 and 5.42 show the evolutions of the learning and test time in function of the database dimension for various codebook sizes (benchmarks on the "Gaussian" database). Equations 5.19 and 5.17 fit correctly the different curves for $t =$

[10] $100t$ is supposed to be the time for the computation of the POW or EXP operations (25 ADD and MUL for each operation to reach the double precision (64bits)

$0.15\,10^{-6}$ seconds per simple operation, which corresponds well to the performances of a SPARC20 machine. The linear dependence of the hardware constraints of the IRVQ classifier implemented on a sequential machine has been illustrated here. Even if a linear dependence is not so critical, this shows the real interest of a fully parallel implementation (the recognition time becoming than fixed), even for off-chip learning (the learning times being not so important for most of the problems).

5.5.4 Conclusions

The IRVQ classifier seems very robust. It will come out from chapter 6 that its performances are comparable to the one of the LVQ classifier (which also use some kind of vector quantization), but its main advantages are that it provides reliable estimates of the probability for each pattern to belong to each class and that its fully parallel hardware implementation is relatively simple [41].

Some guidelines for the optimal choice of the parameter configuration could be inferred from this study of the IRVQ benchmarking results.

The number of learning steps and and the initial learning factor α_{init} are not critical parameters; they may be respectively set to 10 epochs with $\alpha_{init} = 0.3$. The choice of the codebook size must be a compromise between the hardware constraints and the desired classification performances. The best performances are always reached for the the largest codebook sizes, but usually, if the learnset is large enough in order to correctly evaluate the densities inside each class (which is the necessary condition for a good learnset), the asymptotic performances of the IRVQ algorithm will be already reached for a codebook size corresponding to 2-6 percents of the learnset size. The usual optimal value of the γ width multiplicative factor is always in the neighborhood of 0.8; but the exact optimal value be deduced from an experimental Holdout curves.

For most of the problems, a dimension reduction preprocessing and the use of a large codebook will enable to build more robust IRVQ classifiers, less dependent of the γ width factor parameter.

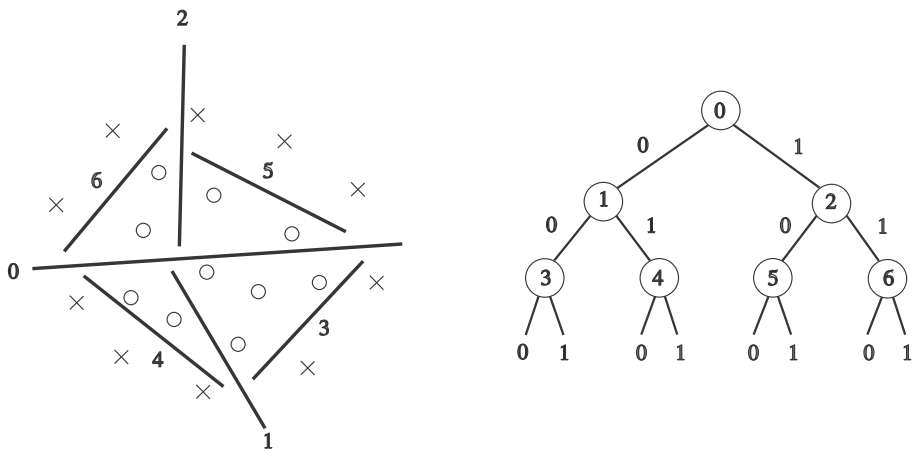

Figure 5.44: PLS classification principle and network organization

5.6 PLS classifier

5.6.1 Brief description

As has been explained in previous reports provided by the ELENA project (see, for instance, the deliverable R2-A-P), *Piecewise Linear Separation (PLS)* incremental algorithms try to find in an incremental way the best linear approximation for the discriminant function able to solve a certain classification task.

From the different types of PLS incremental algorithms, we have selected the *Neural Trees* algorithm [42] for benchmarking purposes, since it provides a good example of the evolving principles associated with these incremental models. As depicted on the right side of figure 5.44, the network structure generated by this algorithm resembles a binary decision tree. The units constituting this structure are the Perceptron-like units the incremental algorithm generates during the training phase, and are trained sequentially on different training sets which result from the successive partition of the input problem. Each unit provides a linear discriminant function (determined by the weight vector obtained once finished the training process on its corresponding training set), so that the final discriminant provided by the network results from the combination of these individual linear discriminants. The left side of fig 5.44 shows how the linear discriminants associated with each generated unit are combined so as to produce the final discriminant function able to solve the stated classification task.

A key point to be considered when implementing a classifier based on PLS incremental models is the training scheme used for the individual units generated during the network construction process. As has been indicated in [43], [44], [45], the usual unit training algorithms (Perceptron [46] or Pocket [47]) may cause an useless evolving scheme, so that finally a totally redundant network structure is generated, which is not able to solve the classification task. So as to avoid these problems, and due to its inherent simplicity, we have adopted the method proposed in [44], which allows for a correct network evolving scheme.

Furthermore, another problem to be handled if a PLS-based classifier is to be implemented is related to the *overfitting* on the training data set. This means that, in the case the discriminant function for the stated classification problem follows a complex

nonlinear shape (as is the case, for instance, of the *sinus* database), the PLS algorithm will create a huge amount of units just for approximating the details of this function. As a consequence, the generalization properties of the network constructed in this way may become very poor. In order to compensate for the problems derived from the trend to memorize the whole training set, we have tried to control this *overfitting*-oriented evolving scheme by means of a very simple method. It consists of fixing a control parameter, N, which determines the minimum training size for a unit generated by the *Neural Trees* algorithm. In this way, if one units receives a training set composed of less than N vectors, this unit is not trained, and its output is fixed so as to point to the class to which most of the vectors in its input training set were assigned.

As it will be explained in the next sections, by controlling this parameter N we are able to balance the ratio between the complexity of the generated network and its classification performance (large values of N will produce very compact networks with poor classification abilities, while small values of N will allow the incremental algorithm to generate larger networks with higher classification performances on the training set). A compromise must be reached for the value of the parameter N, since too small values will degrade seriously the network performance when complex discriminant functions have to be estimated.

5.6.2 Parameters configuration

Regarding the internal parameters controlling the training process performed by the modified Pocket algorithm, they have been fixed to values which guarantee a smooth enough dynamic for the learning procedure. In more detail, the *learning rate*, η, has been fixed to a value of 0.7, while the number of iterations used for the Pocket algorithm (i.e., the number of weight updates) is 10000.

As has been explained previously, the main parameter which shall control the evolution of the network structure generated by the *Neural Trees* algorithm is the minimum training set size, N, for an individual unit generated during the network construction process. This parameter establishes the minimum size of a given training set for a unit generated by the *Neural Trees* algorithm, so that if a given unit receives a training set which is smaller than this value it is not trained, being its output fixed depending on the number of training vectors which belong to each class. In order to study the influence of this parameter on the performance provided by the classifier, several Averaged Holdout training/test processes have been performed for each database, each of them for a different value of N. Six different values (50, 75, 100, 150, 250 and 400) have been taken for this parameter, since, as it will be explained later, they are enough in order to demonstrate its influence on the classifiers behavior.

As a consequence, the smaller is the parameter N, the larger is the resulting network, since the generated units are allowed to be trained with very small data sets. This phenomenon is made evident in figures 5.45 and 5.46, which depict, for the 2-dimension *gaussian* and *phoneme* databases, respectively, the memory usage in words (obtained as the mean number of units generated multiplied by the input dimension plus 2 - one word needed for the threshold value and one more word used to establish the position of each unit in the tree generated by the *Neural Trees* algorithm) versus the value of the parameter N.

Regarding the influence of the parameter N on the classification performance of the

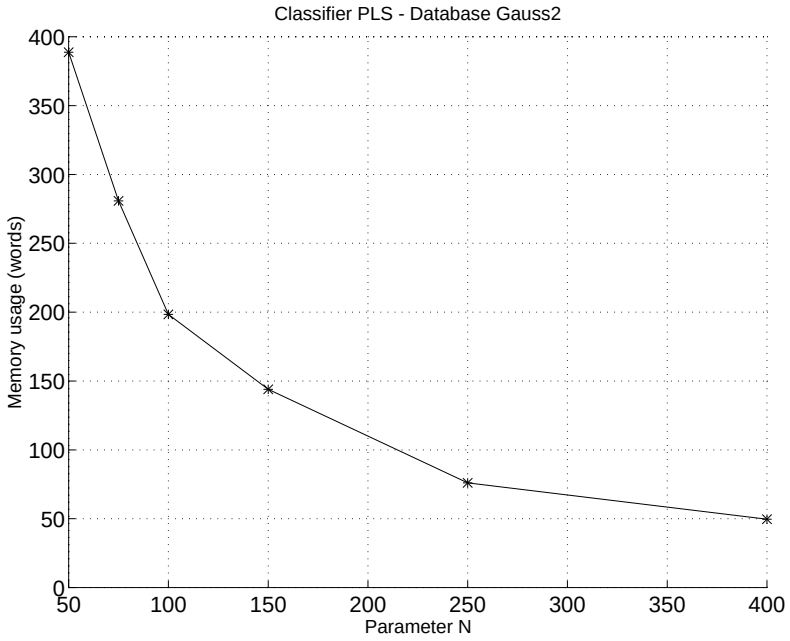

Figure 5.45: Resulting network size as a function of N - gaussian database

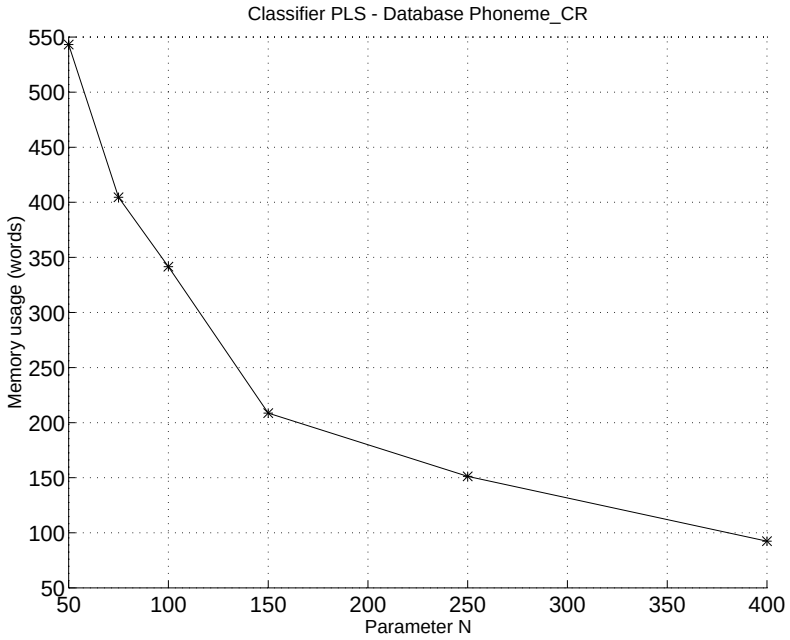

Figure 5.46: Resulting network size as a function of N - phoneme database

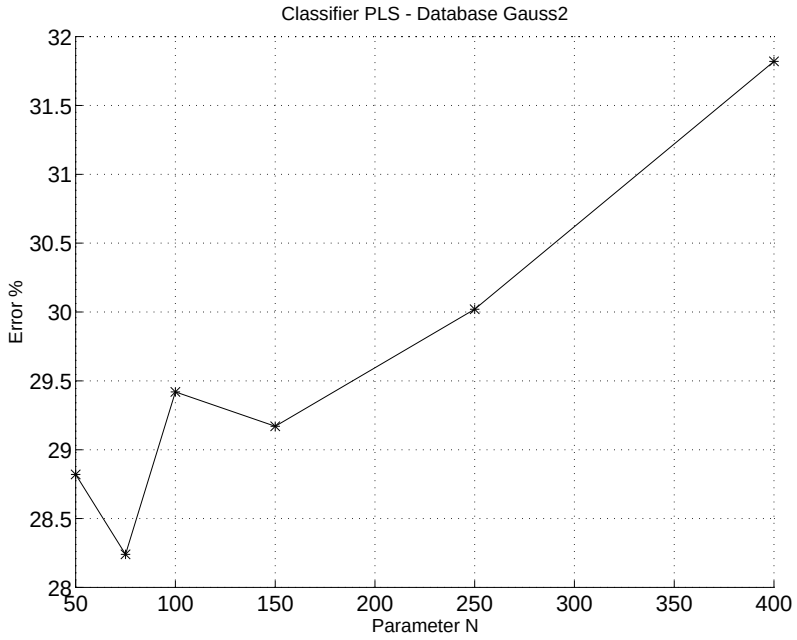

Figure 5.47: Classification error as a function of N - gaussian database

Neural Trees algorithm, very large values (i.e., values around 400) of this parameter will produce also quite high classification errors, as depicted in figures 5.47 and 5.48 for the 2-dimension *gaussian* and *satimage* databases, respectively. As can be seen, the removal of a large fraction of training patterns from the original training set does not allow the classifier to learn the main features of the problem to be solved.

Therefore, as N becomes smaller, the classification error yielded by the PLS classifier is decreased, since it allows for a more accurate approximation of the target discriminant function to be estimated. Note however that too small values for the parameter N will permit the training of new units on very small training sets, and this will usually result in larger classification errors, due to the *overfitting* on the training set (i.e., du to the attempt to approximate the discriminant function by memorizing the whole training set.

The simulations performed for the different artificial and real databases have shown that the optimum value (both in terms of number of units generated and classification error) to be used for the parameter N is always in the range 75-150 (close to 75 for the very simple artificial databases, and to 150 for the real databases). The results presented in the next section are those corresponding to the optimum value of the parameter N.

5.6.3 Benchmarking results

The first interesting result provided by the benchmarking analyses performed for the PLS-based classifier is the relatively strong dependence of the input problem dimension on the classification performance. This dependence can be observed in figures 5.49 and 5.50.

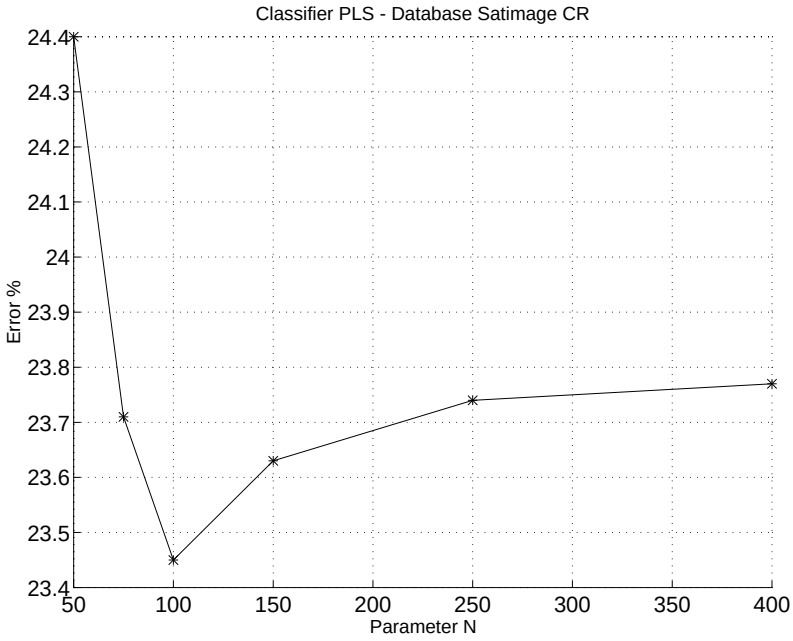

Figure 5.48: Classification error as a function of N - satimage database

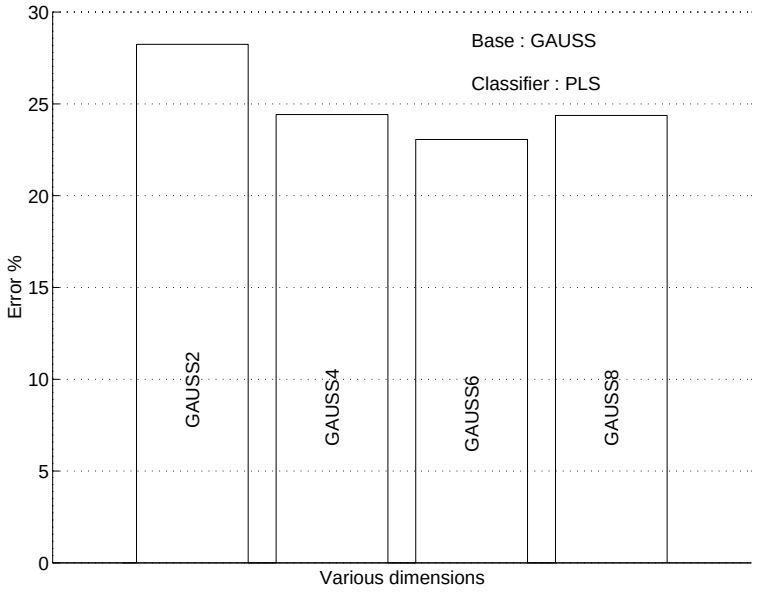

Figure 5.49: Classification error as a function of the input dimension - gaussian database

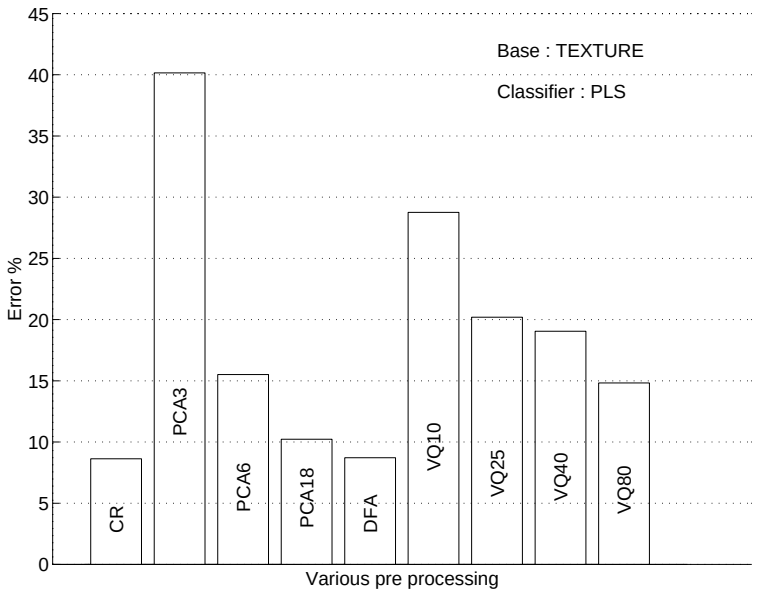

Figure 5.50: Classification error for different preprocessing techniques - texture database

Figure 5.49 represents the classification error provided for the different gaussian databases, and therefore allows for the comparison of the classifier performances for different databases, composed of patterns which follow the same distribution law, but with different dimension (i.e., number of features defining each vector). On the other side, figure 5.50 represents the classification error obtained when the original *texture* database is modified using several pre-processing techniques. We must pay especial attention in this figure to the bars labeled PCA18, PCA6 and PCA3, which represent the results obtained when only 18, 6 and 3 components, respectively, are held for each vector (from the initial 40 components) after a *Principal Component Analysis* is performed on the *texture* database. From these figures, it can be deduced that the performance of the classifier degrades as the input dimension is reduced (except for the 8-dimension *gaussian* database, though it is due only to the validation method - Averaged Holdout - used, a correct result should be expected for a Leave One Out procedure.

This effect can be interpreted as follows: if we keep constant the size of the training set but we reduce the dimension of the vectors constituting this set, this means that we are augmenting the complexity of the problem to be solved (just by increasing the density of vectors per unit volume), and thus a rather more complex discriminant function is required to solve the problem. Since PLS incremental models try to solve a given classification problem by means of a linear approximation of the corresponding discriminant function, poorer performances are to be expected as this discriminant function gets more and more complex.

Another quite interesting result is related to the behavior of the classifier when the training database is quantized (i.e., when only a fixed number of prototype vectors

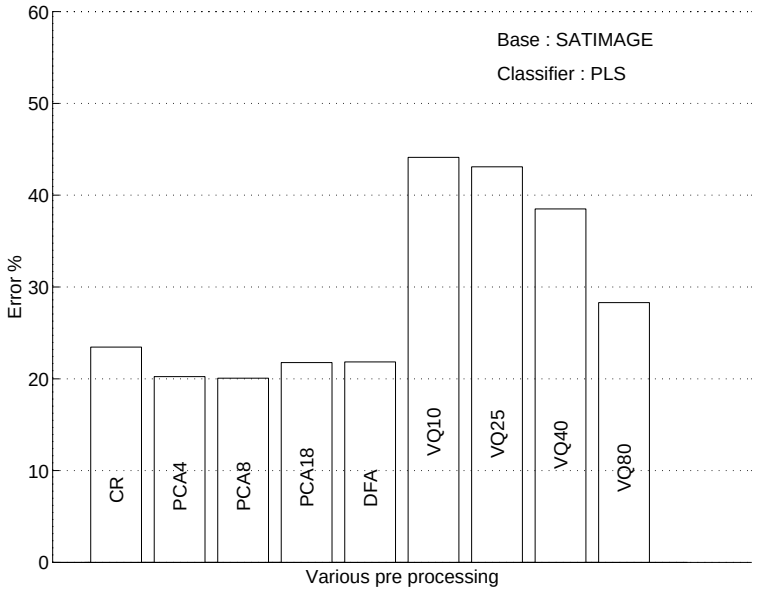

Figure 5.51: Classification error for different preprocessing techniques - satimage database

per class is used to train the network for the desired problem), as can be deduced from figures 5.50, 5.51 and 5.52, which depict, respectively, the classification error yielded for the *texture*, *satimage* and *phoneme* databases when they are pre-processed with various techniques. The results we are interested in are given by the bars labeled VQ10, VQ25, VQ40 and VQ80, which indicate the classification error when the training set is quantized with 10, 15, 40 and 80 patterns per class, respectively.

As can be deduced from these figures, the classification error provided by the classifier does not increase significantly if a quantization of the training set is performed (just compare the bars labeled CR - learning performed on the whole database - and VQ80 - learning performed with just 80 prototype vectors per class), meaning that the classifier is still able to find a good enough approximation for the desired discrimination function. Furthermore, the error decreases as the number of prototype vectors increases, since in doing this we are considering a more accurate representation of the problem to be handled. The results obtained show, however, that 80 prototypes per class seem to be enough in general in order to obtain classification performances close to those given when we use the whole training set.As a consequence, these results show that the solution provided this kind of classifiers is not influenced by the number of vectors used in the learning process, but they are sensitive to the representativeness of these vectors.

Furthermore, by comparing the error bars lapelled 5.52, 5.51 and 5.50, it can be deduced that the performance of the classifier improves for those classification problems where there exists a uniform distribution of the training patterns among the different classes defined in the input space (as is the case for the *texture* database). This behavior is imposed by the algorithms (Perceptron or Pocket) used to train the individual units

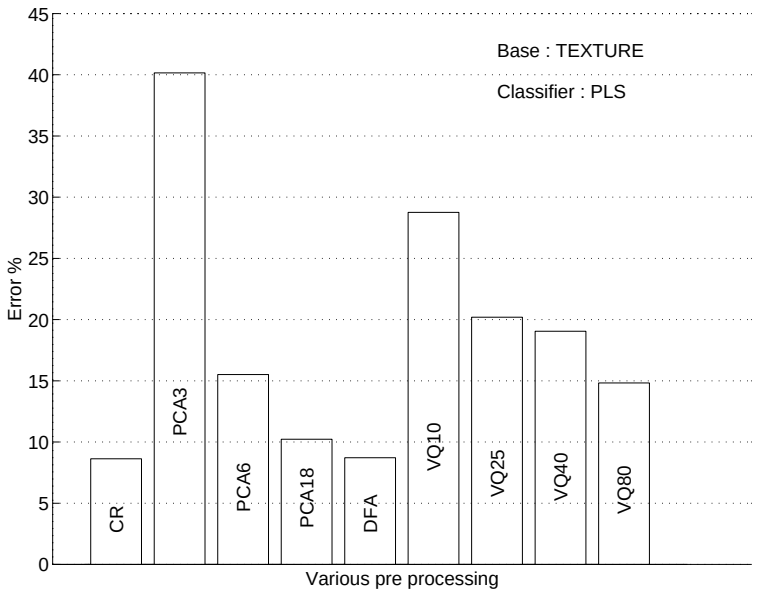

Figure 5.52: Classification error for different preprocessing techniques - phoneme database

generated by the incremental model, since these force the current unit to learn better those categories which are represented by a larger number of samples.

Regarding the memory usage (i.e., the number of parameters to be stored) required by the classifier to solve a certain classification task, it will strongly depend on the complexity of the problem to be handled, i.e., on the complexity of the target discriminant function to be approximated. This effect is made evident from the comparison of figures 5.53 and 5.54.

Figure 5.53 represents the classification error versus the memory usage (in words) for different artificial and real databases: *iris*, *concentric* (point labeled CONCE), *clouds* (point labeled CLOUD), *phoneme* (point labeled PH_CR) and *phoneme* pre-processed by means of PCA (point labeled PH_PCA). As can be easily deduced, for those problems which require less complex discriminant functions (as the artificial databases or the *iris* problem, for instance), the incremental algorithm will generate also fewer units than for more complex tasks (as the *phoneme* database).

The same effect can be observed in figure 5.54, which depicts the classification error versus the memory usage for the databases which result from applying different pre-processing techniques to the *texture* database. In this case, the influence of the problem complexity on the memory usage required by the classifier is made clear for the database which results from holding, after a PCA analysis, only 3 components of the original vectors constituting the *texture* database.

As it has been explained previously, this dimension reduction process causes the a considerable increase of the associated discriminant function, forcing the incremental model to generate a large amount of units in order to approximate accurately this discriminant. As a consequence, we can conclude that, due to the strategy used by the

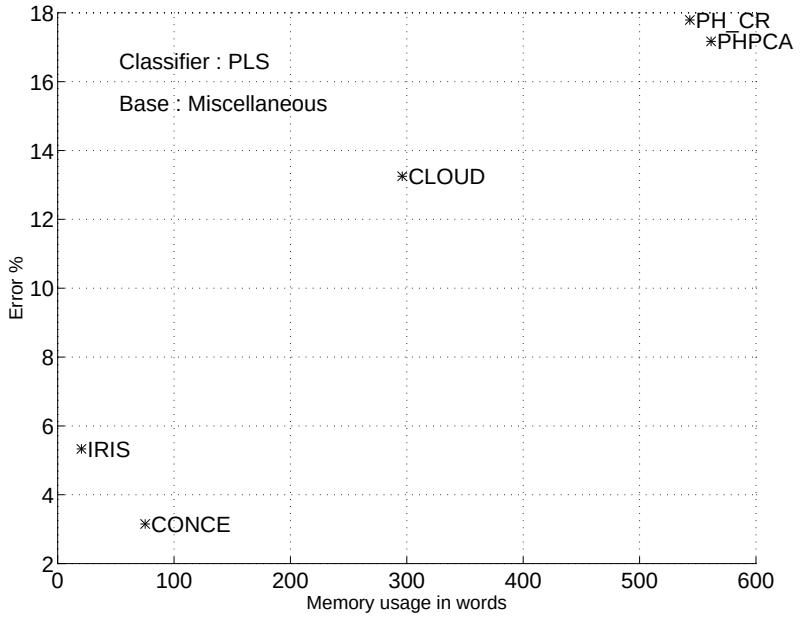

Figure 5.53: Memory usage for different databases

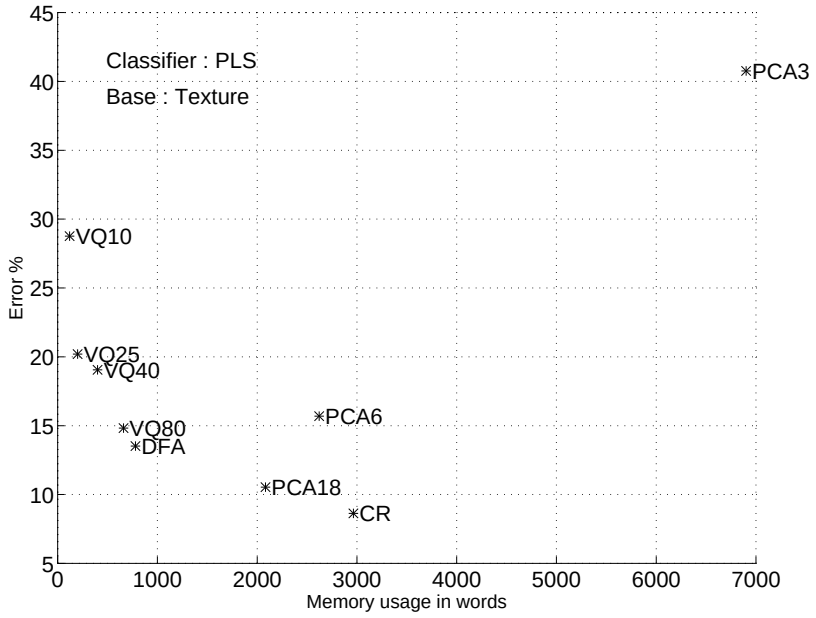

Figure 5.54: Memory usage for the texture database and different pre-processing techniques

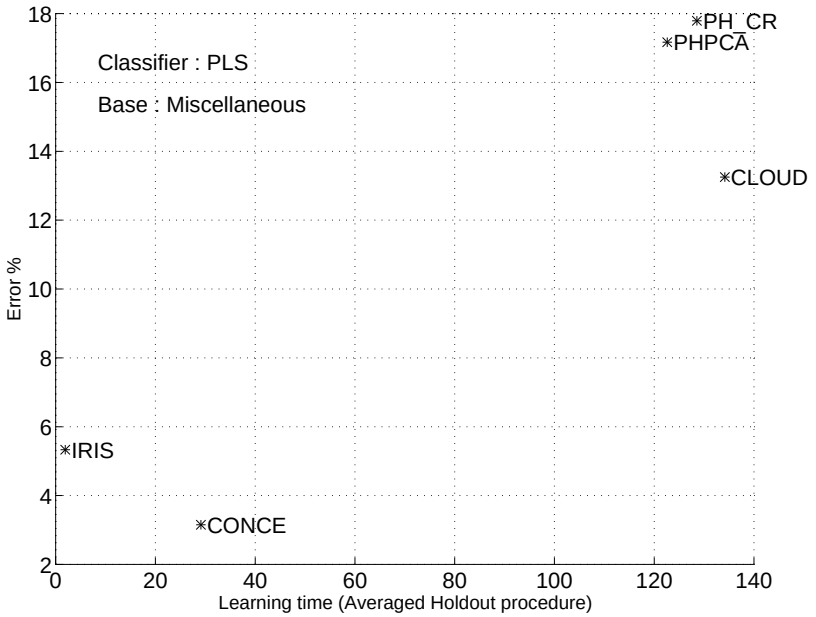

Figure 5.55: Learning time for different databases

PLS models for solving classification tasks, the memory usage implied by the network generated by this kind of classifiers may be considered as an estimate of the order of complexity associated with discriminant function to be obtained (and therefore also an estimate of the order of complexity associated with the problem to be solved). Taking into account that we have fixed the number of iterations for the Perceptron and Pocket learning algorithms which are in charge of training the individual units generated by the *Neural Trees* algorithm, the total learning time for a given problem will depend on the number of units generated during the network construction process. Of course, for the same number of generated units the learning time will be larger for those problems with larger input dimension (since the number of involved operations - multiplication, addition - will be larger, too). Therefore, as the total number of units to be generated is given by the complexity of the discriminant function to be generated, this is the parameter which will finally establish the required learning time for a given problem. In other words, for this kind of incremental classifiers there exists a strong correlation between the amount of memory resources required by the classifier in order to solve a specific task and the time required to learn this task.

Figures 5.55 and 5.56 summarize the ideas stated previously. In figure 5.55 we represent the classification error versus the learning time for different artificial as well as real databases. As indicated in previous paragraphs, those problems which require larger network structures (refer to figures 5.53 and 5.54) force also longer training processes (the results of the learning time for the *clouds* database may seem contradictory, but in fact it is due to the fact that the memory usage for the *phoneme* databases correspond to almost the same number of units generated as for the *clouds* database, taking into account the different input dimension).

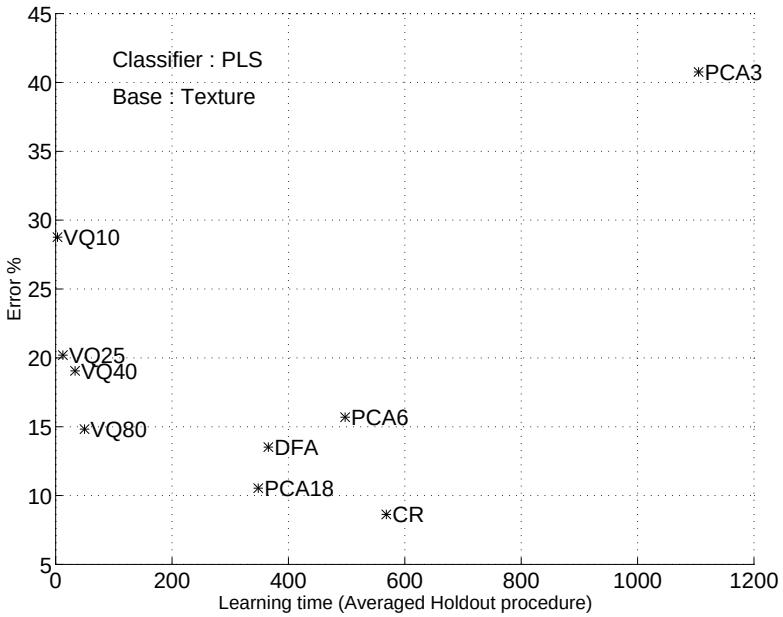

Figure 5.56: Learning time for the texture database with different preprocessing techniques

On the other side, figure 5.56 represents the classification error versus the learning time for the different databases which are obtained by applying several pre-processing techniques to the *texture* database. In this figure we can observe that those preprocessing techniques which increases the complexity of the problem to be handled (as is the case for the PCA3 database) require also much larger learning times than those (like the results for the VQ databases show) implying a reduction in the complexity of the discriminant function to be approximated.

The interest of performing a quantization on the learning database for this kind of incremental classifiers is made evident if we compare figures 5.56 and 5.50. In fact, we can deduce from those two figures that the learning on the entire database (bar labeled CR) and with a quantized version (80 prototypes per class) of this set (bar labeled VQ80) provide network structures which yield almost the same classification error. However, the time required to construct the last structure is at least one order of magnitude less than that required to build the network when the original training set is considered.

Finally, bearing in mind that PLS incremental algorithms generate network structures which resemble a Multilayer Perceptron (MLP) network (or, as the *Neural Trees* algorithm, networks which can be easily converted into a MLP-like structure), we can conclude that the time required by the generated network to process an input vector will be related also to the complexity of its structure, and therefore, to the complexity of the classification problem to be solved (of course, for the same degree of complexity, there will be differences depending on the dimension of the input vectors to be processed). However, due to the very simple operations to be performed by the individual

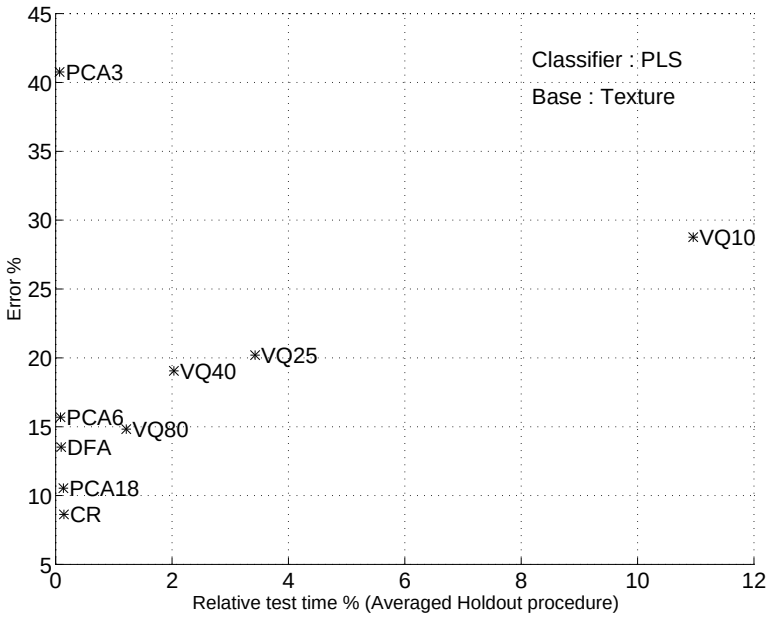

Figure 5.57: Relative test times for the texture database with different pre-processing techniques

units in order to process an input vector (basically, a weighted sum of the vector's component), the time required to classify an input vector represents only a small fraction of the time required to build the network, especially if low-complexity classification tasks are considered.

These ideas are illustrated in figure 5.57, which represents the classification error versus the relative test time (i.e., the ratio between the time required to process the test set and the total time given as the sum of the time required to build the network and the time required to process the test set). Only the test time for the VQ databases represents a significant percentage over the total time, since in this case very fast learning processes can be attained by the incremental algorithm.

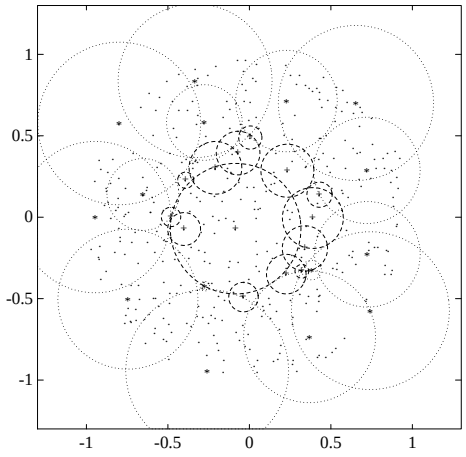

Figure 5.58: Distribution of the RCE neurons for a subset of the "concentric" dataabse

5.7 RCE classifier

5.7.1 Brief description

The Restricted Coulomb Energy (RCE) is one of the first incremental models of neural networks. In the RCE, decision units are characterised by their influence region, defined by an hypersphere around the unit, whose radius is equal to the threshold of the unit. The input space is then divided into zones, each one being represented by different decision units. New units are created with an initial chosen radius if a presented template does not fall into one of the influence regions of the units associated with the correct class; on the other hand, radii associated to units belonging to a wrong class but whose influence regions include the presented pattern are lowered to avoid misclassification. The purpose of the RCE model is to classify n-dimensional input data in an a-priori unknown number of classes. This task is realized by a coverage of the input space using n-dimensional hyperspheres. Figure 5.58 shows the configuration of the RCE neurons (together with their associated hyperspheres) after a learning of a subset of the "concentric" database. The most important aspect of this model resides in this structure adaptation. This principle of incrementality is today much frequently used, and provides a sensible improvement of the model performances.
This network, as described in figure 5.59 is composed of two layers of units (one intermediate and one output layer), connected in a feedforward way, whose number is adapted through the time. This number in each layer evolves depending on the classification quality estimated by a simple comparison between the response of the system and the expected one (supervised model). The connections between intermediate units et output units are adapted, so as the radius of the hypersphere associated to intermediate units.
The connections values between the input layer and the intermediate layer are fixed for each unit created in the intermediate layer. They are never modified after their creation. A connection between the intermediate layer unit and output layer unit is

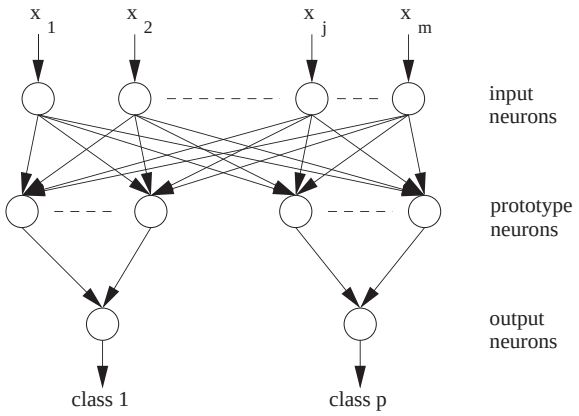

Figure 5.59: The multilayer structure of the RCE network

fixed when a new intermediate layer unit is created. Its value is always equal to one. Each intermediate unit receives the values incoming to the system, its connections and one real value called the "radius". Each output unit only receives the values coming from the intermediate units linked with it.

For more details about the RCE classifier see the R2-B3 report..

5.7.2 Parameters configuration

The two most important parameters of the RCE are the initial radius value and the order of presentation of the examples. In the benchmark studies, the order of presentation of the samples is random and the same for all benchmarks on the different classifiers. The initial radius has been chosen as 10 times the total inertia of the database.

5.7.3 Benchmarking results

The detailed study of the RCE performances is not relevant in the benchmarking context. This is essentially because the RCE classsifier has very poor performances in the case of overlapping distributions. This limitation comes from the learning rule itself, which generates a very large number of units in that case regions. Thus, the Bayes limit cannot be reached because, in that case, the learning process does not make any generalisation. We have chosen to use the RCE as a basis for comparison between classifiers, giving a kind of lower bond in generalisation. It will be also useful for the hardware constraints considerations, because the creation of large number of units is a drastic constraints for memory requirements.

Chapter 6

Comparison of classifiers

This chapter is dedicated to the comparative studies of the classifiers performances for various databases and various preprocessing methods (PCA and DFA for the dimension reduction and the vector quantization for the learnset size reduction).

Most of the figures in this chapter are built according to the same principle. For each database, all the results have been obtained with the best configurations of the seven studied classifiers for each of the studied databases.
For the real databases, the different databases coming from a dimension reduction preprocessing method are referred with the usual notations (CR*nn*, PCA*nn* or DFA*nn*, *nn* indicating the database dimension).
For figures presenting various classifiers error curves (see figure 6.6 for example), the points corresponding to a particular preprocessing on the classifier curve indicate the mean error obtained by an averaged Holdout test with 5 trials [1] (except for the VQ preprocessing for which the a particular test method, described in the last section, has been used).
For other figures presenting the performances of various classifiers for a given database (each particular classifier being indicated on the abscissa: see figure 6.2 for example) each star indicates the mean Holdout error; the maximum and minimum errors are indicated with circles; and the two error bars correspond to the confidence intervals estimated on the these maximum and minimum errors.
Finally figures presenting the performances of different classifiers for different preprocessing (see figure 6.8 for example), are build according to the same principle; each preprocessing being indicated on the abscissa and for a particular preprocessing the order of the various classifiers is indicated by a number in the legend.

In order to have a suitable comparison between the LVQ and IRVQ classifiers (which are based together on a vector quantization principle), they results are provided for identical codebook sizes for all of the problems when nothing else is indicated.

6.1 Artificial databases

6.1.1 Clouds databases

[1] The vectors used in each partition of the original database in a learnset and a testset being always the same for each particular holdout trial from one classifier to another

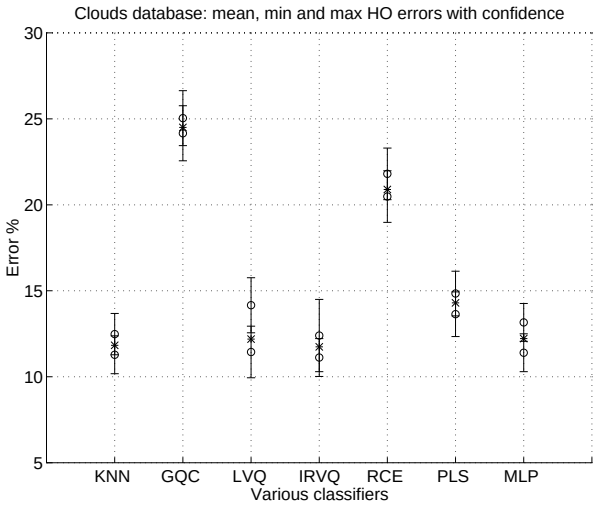

Figure 6.1: Mean, min, max HO errors of the different classifiers on the "Clouds" database

For this artificial database, the problem is to distinguish between an unimodal normal distribution (class 1) and a multimodal normal distribution (class 0) having a relatively high overlapping rate. The theoretical error for this database is 9.66% and the error of the KNN classifier (LOO test) is 10.94%, the difference between these two values being mainly due to the finite number of samples available in the database (5000 patterns). Figure 6.1 shows the HO results of the various classifiers for this database. As we can see on this figure, most of the classifiers behave well for this problem and seem to be sufficiently robust (the difference between all min and max errors being not too important). According to the confidence intervals, the KNN, LVQ, IRVQ, PLS and MLP results are all in the same range and the differences between these classifiers will be on the hardware point of view (mainly the memory requirements).

The results of the GQC and the RCE classifiers are not satisfactory and these classifiers are not acceptable for this kind of problem. For the GQC, this is explained by the fact that class 0 has three very distinct modes and that the GQC has to approximate this distribution with an unimodal Gaussian function. This is an example of the very bad behavior of the GQC algorithm in case of multimodal distributions and this problem will appear for all the other databases having these characteristics. For the RCE, the problem comes from the high overlapping rate between the classes. Since this algorithm learns (and create neurons) until it reaches 100% of recognition on the learnset, the memory requirements of the RCE will always be important in case of overlapping and the associated errors will always be large.

6.1.2 Concentric databases

The "Concentric" problem is totally artificial, the distributions are perfectly circular and uniform and the transition between the classes is abrupt. The results of the various classifiers The error of the GQC classifier is very important. This is due to the fact that it tries to estimate the uniform distributions with gaussian functions; the border is thus far from the real one. There is no overlapping between the classes and the RCE

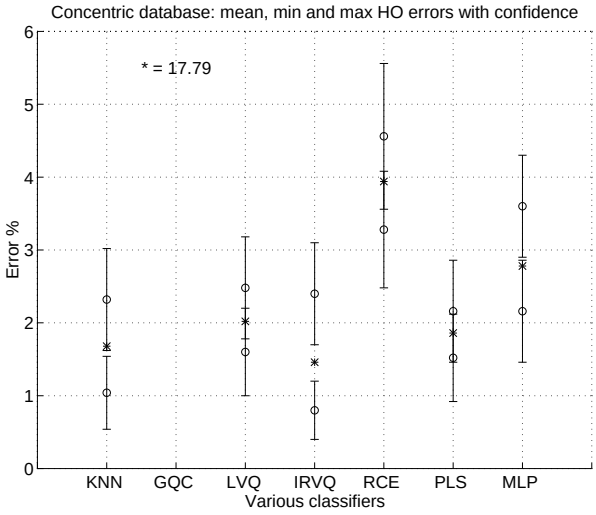

Figure 6.2: Mean, min, max HO errors of the different classifiers on the "Concentric" database

results are thus better than in the case of the "Clouds" database. The fact that the classes are not separated by a free space area and that the probability density inside each class is uniform will influence the standard deviation of the errors of an averaged Holdout test. This mainly explains the large differences between the min and max results. The LVQ and PLS classifiers seems to be more robust than the others for this problem. For the PLS algorithm, the good results are due to the linear separability of the classes. Since the LVQ1 algorithm tends to move the prototypes away from the border, this classifier is more robust than the other.

For the IRVQ classifier, the bad robustness of the classifier could also be explained by the abrupt border, which is not suitable for this algorithm.

6.1.3 Gaussian databases

This database is well suited for the study of the classifier behavior in function of the database dimension. A detailed study has been provided for each classifier in particular in chapter 5 and this section is only dedicated to the comparisons. Figure 6.3 shows the theoretical error and the mean Holdout errors of the various classifiers in function of the database dimension.

Since the overlapping of the two classes is very important, the performances of the RCE classifier on the various databases are very bad. In the "Gaussian" databases each class has a single normal distribution. This explains that the GQC classifier has the best performances on this problem, its error curve being very near of the theoretical one. If we remove the particular case of the GQC, the IRVQ classifier has the best behavior for this problem. It could be due to the fact that the IRVQ classifier uses radial gaussian kernel functions to evaluate the gaussian probability densities inside each class. In spite of the fact that the codebook size of the LVQ classifier has been set to a larger value than the one of the IRVQ (25 codewords per class), the results of this classifier are not as soo good as usual and the difference between the theoretical curve

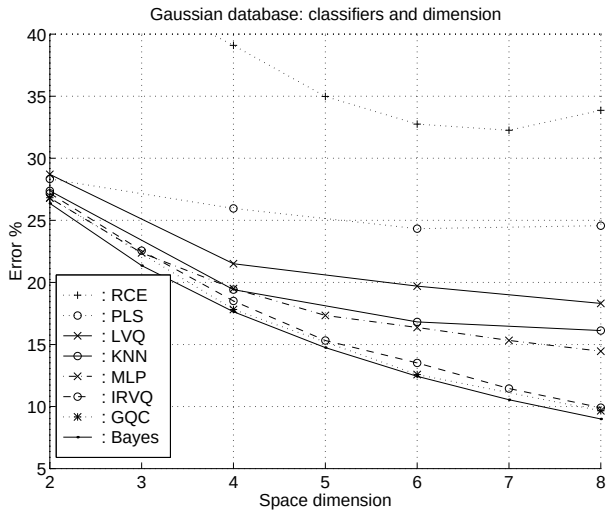

Figure 6.3: Mean HO errors of the different classifiers in function of the dataspace dimension for the various "Gaussian" databases

and the LVQ one increases with the dimension. This is due to the fact that one class is completely distributed inside the other in this case. Since there is no two well separated distribution, the LVQ1 learning principles are not very suited in this case. The MLP performances are better, but they also move away from the theoretical curve as the dimension increases. The number of learning epochs as been set to a constant value for each dimension. The learning task become more and more difficult as the dimension increases and an increase in the number of learning epochs with the dimension could provide better MLP classifiers.

The performances of the KNN classifier follows the one of the MLP. The curves of the LVQ, MLP and KNN could be explained by the empty space phenomenon effect: as shown on figure 4.1, the database size doesn't increase with the dimension and the number of points available for the densities estimation become thus "small" as the dimension increase. The loose of performances is thus not only due to a worst behavior of the classifiers as the dimension increase, but also to the empty space phenomenon. We can also say that the differences between the various classifiers depend on their robustness to the empty space phenomenon problem; for example, the PLS and the LVQ have nearly the same performances in dimension 2 while the difference between them is very important in dimension 8.

6.2 Real databases

6.2.1 Iris databases

This small database with three classes in dimension four is one of the most widely used classification problem in the neural network community. One class is linearly separable from the two others and the latter are not linearly separable from each other. As explained in chapter 4, the most important problem with this database is the number of available patterns. As there is only 50 patterns per class, a Holdout test with a learnset

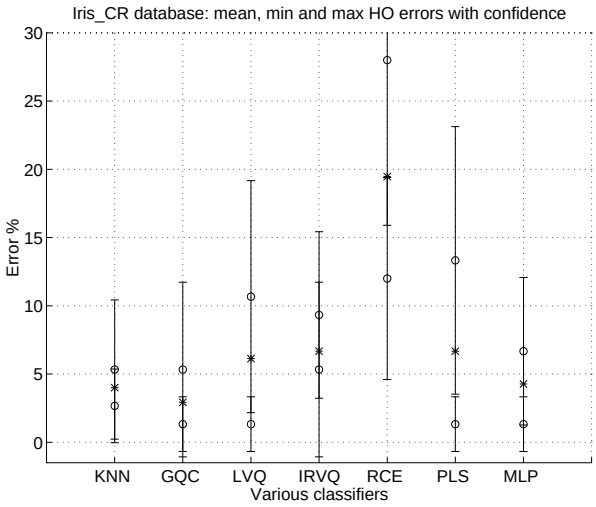

Figure 6.4: Mean, min, max HO errors of the different classifiers on the "Iris_CR" database

containing the half of the database size will impose to build the different classifiers with only 25 patterns per class, which is much too small to reliably estimate the probability densities in dimension 4. The confidence intervals on this "small" database are very large and the classifier results are very influenced by the draw of the patterns to be used for the learnset.

This is illustrated on figure 6.4, which gives the Holdout performances of the different classifiers on this problem. The estimate of the Bayes error on this database by a LOO test of the KNN classifier is $3.33 \pm 4.0\%$. Due to the important confidence intervals on the errors, it is quite difficult to have a reliable ranking of the classifiers performances and this problem should not be utilized any more for classifiers performances comparisons. Nevertheless, some remarks may be made on this figure. The robustness of the KNN, GQC, IRVQ and MLP classifiers is much larger than the LVQ, PLS and RCE one. The very good results of the GQC let us think that the different classes have unimodal gaussian-like distributions; but the overlapping between two classes drives the RCE to poor results.

6.2.2 Phoneme databases

The Holdout classification results on the "Phoneme_CR" database are given on figure 6.5. The estimate of the Bayes error on this database with a LOO test of the KNN classifier is $8.97 \pm 1.1\%$. According to chapter 4, the relative number of points in each class of "Phoneme" is sufficient and there is between-class overlapping.

The LVQ and IRVQ classifiers used for the comparison were build with a codebook of 150 prototypes (this corresponds to 5.5% of the learnset size for a mean error between 16.5 and 17%) and their memory requirements (about 800 memory words) are thus about 15 times smaller than the one of the KNN classifier which has to memorize the entire learnset and for which the mean HO error is 12.29%.

Figures 5.26 and 5.34 respectively show the evolution of the HO error in function of the codebook size for the LVQ and IRVQ classifiers. From these figures, we see that

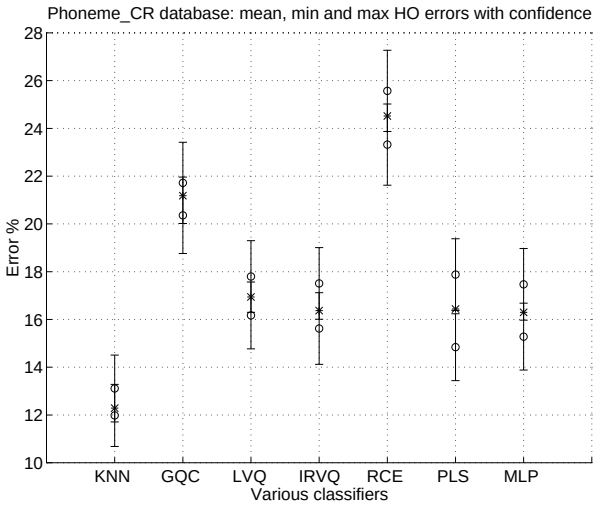

Figure 6.5: Mean, min, max HO errors of the different classifiers on the "Phoneme_CR" database

the chosen codebook size of 150 is a good compromise and that it is also possible to reach a mean error of about 15% for a codebook size of 400 prototypes.

The optimal value for the KNN classifier is 1. It seems thus that the most local is the estimate for this database, the better is the classifier. The bad result of the GQC also tends to justify this view: chapter 4 showed that the classes of this database seem to be unimodal and the large error of the GQC classifier is thus due more to its global estimate of the non-gaussian distributions than to its unimodal character. Once again, the RCE algorithm is handicapped by the overlapping.

The PLS and MLP classifiers, with their respective 1333 and 310 memory words requirements are also good candidate for the "Phoneme" problem, even if the PLS seems to be a bit less robust than the others

6.2.3 Satimage databases

For the "Satimage" database, figure 6.6 summarizes the mean errors obtained by the seven classifiers for various PCA preprocessing. From this figure, two groups of classifiers can be identified, the first group with an error between 14% and 25% and the second group with a error between 8% and 14%. The results for each group are detailed at figure 6.7 and 6.8.

In the first group, we have the PLS, RCE and GQC classifiers. PLS and RCE classifiers have similar poor results, and the PLS classifier seems to be less robust than the RCE and GQC: it always has the largest difference between its minimum and maximum errors. This is due to the fact that this database has very overlapped classes and complex boundaries between them. If the GQC classifier doesn't give interesting results that means that the underlying class distributions are not Gaussian. The existence of several modes in the distributions is confirmed by the observation of the 3 first principal components of this database.

Now, if we consider the second group, we notice that the IRVQ and LVQ classifiers have similar performances, the MLP classifier, a little bit less efficient and the KNN

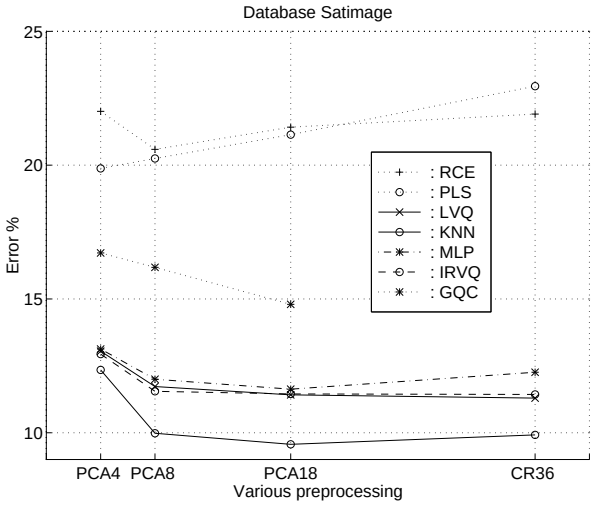

Figure 6.6: Performances of the seven studied classifiers on the "Satimage" database versus the number of dimension obtained after a PCA.

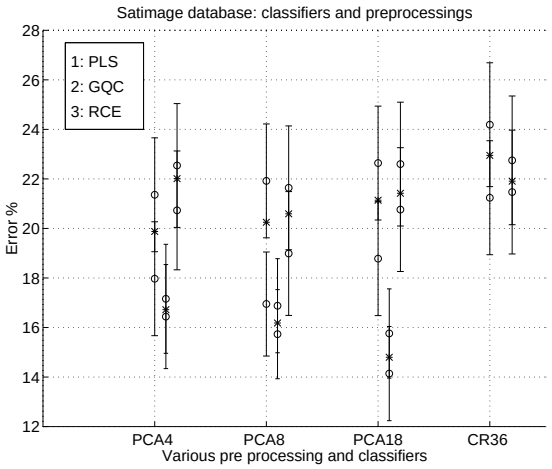

Figure 6.7: Performances and robustness of the group 1 (less efficient classifiers for the "Satimage" database) versus the number of dimension obtained after a PCA.

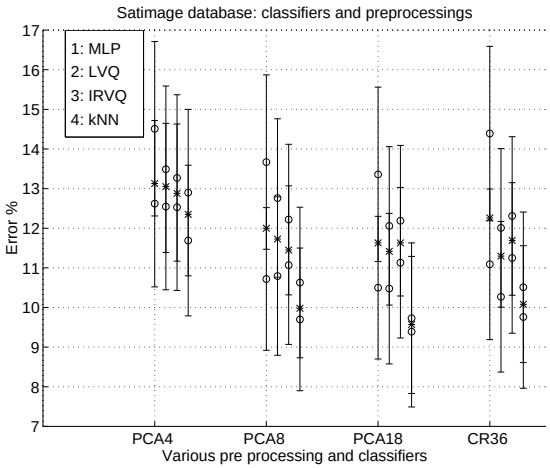

Figure 6.8: Performances and robustness of the group 2 (most efficient classifiers for the "Satimage" database) versus the number of dimension obtained after a PCA.

classifier the best in that case. These classifiers have similar behavior, while their common aim is to estimate the *pdf* before the decision. The KNN method gives good results, but is less adapted to hardware and memory constraints as explained below (section 6.3).

In these figure, the preprocessing DFA5 isn't indicated. In fact, we have remarked in the different studies in chapter 5 that this preprocessing is here quite equivalent (from the point of view of the classification) to a PCA with 5 dimensions (see for example figure 5.6 left).

All the classifiers almost follow the same trends (except PLS and RCE classifiers):

- from 4 dimensions up to 18 dimensions, the error rates decrease,
- from 18 dimensions up to 36 dimensions, the error rates slightly increase.

According to the study of the fractal dimension of this database, it seems that the local dimensions are around 10 for each class. So in this range of dimensions, if we add a new feature, the discrimination power of the overall significantly increases, even if we have no sufficient samples for a good estimate of the *pdf*. After, from 18 dimensions, the features which are added, don't provide new discrimination aspects. So it remains the bias in the estimation of the *pdf* which is here visible by an increase of the error rates.

6.2.4 Texture databases

Figure 6.9 summarizes the results obtained by the seven classifiers in their best configuration. The results obtained with a database in 3 dimensions are shown in this comparison because of the very bad performances on this configuration. The preprocessing DFA will be analyzed after (figure 6.11).

As for the "Satimage" database, the PLS and RCE classifiers have poor results compared to the other classifiers (figure 6.9).

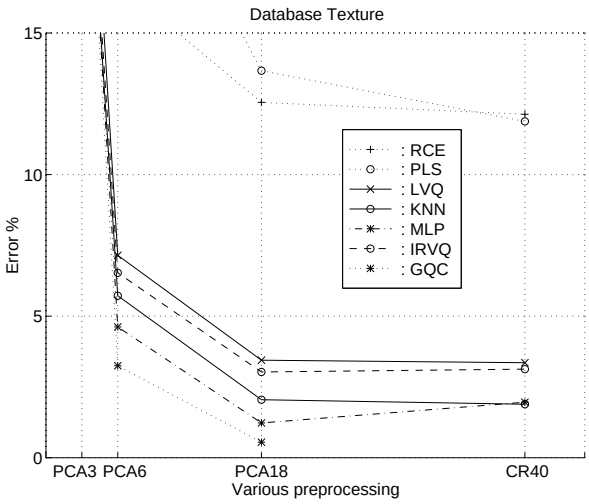

Figure 6.9: Performances of the seven studied classifiers on the "Texture" database versus the number of dimension obtained after a PCA.

With the other classifiers (MLP, LVQ, IRVQ, KNN and GQC), the error rates are less than 8% (mean value). Here, the best classifier for this database is the GQC classifier. So, the distribution is more or less Gaussian. But, if we consider the results obtained by the IRVQ classifier, we remark that its behavior isn't so close to the behavior of the GQC classifier (figure 6.10). Let us recall that the IRVQ uses isotropic Gaussian kernels to estimate the *pdf*. Then it seems that the "Texture" database is characterized by elongate clusters, more or less overlapped between them.

For all the classifiers with this database, the DFA preprocessing seems to be a very well adapted preprocessing. The error rates are significantly smaller than for each other preprocessing (even smaller than for the "CR" database). This could be due to the fact that each class seems to have an unimodal distribution, which is suitable for the DFA.

Like for the "Satimage" database, we observe as the dimensions growing, first an important decrease of the errors and then a slight increase. The interpretation of this phenomenon have been noticed previously.

6.3 Performances and hardware constraints

Let us consider here the 7 previous classifiers from the point of view of hardware constraints (memory, computation time).
For the "KNN", "LVQ", "IRVQ", "MLP" and "PLS" classifiers, the memory usage increases linearly with the number of features (figure 6.12). The learning database must be stored in memory (complete database for "KNN", database after vector quantization for "LVQ" and "IRVQ"). For the "MLP" and "PLS", the memory usage increases roughly linearly with the dimensions. For the "MLP" classifier, it is clear because the same network has been used for these benchmarks (20 units for the first hidden layer and 104 for the second hidden layer). For the "PLS" algorithm, the memory requirement is linked to the number of created units during the learning process, and the number

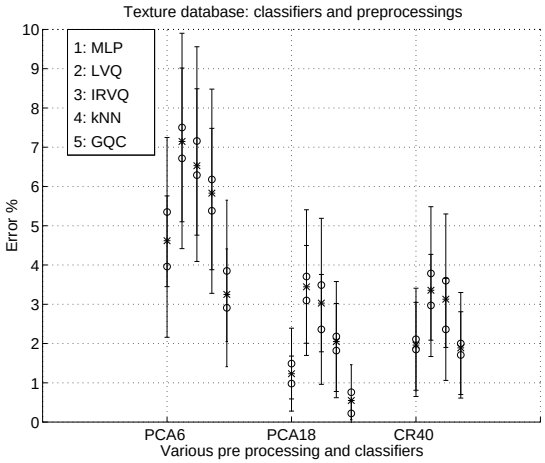

Figure 6.10: Performances and robustness of the most efficient classifiers for the "Texture" database versus the number of dimension obtained after a PCA.

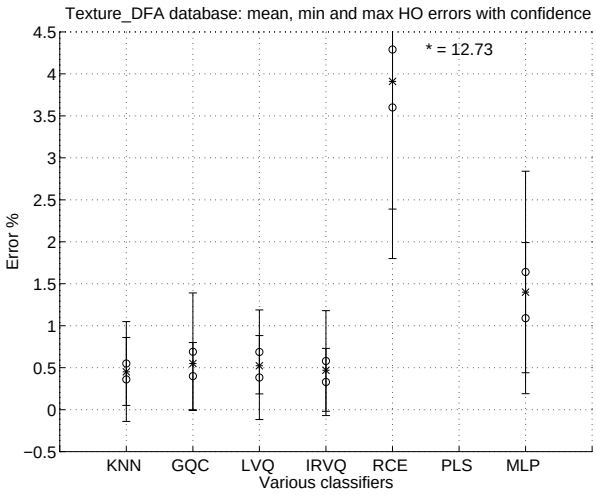

Figure 6.11: Performances and robustness of the seven classifiers for the "Texture" database after a DFA (10 dimensions).

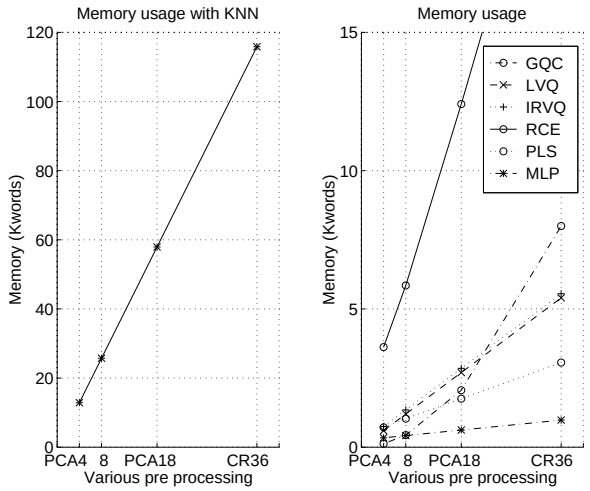

Figure 6.12: Memory usage for the various classifiers on the "Satimage" database versus the number of dimensions

of memory words has been evaluated by the $N \times (d + 3)$, where N is the number of created units and d the input dimension. So globally, for the "PLS" classifier, the memory usage grows quite linearly with the dimension, even if the number of created units increases when the dimension decreases. With a few dimensions, the classes in the database are strongly overlapped, and more units are created. Here, the memory requirement is less with a "MLP" classifier than with a "PLS" classifier and a lower error has been obtained with a "MLP" classifier. For the "GQC" classifier, the memory usage increases linearly with the number of classes and quadratically with the number of dimensions, as the memory must store for each class the center of gravity and the covariance matrix. So this classifier has little memory need. The memory need for the "RCE" classifier depends on the number of units created after the learning. Here for this base, for different number of features (4, 8, 18 and 36), respectively 670, 653, 650 and 724 units were created. So from these results, the memory needs for "RCE" is between the "KNN" classifier and the others. For "LVQ" and "IRVQ" classifiers, figure 6.12 shows the memory needs for a vector quantization with 150 prototypes. Even with this little number of prototypes, the error rate is good, as it is the nearest error from the error with the "KNN" classifier (figure 6.6).
For example, with the "IRVQ" classifier, if we consider a configuration with 36 and 18 features, the error rate grows from 7.97 to 9.32 (increase of 17%) and the memory usage decreases to 50%. The same remark can be done for the other classifiers.
For the hardware constraints, we have chosen to compare only the test time. For the 7 classifiers, the test time is roughly linear with the number of features (figure 6.13). The "MLP" and "PLS" classifiers have the smallest test times. The complexity of the networks is small. However, it is well-known that the learning time is very important for these kinds of algorithm. The "GQC" classifier has also small test times (c distance computations). The difference between the test time for "LVQ" and "IRVQ" noticed at the figure 6.13 mainly depends here on the software implementation, as these two classifiers have quite the same complexity. The test time for the "KNN" classifier is

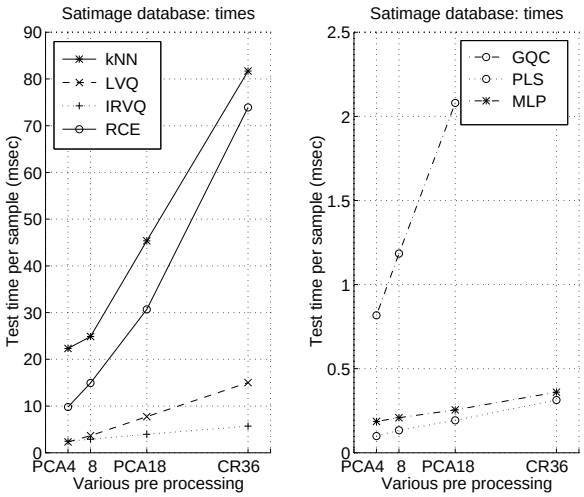

Figure 6.13: Mean test time per sample for the various classifiers on the "Satimage" database versus the number of dimensions (different preprocessings)

the greatest one : all distances between all samples must be computed and a "quick" sort is applied to extract the kth nearest neighbor from the test sample.
The test times estimated at the figure 6.13 are measured on a SUN Sparc20 workstation.

6.4 Vector quantization as preprocessing

The the use of a vector quantization technique as preprocessing has been presented in chapter 2. In this section, we will successively illustrate the interest of the use of such a method, present the general test method chosen for this preprocessing in the framework of ELENA (together with the overfitting problems of this method which appear after the gathering of the test results) and finally provide and analyze some test results.

For a large number of learning algorithms, the size of the learning or design set strongly influence the learning time and the memory requirements of the resulting classifier. Let us examine two particular examples to illustrate this:

- For the backpropagation algorithm used for the MLP learning, the convergence to a useful network necessitate a very large number of learning epochs and this learning could become prohibitive in time for large learning sets. In this case, the solution is either to reduce the number of learning epochs (at the expense of a non-optimal solution), either to reduce the size of the learning set (at the expense this time of a worst probability density estimate).

- The incremental learning algorithm used to build the RCE classifier learns and continue to create new neurons until it is able to correctly classify the entire design set. In case of a high overlapping between the classes, the resulting network will contain a number of neurons strongly linked to the learnset size. The solution to reduce the resulting network size in this case would be either to reduce the design set size by selecting useful patterns in the original learnset, either to modify the

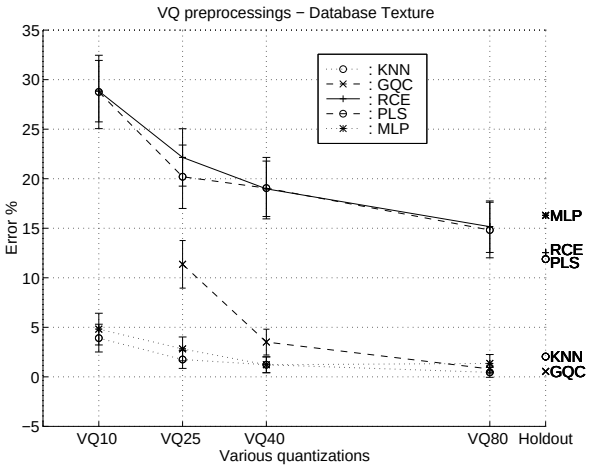

Figure 6.14: Errors on the Texture_VQ databases

RCE learning algorithm in order to stop the learning after the construction of a given network size (again at the expense of a non optimal solution from the RCE learning point of view).

From these two example, it is clear that a reduction of the learnset size could sometime be necessary. A good solution to this problem is to use a vector quantization method to build a reduced design set. This method seems to be useful since the original probability density function inside each class will be roughly kept for the VQ resulting codebook, which would thus be better than a simple random draw of some patterns in the learning set.

As explained in chapter 3, this method has been tested for three particular real databases: "Phoneme_CR", "Satimage_PCA18", and "Texture_PCA18". For each of these databases, four different codebook sizes were chosen (corresponding to a mean value of 10, 25, 40 and 80 codewords per class) in order to obtain the "VQ10", "VQ25", "VQ40" and "VQ80" databases. Of course, the number of codewords inside each class was chosen to respect the a priori probabilities of the classes. The total codebook size for each VQ database is the product of the number of classes by the mean codebook size per class given above.

The selected vector quantization method was the classical LBG or k-means algorithm [16] and the test method was the following: the reduced VQ design sets were build on the entire original database and than used for the learning of the various classifiers. Since the VQ preprocessing corresponds to an important data reduction leading to codewords which are not present in the original database, it has been decided to evaluate the classifiers performances on the original database itself. This method is thus an indirect resubstitution test (the testset being the same as the set used to build the learning set with the VQ). Due to this fact, the risk to obtain a too optimistic error estimate strongly depend on the relative size of the codebook versus the size of the original database. The IRVQ and LVQ algorithms already use a vector quantization method during the learning and these classifiers were thus not used on the "VQ" databases for comparison purpose. Nevertheless if we would like to compare their performances with the others, it would be necessary to use their resubstitution results

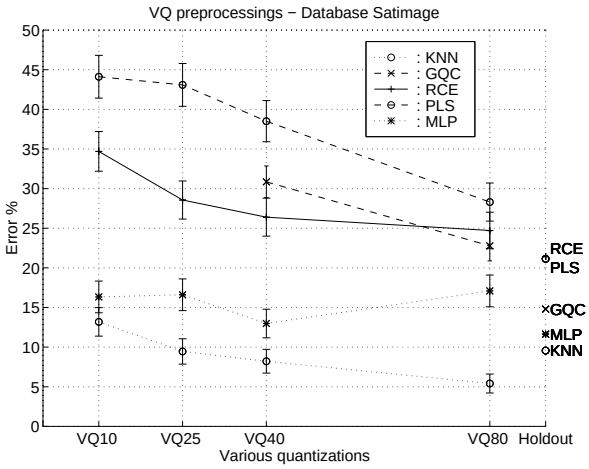

Figure 6.15: Errors on the Satimage_VQ databases

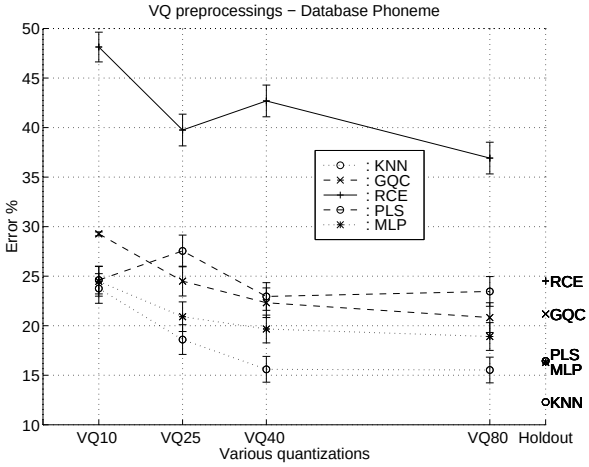

Figure 6.16: Errors on the Phoneme_VQ databases

(the learning corresponding to a VQ of the entire original database and the test being proceed on the same set).

The overfitting problem associated to the test method chosen for the VQ preprocessing was not though to be critical when this test method was decided by the ELENA partners; but when all the results were finally available, it was clear that this method was not the best one, especially for large codebook sizes. Since these results also correspond to an important part of task B4, we have decided to analyse the only results corresponding to acceptable codebook size and provide the other results "as it".

To have an idea of the resubstitution overfitting problem for the three selected databases, we can examine in detail figures comparing the Holdout error with the Resubstitution one in function of the codebook size of classifiers using a VQ learning method. Such figures are available for the IRVQ and the LVQ algorithms: see figure 5.27 for the LVQ results on the "Satimage_PCA18" database and figures 5.35 and 5.34 for the IRVQ results respectively on the "Satimage_PCA18", "Texture_PCA18" and "Phoneme_CR". On these figures, the large difference between the Holdout and

CB	*Phoneme*	*Satimage*	*Texture*
10	0.37%	0.93%	2%
25	0.93%	2.33%	5%
40	1.48%	3.73%	8%
80	2.96%	7.46%	16%

Table 6.1: Ratios (in percents) between the various codebook size chosen for the VQ preprocessing (corresponding respectively to a mean value of $CB = 10, 25, 40, 80$ codewords per class) and the initial database sizes for the "Phoneme", "Satimage" and "Texture" databases.

Resubstitution curves is mainly due to

- the size of the learning set which is the entire database size for the Resubstitution method and the half for the Holdout test (the performances of classifiers build on the half of the database for the Holdout tests will thus be worst), and
- the overfitting problem (a classifier build on a given design set will always be too optimistic if the same set is used to test its performances).

We can reasonably suppose that the main part of the difference between the Holdout and Resubstitution curves is due to the size of the learning set when the codebook sizes are small (when the decrease of the two curves is approximatively the same) and due to overfitting for larger codebook sizes (when the Holdout curve become horizontal while the Resubstitution one continue to decrease). If we desire to avoid overfitting problems in the test of our VQ preprocessed databases, we have thus to select "small" codebook sizes, that is codebook sizes corresponding to the part of the curves where the difference is mainly due to the learning set size. According to figures 5.27, 5.35 and 5.34, we can say that the codebook size will be acceptable if it represents less than 3% of the database size. This corresponds respectively to a mean value of 80 codewords per class for the "Phoneme_CR" database, 30 codewords per class for "Satimage_PCA18" (a total codebook size of 200 codewords on figure 5.27) and 15 codewords per class for "Texture_PCA18". On the other hand, table 6.1 gives the ratios (in percents) between the various codebook size chosen for the VQ preprocessing (corresponding respectively to a mean value of $CB = 10, 25, 40, 80$ codewords per class) and the initial database sizes. From this table and according to principles explained here above, it is clear that "Phoneme_CR" is the only database for which the overfitting problem will not be present for all the VQ preprocessing.

Figures 6.14, 6.15 and 6.16 present the computed performances of the various classifiers in function of the mean codebook size per class respectively for the "Texture_PCA18", "Satimage_PCA18", and "Phoneme_CR" databases. The averaged Holdout error has also been indicated. When this error is much smaller than the different VQ errors, we can reasonably say that there is no overfitting problem. This is not the case for the "Texture" database, but the results seems to be useful for "Phoneme" and "Satimage". For most of these simulation curves the VQ errors have approximately the same behavior: a rapid decrease for small codebook sizes followed by a very small decrease when the codebook size become large enough. We can thus say that, for most of the classifiers, there is a codebook size from which the error rate reaches its asymptotic value. If we consider that the asymptotic VQ error rate has been reached for VQ80 in case of the "Phoneme" database,we can now compare the "VQ80" and "HO"

Classifier	Error (%)		Memory (words)		Rec. time (msec)		Learn time (sec)	
	VQ80	HO	VQ80	HO	VQ80	HO	VQ80	HO
KNN	15.5	12.3	800	13510	1.26	9.63	0.01	0.704
GQC	20.8	21.2	60	60	0.42	0.43	0.04	0.8
RCE	36.9	24.5	104	672	1.57	10.36	8.2	202
PLS	23.46	16.44	154	1332	0.02	0.06	9.82	198
MLP	18.91	16.3	310	310	0.17	0.26	128	354

Table 6.2: Comparisons between results obtained for the "Phoneme_CR" database and for the "Phoneme_VQ80" database. The recognition time is given in milliseconds per sample while the learning time (in seconds) is for the entire database

results on "Phoneme" from the hardware point of view. Table 6.2 provides for each case the computed error, the classifier memory use after learning, the recognition time per sample and the learning time for the entire design set (which contains 160 patterns for the "VQ80" preprocessing and 2702 for the Holdout test).

- For the KNN classifier which has to memorize the entire design set, the hardware constraints reduction are very important compared to the small difference between the errors; a VQ preprocessing should thus be useful for the hardware implementation of the KNN classifier.

- For the GQC classifier, the only interest of the VQ preprocessing concerns the learning time (the memory requirements and recognition time only depend on the data space dimension). Nevertheless, the reported error rates are equivalent and the VQ preprocessing could thus be interessant when the covariance matrix has to be estimated for very large learnsets.

- In spite of the important hardware constraints reduction for the VQ preprocessing for the RCE classifier, the error reported for VQ80 is much too large. It is the same for "Satimage" and "Texture". Since the usual error rates of the RCE classifier are already too important, the VQ preprocessing seems to be without interest for the RCE algorithm.
  The problem is nearly the same for the PLS algorithm, but the VQ80 error is still acceptable and a more detailed study could be useful in this case

- For the MLP classifier, the VQ preprocessing is not very useful for the memory use and recognition times since the network architecture doesn't depend on the database size. The main interest concerns the learning time. 1000 epochs were used for the learning of the 160 patterns of the VQ80 learning set and only 200 epochs for the HO test. Since the errors are equivalent, it seems to be quite interesting to the MLP to increase the number of learning epochs while decreasing the learnset size with a VQ preprocessing.

The use of a vector quantization preprocessing method seems thus to be reliable from the hardware and learning time points of view for the MLP and KNN classifiers. Since the method implemented for the tests presented here suffers of overfitting problems, it could be useful to study the interest of the VQ preprocessing in more details with another test method for which a total independence between the learnset and the testset could be ensured.

Chapter 7

Conclusion

7.1 Synthesis

In this work, we presented benchmarks obtained on seven classifiers studied in the framework of the ELENA project. This report represents a very important effort for a heavy time-consuming task, both on a scientific and communication point of views, since this work has been done by all partners in a very strong collaboration. From this work, guidelines and approaches for choosing preprocessing and classifiers have been described through this whole report. Thus, we will only discuss here the main aspects of this study:

1. the characterization of the databases (size, local dimension, Quantizationdispersion, overlapping),
2. the usual preprocessing (linear factorial analysis) for the reduction of the dimension or the reduction of the number of samples,
3. the performance and the robustness of the classifier in order to make comparisons,
4. the hardware constraints linked to the implementation of the classifiers (memory, computing test time).

The first point represents the preliminary task to achieve before any other process. It is fundamental to estimate the discrimination power of the features, the local dimension of the database, the dispersion of the samples, the overlapping rate, ... in order to adjust the feature extractors, to choose a suitable preprocessing and classifier. By the knowledge of the intrinsic dimension (which has been here estimated by the fractal dimension), we can adjust the preprocessing method to provide a best output space in lower dimension. The factorial analyses (Principal Component Analysis and Discriminant Factorial Analysis) are classical linear methods for this. But in the case of complex non linear structure, as it is often the case in high dimension, these linear methods are limited and it will be more suitable to use non linear projection such as for example, a curvilinear component analysis (see report R2-A-P, for the description of the "Vector and Projection" algorithm).

Concerning the preprocessing, we obtain after a PCA, the evolution of the resulting inertia versus the number of dimensions. This information is important, especially in

high dimension. Linear correlations between features can be easily removed by this way. Moreover, a large reduction of dimensions can be directly obtained without a great loss of discrimination on the features. It is the case for the databases "Texture" and "Satimage", where the performances of classification are quite identical even with only half of the initial features. The number of features can be also reduced by DFA, but with this method, the number of dimensions is fixed by the number of classes ($c-1$). So for example, a problem in high dimensions with 3 classes, will be reduced in two dimensions, and this great reduction will cause some overlapping of the data which will increase the error in classification. In this study, we obtained two different behaviors with this preprocessing. The first one concerns the "Satimage" database (6 classes), where the performances of classification are similar in five dimensions between the output space generated by a PCA or by a DFA. On the other hand, this preprocessing gives excellent results on the "Texture" database, where each class is quite well identified and clusterised (unimodal). In this case, the between-class inertia is increased with regards to the within-class inertia.

In these benchmarks, we used the vector quantization technique to reduce the number of learning samples. The method used here is the K-means algorithm. By this reduction, we obtain a significant reduction of the complexity of the classifiers and then a reduction of the memory usage. In this frame, the question remains of the determination of the number of prototypes for the learning database. Up to now, this question remains open. In this study, we fixed the number of prototypes in order to be consistent with practical considerations on the memory resources for a chip (up to 256 prototypes for all classes). The choice of the test method is very important in order to control the overfitting. Here, we applied vector quantization on the whole database to create the learning base. The test method is a cross validation test where the original database is the test database. So, when the number of prototypes increases in the learning set, the obtained error rates converge towards the resubstitution error which is a too optimistic estimate. An other way would be to realize the vector quantization on different partitions of the whole databases in order to obtain different learnsets independent from the testsets.

Concerning the choice of error measure methods for comparisons, a Leave-One-Out procedure gives an upper bound of the error, the resubstitution method gives a lower bound and the true performance lies in principle in between. To summarize, we can say that the Leave-One-Out procedure will give an absolute performance, but from a practical point of view, it is interesting to evaluate the dispersion of the errors when we use a given configuration of the classifier, different databases. Here we evaluated this robustness by using a Holdout method on five trials and we considered the minimum and maximum error values by computing confidence intervals on these extrema. We obtained large confidence intervals and this measure hasn't be so helpful for the comparisons. A good idea of the dispersion error is also given the extremum error obtained with the Holdout procedure.

Thus a benchmark protocol has been defined in this sense, and all studies on the different classifiers were done according to this procedure. The comparison given at chapter 6 is objective when each classifier has been optimally configured. Extensive tests have been done on the configuration parameters in order to find the minimum error when varying the parameters. Chapter 5 summarizes the behavior of each classifier

according to the different databases and preprocessings.

From these benchmarks, we considered two kinds of classifiers, the classifiers which estimate the probability density functions and the other ones. The performances obtained with the first ones (KNN, GQC, IRVQ, LVQ, MLP) are almost always better than with the second group (RCE, PLS). The RCE and the PLS classifiers (for the PLS, the algorithm used is a "Neural tree" with a control of the number of created units) are two incremental networks with the same kind of stopping criterion (full recognition for a given learnset). The performances in generalization are better with the first group of classifiers; the problem of the stopping criteria remains open and some theoretical considerations are given in report R3-A-P.

The hardware constraints have been taken into account by the evaluation of the memory usage for each classifier and each database, and of the computing time (learn and test). A preliminary procedure has been done in order to "calibrate" the computing time estimated on different workstations. Then, we have obtained consistent time to make possible a comparison of the computing time obtained with different classifiers on different workstations.

7.2 Perspectives

This work proposes both a protocol for benchmarking and an approach for classification design from the characterization of databases and the preprocessing to the optimal configuration of a classifier. Further work must be done in this sense in order to have a better control of the application conditions of the classifiers and the different preprocessings (especially vector quantization). This control would based on the definition of suitable quality criteria linked to each method.

A part of this work has been at the origin of new European project proposal which is currently under evaluation by the EC. The objective of this proposal called "Neuro-Statistical Data Analysis" is to define and develop neural techniques (not only for classification) for data analysis and to integrate them in a methodological frame extended from the statistical approach. This statistical approach have been used here for these classification benchmarks; this study has to be completed and generalized to other domains (prediction, function approximation, ..).

An important issue for these studies is their generality the associated guidelines which can be directly disseminated and used in a large community in the academic and industrial world.

Bibliography

[1] T. Kohonen, *Self-Organization and Associative Memory*, Springer-Verlag, Berlin, 1989, 3rd Edition.

[2] D.L. Reilly, L.N. Cooper, and C. Elbaum, "A neural model for category learning", *Biological Cybernetics*, vol. 45, no. 1, pp. 35–41, 1982.

[3] Y. Cheneval, "Packlib, an interactive environment to develop modular software for data processing", in *IWANN95-Proceedings of the International Workshop on Artificial Neural Networks*, Prieto Mira, Cabestany, Ed., Malaga, Spain, June 1995, Springer-Verlag Lecture Notes in Computer Sciences, Submitted.

[4] Y. Cheneval et al., "Deliverable R2-B2-P - Task B2: Unified graphic environment", Tech. Rep., Elena-NervesII "Enhanced Learning for Evolutive Neural Architecture", ESPRIT-Basic Research Project Number 6891, June 1994.

[5] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, Inc., 1250 Sixth Avenue, San Diego, CA 92101, 2nd edition, 1990.

[6] P. Comon, "Supervised classification: a probabilistic approach", in *ESANN95-European Symposium on Artificial Neural Networks*, M. Verleysen, Ed., Brussels, Belgium, April 1995, D facto publications.

[7] P. Comon et al., "Deliverable R1-A-P - Axis A: Theory", Tech. Rep., Elena-NervesII "Enhanced Learning for Evolutive Neural Architecture", ESPRIT-Basic Research Project Number 6891, June 1993.

[8] Zijian Zheng, "A benchmark for classifer learning", Tech. Rep. TR474, Basser Department of Computer Science, University of Sidney, N.S.W Australia 2006, 1993, Anonymous FTP: /pub/tr on ftp.cs.su.oz.au.

[9] Lutz Prechelt, "PROBEN1 — A set of benchmarks and benchmarking rules for neural network training algorithms", Tech. Rep. 21/94, Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany, Sept. 1994, Anonymous FTP: /pub/papers/techreports/1994/1994-21.ps.Z on ftp.ira.uka.de.

[10] P. M. Murphy and D.W. Aha, "Uci repository of machine learning databases", 1991, Irvine, University of California, Department of Information and Computer Science, Anonymous FTP: /pub/machine-learning-database on ics.uci.edu.

[11] T. Kohonen, G. Barna, and R. Chrisley, "Statistical pattern recognition with neural networks: Benchmarking studies", in *IEEE Int. Conf. on Neural Networks*, San Diego, CA, 1988, vol. 1, SOS Printing.

[12] P. Alinat, "Periodic Progress Report 4", Tech. Rep., ROARS Project ESPRIT II- Number 5516, February 1993, Thomson report TS. ASM 93/S/EGS/NC/079.

[13] G. Nakhaeizadeh, "Project STATLOG ", Tech. Rep., ESPRIT IPSS-2 Number 5170 : Comparative Testing and Evaluation of Statistical and Logical Learning Algorithms for Large-Scale Applications in Classification, Prediction and Control, April 1993.

[14] D. Michie, D.J. Spiegelhalter, and C.C. Taylor, Eds., *Machine learning, Neural and Statistical Classification*, Ellis Horwood Series In Artificial Intelligence, England, 1994.

[15] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons, 1973.

[16] Y. Linde, A. Buzo, and R.M. Gray, "An algorithm for vector quantizer design", *IEEE Transactions on Communications*, vol. 28, pp. 84–95, January 1980.

[17] A. Ahalt, A. K. Krishnamurthy, Chen P., and D. E. Melton, "Competitive learning algorithms for vector quantization", *Neural Networks*, vol. 3, pp. 277–290, 1990.

[18] B.B. Mandelbrot, *Fractal geometry of nature*, Freeman, San Fransisco, 1982.

[19] C Trichot, *Courbes et dimension fractale*, Springer-Verlag, 1993.

[20] H.G.E. Hentschel and I. Procaia, "The infinite number of generalized dimensions of fractals and strange attractors", in *Physica 8D*, 1983, pp. 435–444.

[21] Sarkar N. and Chaudhuri B.B., "An efficient approach to estimate fractal dimension of textural images", *Pattern Recognition*, vol. 25, pp. 1035–1041, 1992.

[22] Demartines P., *Analyse de données par réseaux de neurones auto-organisés*, PhD thesis, Institut National Polytechnique de Grenoble, 1994.

[23] P. Comon, J.L. Voz, and M. Verleysen, "Estimation of performance bounds in supervised classification", in *ESANN94-European Symposium on Artificial Neural Networks*, M. Verleysen, Ed., Brussels, Belgium, April 1994, pp. 37–42, D facto publications.

[24] C. W. Therrien, *Decision estimation and classifiation*, Willey, 1989.

[25] B. Efron, "Bootstrap methods: Another look at the jackknife", *The Annals of Statistics*, vol. 7, no. 1, pp. 1–26, 1979.

[26] B.D. Ripley, "Flexible non-linear approaches to classification", in *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, V. Cherkassky, J.H. Friedman, and H. Wechsler, Eds. 1993, Sringer Verlag.

[27] A. Feelders and W. Verkooijen, "Which method learns most from the data ?", Tech. Rep., University of Twente, Deparment of Computer Science, P.O. Box 217, 7500 AE Enschede, The Netherlands, Jan. 1995, Anonymous FTP: /pub/doc/pareto/aistats95.ps.Z on ftp.cs.utwente.nl.

[28] P.A. Devijver and J. Kittler, *Pattern Recognition: A Statistical Approach*, Prentice Hall, London, 1982.

[29] L. Marascuilo and M. McSweeney, *Nonparametric and distribution-free methods for the social science*, Brooks/Cole Publishing Company, Monterey, CA, 1977.

[30] J.L. Fleiss, *Statistical methods for Rates and Proportions*, Wiley, New York, 2nd edition, 1981.

[31] F. Hergert, H. G. Zimmermann, U. Kramer, and W. Finnof, "Domain independent testing and performance comparisons for neural networks", in *Artificial Neural Networks, 2*, I. Aleksander and J. Taylor, Eds. 1992, pp. 1071–1076, Nort-Holland, Amsterdam.

[32] D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, 1986.

[33] P. Comon and G. Bienvenu, "Ultimate performance of QEM classifiers", *IEEE Trans. Neural Networks*, May 1995, Submitted.

[34] F. Blayo et al., "Deliverable R1-B3-P - Task B3: Software simulation library", Tech. Rep., Elena-NervesII "Enhanced Learning for Evolutive Neural Architecture", ESPRIT-Basic Research Project Number 6891, June 1994.

[35] J. Hertz, A. Krogh, and R.G. Palmer, *Introduction to the theory of Neural Computation*, Addison Wesley, 1991.

[36] A. Guérin-Dugué et al., "Deliverable R1-B1-P - Task B1: Databases", Tech. Rep., Elena-NervesII "Enhanced Learning for Evolutive Neural Architecture", ESPRIT-Basic Research Project Number 6891, June 1994.

[37] P. Comon et al., "Deliverable R2-A-P - Axis A: Theory", Tech. Rep., Elena-NervesII "Enhanced Learning for Evolutive Neural Architecture", ESPRIT-Basic Research Project Number 6891, June 1994.

[38] J.L. Voz, M. Verleysen, P. Thissen, and J.D. Legat, "A practical view of suboptimal bayesian classification", in *IWANN95-Proceedings of the International Workshop on Artificial Neural Networks*, Prieto Mira, Cabestany, Ed., Malaga, Spain, June 1995, Springer-Verlag Lecture Notes in Computer Sciences, Submitted.

[39] J.L. Voz, M. Verleysen, P. Thissen, and J.D. Legat, "Suboptimal bayesian classification by vector quantization with small clusters", in *ESANN95-European Symposium on Artificial Neural Networks*, M. Verleysen, Ed., Brussels, Belgium, April 1995, D facto publications, To be published.

[40] P. Comon, G. Bienvenu, and T. Lefebvre, "Supervised design of optimal receivers", in *NATO Advanced Study Institute on Acoustic Signal Processing and Ocean Exploration*, Madeira, Portugal, July 26-Aug. 7 1992.

[41] M. Verleysen, P. Thissen, J.L. Voz, and J. Madrenas, "An analog processor architecture for neural network classifier", *IEEE Micro*, vol. 14, no. 3, pp. 16–28, June 1994.

[42] J.A. Sirat and J.P. Nadal, "Neural trees: A new tool for classification", Report, Laboratoires d'Electronique Philips, 1990.

[43] J.M. Moreno, F. Castillo, and J. Cabestany, "Enhanced unit training for piecewise linear separation incremental algorithms", in *ESANN93-European Symposium on Artificial Neural Networks*, M. Verleysen, Ed., Brussels, Belgium, April 1993, pp. 33–38, D facto publications.

[44] J.M. Moreno, F. Castillo, and J. Cabestany, "Optimized learning for improving the evolution of piecewise linear separation incremental algorithms", in *New Trends in Neural Computation. IWANN93*, Prieto Mira, Cabestany, Ed., Sitges, Spain, 1993, pp. 272–277, Springer-Verlag Lecture Notes in Computer Sciences.

[45] J.M. Moreno, F. Castillo, and J. Cabestany, "Improving piecewise linear separation incremental algorithms using complexity reduction methods", in *ESANN94-European Symposium on Artificial Neural Networks*, M. Verleysen, Ed., Brussels, Belgium, April 1994, pp. 141–146, D facto publications.

[46] M. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the theory of brain mechanisms*, Spartan books, New York, 1962.

[47] S.I. Gallant, "Optimal linear discriminants", in *Proceedings of the 8th International Conference on Pattern Recognition*, 1986, pp. 849–854.