Twelve numerical, symbolic and hybrid supervised classification methods

Olivier Gascuel, Bernadette Bouchon-Meunier, Gilles Caraux, Patrick Gallinari, Alain Guénoche, Yann Guermeur, Yves Lechevallier, Christophe Marsala, Laurent Miclet, Jacques Nicolas, et al.

TWELVE NUMERICAL, SYMBOLIC AND HYBRID SUPERVISED CLASSIFICATION METHODS*

OLIVIER GASCUEL[a,†], BERNADETTE BOUCHON-MEUNIER[b], GILLES CARAUX[a], PATRICK GALLINARI[b], ALAIN GUÉNOCHE[c], YANN GUERMEUR[b], YVES LECHEVALLIER[d], CHRISTOPHE MARSALA[b], LAURENT MICLET[e], JACQUES NICOLAS[f], RICHARD NOCK[a], MOHAMMED RAMDANI[b], MICHÈLE SEBAG[g], BASAVANNEPPA TALLUR[f], GILLES VENTURINI[h] and PATRICK VITTE[c]

[a] *LIRMM, 161 rue Ada, 34392 Montpellier cedex 5, France*

[b] *LAFORIA-IBP, Université Paris 6, case 169, 4 place Jussieu, 75252 Paris cedex 5, France*

[c] *LIM-CNRS, 163 Av. de Luminy, 13288 Marseille cedex 9, France*

[d] *INRIA Rocquencourt, Domaine de Voluceau, BP 105, 78153 Le Chesnay, France*

[e] *IRISA-ENSSAT, Rue de Kerampont, 22300 Lannion, France*

[f] *IRISA, Campus de Beaulieu, 35042 Rennes, France*

[g] *LMS-CNRS 317, Ecole Polytechnique, 91128 Palaiseau, France*

[h] *E3I, 64 Av. Jean Portalis, BP 4, 37913 Tours cedex 9, France*

Supervised classification has already been the subject of numerous studies in the fields of Statistics, Pattern Recognition and Artificial Intelligence under various appellations which include discriminant analysis, discrimination and concept learning. Many practical applications relating to this field have been developed. New methods have appeared in recent years, due to developments concerning Neural Networks and Machine Learning. These "hybrid" approaches share one common factor in that they combine symbolic and numerical aspects. The former are characterized by the representation of knowledge, the latter by the introduction of frequencies and probabilistic criteria. In the present study, we shall present a certain number of hybrid methods, conceived (or improved) by members of the SYMENU research group. These methods issue mainly from Machine Learning and from research on Classification Trees done in Statistics, and they may also be qualified as "rule-based". They shall be compared with other more classical approaches. This comparison will be based on a detailed description of each of the twelve methods envisaged, and on the results obtained concerning the "Waveform Recognition Problem" proposed by Breiman *et al.*,[4] which is difficult for rule based approaches.

Keywords: Supervised classification, statistical methods, discriminant analysis, neural networks, classification trees, machine learning approaches, hybrid methods, waveform recognition problem.

1. INTRODUCTION

1.1. Survey Motivation and Outline

Supervised Classification is the problem of learning a classification function from examples. If these examples are patients described through a certain number of symptoms or images of digits described by pixel values, a classification function

†Corresponding author. E-mail: gascuel@lirmm.fr

*The authors are members of the French research group SYMENU which is composed of fifty participants from ten laboratories. SYMENU stands for "Numerical-Symbolic Discrimination Methods". The group is supported by the French Ministry of Research and Education.

must predict the class of these elements, in other words an illness or a digit. The goal is to construct these classification functions by using a set of previously classified examples as the only *a priori* knowledge. This is called the *learning set*. The subject is vast, involving numerous applications, and has already been extensively explored in the fields of statistics, pattern recognition and artificial intelligence. Depending on the field in question, supervised classification appears under different names ranging from *discriminant analysis, discrimination and concept learning*. Over the last ten years, several new approaches have appeared, in particular due to developments in the domains of Neural Networks and Machine Learning. These approaches share one common factor in that they facilitate the coexistence of symbolic aspects, such as those found in example representation or in the manner in which the learned classification function is expressed, and numerical aspects, such as the introduction of frequencies and probabilistic criteria.

The aim of this paper, written by the French research group SYMENU, is to study these hybrid approaches, which combine symbolic and numerical aspects. We shall present a certain number of methods conceived (or improved) by members of our group. These methods issue mainly from Machine Learning and from research on Classification Trees done in Statistics. They may also be qualified as "rule-based" in that they all use, in some respects, typically AI rules for form *IF description THEN class*. We shall provide a presentation of these methods, which should be simple but sufficiently complete, thus demonstrating the various ways of combining both the symbolic and numerical aspects, as well as the advantages and the inconveniences of these combinations. Besides, numerous other supervised classification approaches exist, which are mainly numerical and derived from Statistics, Pattern Recognition and Neural Networks. We shall present the most frequently used and cited of these methods. This will enable us to illustrate the specificities of hybrid, rule-based approaches. Moreover, the mere principle as well as the properties and recent developments of these classical methods often seem not to be well known by the Machine Learning community, and this paper could serve as an introduction to these aspects.

In the last few years, several systematic comparisons of various methods on data sets have been published.[6,35,62]. In particular, King *et al.* provide a confrontation between 17 well-known methods (CART, CN2, C4.5, SMART, ...) on 12 real-word sized problems. Our purpose is somewhat different. Most of the methods we present are original, or very recent. Therefore, we decided in favour of a precise, explanatory description of these methods, rather than an extensive experimental comparison which could be seen as untimely. To illustrate the differences among the methods presented, we have evaluated them according to a classical problem of supervised classification, that of "Waveform Recognition"[4] which is known to be difficult for rule-based approaches. Our objective was to show experimentally that our original hybrid algorithms have better performances (on this difficult problem) than classical symbolic methods such as, for example, CART, and that they are not very far from well established numerical methods such as, for example, Fischer's linear discriminant function.

This paper is organized as follows. First, we shall define more formally the subject of supervised classification and introduce the notation (Sec. 1.2). A brief history will allow the methods to be contextualized (Sec. 1.3). Then, we shall outline the main points of the hybrid approaches (Sec. 1.4). The waveform recognition problem is presented in Sec. 1.5. A comparison of methods is not that easy to accomplish, even on a particular application. Thus, a certain number of criteria have been retained which are presented in Sec. 1.6.

The remainder of the article is devoted to a description of the methods themselves. In order to provide a sufficiently large scope of supervised classification, we first describe several classical methods: Sec. 2 addresses the main statistical approaches with particular emphasis on four classical methods; Sec. 3 describes Neural Networks, and more especially the Multi-Layer Perceptron; Sec. 4 presents the CART Classification Tree approach of Breiman *et al.*[4] Remaining parts are devoted to original contributions. Section 5 proposes the combination of Classification Trees and Fuzzy Sets. Sections 6–8 present methods based on the use of Decision Rules, similar to the rules of an expert system: the first adopts a combinatorial approach; the second describes "Decision Committees" which are a simple and comprehensible way of combining rules; the third relies on the use of a Genetic Algorithm. The two following parts look at methods derived from the Version Space approach[43]: Sec. 9 proposes the combination of this approach and of a Hierarchical Clustering based pre-processing; Sec. 10 is also related to the Star Algorithm[41] and is based on the use of "constraints". The final sections (11 and 12) provide a general discussion of our results and observations, and conclude the paper.

1.2. Supervised Classification

Let us assume that we have a set of examples E, called the learning set, of cardinal n. As a general rule, this set constitutes only a small part of the entire range of possible examples which is often infinite. Each example, which we shall denote (x, c) represents a pair (description, class). The description x belongs to the description space X. In the case where the description is of the value-attribute type, the description space is a product space $X = X_1 \times X_2 \times \cdots \times X_p$, in which each X_j is the set of values possible for the jth attribute. The value of this attribute for a given description x is denoted as x_j. The class c of an example is an element of the classes $\{C_1, C_2, \ldots, C_g\}$. The set E is thus partitioned into g subsets $E_1, E_2, \ldots, E_g$, of cardinal $n_1, n_2, \ldots, n_g$, respectively. In the special case where there are only two classes, these are denoted C_+ and C_-. In this case, one frequently refers to the learning of a *concept*, the *positive* examples being those of E_+ and the *negative* examples or counter examples are those of E_-. Note that each example is assigned to a single class. However, the same description may correspond to several examples belonging to different classes. Therefore, the link between the description and the class is not necessarily functional, or deterministic. In practice, this is almost never the case as the description is generally incomplete and partially erroneous.

Our goal is to build a classification function from these learning examples, thus allowing a class to be attributed to a new example whose class is unknown. The general principle consists in constructing a function which will enable a good reclassification of the learning examples. However this principle does not suffice, since the goal is to achieve, above all, good performance on the new examples. In order to estimate it, a *test set* is used which is independent of the learning set. In practice, we have a finite set of examples that we divide into one (or several) learning sets and one (or several) test sets. Due to the nondeterministic nature of the problems processed, and because the learning set generally constitutes only a small subset of the possible examples, it must be admitted that the classification function chosen does not provide a perfect reclassification of the learning examples. Excellent results on the learning set do not necessarily yield excellent test results, therefore a compromise between learning performance and test performance must be found. Moreover, one often wishes the learning procedure to provide a classification function containing explanations on the class partition observed on the data.

1.3. A Brief History

Supervised classification has quite a long history; our purpose here therefore will be to provide a limited scope of the subject. We shall concentrate on a broad outline highlighting the specificity of the hybrid approaches.

The discriminant function of Fisher[17] was one of the first methods to appear. In this method, examples are represented by points of $\mathbf{R}^p$, and classes are separated by linear or quadratic surfaces which are optimal when the classes are Gaussian (cf. Sec. 2). The Rosenblatt Perceptron[56] also uses a representation in $\mathbf{R}^p$ with linear decision surfaces while enabling an adaptive, or incremental, learning, which means that it is capable of taking examples into account consecutively as each one "arrives". Current Neural Network models, in quite a number of cases, are direct descendants of the Perceptron (cf. Sec. 3).

The 1960's saw the development of several studies in pattern recognition, notably statistical pattern recognition. Duda and Hart[15] and Fukanaga[21] provide a general presentation of that period. Among these studies, Sonquist and Morgan[64] produced the first works to appear on Classification Trees whose methods are still considered today as the pivots of the hybrid approach. Tree-based approaches are interesting from two points of view. On the one hand, they naturally integrate qualitative or symbolic representations; on the other hand, they contain a very high explanation power (cf. Secs. 4 and 5). They also constitute a nonparametric method class in that they do not presuppose a data model. Other nonparametric approaches were also developed, notably Parzen's Kernel method[49] and the k-Nearest-Neighbor technique.[11] These approaches are not only simple to implement, they also contain remarkable asymptotic properties (cf. Sec. 2). Concerning binary data processing (represented in $\{0,1\}^p$), it is worth mentioning the works of Bongard[3] and those of Quinqueton and Sallantin,[52] who present an alternative to Classification Trees and who are primarily responsible for some of the methods which will be presented below (cf. Secs. 6 and 7).

In the field of Artificial Intelligence, the development of supervised classification methods dates essentially from the beginning of the 1970's, with the famous "Arch Concept Learning" problem as devised by Winston.[75] These methods contributed to the capacity to learn from structural example descriptions, thus abandoning the value-attribute model used in all of the above-mentioned approaches. Several representation modes were envisaged, notably those based on semantic networks[75] and on predicate logic.[70] Mitchell[43] showed that in the AI methods, a more or less explicit solution, or *version*, space exists, partially ordered by a generalization relation. Moreover, Mitchell proposed an algorithm to search this space (cf. Secs. 9 and 10). Other methods were proposed, notably the Star algorithm,[41] (cf. Secs. 8 and 10) while the notion of generalization was also explored,[14] particularly within the scope of Inductive Logic Programming.[45]

Neural networks have come to the forefront due to the work of Hopfield[33] on associative memory models, inspired by statistical physics. In the field of supervised classification, a decisive step was reached with the development of the Multi-Layer Perceptron.[57] The MLP associates the notion of hidden cells[40] with a learning algorithm of the stochastic gradient type, such as the backpropagation of the error gradient, thus enabling a break-away from the linear framework. Large scale applications have been handled successfully, notably in the domain of speech processing[60] and character recognition.[12]

1.4. Hybrid Approaches

Hybrid approaches comprise both neural network methods and other more typical AI methods as mentioned above. Neural networks are close to numerical methods, and the symbolic aspect appears mainly in the network architecture which is symbolic by nature, and which expresses *a priori* knowledge on the problem processed. This symbolic aspect appears also in recent studies which aim at giving explanatory virtues to the networks, especially by implementing rule extraction mechanisms.[62] Concerning AI methods, although originally symbolic, it was soon obvious that an overly logical approach,[41] aiming to construct perfect classification functions on the learning set, was not the best solution. Thus the idea emerged of constructing classification functions with "good" learning performance, this being quantified by statistical numerical criteria.[9,22] More generally, the proximity of the problem lends itself to a more natural cooperation between the statistical approach, the neural network approach or that used in AI. For example, validation methods, of the cross-validation or bootstrap type,[28,65] naturally apply to "nonstatistical" methods. This is also the case for certain fundamental results produced by Vapnik and Chervonenkis,[68] which can prove the asymptotic consistency of numerous methods, notably neural[24] and those based on classification trees.[4] Finally, numerous statistical algorithms may be used, for example, to conduct data pre-processing[47] (Sec. 9).

The hybrid approaches thus constitute a research domain, rather than a set of well defined methods. They take advantage of the tools and results from various fields — statistics, pattern recognition, neural networks and artificial intelligence. Among the main objectives pursued, the following may be mentioned: performance, whether in terms of calculation time or classification error rate; the explanatory nature of the learned classification function; the capacity to handle complex data, represented for example in predicate logic, which cannot be handled by conventional numerical methods.

1.5. The Waveform Recognition Problem

Waveform recognition is an artificial problem which was introduced by Breiman *et al.*[4] in the study of classification trees. In their book, these authors used two illustrative applications: the digit recognition problem, and the waveform problem. With the former, their classification tree program, CART, achieves excellent results, in terms of both classification accuracy and tree size. And they chose the waveform problem precisely because it is difficult for classification trees, thus providing a better illustration of program behavior. Moreover, the first rule-based programs issued from machine learning do not solve the problem better. For example, CN2[9] obtains results which are not as good as those of CART, while the SDL[5] which relies on a heavy, simulated annealing algorithm, only slightly improves the CART performance. Therefore, we chose this problem for the same reasons as Breiman *et al.*, and because it appeared challenging for our approaches.

The problem is to discriminate between three classes of *waveforms*. Each waveform simulates a quantitative chronological phenomenon observed in 21 regularly spaced instants. It is an object characterized by a point of $\mathbf{R}^{21}$.

1.5.1. Analytical class definition

The three classes are obtained by combining three basic *waves* two by two. The latter, which we shall denote h_1, h_2, and h_3 are represented in Fig. 1. They are unimodal and dephased. They are associated by a random convex combination before being perturbed by a random Gaussian noise. Thus, considered analytically, class C_1, C_2 and C_3 elements are respectively conceived through the expressions:

$$x = uh_1 + (1-u)h_2 + \varepsilon\,,$$

$$x = uh_1 + (1-u)h_3 + \varepsilon\,,$$

$$x = uh_2 + (1-u)h_3 + \varepsilon\,,$$

where u is a random variable of uniform density on the $[0,1]$ interval, and where ε is a random Gaussian centred vector, with a variance-covariance matrix unity. We also consider the classes to be equiprobable.

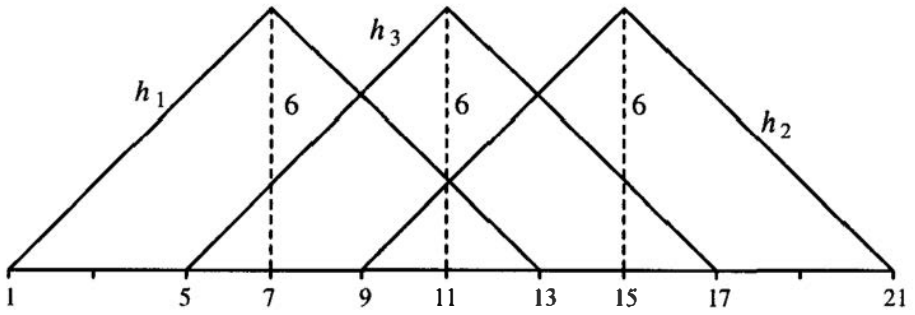


Fig. 1. The three basic waves.

It may be noted that in the absence of the ε Gaussian noise, the three classes would be represented in $\mathbf{R}^{21}$ by the three sides of the h_1, h_2 and h_3 summit triangle, as shown in Fig. 2. In this case, the problem would be deterministic. A perfect assignment function (without classification error) should consist in assigning each description x to the corresponding class on the side to which x belongs. The random Gaussian noise will perturb these considerations. The effect produced is that each point of $\mathbf{R}^{21}$ is an acceptable description in terms of the three classes, and the problem is no longer deterministic.

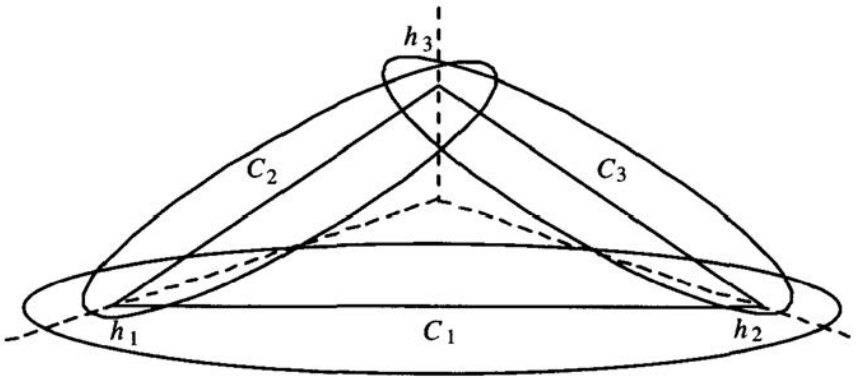


Fig. 2. Schematic representation of the three classes.

Given that the classes are equiprobable, we can demonstrate that the minimal error assignment rule, or Bayes rule (cf. Sec. 2.1), consists in assigning a description x to the class whose conditional probability density is maximum. The assignment areas are now not so easy to define. Nevertheless, we can predict that the boundaries, points of equilibrium between the conditional densities, trace "quasilinear" surfaces. In fact, we know that for symmetrical reasons, at least one of these boundaries is perfectly linear. This boundary corresponds to the vertical axis passing by h_3 in Fig. 2. The other boundaries, represented in the figure by a dotted line, are not necessarily linear, however they are most likely to be quite regular. Moreover, in the regions which are near the summit h_1 and h_2 of the triangle, regions of high uncertainty and high density, these boundaries must be well approximated by hyperplanes passing by (or close to) h_1, respectively h_2. The quasi linearity of the optimal boundaries explains, to an extent, the success of certain methods such

as the statistical parametric methods and neural networks. We shall discuss the subject at greater length in what follows.

According to Breiman *et al.*[4] the error rate of the Bayes rule is around 14% for this problem. No classification function, even if it is learned on an infinite number of examples, can expect to beat this performance. As the learning sets which we manipulated were of limited size ($n = 300$), we cannot even expect to reach this optimal result due to the sampling noise.

1.5.2. Learning and test sets

Eleven learning sets, each of them under three different codings as described below, were distributed within the SYMENU group. The first set corresponds to data used by Breiman *et al.* to test the CART program performance. The other ten were drawn at random, according to the process described above. Each set consists of 300 examples drawn independently (note that the *a priori* probability of the classes is 1/3). As it was possible to make use of several comparable learning sets, this enabled us to show quite precise averaged results, and to highlight the variability of these results. Using the same process, and independently of the learning sets, we drew a test sample of 5000 examples. The classification methods which we shall present in the following were validated using these test examples.

1.5.3. Attribute discretization

The attributes presented above are all continuous. We transformed the eleven learning sets and the test sample, in order to evaluate the performance of the methods on discrete data which are nearer to symbolic. The initial quantitative attributes were split into 21 binary descriptors and 21 ternary descriptors. As a result, each of the files initially drawn for the quantitative attributes produced two discretized files. Boundary identification, essential in coding the quantitative attributes, was achieved through maximization of the link between the partition into classes and that obtained by discretization in intervals of the variable to be coded. This link was measured using the χ^2 criterion. The algorithm we used[18,37] is based on dynamic programming, and it enables optimal coding according to this criterion. Let k be the number of desired intervals and n the number of examples. The complexity of this algorithm is $O(kn^2)$ when $k \neq 2$ (a very simple, linear algorithm is sufficient when $k = 2$). Therefore, it may be seen as challenging, in comparison with Catlett's[6] which has $O(n \log(n))$ complexity, but which is suboptimal. The 21 descriptors were split in this manner on a base of 1500 examples which were drawn specifically. This coding was then applied to each of the files.

1.6. Evaluation Criteria

We selected a certain number of criteria to evaluate the methods, which are shown in Table 1. The first column describes the type of data processed: binary, ternary or continuous. The following columns give the average results obtained on the 11 learning files. Each criterion shall now be examined in greater detail.

Table 1. Format of tables to be completed for the various methods (numbers are given only as illustration).

Data	Train	Test	Test(1)	Test(2)	Test(3)	CPUtrain	CPUtest	Size
Binary	17.3% (1.1%)	22% (1%)	28.9%	18.6%	18.6%	0.3″	1″	66

1.6.1. Classification error rate

The classification error rate is the proportion of cases where the example class is not identified by the learned classification function. This proportion is estimated by using the test sample described above. The Test column gives the average result obtained on the 11 learning files, as well as the standard deviation (number in brackets). A high standard deviation indicates that the method is unstable, and can sometimes yield bad results. The 3 Test(i) columns give the results obtained for the three subsets of the test set, corresponding to each of the 3 classes. When the results in the columns are unbalanced, this means that there is a "weakness" in the method. For example, in Table 1, it may be noticed that the method classes the examples of the first class poorly. The Train column gives the average results, and the standard deviation obtained on the learning sets. A low score in this column does not necessarily indicate that the method is good. What is important is that the difference between learning and test is not too high. If this is the case, the method has a tendency towards rote learning. Most likely, it has too high a degree of freedom.

1.6.2. Computing time

It is possible to distinguish between two calculation times, the required learning time: CPUtrain, and the required time to decide on a new example class: CPUtest, which is the time required to classify the 5000 examples of the test set. Depending on the application in question, the relative importance of learning and test time will vary. If, for example, the data being processed is scientific data with a very long acquisition time, a learning time of several days is not always a drawback. However, if an exploratory procedure is adopted to try to "understand" the data, by varying the method parameters as well as the description mode, then the learning must be quick. The same applies to the decision: the waiting time for a medical diagnosis may be a few seconds; the decision must be "real-time" in the case of phoneme recognition for continuous speech processing.

1.6.3. Explanation size and power

The Size column shows the complexity of the learned classification function. Of course there is no unique measurement. In the case of numerical methods, we count the number of parameters. In the case of more symbolic (or logical) methods, it is possible to count for example, the number of literals used. Size is directly

linked to the explanation power of the method. The greater the size, the lower the explanation power of the classification function. Moreover, for equal sizes, it is usually easier to interpret a logical formula than a mathematical formula based on real parameters. Again, the results should be modulated in function of the application: a good explanation power is of little use for phoneme recognition, but is indispensable in the medical field. Finally, it is worth pointing out that size and decision time are mandatorily correlated.

1.6.4. Finding a compromise

Generally speaking, a compromise must be found between the qualities mentioned above. For example, a low size associated with a high explanation power is usually obtained at the expense of a high error rate. Nevertheless, we are also aware of the fact that too high a size often corresponds to rote learning, and to an equally high error rate. From another point of view, it is often more difficult to optimize in the discrete space, and thus obtain logical formulae containing explanation, than to optimize in the continuous space, which yields numerical functions and few explanations. Also, certain methods such as the k-Nearest-Neighbor, do not proceed to any learning. However, this is balanced by a high decision time. It is an empty quest to hope for a perfect method, and depending on the application one quality will be given preference over another.

2. STATISTICAL METHODS

We consider here two approaches which are very classical in statistical pattern recognition: the *parametric* approach, which we expound in the Gaussian framework, and the *nonparametric* one. These two approaches are opposed in principle, and reveal two distinct ways of treating the subject. However, first, we shall introduce the *Bayes Minimal Error Decision Rule* which plays a central role in statistical discrimination.

2.1. Bayes Minimal Error Decision Rule

As we have seen in Sec. 1.2, the problems treated are rarely deterministic, and objects from two different classes often adhere to the same description. Thus, classification which proceeds from example description is automatically subject to a degree of incertainty. Bayes Decision Rule consists in assigning the object described by x to the class C_i such that $\Pr(C_i/x)$ is maximum. It is easy to demonstrate that this rule is optimal, in the sense that it minimizes the misclassification probability. In reality, however, it is very rare to know the class probabilities given the description. On the contrary, it is much easier to find out, or to estimate, class probabilities and the description distribution given the class. Using the Bayes theorem, an operational expression may then be found:

$$\Pr(C_i/x) = \frac{\Pr(C_i) f(x/C_i)}{f(x)},$$

where $f(x)$ and $f(x/C_i)$ represent, respectively the density and conditional density of x. The denominator of this expression being independent of C_i, the Bayes Decision Rule consists in assigning x to the class which maximizes $\Pr(C_i)f(x/C_i)$. Let us now consider the case where the classification is binary ($g = 2$). By putting

$$\lambda(x) = \ln\left(\frac{f(x/C_+)}{f(x/C_-)}\right) + \ln\left(\frac{\Pr(C_+)}{\Pr(C_-)}\right), \tag{1}$$

we find that the Bayes Decision Rule is expressed as

$$\forall x,\ \text{If } \lambda(x) \geq 0 \text{ Then } C_+,\ \text{Else } C_-.$$

From this expression, it results that the surface defined by the equation $\lambda(x) = 0$ is the boundary separating the two areas assigned to C_+ and C_- in X.

It should be noted that precise knowledge of the probability laws ruling the descriptions of the objects to be classified is essential when implementing the Bayes Decision Rule. However, in most cases these laws are unknown. But they can be estimated by diverse methods, the most classical of which we are going to examine below.

2.2. Parametric Methods; the Gaussian Case

We shall assume in this section that the conditional probability laws of descriptions are elements of a known family defined by a vector θ. More precisely, the generic analytical expression $f(x/\theta)$ is known, however, the θ_i parameters, which characterize the distribution of each class C_i remain unknown. The parametric approach uses the Bayes Decision Rule after having estimated these unknown parameters from the learning examples. In the most general case, this estimation is often carried out through the maximum likelihood method. However, here we shall only discuss the Gaussian case, which is easier to process.

Let us assume that the description x of an object to be classified consists of p continuous attributes ($x \in \mathbf{R}^p$), and that the set of descriptions of each class C_i is randomly dispersed in $\mathbf{R}^p$ according to a Gaussian distribution with mean vector μ_i and variance-covariance matrix Σ_i.

In this case, the conditional density of x is expressed as

$$f(x/C_i) = ((2\pi)^p \det\Sigma_i)^{-1/2} \exp(-\tfrac{1}{2}(x-\mu_i)^t \Sigma_i^{-1}(x-\mu_i)).$$

The Bayes Decision Rule now consists in minimizing the expression

$$\Delta^2_{\Sigma_i}(x,\mu_i) - 2\ln(\Pr(C_i)) + \ln(\det\Sigma_i), \tag{2}$$

where $\Delta^2_{\Sigma_i}(x,\mu_i)$ is the Mahalanobis distance between x and the mean description of the class C_i:

$$\Delta^2_{\Sigma_i}(x,\mu_i) = (x-\mu_i)^t \Sigma_i^{-1}(x-\mu_i).$$

If we admit that *a priori* probabilities of classes are identical and that the variance-covariance matrixes are equal ($\forall i,\ \Sigma_i = \Sigma$), it derives from (2) that the Bayes

Decision Rule consists in minimizing $\Delta_\Sigma(x, \mu_i)$. In this particular case, we then assign an object described by x, to class C_i whose mean description μ_i is the nearest to x. In a more general manner, we show that the equality of the variance-covariance matrices induces a linear discrimination. This is illustrated simply by examining the binary case ($g = 2$). By developing **Eq.** (1), one finds the linear expression

$$\lambda(x) = (x - \mu)^t \Sigma^{-1} (\mu_+ - \mu_-) + \ln \begin{pmatrix} \Pr(C_+) \\ \Pr(C_-) \end{pmatrix},$$

where $\mu = \frac{1}{2}(\mu_+ + \mu_-)$. The optimal separation surface between C_+ and C_- is a hyperplane of $\mathbf{R}^p$, of equation $\lambda(x) = 0$, defined by $p + 1$ parameters.

In the general case, by developing expression (2), one easily finds that the boundary is a surface of the second degree in x (Ref. 15, p. 30) and that it is defined by $1 + p(p+1)/2$ parameters. Hence, the discrimination is said to be quadratic. The Gaussian model, like all parametric models, requires an initial phase of parameter estimation (or learning). This consists in simply estimating the coordinates of the mean vectors μ_i and the matrix elements Σ_i on the examples of the learning set.

We applied the method described above to waveform data, by assuming, on one hand, the equality of the variance-covariance matrices (linear discrimination) and, on the other hand, the inequality of these matrices (quadratic discrimination). The three types of data were processed using the DISRIM procedure of the SAS software, running on a Sparc 2. Results are given in Tables 2.1 and 2.2. What is noticeable is the robustness of the Gaussian approach. In fact, performance undergoes little degradation when we drop the Gaussian hypothesis by binarization or ternarization of the continuous descriptors. Moreover, this hypothesis appears to be extremely interesting on the level of calculation time, due to the existence of an analytical solution (2). The learning time is the shortest among all the methods, if the procedures without learning are excluded (Sec. 2.3). We would also like to highlight the fact that the quadratic discrimination results are not as good as those for linear discrimination. In fact, the overparameterization of the quadratic discrimination is not worthwhile in this case. It serves only to improve the discrimination of the elements of the learning set and not those of the test sample. This phenomenon occurs frequently, and is revelatory of the equilibrium to be found between the precision of the learning and the real test performance.

Table 2.1. Parametric approach, linear discrimination.

Data	Train	Test	Test(1)	Test(2)	Test(3)	CPUtrain	CPUtest	Size
Binary	17.3% (1.1%)	22% (1%)	28.9%	18.6%	18.6%	0.3″	1″	66
Ternary	15.9% (1.8%)	19.9% (0.5%)	28.8%	15.5%	15.5%	0.3″	1″	66
Contin.	12.8% (1.3%)	20.4% (1%)	23.5%	19.8%	17.9%	0.3″	1″	66

Table 2.2. Parametric approach, quadratic discrimination.

Data	Train	Test	Test(1)	Test(2)	Test(3)	CPUtrain	CPUtest	Size
Binary	10.1% (1.2%)	25% (0.7%)	31.6%	21.8%	21.5%	0.6″	5″	696
Ternary	7.8% (0.8%)	21.9% (0.7%)	28.6%	18.2%	19%	0.6″	5″	696
Contin.	4.2% (0.9%)	21.4% (1%)	21.5%	21.7%	21%	0.6″	5″	696

2.3. Nonparametric Methods

In statistical supervised classification, the nonparametric approaches are characterized by the absence of an *a priori* hypothesis on the conditional distribution of the examples in the description space. As above, we use the Bayes Decision Rule by trying to identify the class C_i which maximizes the *a posteriori* probability $\Pr(C_i/x)$. In order to achieve this, and assuming the class probabilities as *a priori* knowledge, we need to estimate the conditional densities $f(x/C_i)$ of the descriptions. Statistical nonparametric methods proceed through local estimation of these densities, the two most outstanding approaches being that of Parzen's Kernel[49] and k-Nearest-Neighbor.[11]

2.3.1. Parzen's Kernel method

Let x be the description in $\mathbf{R}^p$ to be classed. We denote then as $W_h(x)$ the hypercube (or window) centred on x of side h. For C_i fixed, let $k_i(x)$ be the number of examples of E_i included in $W_h(x)$. We can easily demonstrate that

$$k_i(x) = \sum_{e \in E_i} I_{W_1(0)}\left(\frac{x - x_e}{h}\right),$$

where $I_{W_h}(x)$ is the indicator function associated to the hypercube $W_h(x)$ and where x_e is the description of the example e. It seems only natural to take $k_i(x)/n_i$ as the estimation of the conditional probability of membership to the window $W_h(x)$. By dividing this probability by the volume $V_h = h^p$ of the window, we obtain the estimation of the conditional density

$$\hat{f}(x/C_i) = \frac{k_i(x)/n_i}{V_h} = \frac{1}{n_i} \sum_{e \in E_i} \frac{1}{V_h} I_{W_1(0)}\left(\frac{x - x_e}{h}\right). \tag{3}$$

Moreover, Hand[27] has shown that this estimation converges towards the density $f(x/C_i)$, if the dimension h of the window diminishes according to a law in $O(1/\sqrt{n_i})$, when n_i increases.

Equation (3) defines $\hat{f}(x/C_i)$ as the sum of Boolean contributions of each learning example of class C_i, and the h parameter determines the "scope of action" of

these examples. The result is that $\hat{f}(x/C_i)$ is, by construction, a discontinuous function even though $f(x/C_i)$ usually is a continuous function. This is not very satisfactory even when we know that the discontinuity tends to disappear as n_i increases. In order to smooth $\hat{f}(x/C_i)$, we define a less contrasted contribution of examples by using a continuous function for characterizing the scope and intensity of the example influence. Expression (3) may then be generalized by

$$\hat{f}(x/C_i) = \frac{1}{n_i} \sum_{e \in E_i} \frac{1}{V_h} K\left(\frac{x - x_i}{h}\right), \tag{4}$$

where K is a positive function, called *kernel*, such as

$$\int K(u)du = \int \frac{1}{V_h} K\left(\frac{x - x_e}{h}\right) dx = 1\,.$$

This latter expression, associated with (4), makes $\hat{f}(x/C_i)$'s integrate equal to one, and thus estimation (4) may be assimilated to a probability density. The term h is called the *smoothing factor*. This parameter plays a determining role by defining both the amplitude and the scope of the influence of the learning examples. If too high, we would tend to level off the variations of $f(x/C_i)$. However, if too low the estimation of $f(x/C_i)$ will become a "comb" with multiple peaks localized at the points of the learning sample. For waveform data, we have chosen a Gaussian kernel, as is usually the case

$$K(u) = \frac{1}{(2\pi)^{p/2}} \exp\left(-\frac{1}{2} u^t u\right).$$

It has the advantage of defining a symmetric influence around examples as well as a progressive extinction of the contribution of these examples as one departs from the description to be classed. We tested the method for different values of h (0.5, 1 and 2), using the Euclidean distance. The results are very stable. In Table 2.3, we provide only the statistics obtained for the case where $h = 1$. The CPU times are those provided by the SAS software running on a Sparc 2. The size is that of the data, since there is no prior learning phase, i.e. 300 (examples) $\times$ 21 (descriptors).

Table 2.3. Nonparametric approach, Parzen's Kernel method.

Data	Train	Test	Test(1)	Test(2)	Test(3)	CPUtrain	CPUtest	Size
Binary	3.3% (2.0%)	22.7% (1.5%)	47.8%	8.7%	11.5%	0″	893″	300 × 21 = 6300
Ternary	0%	21.7% (0.6%)	35.5%	15.3%	14.2%	0″	866″	300 × 21 = 6300
Contin.	0%	22.2% (1.2%)	26.4%	18.7%	21.4%	0″	933″	300 × 21 = 6300

2.3.2. k-nearest-neighbor method

Let us reconsider the first part of Eq. (3)

$$\hat{f}(x/C_i) = \frac{k_i(x)/n_i}{V_h} .$$

This relation encompasses two nondetermined terms, with a common structural link: $k_i(x)$ and V_h (the term n_i is known from the learning set). Two attitudes may be adopted for the calculation of $\hat{f}(x/C_i)$: either we fix the volume V_h and we count the number $k_i(x)$ of examples belonging to this volume, or we fix the value and then adapt the volume V_h so that it contains exactly $k_i(x)$ examples. The first approach is that adopted above, the second is the base of the k-Nearest-Neighbor technique, or k-NN. This method implements an intuitive idea, which consists in assigning to x the most represented class among the k nearest neighbors of x. As h above, the k parameter plays a very delicate role. Too weak a value for k induces a classification function which is too specific of the learning sample. Too high a value of k will tend to make the classification function uniform; this function will then retain the most frequent class. It should be pointed out that when the size of the learning sample increases indefinitely, and if we take a value of k in $O(1/\sqrt{n})$, the k-NN assignment rule converges towards the Bayes Decision Rule.

In waveform processing, we applied the k-NN method for several values of k (1, 30, 75 and 100), using the Euclidean distance. Best results were obtained when k is around 30, the value retained in Table 2.4. The CPU time is that of the SAS software on Sparc 2. As in the previous case, the size is that of the data.

Table 2.4. Nonparametric approach, k-nearest-neighbor method.

Data	Train	Test	Test(1)	Test(2)	Test(3)	CPUtrain	CPUtest	Size
Binary	0%	23.3% (2.1%)	46.0%	10.6%	13.2%	0″	232″	300 × 21 = 6300
Ternary	0%	20.4% (1.7%)	42.5%	9.9%	8.9%	0″	245″	300 × 21 = 6300
Contin.	0%	18.3% (1.7%)	31.8%	10.3%	12.9%	0″	258″	300 × 21 = 6300

2.4. Discussion

The error rates of both nonparametric methods used are fairly close, and roughly equivalent to that of linear discrimination (Table 2.1). This may be explained by the characteristics of the problem in which the decision surfaces are quasilinear (Sec. 1.5), so that the Gaussian assumption does not appear as a handicap. Different results could be obtained with another problem. It has to be underlined that both nonparametric methods obtained unbalanced error rates among the three classes. Examples from the first class are often poorly predicted. In real-world

applications, such a characteristic would usually be considered a heavy flaw. Moreover, from a practical point of view, one important element of these methods is that they do not conduct *a priori* learning phase. Therefore, their learning time and explanation power are null, yet the decision time is relatively long. Nevertheless, various pre-processing techniques can be performed to reduce the complexity of nearest neighbors based recognition.[13] In this case, the learning time is not null and the recognition time is cut down. The first solution consists in reducing the sample size while extracting prototype examples.[7,29] In this case, we speak of the "condensed" nearest neighbor rule. We performed experiments using the edition method[74] which consists in removing from the learning sample the examples which are poorly classified during a first application of the standard k-NN method. Our results with binary data show that the performance is slightly improved (23.0%) while the number of retained, prototype examples is greatly reduced (56 in average, instead of 300) as is the decision time. Other methods structure the learning sample and organize the search. Several solutions exist, which are usually based on trees[20] and which make it possible, in some cases, to find nearest neighbors in approximately constant average time.[71] However, the performance of these latter methods degrades rapidly with the dimension of the representation space. They could hardly be used to deal with the waveform problem, unless preprocessing the data by reducing the representation space dimensionality (Ref. 15, pp. 246–248).

3. NEURAL NETWORKS

3.1. The Multi-Layer Perceptron Model

Neural Networks include a wide range of models which differ in functional form, the classes of functions approximated, the criteria optimized and the learning algorithms. Generally, learning consists in estimating the value of numerical quantities, the *weights*, characterizing the model, from a learning set of patterns. Supervised classification is one of the favourite applications of many of those models. We refer to Hertz *et al.*[31] for a more detailed presentation of Neural Networks.

We used the Multi-Layer Perceptron (MLP) model[57] for our tests. It is one of the most common and simplest nonlinear network models. MLP's are nonparametric systems as defined in Sec. 2. Like all networks, an MLP is a combination of basic elements called cells. These are computational units which receive input data from $\mathbf{R}^p$, and produce a real output in $\mathbf{R}$. The *transfer function* characterizing such a unit has the following form:

$$y = f(w_0 + \Sigma_j w_j x_j)\,.$$

The parameters $w = (w_1, \ldots, w_p)$ are the weights of the cell, w_0 is the bias, $x = (x_1, \ldots, x_p)$ is the *input* to the cell and y its *computed output*. In the basic MLP model, the *activation function* f is usually defined as follows:

$$f(u) = \frac{e^{Ku} - e^{-Ku}}{e^{Ku} + e^{-Ku}}\,.$$

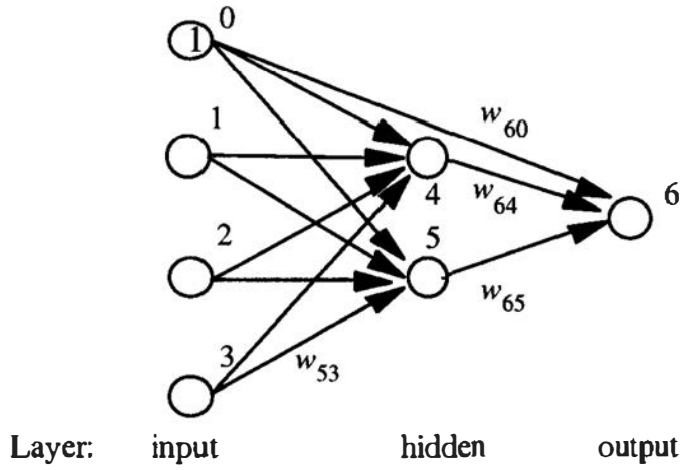


Fig. 3. A three-layer perceptron.

This function was initially inspired by McCulloch and Pitts'[40] formal model of the neuron.

In a basic MLP, the units are arranged in successive layers with connections between layers. Data is sent to the input layer for a copy and is afterwards processed sequentially by the successive *hidden layers*. Cells belonging to the same layer compute in parallel, the outputs of the cells of layer m being the input to layer $m+1$. The last layer provides the answer of the system and is called the *output layer*. We only consider systems with a single hidden layer. Figure 3 depicts a three-layers MLP. A bias cell (index 0) enables the bias terms W_{i0} to be introduced. It is permanently set to 1 and is connected to all cells in the subsequent layers. The composition of elementary transfer functions of the different cells is called the *global transition function* of the network. This function, ϕ, is defined from $\mathbf{R}^p$ to $\mathbf{R}^q$ and can be written for the ith output by combining the local activation functions of the different units:

$$y_i = \phi_i(x) = f\left(w_{i0} + \sum_j w_{ij} f(w_{j0} + \Sigma_k w_{jk} x_k)\right),$$

where j indexes the hidden units and k the input units. ϕ being a combination of elementary nonlinear functions, its complexity may be adjusted by varying the number of hidden cells.

3.2. The Learning Algorithm

Let us consider a network with a fixed architecture. Its transition function is then defined by the value of the connection weights. The difference between the desired outputs and the outputs computed by this model, or the matching of the model to the data, is characterized by a cost function Q. Learning consists in minimizing this function by adjusting parameters of the model. One of the most common cost functions is the quadratic error, i.e. the square of the Euclidean distance between desired and computed outputs. This is the function we have used here. Usually,

the desired outputs for a classification task are indicators of classes (i.e. if $x \in C_i$, all components of y are equal to 0 except the ith which is equal to 1).

Classic algorithms for MLP are based on *gradient* techniques. The basic version described below is known as the *steepest descent method.* Let us consider a given network. Starting from an initial configuration, a gradient algorithm will modify the values of the parameters by successive adjustments which aim at minimizing the error criterion Q according to the rule:

$$w = w - \varepsilon \partial Q / \partial w \,,$$

where ε is the *learning rate.* It monitors the amplitude of the modifications and can be fixed or variable during the algorithm. It is set to 0.01 in the experiments described below. The classical learning algorithm for MLP is called *back-propagation.*[57] It constitutes an implementation of an adaptive gradient algorithm on a MLP.[73] Other, more sophisticated learning algorithms were proposed[67] which are mainly based on second order minimization techniques,[2] or on the conjugate gradient principle.[44] These algorithms tend to converge faster than the back-propagation, and usually find apparently better parameter values, according to the cost function Q measured on the learning set. Impressive differences between the back-propagation and a quasi-Newton approach have been reported by Chung and Setiono[8] using artificial data. However, this type of result is generally not observed with real world problems, and the improvement obtained on the learning set is rarely confirmed on the test set. Moreover, the convergence speed of these algorithms often makes it difficult to use *early-stopping* which consists in stopping the algorithm before convergence, and which is one of the most efficient procedures to avoid overfitting with neural methods. For example, with waveforms, we have tested the conjugate gradient procedure[44]; the computation time was divided by between 2 and 5 depending on the number of iterations, but we were unable to reach the same level of classification accuracy.

3.3. The Classification Rule

The Classification Rule used for MLP consists in assigning a description to the class identified by the maximal computed output. The ith output of the MLP is, in some respects, an estimate of $\Pr(C_i/x)$. This quantity is thus directly estimated here, unlike the statistical methods introduced in Sec. 2 which estimate the conditional densities of the data.

3.4. Results

Data were normalized per component for each training and test set. Two series of experiments were run, the first on a network without hidden layer, the second on a network having an hidden layer of five units. Invariably, the output cells are sigmoïd transfer functions. The Bayes Decision Rule being quasi-linear (refer to Sec. 1.5), we did not test more complex networks. Moreover, we tested two algorithms, the

standard backpropagation and Möller's[44] conjugate gradient. The results obtained with backpropagation for both networks are given in Tables 3.1 and 3.2. In the size column, we indicate the number of weights of the corresponding network. The CPU time is approximate and was measured on a Sparc 10. The performance obtained for continuous data is almost optimal (i.e. close to 14%). The results obtained for ternary data are also similar. Note that whatever the data set, both systems give equivalent performances, despite a slight overparameterization of the second. When using the conjugate gradient algorithm with five hidden units, the performance is a little worse (18.2% with continuous data) which indicates a slight overfitting, but the running time is about 6 times faster. Besides, early stopping appears necessary with this learning algorithm, but it necessitates a fine tuning which is not that easy to achieve when disposing of only 300 learning examples.

Table 3.1. Two-layer perceptron.

Data	Train	Test	Test(1)	Test(2)	Test(3)	CPUtrain	CPUtest	Size
Binary	14.93% (1.59%)	20.87% (0.61%)	25.42%	18.27%	18.86%	240″	20″	66
Ternary	12.23% (1.55%)	18.09% (0.68%)	23.71%	15.31%	15.17%	240″	20″	66
Contin.	10.8% (1.05%)	17.31% (0.82%)	20.49%	16.35%	15.06%	240″	20″	66

Table 3.2. Three-layer perceptron with five hidden units.

Data	Train	Test	Test(1)	Test(2)	Test(3)	CPUtrain	CPUtest	Size
Binary	12.5% (1.38%)	21.26% (0.90%)	25.61%	18.52%	19.60%	300″	20″	128
Ternary	9.6% (1.09%)	18.79% (0.87%)	23.21%	17.13%	18.79%	300″	20″	128
Contin.	7.8% (1.33%)	17.15% (0.92%)	21.8%	14.16%	15.44%	300″	20″	128

Network performance is among the best we obtained in this study. This might be a little surprising, since these methods have not used their nonlinear potential here. The comparison with statistical methods is particularly interesting (refer to Sec. 2). Although the optimal surfaces are quasi-linear, linear discriminant analysis leads to poorer results. The explanation might lie in the fact that data significantly deviate from the Gaussian model, and the quasi-linear optimal surfaces are only imperfectly approximated by the analytical solution (2) derived from the Gaussian hypothesis. Moreover, the sigmoïd shape of the transfer function used on the output

units enables us to concentrate on the descriptions located near the boundaries, thus allowing more precise learning of the latter.

4. TREE-BASED CLASSIFICATION, THE CART METHOD

4.1. Overview

Tree-based methods are used to construct classification functions which can be represented by a decision tree. These methods are well-known and widely employed in both statistical Pattern Recognition and Machine Learning. Since the seminal article of Sonquist and Morgan,[64] an impressive amount of research on tree-based classification has been conducted in both fields, among which we shall only cite here those of Breiman *et al.*[4] and Quinlan.[50]

Classification trees are usually binary, and can be represented as shown in Fig. 4. The circular nodes are decision nodes and the square nodes are terminal nodes. Each decision node has a *binary question* associated with it, and each terminal node has a class C_i associated with it. In our example of Fig. 4, there are two classes which are respectively *Ill* and *Well*, while the binary questions are based on the attributes *Temperature* which is continuous, *Throat-irritation* which is binary and *Cough* ($\in$ {*none, dry, loose*}) which is qualitative. The tree classifies a description x through a chain of binary decisions. Starting at the root node and proceeding down the tree, tests are conducted using questions to determine whether the description goes to the left or right descendant. The description is then assigned to the class of the terminal node in which it lands.

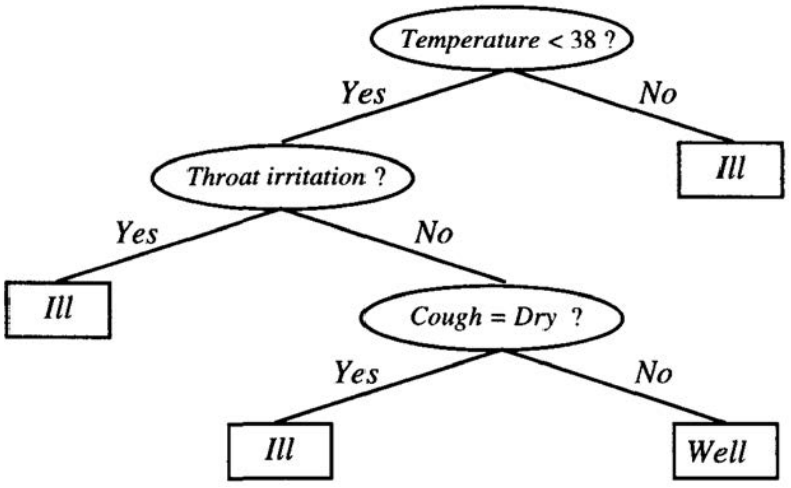


Fig. 4. A binary classification tree.

A tree path from root to leaf constitutes a production (or decision) rule similar to those of expert-systems of the MYCIN type.[63] For instance, the following rule may be extracted from the tree of Fig. 4

$$[\textit{Temperature} < 38] \text{ and } [\textit{Throat-irritation} = \textit{Yes}] \rightarrow \textit{Ill}.$$

This demonstrates the explanation power of the approach, thus the reasons for its success in Artificial Intelligence. The rules conceived are easy to interpret as they are written using description language that the user has defined to describe data.

Given a learning set, most approaches to classification tree design determine the binary questions in a stepwise top-down fashion: the training set E is associated to the root of the tree; a binary question is chosen which splits E into two subsets; this splitting process is repeated for both subsets, for their descendants and so on. Now, there are three basic issues in classification tree design:

(1) selecting an appropriate binary question for each decision node;
(2) determining an appropriate set of terminal node; and
(3) selecting an appropriate class for each terminal node.

For each decision node, the binary question is usually selected by optimizing a splitting criterion among a set of possible questions. Frequently, these questions are based on the use of a single description attribute, as in Fig. 4, but they may also be based on linear[4] or neural net[26] combinations of these attributes. The terminal node set is usually determined by halting the splitting by subject to some stopping criterion, or by continuing the splitting until all terminal nodes have pure class membership and then pruning back. Pruning is the more recent approach and has better properties. Finally, the class for the terminal nodes is obtained using the majority rule, or a weighted majority rule in the case of nonuniform misclassification cost. In the following, we shall briefly describe the popular CART method which was the first to propose the pruning approach. More details are given in the book of Breiman *et al.*[4]

4.2. The CART Method

4.2.1. The set of possible binary questions

In the standard use of CART, each question is based on a single attribute, and not on some (e.g. linear) combination of the initial attributes. The form of these binary questions depends on the type of associated attributes. A binary attribute obviously generates a single binary question. A qualitative attribute taking m values generates $2^{m-1} - 1$ binary questions which correspond to the nonempty bipartitions of its values, e.g. the attribute *Cough* defined above generates the questions: "$Cough = None$", "$Cough = Dry$?" and "$Cough = Loose$" which correspond to the bipartitions: $\{None\}|\{Dry, Loose\}$, $\{Dry\}|\{None, Loose\}$ and $\{Loose\}|\{None, Dry\}$. A continuous attribute x_j generates questions having the shape "$x_j < v$?". There is *a priori* an infinite number of possible values for v, and CART only considers those defined by the equation $v = (v_i + v_{i+1})/2$, where the v_is are the consecutive values taken by the attribute on the examples attached to the node to be split. Thus, if this node contains m examples, $(m - 1)$ binary questions are envisaged for each continuous attribute. Finally, ordered attributes, such as $size \in \{small, medium, big\}$, are dealt with in the same way as continuous ones.

4.2.2. Tree growing and splitting criterion

In the tree growing phase, a large tree, T, is grown by recursively finding binary questions until all terminal nodes have a pure or nearly pure class membership or cannot be split further. Questions are chosen among the set of possible questions by optimizing a splitting criterion. The criterion retained by CART is based on the Gini index. Let t be the node to be split, E be the set of training examples attached to t, and $\Pr(C_i/E)$ be the proportion of examples from E which belong to class C_i. The Gini index defines the *impurity* of E as

$$i(E) = \sum_{i \neq j} \Pr(C_i/E)\Pr(C_j/E)\,.$$

This impurity is maximum when for every index i we have $\Pr(C_i/E) = 1/g$ (g is the number of classes), and is minimum ($= 0$) when all examples from E are in the same class. The aim is to find the binary question which most reduces the impurity of E. Let us consider a given binary question B, E_L^B the subset of E corresponding to the answer *Yes* to B, and E_R^B the subset corresponding to *No* (L stands for left and R for right). The reduction of impurity brought by B is defined by

$$\Delta(E,B) = i(E) - \Pr(E_L^B/E)i(E_L^B) - \Pr(E_R^B/E)i(E_R^B)\,,$$

where $\Pr(E_L^B/E)$ and $\Pr(E_R^B/E)$ are the proportions of examples from E which are respectively in E_L^B, and in E_R^B. Finally, the binary question chosen for splitting t is the one which maximizes this criterion.

4.2.3. Tree pruning

In the tree pruning phase the large tree, T, is pruned back to avoid overfitting the training data. This process consists in removing some branches of T which do not significantly improve the error rate, but which make its size (or complexity) high. A pruned subtree is selected by minimizing an error rate estimate over a parametric family of pruned subtrees. This family is generated as follows. Suppose each node in T is assigned a class C_i based on majority vote. Then we may compute the error rate on the training sample of any pruned subtree S of T. Let $R(S)$ be this *resubstitution* error estimate. Now define the *error-complexity* of a pruned subtree $S(\subset T)$ by

$$R_\alpha(S) = R(S) + \alpha|S|\,,$$

where $\alpha \geq 0$ and $|S|$ is the number of terminal nodes in S. The desired family of pruned subtrees $T_\alpha(\alpha \geq 0)$ is obtained by minimizing the error-complexity criterion for each fixed value of α: $R_\alpha(T_\alpha) = \min_{S \subset T} R_\alpha(S)$. Note that $T_0 = T$ and $T_\alpha = root(T)$ when α is large enough. From a computational stand point, it is not easy to obtain T_α for any given value of α, so that CART uses an algorithm which starts from $T_0(= T)$ and, by iterative pruning of "weak" branches, produces a sequence of smaller and smaller subtrees until $root(T)$ is reached. Moreover, it may be shown that this sequence, (T_{α_i}), is a subset of the family (T_α). From this sequence,

a pruned subtree, T_{α^*}, is then selected by minimizing and "honest" estimate, $\hat{R}$, of the probability of error

$$\hat{R}(T_{\alpha^*}) = \min_{\alpha_i} \hat{R}(T_{\alpha_i}) .$$

CART can use several methods to obtain honest estimates of the error probability. The simplest approach is to use an independent test sample. But this approach does not allow all of the training data to be used for both growing and pruning the tree, so that it is precluded when the number of training examples is low. In this case, the usual approach employed in this study is based on a cross-validation estimate of the error probability.

4.3. Results

Results obtained by CART 1.1 using the default options are displayed in Table 4. The (average) tree size is measured by the number of edges, and the CPU time is obtained on a Sparc 2. Among the methods experimented in this study, classification trees have one of the best explanation powers. These classification functions, however, are also endowed with one of the poorest discrimination powers, which illustrates the dilemma explanation/discrimination power. It is a recognized fact that the waveform problem is difficult for classification trees. The use of a more appropriate coding such as that used in (Sec. 9), enables tree results to come very close to those obtained by other methods. Moreover, analogous results may be obtained by using linear combination of attributes in the decision nodes, instead of single attributes (for more details see Ref. 4). The results obtained here do not therefore prejudice classification tree results on other applications.

Table 4. Tree-based approach, the CART method.

Data	Train	Test	Test(1)	Test(2)	Test(3)	CPUtrain	CPUtest	Size
Binary	22.9% (1.0%)	29.9% (2.2%)	34.6%	29.1%	26.0%	29″	26″	17.6
Ternary	22.5% (0.7%)	30.5% (1.63%)	33.37%	31.22%	26.74%	33″	17″	12.6
Contin.	19.8% (1.1%)	31.1% (1.2%)	29.7%	32.1%	31.5%	60″	34″	7.4

5. FUZZY CLASSIFICATION TREES

5.1. Overview

We are primarily concerned here with the problem of the construction of classification trees when (some) attributes are continuous. With such data, usual symbolic classification trees do not give excellent results. The key point is the discretization of these attributes, i.e. the partitioning of an infinite number of values into a finite

number of subsets of their domain. Most of the known methods to discretize continuous attributes are used prior to the construction of the tree.[6,16] This may be done for several reasons, e.g. to reduce the running time. Our method, like CART described above (Sec. 4), proceeds dynamically by adjusting the thresholds at each tree's node according to the attached examples. However, we have observed that, if we split the domain of an attribute into two subsets at a given threshold, we obtain imprecise tested values near this threshold, because all possible values of the attribute are not present in the training examples. The solution we propose is to use *membership degrees* for examples with values near the threshold, establishing a gradual split of the values. During tree construction, we employ a fuzzy measure of information[53] to discretize and to compare the continuous attributes. To classify a new example, we aggregate membership degrees obtained by a given example on a path of the tree, using the fuzzy set theory.[76]

5.2. A Dynamically and Fuzzy Entropy-Driven Discretization

5.2.1. Splitting criterion

We must choose the best attribute that will be used to split into subsets the set of training examples E attached to a given node t. A cost function is used to determine this choice, which is based on a fuzzy measure of entropy. Let us use the same notation as in Sec. 4.2. The classical Shannon entropy has, in some respects, a meaning similar to that of the Gini index (Sec. 4.2), and is defined by

$$e(E) = \sum_i -\Pr(C_i/E) \log(\Pr(C_i/E)) .$$

In our model, classical sets are replaced by fuzzy sets and we have to define a fuzzy probability. Let $\Omega = \{\omega_1, \ldots, \omega_r\}$ be a set of events, each one associated with a probability $\Pr(\omega_i)$. Let F be a fuzzy set defined on Ω with membership function f_F. The *fuzzy probability* of the fuzzy set F is

$$\Pr^*(F) = \sum_{i=1}^{r} f_F(\omega_i) \cdot \Pr(\omega_i) .$$

Using this notion, the classical entropy can be extended to a *fuzzy measure of entropy*

$$e^*(E) = \sum_i -\Pr^*(C_i/E) \log(\Pr^*(C_i/E)) .$$

Let us now consider a noncontinuous attribute j, associated with the sparse domain $X_j = \{v_1, v_2, \ldots, v_k\}$. This attribute splits E into k subsets E_l whose conditional probabilities are denoted as $\Pr(E_l/E)$. The classical information of this split (or the measure of conditional entropy) is defined as

$$I(j) = \sum_i \Pr(E_l/E) e(E_l) .$$

A fuzzy measure of information, the *entropy-star measure*, can also be defined in the same way. If $F_1, F_2, \ldots, F_k$ are fuzzy sets of E, we now have

$$I^*(F_1, F_2, \ldots, F_k) = \sum_{i=1}^{k} \Pr^*(F_i) e^*(F_i) .$$

When splitting E by means of the attribute j, the information gain represents the decrease of uncertainty on the classes C_i entailed by the use of j

$$\Delta(j) = e(E) - I(j)$$

or, with the fuzzy quantities

$$\Delta^*(j) = e^*(E) - I^*(j) .$$

To split the training set, we look for convenient fuzzy sets $F_1, F_2, \ldots, F_k$ for each continuous attribute j, and this search will be explained in the next section. To construct the tree, we choose the attribute with the highest fuzzy information gain $\Delta^*(j)$. Then, the best attribute corresponds to the minimum of $I^*(j)$.

5.2.2. Fuzzy set determination and threshold computation

To construct the fuzzy subsets of E attached to a given continuous attribute j, we build a partition of X_j into fuzzy sets which is satisfying for the identification of classes. In order to define these fuzzy sets, we search for relatively homogeneous clusters of X_j with respect to the distribution of classes. Each of these clusters will represent the kernel of a fuzzy subset of the partition.

To find such clusters, we use a technique from the mathematical morphology theory.[10,39,61] This technique is generally used in image analysis to smooth and to filter spot noise. It lies on two operators: *erosion* and *dilatation*. Erosion enables the destruction of small heterogeneous regions, while dilation enlarges and brings together homogeneous zones. Here, the training set E defines a two-dimensional shape for each attribute. The first dimension is the set $S_j (\subset X_j)$ of values occurring in E, and the second dimension is the set of classes. For a given attribute, E may be transformed into a word whose alphabet is the set of classes. For example, when E equals {(1.7, C_1), (2.4, C_2), (2.9, C_1), (3.3, C_1), (3.7, C_2), (3.8, C_3), (4.0, C_1), (4.2, C_2), (4.3, C_2)}, the corresponding word is $C_1C_2C_1C_1C_2C_3C_1C_2C_2$. Dilatation and erosion are applied to this word to obtain a clustered form. Each of these operators is implemented as a transducer which expresses rewriting rules, such as: change a letter L into U when it is surrounded by two letters different from L. In this case, the special letter U means "uncertainty" and indicates the places where the classes are highly mixed. For instance, with the previous word we would obtain $C_1C_1C_1UUUC_2C_2$.

Except the uncertain one, each of these clusters represents a collection of values from S_j which, in the majority of cases, belong to the same class. Let the two biggest clusters, other than the uncertain one, be expressed as

$$[\underline{l}, \bar{l}] \quad \text{and} \quad [\underline{h}, \bar{h}] \quad \text{with } \bar{l} < \underline{h} .$$

From these clusters, we may now define the two fuzzy sets F_l and F_h whose kernels are $K_l = [-\infty, \bar{l}]$ and $K_h = [\underline{h}, +\infty]$, and whose supports are respectively $[-\infty, \underline{h}]$ and $[\bar{l}, +\infty]$. To define the membership function of F_l and F_h, we compute a threshold, denoted as v_j, using the barycentric expression

The membership functions of F_l and F_h are then defined as shown in Fig. 5. Finally, we can compute $I^*(F_l, F_h)$ to find the most relevant attribute. Moreover, two training subsets of E are obtained from the split induced by v_j, and used for further tree expansion. This expansion is performed by adding new attributes until the measure of entropy of the local training subset is below a fixed value.

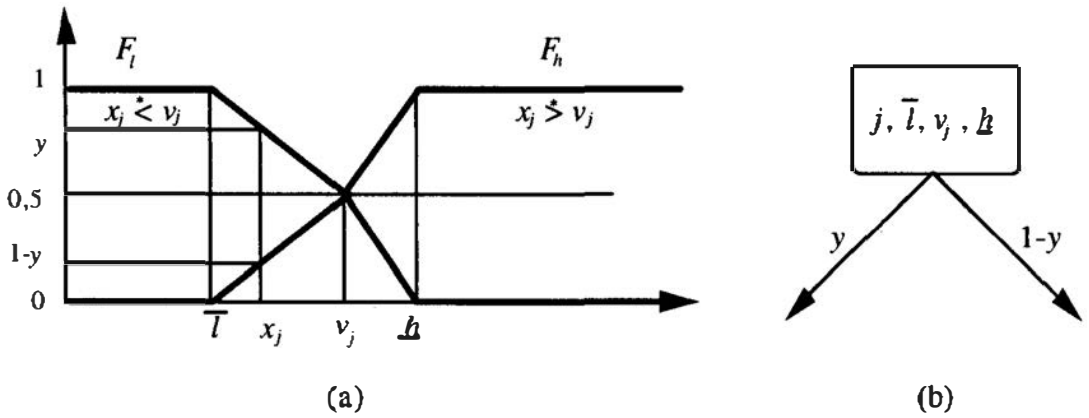


Fig. 5. (a) Membership functions associated with the fuzzy events $\{x_j < v_j\}$ and $\{x_j > v_j\}$; (b) Use of these functions in a tree node.

The dilatation and erosion operators are used a fixed number of times, so that the discretization's complexity remains linear (once the attribute values have been sorted). It follows that the complexity of the whole algorithm is the same as CART's or ID3's, i.e. $O(pnd)$ where d is the depth of the tree.[66] If the tree is well-balanced, we have $d = O(\log(n))$, and the time complexity is $O(pn\log(n))$. This is also the complexity needed for sorting the attribute values, prior to the algorithm runs.

5.3. Classification Function

Generally, when a classification tree is used to classify a new example, the values of this example are compared with the computed thresholds. However, a threshold is not generally a value which really occurs in the training set. It is imprecise in nature because there is no precise information about the values between $\bar{l}$ and $\underline{h}$. Restricting the descent to a single branch of the tree for these values may not be very efficient. It is better to enlarge the threshold to an interval. An estimate probability may then be associated with the outcome of this test.[51] In our approach, we use the values found during discretization as boundaries for the imprecise interval, and a graduality of membership to a path for the values near the threshold. At a given tree node, the description to be classified is associated with each of the edges issuing

from the node, with a degree equal to its degree of membership to the fuzzy subsets associated with that edge, as defined in Fig. 5. A path to a leaf is then associated with a collection of degrees and we can aggregate these membership degrees by means of fuzzy aggregation operators.

A path from the root to a leaf is equivalent to a rule "If (and $P_1 P_2 \ldots P_k$) Then C_i", where the premises correspond to the fuzzy events associated with the edges of the path, and where the conclusion is the class attached to the leaf. In order to aggregate the premises, we use a conjunctive operator, and to aggregate the conclusions of all the rules corresponding to a given class, we use a disjunctive operator from fuzzy set theory. More specifically, in order to achieve the intersection of the premises, we use the usual operator known as *triangular norm* (t-norm for short), and to achieve the union of the conclusions, we use a *triangular conorm* (*t-conorm* for short). In this study, we employed the t-norm and the t-conorm defined by Zadeh,[76] which are respectively the *minimum* and the *maximum*. The final decision is the class which has the highest membership degree. We can foresee other uses for these membership degrees which take into account all the degrees found on the paths. For example, a classification into continuous classes can be made by computation of a barycentric value from all the obtained classes, or, in a fuzzy framework, it would be useful to preserve the decision as a fuzzy subset of the set of classes, with the obtained degree.

5.4. Results

Our approach is essentially concerned with the problem of the construction of classification trees when continuous values occur, and we present here our results on the continuous data sets only (Table 5). The average size of the trees is the number of edges. The CPU time is indicated for a program written in C, running on a Sun station Sparc 10. We observe that the results are slightly enhanced by the use of fuzzy trees compared to results obtained by CART which considers crisp (non fuzzy) tests for the thresholds. However, the average sizes of CART trees and fuzzy trees are not the same, so that further studies would be needed to obtain a more precise comparison.

Table 5. Fuzzy classification trees.

Data	Train	Test	Test(1)	Test(2)	Test(3)	CPUtrain	CPUtest	Size
Contin.	13.1% (1.56%)	29.96% (1.25%)	34.7%	25.9%	27.1%	5″	54″	26.4

6. EMPTY MONOMIALS

6.1. Method Overview

This method builds discriminant functions between classes, the elements of which are described with binary attributes. These functions are conjunctions of Boolean

variables. The conjunctions that we use are selected because they never appear in one specific class, but are attested in the other classes a minimum number of times. So these conjunctions are characteristic elements of non-membership of a given class C_i, i.e. characteristic of "not-C_i".

We shall first describe our method for a problem with only two classes C_+ and C_-, represented by positive and negative examples. These belong to the sets E_+ and E_- whose union constitutes the learning set E. A new description is classified into C_+ (resp. C_-) according to the number of characteristic rules of not-C_- (resp. not-C_+) it satisfies. We can measure two error rates, one on the learning set, the other on the test set. On the learning set, there may be no error, but there might be indecision, since it is possible to find an element of E that satisfies no rules. On the test set, errors are obviously possible. Moreover, some uncertainty may occur, either because an example does not satisfy any characteristic features of C_+ and C_-, or because it possesses the same quantity of both.

In the Boolean framework, a conjunction of attributes (generators of Boolean algebra) with values 0 or 1 is a *monomial*, and monomials are conjunctions of *literals*. On the set E, a monomial *covers* some elements, those having the same values as those of the monomial attributes. If there is no element of E to present this conjunction of literals, this monomial is said to be *empty* on E. The empty monomials of E_+ (resp. E_-) indicate not-C_+ (resp. not-C_-). To build our discriminant functions, we first enumerate all the empty monomials of E_+, then those of E_-, and we only keep monomials that cover at least q elements of E_- (resp. E_+), q being a parameter. Moreover, we only consider empty monomials with a minimal length, since an empty conjunction lengthened with other literals stays empty.

The building of empty monomials has been studied for a long time, because it is connected with the minimization of Boolean function problems. Let F be a Boolean function given as a disjunction of complete monomials (disjunctive normal form), or equivalently as a T Boolean array (with p attributes, T has p columns). To minimize F we look for all its prime implicants that are the empty monomials of the complementary function $\bar{F}$. To solve this very important problem for circuit design,[72] numerous algorithms have been proposed (Karnaugh, Mc Cluskeyand Quine, cf. Ref. 19) that start from $\bar{T}$. This is not very practical in our framework, since $\bar{T}$ is the complementary array of T in $\{0,1\}^p$. Therefore, we have chosen Kuntzmann's[36] algorithm, which works directly on T and can be adapted, particularly when the number of literals must be bounded, or when only positive forms of attributes are required.

Let us suppose that we are building the empty monomials of E_+ (the procedure is the same for E_-). The algorithm is sequential, and it builds the successive lists of empty monomials. After examining the i first elements of E_+ the resulting list is denoted as L_i. Let e_i be an element of E_+ and S_i be the set of the complementary forms of its literals. For instance, if $e_i = x_1\overline{x_2}x_3x_4$ we have $S_i = \{\overline{x_1}, x_2, \overline{x_3}, \overline{x_4}\}$. The first list L_1 is initialized with S_1. Each step corresponds to the examination of a new element e_i and produces the list L_i from list L_{i-1}. When the whole set E_+ has been check, the resulting list contains all the empty monomials of E_+.

This algorithm is based on the following proposition:

Let $\lambda \in S_i$ and $\mu \in L_{i-1}$,
(1) If μ contains λ then $\mu \in L_i$;
(2) If μ does not contain $\bar{\lambda}$, then the conjunction $\mu\lambda$ belongs to L_i.

The proof is easy. If μ is an empty monomial that contains λ, as $\lambda(\in S_i)$ is not in e_i, μ is still empty. If μ contains $\bar{\lambda}$, $\mu\lambda$ contains $\lambda\bar{\lambda}$, and it is useless to add it to L_i. Finally, if μ contains neither λ nor $\bar{\lambda}$, $\mu\lambda$ is empty since μ was empty and λ is not in e_i. The Kuntzmann's algorithm is the iterative application of this proposition, using the rules (1) and (2) in a specific order:

L_1 is initialized with S_1
For $i = 2$ to $|E_+|$
  For each literal $\lambda \in S_i$
    { Copy in L_i all the monomials of L_{i-1} that contain λ
      and delete them from L_{i-1};
      If one of them is equal to λ, λ is removed from S_i }
  For each literal $\lambda \in S_i$ and any monomial $\mu \in L_{i-1}$
    { When $\bar{\lambda} \notin \mu$ and $\mu\lambda \notin L_i$, add $\mu\lambda$ to L_i }
End of For i

The complexity of this algorithm depends on the length of empty monomial lists, that cannot be predicted. In the worst case, this length is exponential in the number of attributes, so it cannot be used for large size problems. To get L_i from list L_{i-1}, either we copy monomials remaining empty or we enlarge them with one literal in S_i. So it is very simple to generate only monomials having a bounded number of literals, which is attractive in our context. Moreover, the number of monomials having a bounded length $l_{\max}$ is polynomial, and consequently the enumeration procedure has a worst case complexity $O(p^{l_{\max}})$.

6.2. Application to the Waveform Problem

We applied the method described above to the binary description of the waveforms. We shall now explain how we adapted this method, and how we choose the parameter values using the original data of Breiman *et al.*[4]

For each class of the learning set compared to the union of the others, we enumerate its empty monomials having at most k literals, or k-monomials. Then, for any description to be classified, we count the number of monomials of each class covering this example. The greater the quantity for a class, the smaller the chance of belonging to it. Thus, we have three scores corresponding to the classes, and we assign a description to the one which has the smallest score, i.e. the class having the minimum number of empty monomials covering this example. When considering a learning example, there is at least one class with a score equal to 0, but there may be several. In that case, there is indecision. For a test example, there may be two or three scores obtaining the minimum value. If the actual class does not give a minimum score, we count an error, and so indecisions are only between alternatives,

one of them being the correct class. If there are two tied classes, the third one is discarded, and this becomes a double indecision; if there are three, we have a triple indecision.

First, we tried empty 2-monomials. Among the 300 instances of the learning set, there are 162 double indecisions and 2 triples. This means that there are only 136 elements that are (correctly) classified. This great rate of uncertainty (55%) led us to consider empty monomials with length 3. There are respectively 567, 604, 564 for the three classes. Now, 48 double indecisions and no triple remain, which corresponds to 84% of correctly classified elements. Then, we discarded monomials covering less than 10% of the negative examples, i.e. less than about 20 examples. This selection criterion gives 265 monomials for class C_1, and we took the same number for the other classes, retaining monomials with the best covering rate. For the learning set, there are now 56 double indecisions, no triple and 244 waveforms correctly classified. For the test set that contains 5000 waveforms, the number of double indecisions is 858, corresponding to 17% and there is no triple. The number of errors is 681, and the number of elements correctly classified is 3461. Uncertainty can be treated according to two options:

- Either we do not take a decision, and then there are only 4142 classified elements. Consequently, the error rate is 16.5%;
- Or we toss up to decide the class when there is indecision, and the decision will be correct one time out of two. In that case, there will be 681 + 429 errors, corresponding to an error rate of 22%.

6.3. Results

For the 11 learning sets, we limited the number of selected monomials to 250. Sometimes, all of them do not cover 10% of the elements of the other classes, and often there are less than 250. When there are more than 250, we keep those having the greatest covering rate. The average number of selected monomials is 225 per class, most of them having length 3. On average, there are 1094 double indecisions, that is 22%, which are not uniformly distributed; there are less for class C_2 than for class C_1 or C_3. The average number of errors is 625 corresponding to 13% of the waveforms. If we insist on classifying the whole test set, tossing up for a decision, there will be 547 new errors which will give 23.4% on the whole set. The CPU time corresponds to a program written in Basic and running on a Macintosh Power Book 165. The size is the number of monomials multiplied by their length. Results are given in Table 6.

Table 6. Empty monomials.

Data	Train	Test	Test(1)	Test(2)	Test(3)	CPUtrain	CPUtest	Size
Binary	11% (0.7%)	23.4% (0.3%)	25%	21%	24%	330″	444″	2025

It seems clear that the quality of the decisions depends on the number of selected empty monomials; we use a large number, because below the threshold of 100 the uncertainty number increases very fast (greater than 1500). So, considering the size of the classification function, it is a poor method. Even if we were to design a selection strategy to reduce the number of monomials, keeping the same covering rate, we would never reach the efficiency of the Decision Committees method (Sec. 7), for example. Finally, the way the empty monomials are built seems to be the most interesting point. It is more efficient than the enumeration of all the monomials which come first, and then selecting those having a large covering rate in one class and a small value in the others. But, in doing so, monomials that cover only a few elements in a class and that occur frequently in another, are missing, although these monomials could be good indicators of class membership.

7. DECISION COMMITTEES

7.1. Introduction

Decision committees use expert-system-like rules. They try to associate the explanatory character of classification trees with an additive combination of the rule, in the same way as MYCIN-like expert systems.[63] The decision is taken on the basis of a set of fired rules, and not on the basis of a single rule (or a path from the root to a leaf) as in classification trees. This additivity exists in linear discrimination and in neural networks, and partially explains the performance of these approaches which are often better than those based on classification trees. Our aim is to find small-sized decision committees. Even though they do not constitute a radically new type of classification procedure (Refs. 3 and 52, Sec. 6), decision committees have given rise to few studies, especially in the form presented here. For more details, further theoretical and experimental results, the reader shall refer to Ref. 48.

7.1.1. Notations and definitions

We shall consider here that examples are described by binary attributes, these being possibly derived from nonbinary ones (qualitative, ordinal, ...) by following a classical discretization procedure (Ref. 4, Secs. 1.5 and 4.2). Literals associated with variables are denoted as $x_1, \overline{x_1}, x_2, \overline{x_2}, \ldots, x_p$ and $\overline{x_p}$.

A decision committee consists of a set of rules $\{(t_i, v_i)\}$ where each t_i (the condition part) is a monomial (or conjunction of literals) and each v_i (the conclusion part) is a g component-vector, v_{ij} taking its values in $\{-1, 0, 1\}$. These values express that the ith rule is respectively in favour, neutral and in disfavour of the class C_j. A default rule is added to this set of rules: in a way, it expresses the *a priori* distribution of classes, and it is used in case of indecision. To classify an example, we calculate for each class C_j the sum $v_{.j}$ of the jth components of the fired rules. After that, the sums $v_{.j}$ are compared. When one of these is strictly greater than the others, it designs the selected class. In the other case, we use the default class to choose among the classes having the highest scores. An example is given in Fig. 6. In (a) we give the decision committee itself, in (b) the result of

this decision committee for the example $x_1x_2x_3x_4x_5$, and in (c) the result of this decision committee for the example $x_1\ x_2x_3x_4\ x_5$. In both cases, two rules are fired. In case (b), the chosen class is C_1, and in case (c), the decision is given by the default class, and is C_2.

(a)

Conditions	Conclusions		
x_1x_2	1	0	-1
$\overline{x_2}x_3$	0	1	0
x_4	0	0	1
$\overline{x_5}$	1	0	0
Total			
Default	0.2	0.3	0.5
	C_1	C_2	C_3

(b)

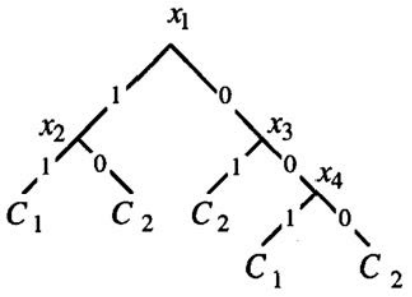
Conditions	Conclusions		
x_1x_2	1	0	-1
[redacted]			
x_4	0	0	1
[redacted]			
Total	1	0	0
Default	0.2	0.3	0.5
	C_1	C_2	C_3

(c)

Conditions	Conclusions		
[redacted]			
$\overline{x_2}x_3$	0	1	0
[redacted]			
$\overline{x_5}$	1	0	0
Total	1	1	0
Default	0.2	0.3	0.5
	C_1	C_2	C_3

Fig. 6. (a) A decision committee; (b) Result for the example $x_1x_2x_3x_4x_5$; (c) Result for the example $x_1\ \overline{x_2}x_3x_4\ x_5$.

Small-sized decision committees, such as the one above, are easy to interpret, and they need only a pencil to be used, like classification trees, or Rivest's decision lists.[55] The difference is that rules are neither ordered (as in decision lists) nor organized in a dichotomic way (as in classification trees). Decision committees might also be viewed as linear discriminators whose coefficients belong to $\{-1, 0, 1\}$, and are able to use conjunctions of literals. Finally, note that when there are two classes, it is convenient to state that a rule in favor of one class be in disfavor of the other. We can then state that the conclusion has one value belonging to $\{-1, 1\}$ which indicates that the rule is respectively in disfavor and in favor of class C_1 or, equivalently, respectively in favor and in disfavor of class C_2 (an example is given in Fig. 7).



(a)

x_1x_2	1
$\overline{x_1}\,\overline{x_3}x_4$	1
Total	
Default	0

C_1

(b)

$x_1\overline{x_2}$	-1
$\overline{x_1}x_3$	-1
$\overline{x_1}\,\overline{x_3}x_4$	-1
Total	
Default	1

C_1

(c)

Fig. 7. Two ways (b) and (c) of coding a decision tree (a) by a decision committee.

7.1.2. Comparison with other classification functions

Let us consider first the Boolean case ($g = 2$). Classes C_1 and C_2 represent truth values *True* and *False*, and the truth table of any Boolean function can be transcribed directly under the form of a decision committee whose monomials are composed of p literals. This means that by multiplying the number of rules and the number of literals, we can represent every Boolean function by a decision committee. This result can be extrapolated to a greater number of classes.

However, in supervised classification we only consider functions whose size (number of literals) is limited. The result given above demonstrates that if we take a decision committee with sufficiently high size k', we can represent any function whose size is k. In order to compare decision committees with other classes of functions, we therefore need to establish a link between k and k'. Take the case of classification trees and consider Fig. 7, in which we show two ways of coding a classification tree by a decision committee. On the basis of this figure, it is clear that classification trees whose depth is k can be coded by decision committees whose monomials have a length of k at most. If we define the size of a classification tree more naturally to be the number of its edges (8 in the previous example), and the size of a decision committee to be its number of literals (5 and 8 before), we can show that any tree of size k can be coded by a decision committee whose size is at most $(k^2 + 6k)(g - 1)/8g$. To give a concrete example, let us consider the classification tree found by CART,[4] for the waveform recognition problem. It is composed of 20 edges. The preceding result shows us that by exploring the set of decision committees whose size is no more than 44, we explore a set of functions that contains any classification tree of size 20, and thus the one found by CART.

Other results of the same type can be derived for other classes of functions. Let us mention however that the possibility to duplicate a rule allows us to come as close as desired to linear separators having real coefficients, and allows us to simulate the ordering of monomials in a decision list (the first rules are duplicated in order to make the decision when they are fired).

7.2. The Learning Algorithm

We can demonstrate that finding the decision committee whose size is bounded and which makes the fewest number of errors is an NP-Hard problem. We are therefore obliged to use approximate methods. We tested numerous algorithms among which some were based on simulated annealing, as in Ref. 5. The algorithm we retained proceeds in two stages : it begins by extracting a certain number of "good" rules and then puts them in a decision committee having a "good" performance on the training sample. We are now going to examine these two procedures separately.

7.2.1. Extracting good rules

The algorithm used is a version of PLAGE restricted to Boolean representations.[23] The aim is to find all of the most general rules satisfying certain numerical criteria. The principle is to make a breadth-first search of the set of monomials, organized

according to the generalization relationship. This search is top-down and starts with the most general monomials, i.e. those having only 1 literal. A monomial m is evaluated by two numerical criteria: (1) we impose that it covers a sufficiently high number of examples N of the learning sample; (2) we impose that it has a sufficiently high discriminant power. This power is measured by the χ^2 criterion, used here as an heuristic, rather than for its statistical properties. For each class C_i, we calculate the quantity Q_i using the χ^2, as indicated below

m	True	False
C_i	a	c
$\overline{C_i}$	b	d

$$Q_i = \frac{(a+b+c+d)(ad-bc)^2}{(a+b)(a+c)(d+b)(d+c)}.$$

When condition (1) is satisfied, and when the maximum of the Q_is is greater than a threshold T, we construct a rule having m as condition. For any i, if $(Q_i < T)$ then the ith component of the rule is 0; otherwise if $(ad - bc > 0)$ this component equals 1, and it is -1 otherwise. When a monomial is retained to form a rule, all of its specializations are pruned. If m is not retained, we calculate a promise function that gives the best score that could be reached by a specialization of m. If this promise is lower than T, all the specializations of m are also pruned. Finally, the algorithm stops when all monomials have been pruned or evaluated.

The worst case complexity of this algorithm is exponential in the number of attributes. In practice, the computation time depends on the chosen thresholds. If they are badly adjusted the algorithm will explore the whole space of all monomials, and this time will be prohibitive. Inversely, the case might occur where only the monomials having a single literal are produced, and the rest of the space is pruned. Experimentally, this algorithm allows us to find rules having 3 literals, from descriptions based on more than 100 binary attributes, in less than one hour of CPU on Sparc 10.

7.2.2. Rule aggregation to form a good decision committee

The objective of this algorithm is to extract from the set of rules previously chosen, a subset that, when assembled, constitutes a decision committee having a low error rate on the training set. The general principle is analogous to that used in agglomerative methods of hierarchical classification. First, we partition the rules into singletons reduced to single rules. At each step, the algorithm achieves the union of two subsets belonging to this partition, and the number of elements of this partition diminishes by one element. These two subsets are chosen by maximizing a gain criterion R among the set of pairs of elements of the partition. The algorithm stops when the best union of pairs of elements of the partition leads to negative or null value of R. We then extract from the partition the subset of rules which forms the decision committee having the lowest error rate.

Let L be a set of rules. We define its error rate, e_L, to be that of the Decision Committee formed by these rules and completed by the best possible default rule (according to the error rate computed on the learning set). The criterion $R_{L,L'}$

measures the gain expected from the union of the two sets L and L'. It is defined by the equation

$$R_{L,L'} = \min\{e_L, e_{L'}\} - e_{L \cup L'} .$$

When this criterion is positive, the Decision Committee formed by the union of L and L' has a lower error rate than that of the two Decision Committees taken in isolation. All along the algorithm, we therefore try to group rules representing a high gain rather than a low error rate. This enables us not to forget rules that could, once reunited in a Decision Committee, constitute a good classification procedure. Let us point out the fact that the results obtained using this algorithm are much better than those of the greedy approach which consists in choosing the best rule and adding rules one by one, while minimizing the error rate, until no gain is possible. Finally, the complexity of this algorithm is in $O(nr^3)$, where r is the number of rules found by the preceding procedure, and n the number of examples.

7.3. Results

The learning algorithm described above has been evaluated on binary and ternary data. The CPU time corresponds to a nonoptimized program written in C, running on a Sun Sparc 10. The size of the decision committees obtained is the number of literals. Results are given in Table 7. We notice that the performance of decision committees is not so far from that of neural networks and much better than that of classification trees. The size of the resulting decision committee is always small, and thus achieves one of the objectives of symbolic methods, which is to produce discriminant functions with a good explanatory power. The rules found generally have only one literal. This explains the small amount of time needed for computation. When different thresholds are chosen (smaller for N and higher for T), computation time and rule size increase, but results are not better.

Table 7. Decision committees.

Data	Train	Test	Test(1)	Test(2)	Test(3)	CPUtrain	CPUtest	Size
Binary	20.5% (1.5%)	23.0% (0.8%)	32.8%	20.7%	15.9%	25″	3″	20
Ternary	16.6% (1.8%)	20.5% (1.4%)	33.0%	14.8%	13.6%	23″	5″	28

8. LEARNING RULES WITH A GENETIC ALGORITHM

8.1. The Genetic Algorithm Principles

Genetic algorithms are optimization procedures inspired from the mechanisms of natural selection.[32] In order to solve an optimization problem, these algorithms use a population where each individual represents a possible solution to the problem. Such an individual is evaluated using an evaluation function that measures how well the

individual is adapted to the problem. Starting initially with a randomly generated population where each individual has been evaluated, the genetic algorithm selects a subset of individuals by choosing the best individuals of the population with a high probability. Then, these selected individuals are used as parents in order to produce the next generation. To achieve this, an individual must be represented as a string of genes in order to be able to use genetic operators such as crossover or mutation. The crossover operator selects two parents and randomly exchanges two substrings of genes in order to create two new individuals. The mutation operator randomly modifies some genes of an individual and aims at introducing new or forgotten genes in the population. These operators aim to create better and better individuals by combining useful genes of their parents. A new population of the same size as the preceding one is created using these operators, and the algorithm will continue such evaluation-selection-crossover-mutation cycles until a stopping criterion is fulfilled.

Genetic algorithms can be applied to machine learning problems.[34,69] In this case, each individual represents knowledge such as rules, neural networks or Lisp functions. The evaluation function is computed using a learning set and may measure, for instance, whether or not an individual correctly classifies the examples in this set. The aim of the algorithm is then to find the individual that correctly classifies the greatest number of examples. The algorithm we shall present, called SIA, follows this line of approach.

8.2. Input–Output Model

SIA takes as input:

- *Examples.* Each example is described with attributes. For instance, these attributes can be *Colour* or *Size.* Each example has a special attribute that represents its class and that takes discrete values. For instance, the *Fly* attribute that may take the values *yes* or *no* can be the class attribute to be predicted using the two other attributes. The attribute values can be missing, indifferent or undefined. SIA can also deal with tree-structured values. For instance, the *Colour* attribute may take the specific values *red* or *orange*, and a more general value *red-like.*
- *Biases.* These biases are preference given by the user who may like to favor some rules. For instance, the user may ask for specific rules that take into account many attributes, or for general rules. This aspect is expressed using a parameter which is denoted as β in the following. In order to handle noise, the user may also give a maximum allowed error rate for each learned rule, denoted as α.
- *Search intensity.* The user can let the genetic algorithm spend more or less time for learning rules, by modifying the stopping criterion. This intensity is denoted by *Nbmax* and its use is detailed in the following.

Finally, SIA outputs rules of the following form:

$$\text{If } (Colour = \textit{red-like}) \text{ and } (Size \in [4.3, 7.8]) \text{ Then } Fly = yes\,.$$

These rules can be used by SIA to predict the class of an unclassified example, or to analyze the database by providing the user with symbolic and thus understandable results.

8.3. The Learning Algorithm

SIA is a covering algorithm inspired from AQ[42] which uses a genetic algorithm as search algorithm: an example of the learning set is chosen as a seed, and then the genetic algorithm tries to find the best rule that covers this example; another uncovered example is then chosen to learn another rule, until all examples are covered. For instance, let us suppose that the chosen example is:

$$(Colour = red) \text{ and } (Size = 5.4) \text{ and } (Fly = yes).$$

The genetic algorithm is going to generalize this example into a rule. This example is initially translated into a very specific rule, denoted by *Rinit* in the following, that would be in our example:

$$\text{If } (Colour = red) \text{ and } (Size \in [5.4, 5.4]) \text{ Then } (Fly = yes).$$

Then, the genetic algorithm uses a population of rules that are all at least as general as this initial rule. Its aim is to find a rule that maximizes the criterion which is defined by the user, like for instance "the most general rules with less than 10% classification errors". This population of rules evolves using the principles described earlier and using genetic operators adapted to the high level representation of the rules.

The mutation operator randomly generalizes a rule by performing one or more of the following operations:

- *Enlarging an interval.* For instance, the [5.4, 5.4] interval can be changed to [4.7, 5.6], where 4.7 and 5.6 are other values of the *Size* attribute that are observed in the learning set.
- *Generalizing a tree-structured attribute.* For instance, in the previous rule, the value *red* can be changed to *red-like*.
- *Dropping a condition.* For instance, the condition over the *Size* attribute in the previous rule can be dropped, which generates the rule "If $(Colour = red)$ Then $(Fly = yes)$".

The crossover operator exchanges conditions between two parent rules. For instance, with the following two parent rules:

$$\text{If } (Colour = blue) \text{ and } (Size \in [4.4, 7.84]) \text{ Then } (Fly = yes),$$

$$\text{If } (Colour = red) \text{ and } (Size \in [3.5, 5.6]) \text{ Then } (Fly = yes),$$

the following rules can be generated by exchanging the condition over the *Colour* attribute:

$$\text{If } (Colour = red) \text{ and } (Size \in [4.4, 7.84]) \text{ Then } (Fly = yes),$$

$$\text{If } (Colour = blue) \text{ and } (Size \in [3.5, 5.6]) \text{ Then } (Fly = yes).$$

Each generated rule is evaluated through the evaluation function. For a given rule R, this function equals:

$$f(R) = \frac{c(R) - \alpha nc(R) + \beta g(R)}{c(R) + nc(R)}$$

where $c(R)$ is the number of examples that R classifies correctly, $nc(R)$ is the number of examples that R misclassifies and where $g(R)$ is the generality of R, measured by the proportion of dropped attributes in R condition part. Having $\beta < 1$, this function ensures that the accuracy of learned rules, i.e. $c(R)/c(R) + nc(R)$, is above $\alpha/1 + \alpha$. This is due to the fact that the initial rule satisfies $f(Rinit) \geq 0$, and that the algorithm may only improve this rule. The higher β is, the greater importance given to generality. Usually, we choose $0 < \beta < 1$. In this case, first importance is given to the accuracy of rules, and among the rules having the same accuracy, the most general is prefered.

The genetic algorithm stops when more than *Nbmax* rules have been generated without improving the best rule of the population. The best rule found is then added to the list of rules that will be output. SIA then chooses another example which is not covered by any learned rules, and uses this example as a new seed for learning another rule. The genetic algorithm is called up several times until all the examples in the learning set are covered. The overall algorithm is the following:

1 Let x be an uncovered example
  2 Generalize x into a rule R^* using the GA:
    2a Translate x into a specific rule *Rinit* and
    Initialize the rule population P to *Rinit*
    2b Randomly select one or two parent rules in P,
    Generate one offspring R using genetic operators and
    Evaluate R using f.
    2c Add R to P if P contains less than 50 rules or
    replace the worst rule R^- of P with R if $f(R) > f(R^-)$.
    2d Repeat steps 2b and 2c until the best generated rule R^*
    has not been improved for more than *Nbmax* generations.
  3 Output R^* and Go to 1 if some examples are still uncovered.

Let p denote the number of attributes. For the sake of simplicity, let us suppose that these attributes are binary. In the worst case, the genetic algorithm generates all possible rules, that is $\Theta(2^p)$, and this for every search it performs. This yields an exponential worst case complexity of the whole algorithm. Of course, this worst case complexity is never reached in practice. For instance, in the waveform learning problem, the maximal number of rule evaluations would be in the order of 10^{11}, but SIA evaluates only 10^5 rules in practice.

8.4. Classification Procedure

Given a set of rules and a description z to be classified, SIA looks for the rule which is the closest to z, and chooses the class that appears in the conclusion part of

this rule. This technique is similar, in a way, to the Nearest Neighbor algorithm (Sec. 2.3). The distance used takes into account the quality of the matching between the rule and the example. Let us consider a rule R. The distance $d(R, z)$ equals 0 when R matches z exactly. Or else, the distance is equal to the proportion of R's conditions which do not match z. When several rules are at the same distance from z, the rule with the best performance on the learning set is chosen, according to the evaluation function described previously.

8.5. Results

Results on the waveform problem are given in Table 8. The CPU time corresponds to a Pascal program running on a Sun Sparc 10 workstation. The Size column corresponds to the number of rules (80) multiplied by the mean number of conditions in the learned rules (4). In this test, SIA tries to find the most general rules with a maximum error rate of 10% on the learning set ($\alpha = 9$ and $\beta = 0.1$). Thus, SIA performance on the learning set is good. This also explains why the number of learned rules is high. While SIA is a stochastic algorithm which does not always learn the same rules on two different runs with the same learning set, the standard deviations obtained are quite low. SIA has been successfully applied to several databases, but SIA does not get a better performance on the waveform problem compared to the other methods. This may be due to the fact that noise is not handled very well either in the learning or classification procedures.

Table 8. SIA genetic algorithm.

Data	Train	Test	Test(1)	Test(2)	Test(3)	CPUtrain	CPUtest	Size
Binary	4% (0.7%)	24.3% (0.7%)	27.7%	20.8%	24.3%	300″	30″	$4 \times 80 = 320$

However, one should note that this learning task is rather "simple" for SIA because there are no unknown or tree-structured attributes in the data. The genetic algorithm is also more flexible than other heuristic based search. For instance, the criteria to be optimized can be easily modified by the user, without any modification of the search algorithm. This may allow SIA to take into account user preferences for some attributes over some others, or to use any, e.g. nonuniform, misclassification cost functions. The flexibility of genetic algorithms also allows SIA to be enhanced in order to deal with first order logic representation,[1] without changing its main principles.

9. VERSION SPACES

9.1. The Version Space Framework

Version spaces were developed by Mitchell[43] as a general framework of combinatorial learning algorithms for the discrimination between two classes, C_+ and C_- (positive

and negative instances). The goal is to represent the set of solutions discriminating C_+ from C_-, in a given *characterization language*. For this purpose, one structures the space of all sentences of the language with a partial ordering called generalization relation. This relation is such that a sentence s is more general than another sentence s' if it covers a superset of examples, and this is denoted as $s \geq s'$. One then builds two sets, the set S of all more specific solutions (minimal w.r.t. this order) and the set G of all more general solutions (maximal w.r.t. this order). This is done incrementally on the set of instances. Initially, $S = \{\perp\}$ (solution rejecting all instances) and $S = \{T\}$ (solution accepting all instances). At each step, if the new instance i is positive (resp. negative), the elements of S (resp. G) are minimally generalized (resp. specialized) in order to cover (resp. reject) i, with a set of operators depending on the chosen language. In case of Boolean descriptions, which we used for the waveform problem, a convenient representation of the G set allows an $O(n^2)$ time complexity, where n is the number of instances.[46]

The decision d_{VS} of the membership class of a new description x is taken on the basis of the version space $VS = (S, G)$ following the rule:

$d_{VS}(x) = C_+$ If $\forall s \in S,\ x \leq s$, i.e. x belongs to the class of examples;

$d_{VS}(x) = C_-$ If $\forall g \in G,\ \neg(x \leq g)$, i.e. x belongs to the class of counter-examples;

$d_{VS}(x) =?$ Or else, i.e. x can be classified in either two classes indifferently.

This approach presents some important characteristics from the point of view of supervised classification. First of all, this method is primarily intended to find the set of solutions which perfectly discriminate the learning examples. When no such solution exists, the version space approach is faced with a difficulty, the treatment of which constitutes the main subject of this section. On the other hand, the fact that all solutions are retained makes the method relatively insensitive to the presence of irrelevant or redundant attributes. Finally, in common with most generalization methods, the language of characterization is part of the data. This language may then be adapted to the application, without changing the method.

9.2. Version Spaces and the Waveform Problem

We considered that the examples were described with binary attributes. In this case, a natural characterization language is the set of monomials that one may build from these attributes. Generalization ordering corresponds to the cover relation between monomials ($m_1 \geq m_2$ iff $m_2 \Rightarrow m_1$). Note that in this case, the set S is always reduced to a singleton.

We tried to characterize the examples corresponding to each class proposed by Breiman *et al.*[4] with respect to its complementary set, e.g. C_1 w.r.t. $C_2 \cup C_3$. The opposite is also possible, and one may choose $C_2 \cup C_3$ for the class of examples and C_1 for the class of counter-examples. In the second case, the problem comes down to the characterization by empty monomials, as carried out in Sec. 6, but the results are clearly inferior. On the other hand, using both characterizations slightly

improved the results, at the expense of a doubling of the number of literals in each characterization.

This being stated, the waveform problem is difficult for Version Spaces. Results clearly show that it is not a good method for this problem. The main reasons are:

- First of all, it is a problem with more than two classes and consequently, not directly within reach of this method. As already outlined before, the solution is to produce as many version spaces as the number of classes. However, this requires a new decision rule to be built, combining the individual decisions of each version space.
- Second, the chosen description language is insufficient to characterize each class. The "sequential" nature of the data is not taken into account. Characterizations are particularly sensitive to a translation factor, and this is clearly not desirable.
- Finally, a certain degree of covering between classes exists (the problem is not deterministic). One is not always guaranteed of being able to discriminate perfectly, even between the training examples.

Therefore, we are not looking for a method to challenge the results of classical methods such as linear discrimination (Sec. 2.2), largely more suited for this particular application. We would prefer to show how the results of the basic method may be improved in a symbolic-numerical framework.

9.3. Description Language

We experimented with the binary coding described in Sec. 1.5. This language being particularly poor, we tried several alternative codings of the continuous data, in order to enrich the basic descriptions while remaining in a domain of binary attributes. The most interesting results were obtained with a coding scheme suggested by Breiman *et al.*,[4] based on a moving average calculation on raw data. More precisely, we computed the moving average for windows of size 1, 3 and 5, and coded the result with 2 Boolean threshold attributes ($x_i < 3$ and $x_i > 6$), that leads to a coding of length $p = 126$ bits. On average, this coding allows us to "reduce" the noise attached to these contiguous attributes whose values, before the noise has been added, are necessarily very close. It includes therefore a considerable knowledge of the data generation model. For instance, in the case of classification trees, it provide an increase of about 8% of the recognition rate.[4]

9.4. Training Example Selection

The search algorithm in the version space is initialized with a set S reduced to the element $\perp$ (recognizing no description) and a set G reduced to element T (recognizing all the descriptions). S and G are then refined, taking into account the examples of the learning set incrementally.

When the language is not sufficient to describe the data (this is in fact the case for the waveform problem), the algorithm may stop in two different states. Either it converges on a single solution before having treated all training examples, or

it detects that no solution exists. The simplest way, retained here, that always guarantees the existence of at least one solution, is to reject all examples producing an empty version space. It must be clear that this process can lead to an acceptable solution only if training examples have been adeptly selected for their presentation. Furthermore, a good scheduling of these examples reduces the complexity of the calculation by allowing a faster convergence of the algorithm.

The goal is to produce this ordering by means of a hierarchical classification of the training examples. One selects the higher nodes in the tree, corresponding to a cluster in which all examples belong to the same class, and represented by circled nodes in Fig. 8. In this way, one builds a partition of the examples in homogeneous clusters, that are then sorted in decreasing order of size. While learning class C_i, one alternates, in decreasing order of size, the presentation of a group of C_i and the presentation of a group of another class. Thus if classification is correct, one learns as a priority the most relevant aspects of the characterization, covering the largest set of examples. Thus, the tree in Fig. 8, where leaves are labeled with the name and the class of the examples, leads to the following presentation if one tries to learn class C_1 (+ for example and − for counter-example): (e_4^+, e_5^+, e_6^+, e_1^-, e_2^-, e_8^+, e_9^+, e_3^-, e_7^-).

Furthermore, note that the very same tree may serve to produce disjunctive formulae by learning monomials for each group of a partition of the tree. For instance, if one cuts the tree just below the root, one obtains a partition of the examples in 2 groups that gives the 2 following presentations for class C_1: (e_4^+, e_5^+, e_6^+, e_1^-, e_2^-, e_3^-, e_7^-) and (e_8^+, e_9^+, e_1^-, e_2^-, e_3^-, e_7^-). One may alternatively choose the counter-examples only in the cluster containing the examples, but this generally leads to significantly lower results. In the case of waveforms, this possibility of introducing disjunctions has not brought a noticeable gain in recognition.

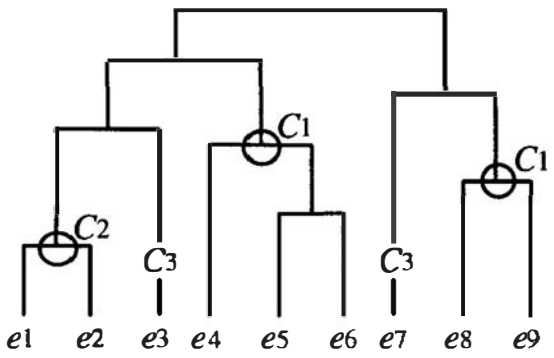


Fig. 8. Hierarchical clustering of the examples $e_1, e_2, \ldots, e_9$, the class being indicated on the internal nodes, or on the edges.

We used a hierarchical ascending clustering algorithm (CHAVL), based on the Likelihood Linkage Analysis method.[38] We start with the calculation of the raw similarity between any two examples. This raw measurement is standardized with respect to its theoretical mean and standard deviation, calculated on an independence assumption basis, and the final index corresponds to the likelihood of the similarity. The hierarchy is then built step by step, following the maximum likeli-

hood criterion. First, we measured the raw similarity with the number of attributes simultaneously true in both examples. The results, while noticeably improving the results of a random partition (approximately 10%), presented a relatively high standard deviation (4%). The best classification results were obtained for a raw similarity measuring the number of identical windows of 2 or 3 contiguous attributes in the two compared sequences.

9.5. The Classification Function

As many version spaces are produced there are classes to be discriminated. It is necessary then to build a classification rule which seeks for a consensus between the different judges represented by each version space. After numerous trials, we adopted a solution where each judge has g ballot papers to be distributed among the g classes following its decision function. This corresponds to a rule choosing the class C_i maximizing the following function f_i:

$$f_i(x) = \sum_{j=1,g} f_{ij}(x) \text{ where}$$

If $d_{VS_i}(x) = C_+$ then $f_{ii}(x) = g$ and $f_{ij}(x) = 0$ for $j \neq i$;

If $d_{VS_i}(x) = C_-$ then $f_{ii}(x) = 0$ and $f_{ij}(x) = g/g - 1$ for $j \neq i$;

If $d_{VS_i}(x) =?$ then $f_{ij}(x) = 1$ for every j.

9.6. Results and Discussion

Results given in Table 9 have been obtained from binary data, using a classification based on windows of 2 attributes (Sec. 9.4), and on continuous data coded according to the method of Breiman *et al.* (Sec. 9.3), using a classification based on windows of 3 attributes. The CPU time corresponds to a program written in C and Prolog, which is relatively well optimized (original research results were necessary for this purpose,[46] and which runs on a Sparc IPX 32 Mega ($\approx$ Sparc 2). This includes the time of clustering training examples, which is about 20 seconds. The obtained length is expressed in terms of the number of literals and corresponds to the generation of 12 sets (S and G for each class and characterizing either the examples or the counter-examples).

Table 9. Version spaces.

Data	Train	Test	Test(1)	Test(2)	Test(3)	CPUtrain	CPUtest	Size
Binary	30.9% (3.6%)	31.9% (2.3%)	37.5% [62]	29.2%	29.1%	35″	0.2″	27
Breiman's Coding	17.4% (1.9%)	21.6% (2.0%)	28.9% [300]	19.2%	16.7%	70″	0.3″	214

There is generally a single monomial in G (and S). These monomials are relatively short in the case of the initial binary coding (about 2 literals), and much longer in the case of Breiman's coding (about 18 literals, but for a code 6 times longer). Exhibited learning times show a linear progression with respect to the number of attributes. The recognition rates found have to be appreciated in relation to the error rate on the training set, corresponding to the percentage of training examples that were not taken into account, due to the inadequacy of the characterization language. A related interesting indicator is the number of examples treated before the convergence of the algorithm. This number is given between brackets in column Test(1). For an acceptable coding, there must be no convergence, i.e. one must find 300. The slight difference between results obtained on the training set and those on the test set is to be noted. Note also that results are less accurate for class 1, more difficult to characterize in a conjunctive way.

10. CONSTRAINT-BASED LEARNING

10.1. Method Overview

Constraint-based induction is a new Machine Learning algorithm[58,59] inspired from both the Star Algorithm AQ[41] and the Version Space approach.[43] Since these approaches were presented previously (Secs. 8 and 9), we shall only recall the limitations which motivated their hybridization and embedding in the constraint-based induction frame.

Version Space fails to handle disjunctive concepts, overlapping concepts, and/or noisy data. Furthermore, the size of the G set may be exponential when using the standard disjunctive representation.[30] Star Algorithms partly overcome such limitations through a bottom-up exploration of the examples. However, the stars are built under control of the expert, who explicitly provides criteria for sorting the number of solutions to be kept.

So constraint-based learning investigates the coupling of the Version Space and the Star Algorithm. On the one hand, a bottom-up exploration enables us both to deal with disjunctive concepts and handle noisy data. On the other hand, the need for evaluation and control is avoided by defining the star of a seed (x, c) as the G set derived from this seed: the G-star associated to seed (x, c), noted $G(x, c)$, is the more general formula covering x and not covering any training example belonging to another class. A constraint-like representation, inspired from Ref. 46, enables a polynomial building and handling of G-stars with attribute-value descriptions. Note that this representation handles continuous attributes directly, as opposed to some approaches (e.g. Secs. 6, 7 and 9) which require prior segmentation of continuous attribute domains. Finally, to classify an unknown description z, we use a simple idea which exists, for example, in the k-NN method and in some methods already presented: z is classified in the most frequent class among the stars to which it belongs.

10.2. Notation and Definitions

We restrict ourselves to linear (real, integers) and nominal (discrete or tree-structured) attributes. Let (x, c) denote a seed. Let E_- be the set of training examples that belong to a class different from c. The G-star $G(x, c)$ is the conjunction of the constraints derived from the examples in E_-, called counter-examples to (x, c). Before defining a constraint, let us recall the notion of *selector*.[41]

- A *selector* is a Boolean function defined on the problem domain X, which is denoted as $[attribute = V]$; in the case where *attribute* is nominal, this function takes value *true* for x in X iff the value $attribute(x)$ equals value V (or is more specific than V if the domain of *attribute* is a hierarchy); in the case where *attribute* is linear, it takes the value true iff $attribute(x)$ belongs to interval V.
- The *constraint* derived from counter-example (x', c'), noted $D(x, x')$, is the disjunction of the most general selectors that cover x and reject x', called maximally discriminant selectors. Let us consider the following data:

	Smooth	*Height*	*Width*	*Colour*	*Class*
$x =$	*Yes*	3	19	*Red*	C_+
$x' =$	*No*	7	12	*Blue*	C_-

  Attribute *Smooth* is binary; so the maximally discriminant selector based on this attribute is $[Smooth = yes]$. Attributes *Height* and *Width* are linear. Hence, the maximally discriminant selectors based on these attributes are respectively: $[Height < 7]$ and $[Width > 12]$. Last, assuming *Hot-colour* is the most general value for attribute *Colour* such that it covers *Red* and rejects *Blue*, then the maximally discriminant selector based on attribute *Colour* is $[Colour = Hot\text{-}colour]$. Finally, the more general formula covering x and rejecting x' is:

$$D(x, x') = [Smooth = Yes] \text{ or } [Height < 7]$$
$$\text{or } [Width > 12] \text{ or } [Colour = Hot\text{-}colour] .$$

  The disjunction $D(x, x')$ is the G-star built from the unique positive example (x, c) (the seed) and the unique negative example (x', c'). The selector based on a given attribute is present iff this attribute is informed for both x and x' (which enables easy handling of missing values), and if it takes different values for x and x' (up to a given precision in the case of real values).
- The G-star of a seed (x, c) is the conjunction of the constraints $D(x, x')$ derived from all its counter-examples. However, a counter-example gives rise to a constraint iff, at the time it is considered, it still belongs to the star. Otherwise, it is discarded.

10.3. Algorithms

As in the Star Algorithm, learning examples are considered randomly. An example becomes a seed, i.e. gives rise to a star iff, at the moment it is considered, it is

not yet rightly classified given the stars formerly built. The star of a seed (x, c) is the conjunction of all constraints derived from the counter-examples of (x, c), and the constraint derived from a counter-example (x', c') is the disjunction of all maximally discriminant selectors covering x and rejecting x' (see above). Note that this representation of G is polynomial with respect to the number of attributes and examples, whereas the usual representation (as a disjunction of conjunctions) is possibly exponential.[30] This allows polynomial learning (and classification). Building a constraint is in $O(p)$, building a star is in $O(n^2p)$, and the entire learning process is in $O(n^3p)$.

The classification of a new description z is based on the learned G-stars: z is classified in the most frequent class among the stars $G(x, c)$ it belongs to. Membership to a star may be tuned according to two parameters:

- The first parameter denoted as ε allows us to handle "noisy" descriptions: z belongs to $G(x, c)$ if it satisfies at least a percentage $(100 - \varepsilon)$ of the constraints in the star.
- The second parameter denoted as M controls the generality of constraints: z satisfies a constraint $D(x, x')$ iff it satisfies at least M selectors in this constraint. When M is 1, $D(x, x')$ is simply used as a disjunction of the selectors; otherwise, $D(x, x')$ is used as an *M-of-N* concept. This heuristic is motivated by the fact that, when M is 1, most z happen to satisfy any $D(x, x')$ (especially when the problem domain involves many continuous attributes); hence they belong to most stars, and are classified in the most represented class! This drawback disappears as expected when M increases.

Since the total number of selectors in the stars is upper-bounded by n^2p, the classification of an example has a complexity in $O(n^2p)$.

10.4. Results

Parameter ε varied from 0 to 20% and parameter M varied from 7 to 11 in our experiments. Parameter ε influences the performance in a classical way: when it increases, the performance decreases on the training set, while on the test set it increases at first, then decreases. In fact, the only crisp difference occurs between $(\varepsilon = 0\%)$ and $(\varepsilon = 5\%)$. For a given value of ε, the performance is quite stable depending on M. However, an increase in M can to some extent counter-balance an increase of ε. For instance, the best results on continuous data are obtained for $(\varepsilon = 10\%, M = 8)$ and $(\varepsilon = 20\%, M = 9)$. The program is implemented in C++ and runs on a HP 710 workstation. The size is expressed as a number of stars and a total number of selectors. Note that this huge number of selectors allows one to efficiently grasp an arithmetic concept in a logical manner. On the other hand, it needs bunches of selectors to mimic arithmetic skills (imagine approximating an oblique line through stepwise functions). The theory hidden in the G-stars is therefore unintelligible. However any decision can be explained through the common points between the case z at hand, and the seeds (x, c) of the stars to which z belongs. Results are given in Table 10, and are among the best we obtained in this

study. Thus, it appears that the constraint-based approach efficiently overcomes the limitations of the Version Space regarding disjunctive and overlapping concepts and noisy data. Moreover, we would like to emphasize that constraint-based learning may be extended to a subset of first order logic.[58]

Table 10. Constraint-based learning.

Data	Train	Test	Test(1)	Test(2)	Test(3)	CPUtrain	CPUtest	Size
Binary	22.8% (0.5%)	21.6% (0.3%)	25.6%	21.6%	17.1%	62″	388″	stars 87 Sel 33000
Contin.	18.0% (0.6%)	18.0% (0.2%)	18.1%	18.1%	17.7%	50″	303″	stars 77 Sel 55000

11. ABOUT THE RESULTS ON THE WAVEFORM RECOGNITION PROBLEM

It is obvious that the choice of a particular problem favors certain methods over others. All the conclusions of this study, particularly on the misclassification rate, cannot therefore be extrapolated to other problems and other domains. Nevertheless, certain characteristics of the methods appear in the results presented. Some methods learn more rapidly, others decide more quickly, or are endowed with a better explanation power. Table 11 and Fig. 9 highlight these different characteristics. Except the *Best coding* item, they are derived from results obtained by methods on binary data, which are those which more clearly justify the use of a hybrid approach, if we exclude the Fuzzy Classification Trees (Sec. 5) which are basically intended to deal with continuous attributes. For this latter method, we used results from Table 5 which were obtained for continuous data. The twelve methods are classified according to the following six criteria:

- *Total Misclassification Rate* for which we provide the detailed ordering, although this could be questioned as said before. Moreover, because of the stochastic nature of the problem, this ordering might be slightly modified if other learning and test sets were used. This is clearly shown in Fig. 9 where the 95% standard confidence interval of the expected error rate is represented (Test $\pm$ $1.96\sigma/\sqrt{11}$). When performing a Student test, we see that the only 95% significant differences between consecutive methods are between: MLP and Constraint-based, Empty Monomials and Genetic approach, Genetic approach and Q. Fischer, Q. Fisher and Fuzzy Classification Trees, Classification Tree and Version Space. If we now consider for a given learning set the interval in which the misclassification rate lies with probability 95% (obtained from Fig. 9 by enlarging the confidence intervals by a factor $\sqrt{11} \approx 3,5$), we see that it is not unlikely to observe a learning set such that a rather poor method, e.g. Q. Fisher, performs better than a good method, e.g. L. Fisher. In practice it may be said, therefore, that the first 9 methods are not extremely different, while the last 3 yield very inferior results. Besides, the standard deviation is widely different from one method to another. Some

Table 11. Ordering of the methods according to six criteria. Test, Test(1), CPUtrain, CPUtest and Size have the same meaning as in previous tables and have been obtained on binary data. Best coding provides the ordering of the methods according to lowest misclassification rate observed in this study, depending on the coding scheme; B stands for Binary, T for ternary, C for continuous and BR for Breiman's coding (Sec. 9.3).

Method	Test	Test(1)	Best coding	CPUtrain	CPUtest	Size
L. Fisher	3	5	4 T	1	1	2
Q. Fisher	9	7	6 C	1	1	3
Parzen	4	12	8 T	1	3	3
k-NN	6	11	3 C	1	3	3
MLP	1	2	1 C	3	2	2
Decision tree	11	9	9 BR	2	1	1
Fuzzy decision tree	10	6	12 C	3	2	2
Empty monomials	7	1	10 B	2	2	3
Decision com.	5	8	5 T	2	1	1
Genetic approach	8	4	11 B	3	2	2
Version space	12	10	7 BR	2	1	1
Constraint based	2	3	2 C	3	3	3

methods (e.g. Parzen's Kernel or k-NN) have quite a high variability, while some others, typically hybrid methods (e.g. Empty Monomials and Constraint-based) present a low standard deviation.

- *Misclassification rate in the first class*, indicated as Test(1), for which we also provide the detailed ordering. We selected this criterion because the first class seems to be much harder to predict correctly. It follows that some methods, e.g. Parzen's Kernel, obtained very unbalanced results among the three classes. These latter methods rarely predict the first class and take few risks. On the other hand, methods such as Empty Monomials, take more risks, obtain well balanced results and inevitably are not excellent considering the total misclassification rate. Therefore, the misclassification rate in the first class enables the point of view given by the total misclassification rate to be completed and corrected (see Fig. 9).
- *Lowest misclassification rate observed during this study, depending on the coding scheme*, indicated as Best coding. Again, we selected this criterion to complete and correct the misclassification rates obtained with binary data. Indeed, some methods have the ability to directly handle continuous data or have sufficiently low computational cost that it is possible to use a sharp non-binary discretization of continuous attributes. Most of these methods takes advantage of these data which contain more information than binary data (Fig. 9). The most important improvement is obtained for the k-NN method, while the exceptions are CART whose results are better with the binary data (Table 4), and Parzen's Kernel which is only slightly improved by ternary and continuous codings (Table 3). The Empty Monomials and Genetic methods which have only been tested with the binary coding, probably because of computational cost, are penalized. Finally,

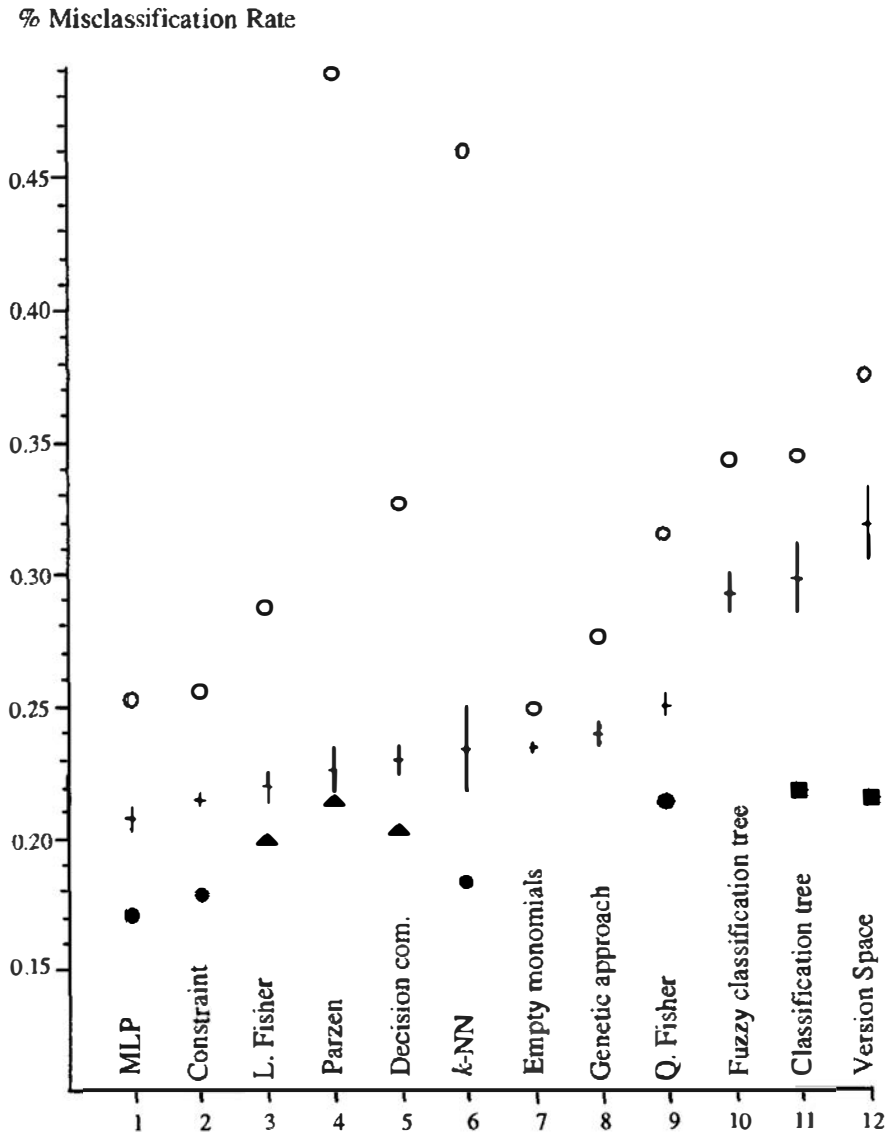


Fig. 9. Ordering of the twelve methods with: their average total misclassification rate on binary data (horizontal stroke); the 95% standard confidence interval of this average rate (vertical line); their average misclassification rate in the first class (white circle); their lowest average misclassification rate observed during this study, depending on the coding scheme (black circle: continuous data, black triangle: ternary data, black square: Breiman's coding).

the most impressive difference is obtained with Breiman's coding (Sec. 9.3) which was only applied to the two poorest methods (i.e. CART and Version Space) and which makes them close to the best. This proves, if proof is needed, that the coding stage is of primary importance and that it must be conducted by using as much as possible the knowledge we have about the data, and by considering the properties of the classification method we envisage.

- *Learning Time* for which three categories of methods were considered: (1) those which provide an instant response ($\leq$ 1 second); (2) those for which the waiting time is reasonable ($\leq$ 1 minute); (3) others which are not likely to be envisaged in the case of an exploratory procedure aimed at "understanding" the data. The times as indicated throughout the article were all measured on different equip-

ment. To obtain comparable results we considered the Sparc 2 (Sec. 2) as the basic machine (power = 1), and we have corrected the other running times by using the following power ratios: Mac (Sec. 6) $\approx 1/10$, Sparc 1 (Sec. 4) $\approx 1/2$, Sparc IPX (Sec. 9) and HP710 (Sec. 10) ≈ 1, Sparc 10 (Secs. 3, 5, 7 and 8) ≈ 2. This is clearly a quite rough estimate, but our categories are not affected by reasonable modifications of these ratios.

- *Decision time* for which we also considered three categories: (1) quick response for 5000 test examples ($\leq$ 10 seconds); (2) reasonable time ($\leq$ 2 minutes); (3) others not to be envisaged in an exploratory approach. If we take the decision time for a single example, all the methods are sufficiently rapid ($\leq$ 1 second) to be used, for instance, in the case of medical diagnosis.
- *Size* for which we also provide three categories: (1) results directly exploitable and interpretable; (2) size sufficiently reduced so that useful information may be drawn from the results without much difficulty; (3) size such that the compression aspect (or resume) of data is practically absent. For example, classification trees are in the first category, linear discrimination (66 parameters) in the second, while quadratic discrimination (696 parameters) is in the third category.

It may be noted that among the "best" methods, we find Linear Discrimination, Multi-Layer Perceptron and Decision Committees, which are similar in several aspects. The performance of these methods may certainly be explained by their appropriateness for the problem. Moreover, the analytic solution of Fisher's Discriminant Function, combined with the well optimized implementation of the SAS software, yields high computational efficiency. However, if we try to compare the hybrid methods with the most classical methods presented (Fisher's Discriminant Function, Parzen's Kernel, k-NN and Classification-Tree), we see that there are some methods which are more powerful in terms of error rate, for example, Neural Network or Constraint-based methods. Moreover, most hybrid methods obtain results which are better balanced than those obtained by the classical methods. Concerning Neural Networks, similar results have been found in other studies.[35,54] Likewise, the Decision Committee method is as explanatory as that of Classification Trees, yet the results obtained by the former are vastly superior at the performance level. This somewhat contradicts a conclusion of King *et al.* (Ref. 35, p. 312) which was that "symbolic algorithms all performed very similarly and that there is no obvious best algorithm". The explanation, developed below, is very likely linked to the notion of vote in decision taking, used by the best hybrid methods. Therefore, these experimental results are very encouraging. It remains to verify that these good performances can also be observed with various large real-world classification problems. Preliminary results in that direction have already been obtained for the Decision Committe method.[48]

12. CONCLUSION

In this article we presented twelve supervised classification methods, some classical and some original, which combine numerical and symbolic aspects. We would now

like to make some general remarks about the methods presented, and try to draw some conclusions.

First of all, it appears that the symbolic aspect of the methods presented is predominantly linked to the notion of rule. This is clear in all of the methods, with the exception of the purely numerical (Sec. 2). The rule notion, which emerges in the earliest studies on Machine Learning, and is linked to expert systems and to the search for explanation virtues, takes on various forms. We find juxtaposed, Tree-Based and Decision Committee methods (Secs. 4, 5 and 7) on one hand, and on the other hand, those based on empty monomials (Sec. 6) and constraints (Sec. 10). In the first case, we find rules which are indeed close to those of expert systems, whilst in the latter the rules are extremely numerous and primarily characterize the description space. It is also to be noted that Neural Networks, which are inherently numerical, integrate a certain notion of rule if they possess a hidden layer. This hidden layer contains the "rule" conclusions defined through the first weight layer, while the next layer shows how these rules should be combined.

Concerning the numerical aspects, the notion of vote in decision taking has considerable importance. This occurs on two levels. First, in several methods, rules are fired through a partial matching procedure, in other words, through a counting and threshold comparison procedure. Now, in most methods, the decision is taken based on a set of fired rules and not on the basis of a single rule. Exceptions to this are the classification-tree methods, which in light of the results on waveforms, seem to be penalized by this characteristic. Numerical methods, particularly Fisher's Linear Discrimination and Neural Networks, integrate the voting notion through the more general notions of weighted sum and activation function.

Different solutions are proposed by these methods for finding rules, for evaluating those rules individually and collectively, for firing rules, and for combining their decisions. Other solutions exist, which are not presented here, among which we may cite those inspired by neural networks, such as Refs. 26 and 62. Deepening the theoretical results is a promising research field to be explored in future studies. This would make it possible to find paths through the diversity of the solutions proposed at present, to specify formal bases of certain approaches and to explain the often positive experimental results obtained by these methods.

Several methods presented integrate the notion of generalization, and that of solution or version space. They proceed by exploration of this space, and extract the pertinent points which are used to form the condition part of rules. Thus, they are able to pass from simple attribute-value descriptions to more complex system descriptions, based on subsets of predicate logic, or on graph-based formalisms. This passage has already been studied for several methods we have presented,[58,69] and applications for other description types, notably biological sequences, have been conducted successfully.[22,25] We think that this capacity to process complex data, as well as the results obtained here on a problem known to be hard for rule based approaches, is very encouraging for the future of supervised classification hybrid methods.

REFERENCES

1. S. Augier, G. Venturini and Y. Kodratoff, "Learning first order logic rules with a genetic algorithm", *Proc. Int. Conf. Knowledge Discovery in Databases* (*KDD'95*), AAAI Press, 1995, pp. 21–26.
2. R. Battiti, "First and second order methods for learning: between steepest descent and Newton's method", *Neural Comput.* **4** (1992) 141–166.
3. N. Bongard, *Pattern Recognition*, Spartan Books, 1970.
4. L. Breiman, J. H. Friedman, R. A. Olshen and C. J. Stone, *Classification and Regression Trees*, Wadsworth Inc., 1984.
5. F. De Carvalho and O. Gascuel, "SDL, a stochastic algorithm for learning decision lists with limited complexity", *Ann. Math. Artif. Intell.* 10 (1994) 281–302.
6. J. Catlett, On changing continous attributes into ordered discrete attributes", *Lectures Notes in AI* **482** (1991) 164–178.
7. C. Chang, "Finding prototypes for nearest neighbor classifiers", *IEEE Trans. Comput.* **26** (1974) 1179–1184.
8. S. L. Chung and R. Setiono, "Efficient neural network training on a cray Y-MP", *Int. J. High Speed Comput.* **7** (1995) 109–123.
9. P. Clark and T. Niblett, "Induction in Noisy domains", *Progress in Machine Learning*, eds. I. Bratko and N. Lavrac, Sigma Press, Wimslow, 1987, pp. 11–30.
10. M. Coster and J. L. Chermant, *Précis d'analyse d'images*, Presses du CNRS, Paris, 1989.
11. T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification", *IEEE Trans. Inform. Theor.* **13** (1967) 21–27.
12. Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L.D. Jackel, "Back-propagation applied to handwritten zipcode recognition", *Neural Comput.* 1 (1989) 541–551.
13. D. V. Dasarathy, *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, IEEE Computer Society Press, Los Alamitos, CA, 1991.
14. T. G. Dietterich and R. S. Michalski, "A comparative review of selected methods for learning from examples", *Machine Learning: An Artificial Intelligence Approach*, eds. R. S. Michalski, J. G. Carbonell and T. M. Mitchell, Tioga, 1983, pp. 41–81.
15. R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, John Wiley, 1973.
16. U. M. Fayyad and K. B. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning", *Proc. 13th Int. Joint Conf. Artificial Intelligence*, Chambery, Aug. 1993.
17. R. A. Fisher, "The use of multiple measurements in taxonomic problems", *Ann. Eugenics* **7** (1936) 179–188.
18. W. D. Fisher, "On grouping for maximum homogeneity", *J.A.S.A.* **53** (1958) 789–798.
19. H. G. Flegg, *L'algèbre de Boole et son Utilisation*, Dunod, Paris, 1967.
20. J. H. Friedman, J. L. Bentley and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time", *ACM Trans. Math. Software* **3** (1977) 209–226.
21. K. Fukunaga, *Statistical Pattern Recognition*, Academic Press, 1990.
22. O. Gascuel, "A way to give and use knowledge in learning", *Proc. 1st European Working Session on Machine Learning*, Orsay, March 1986.
23. O. Gascuel, "Quelques aspects numériques de l'analyse symbolique des données", *Induction Symbolique et Numérique*, eds. E. Diday and Y. Kodratoff, Cepadues-Editions, Toulouse, 1991, pp. 327–359.
24. S. Geeman, E. Bienenstock and R. Doursat, "Neural networks and the bias variance dilemna", *Neural Comput.* **4** (1992) 1–58.

25. A. Guénoche and P. Vitte, "Discriminant analysis for sequences analysis: application to protein kinases", Technical Report 126, Laboratoire d'Informatique de Marseille, May 1995.
26. H. Guo and S. B. Gelfand, "Classification trees with neural network feature extraction", *IEEE Trans. Neural Networks* **3** (1992) 925–933.
27. D. J. Hand, *Discrimination and Classification*, John Wiley, 1981.
28. D. J. Hand, "Recent advances in error rate estimation", *Patt. Recogn. Lett.* **4** (1986) 335–346.
29. P. E. Hart, "The condensed nearest neighbor rule", *IEEE Trans. Inform. Theor.* **14** (1968) 515–516.
30. D. Haussler, "Quantifying inductive bias: AI learning algorithms and valiant's learning framework", *Artif. Intell.* **36** (1988) 177–221.
31. J. Hertz, A. Krogh and R. G. Palmer, *Introduction to the Theory of Neural Computation*, Addison Wesley, 1991.
32. J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.
33. J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities", *Proc. National Academy of Sciences, USA* **79** (1982) 2554–2558.
34. K. De Jong, "Learning with genetic algorithms: an overview", *Mach. Learn.* **3** (1988) 121–138.
35. R. D. King, C. Feng and A. Sutherland, "STATLOG: Comparison of classification algorithms on large real-world problems", *Appl. Artif. Intell.* **9** (1995) 289–333.
36. J. Kuntzmann and P. Naslin, *Algèbre de Boole et Machines Logiques*, Dunod, Paris, 1967.
37. Y. Lechevallier, "Recherche d'une partition optimale sous contrainte d'ordre total", Technical Report 1247, INRIA, Rocquencourt, Nov. 1990.
38. I. C. Lerman, "Foundations of the likelihood linkage analysis (LLA) classification method", *Proc. Applied Stochastic Models and Data Analysis*, Wiley, 1991, pp. 63–76.
39. C. Marsala and B. Bouchon-Meunier, "Fuzzy partitioning using mathematical morphology in a learning scheme", *Proc. Int. Conf. Fuzzy Systems* (*FUZZ – IEEE'96*), New Orleans, 1996, pp. 1512–1517.
40. W. S. McCulloch and W. Pitts, "A logical calculus of ideas immanent in nervous activity", *Bull. Math. Biophys.* **5** (1943) 115–133.
41. R. S. Michalski, "A theory and methodology for inductive learning", *Machine Learning: An Artificial Intelligence Approach*, eds. R. S. Michalski, J. G. Carbonell and T. M. Mitchell, Tioga, 1983, pp. 83–134.
42. R. S. Michalski, I. Mozetic, J. Hong and N. Lavrac, "The multi-purpose incremental learning system AQ15 and its testing application to three medical domains", *Proc. AAAI-86 5th National Conference on Artificial Intelligence*, 1986.
43. T. Mitchell, "Generalization as search", *Artif. Intell.* **18** (1982) 203–226.
44. M. F. Möller, "A scaled conjugate gradient algorithm for fast supervised learning", *Neural Networks* **6** (1993) 525–533.
45. S. H. Muggleton, ed., *Inductive Learning Programming*, Academic Press, 1992.
46. J. Nicolas, "Une représentation efficace pour les Espaces de Versions", *Proc. Journées Francophone d'Apprentissage*, Apr. 1993.
47. J. Nicolas and I. C. Lerman, "Combining numeric and symbolic tools: a case study in pattern recognition", *Proc. Applied Stochastic Models and Data Analysis*, Wiley, May 1991.
48. R. Nock and O. Gascuel, "On learning decision committees", *Proc. Int. Conf. Machine Learning*, July 1995.

49. E. Parzen, "On estimation of a probability density function and mode", *Ann. Math. Stat.* **33** (1962) 1065–1076.
50. J. R. Quinlan, "Induction of decision trees", *Mach. Learn.* **1** (1986) 86–106.
51. J. R. Quinlan, "Probabilistic decision trees", *Mach. Learn.* **3** (1990) 140–152.
52. J. Quinqueton and J. Sallantin, "Expansion and compression of binary data to build features by learning", *Proc. 6th Int. Joint. Conf. Pattern Recognition*, Aug. 1983.
53. M. Ramdani, *Système d'induction formelle à base de connaissances imprécises*, Thèse de l'Université P. & M. Curie, 1994.
54. B. D. Ripley, "Neural networks and related methods for classification", *J. R. Stat. Soc.* **B56** (1994) 409–456.
55. R. Rivest , "Learning decision lists", *Mach. Learn.* **2** (1987) 229–246.
56. F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organisation of the brain", *Psychol. Rev.* **65** (1958) 386–407.
57. D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning internal representations by error propagation", *Parallel Distributed Processing*, Vol. 1, Chap. 8, MIT Press, Cambridge, 1986, pp. 318–362.
58. M. Sebag, "A constraint-based induction algorithm in FOL", *Proc. Int. Conf. Machine Learning*, Morgan Kaufmann, July 1994.
59. M. Sebag, "Delaying the choice of bias: a disjunctive version space approach", *Proc. Int. Conf. Machine Learning*, Morgan Kaufman, July 1996.
60. T. Sejnowski and C. Rosenberg, "Parallel networks that learn to pronounce English text", *Complex Syst.* 1 (1987) 145–168.
61. J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, New York, 1982.
62. W. Shavlik, R. Mooney and G. Towell, "Symbolic and neural net learning algorithms: an artificial comparison", *Mach. Learn.* **6** (1991) 11–143.
63. E. Shortliffe, *Computer Based Medical Consultations: MYCIN*, American Elsevier, New York, 1976.
64. J. A. Sonquist and J. N. Morgan, "The detection of interaction effects", Technical Report, Institute for Social Research, University of Michigan, Ann Arbor, 1963.
65. G. T. Toussaint, "Bibliography on estimation of misclassification", *IEEE Trans. Inform. Theor.* **20** (1974) 472–479.
66. P. E. Utgoff, "Incremental induction of decision trees", *Mach. Learn.* **4** (1989) 161–186.
67. P. P. Van der Smagt, "Minimisation methods for training feedforward neural networks", *Neural Networks* **7** (1994) 1–11.
68. V. N. Vapnik and Y. A. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities", *Theor. Probability Appl.* **16** (1971) 264–280.
69. G. Venturini, *Apprentissage adaptatif et apprentissage supervisé par algorithme génétique*, Thèse de doctorat, Université de Paris XI, Orsay, 1994.
70. S. A. Vere, "Inductive learning of relational productions", *Pattern-Directed Inference Systems*, eds. D. A. Watterman and F. Hayes-Roth, Academic Press, New York, 1978, pp. 281–296.
71. E. R. Vidal Ruiz, "An algorithm for finding nearest neighbors in (approximately) constant average time", *Patt. Recogn. Lett.* **4** (1986) 145–157.
72. I. Wegener, *The Complexity of Boolean Functions*, John Wiley, Chichester, 1987.
73. B. Widrow and M. E. Hoff, "Adaptative switching circuits", *IRE WESCON Convention Record*, Part 4, New York, 1960, pp. 96–104.
74. D. Wilson, "Asymptotic properties of nearest neighbor rules using edited data", *IEEE Trans. Syst. Man Cybern.* **2** (1972) 408–421.

75. P. H. Winston, "Learning structural descriptions from examples", *The Psychology of Computer Vision*, ed. P. H. Winston, Mc Graw Hill, New York, 1975, pp. 157–209.
76. L. A. Zadeh, "Fuzzy sets", *Information Control*, 1965, pp. 338–353.