Abstract—A Brain-Computer Interface (BCI) is an interface that directly analyzes brain activity to transform user intentions into commands. Many known techniques use the P300 event-related potential by extracting relevant features from the EEG signal and feeding those features into a classifier. In these approaches, feature extraction becomes the key point, and doing it by hand can be at the same time cumbersome and suboptimal. In this paper we face the issue of feature extraction by using a genetic algorithm able to retrieve the relevant aspects of the signal to be classified in an automatic fashion. We have applied this algorithm to publicly available data sets (a BCI competition) and data collected in our lab, obtaining with a simple logistic classifier results comparable to the best algorithms in the literature. In addition, the features extracted by the algorithm can be interpreted in terms of signal characteristics that are contributing to the success of classification, giving new insights for brain activity investigation.

I. INTRODUCTION

A Brain-Computer Interface (BCI) is an interface that does not entail muscle movements, but it bypasses any muscle or nerve mediation and connects a computer directly with the brain by picking up signals generated by the brain activity.

Electrical phenomena in the brain are known since the 19th century, but it was in the beginning of the 20th century that scientists began to study the activity of the brain with the help of electroencephalography (EEG), the recording of brain potentials by applying electrodes on the scalp. The relatively recent availability of powerful digital hardware and analysis algorithms opened the door to the use of EEG as a mean of communication, i.e., to BCIs.

There are different kinds of brain activity that can be used in a BCI. Different internal and external events cause different patterns in the brain waves, and many of these patterns have been studied for use in a BCI, as, for example, visual evoked potential (VEP), slow cortical potential (SCP), mu and beta rhythms [1]. In this study, we focus on the P300, an event-related potential (ERP) that is visible in an EEG recording as a positive peak at approximately 300 ms from the event (see Figure 1). It follows unexpected, rare, or particularly informative stimuli, and it is stronger in the parietal area. Stimuli can be visual, auditory, or even tactile. As the P300 has been known for many years [2], it has been extensively studied. The exact shape of the P300 depends on the characteristics of the stimuli and their presentation.

Bernardo Dal Seno and Matteo Matteucci are with the Department of Electronics and Information, Politecnico di Milano, IIT-Unit Italy (email: {dalseno,matteucci}@elet.polimi.it). Bernardo Dal Seno is also with IIT (Italian Institute of Technology). Luca Mainardi is with the Department of Bioengineering, Politecnico di Milano, IIT-Unit, Italy (email: luca.mainardi@biomed.polimi.it).

The P300 has been widely used for BCIs, with many variations, but in all cases the paradigm is the same: the BCI system presents the user with some choices, one at a time; when it detects a P300 potential the associated choice is selected. The user is normally asked to count the number of times the choice of interest is presented, so as to remain concentrated on the task. As the P300 is an innate response, it does not require training on part of the user.

As far as BCI applications are concerned, the exact shape of the P300 is not so important, as long as there is a way of detecting it, and algorithms that adapt to the particular shape of a user are needed. Detecting a P300 in a single trial is very difficult; therefore, repeated stimuli are normally used to select the stimulus most likely to have generated a P300. The number of repetitions can be predetermined for each user to get the best trade-off between speed and accuracy.

In [3], Donchin and colleagues presented the first P300-based BCI, called also P300 speller, which permits to spell words. A grid of letters and symbols is presented to the user, and entire columns or rows are flashed one after the other in random order (see Figure 2 for an example). Each one of the 6 rows and 6 columns is flashed exactly once in the first 12 stimulations; then another round of 12 stimulations is repeated, with flashing of rows and columns done in a new random order, and this procedure is repeated for a predefined number of times for each letter. Epochs 1.1 s long are extracted around each stimulation, and classification is made through Stepwise Discriminant Analysis (SWDA) applied to averages of samples from epochs relative to the same stimulation (same row or same column).

In [4], a virtual-reality system is presented where subjects operates objects selected through the P300. Classification is made by comparing the correlation of single responses with the averages of all target and nontarget responses.

In [5], tests have been made both with healthy and
impaired subjects. The subjects control a cursor by choosing among four commands (up, down, left, right) via the P300. Single-sweep detection is performed; independent component analysis (ICA) is used to decompose the EEG signal, a fuzzy classifier identifies a candidate P300 component among the ones extracted by ICA, and a neural network classifies it as target or non-target. The system is more effective with healthy subjects, though no exact reason could be pinpointed.

In [6], an initial attempt at using a BCI in a home environment is reported: a person with ALS uses a P300 speller on a daily basis. The system is very similar to the original Donchin’s speller, with a few differences in the detection algorithm.

Many techniques for detecting the P300 extract relevant features from the EEG signal and feed those features into a classifier. In these approaches, feature extraction becomes the key point, and doing it by hand can be at the same time cumbersome and suboptimal. In this paper we face the issue of feature extraction by using a genetic algorithm able to retrieve the relevant aspects of the signal to be classified in an automatic fashion.

In the following section, we detail the genetic algorithm proposed to automatically extract useful features for P300 classification. Section III shows the performance of the algorithm with data from the 2003 BCI Competition and data collected in our lab. In Section IV, we give an interpretation of the feature extracted by the genetic algorithm, while Section V proposes some concluding remarks suggested by such interpretation.

II. FEATURE-EXTRACTING GENETIC ALGORITHM

We have developed a system for the automatic feature extraction of P300s based on a genetic algorithm. Genetic algorithms (GAs) are a class of optimization algorithms that mimic — in some respects — the way natural evolution works through the “survival of the fittest” principle [7].

We developed a “shallow” approach where a genetic algorithm operates on very simple features extracted to be used for the classification of P300 epochs, with almost no preprocessing. We took inspiration from previous works [8], [9], which classify epochs in an effective way without developing the usual chain of information enhancement based on preprocessing, feature extraction, and classification. In our case, features are encoded in variable-length chromosomes, where each gene encodes one feature, and the fitness of an individual is given by the performance of a classifier trained on the encoded features.

GAs have been used already in the BCI field; in [10], the best combination between different features and different classifiers is sought for a motor-imagery task. In [11], a classifier operating on P300 features is selected by a GA.

A. Logistic Classifier

In this particular implementation, features are “tuned” on the use of a simple logistic classifier. If $x \in X$ is a vector of features representing an object and $y \in Y = \{-1, +1\}$ is the label of the object, a logistic classifier [12] approximates the probability $P(y \mid x)$ with a logistic function:

$$P(y = +1 \mid x) = \frac{1}{1 + \exp(w_0 + \sum_{j=1}^{n} w_j x_j)} \tag{1}$$

$$P(y = -1 \mid x) = 1 - P(y = +1 \mid x) = \frac{\exp(w_0 + \sum_{j=1}^{n} w_j x_j)}{1 + \exp(w_0 + \sum_{j=1}^{n} w_j x_j)} \tag{2}$$

where $x_j$ are the $n$ components of the vector $x$. The decision of the class to assign to a given sample $x$ is taken by comparing the two probabilities $P(y = -1 \mid x)$ and $P(y = +1 \mid x)$. The parameter vector $w$ can be found by maximizing, by using gradient ascent, its log-likelihood, with a term added to penalize large values of $w$ components:

$$L^{(\lambda)}(w) = \sum_{i=1}^{N} \log P(y_i \mid x_i, w) - \lambda \|w\|^2. \tag{3}$$

B. Encoding

The chromosome of each individual in a population encodes a set of features, and its logical structure is shown in Figure 3. A chromosome contains a variable number of genes, with an identical structure, and each gene is formed by five elements. The first three elements define a feature: the first one is an integer designating one feature extractor out of...
a predetermined set, while the two following elements encode two real-valued parameters for such an extractor. Feature extractors are functions with three arguments: a signal from which a feature is extracted, and the two parameters encoded in genes; these parameters are within the range \([0, +1]\), and their actual meaning varies from extractor to extractor. The fourth element of a gene is an integer number, which identifies the EEG channel where the feature encoded in the gene is to be extracted from. The last element of a gene is a Boolean flag that determines whether the gene is active or inactive. Inactive genes are not used to compute the fitness of a chromosome. Their role is of a genetic reserve, as they can be turned on in a later generation by mutation. The position of a gene within a chromosome is not significant.

We used up to six different feature extractors, which share a very simple scheme: the input signal is multiplied by a weight function, and the result is integrated over time. In other words, feature extractors compute the cross-correlation between the input and a weight function. If we call \(s(t)\) the EEG signal from the channel the feature is to be extracted from, a feature extractor constructs a weight function \(w(t)\) from the parameters specified by the gene elements and then computes the resulting feature \(x\) with the formula:

\[
x = \sum_{t=1}^{T} u(t)s(t).
\]

The six weight functions used by the six feature extractors are shown in Figure 4. The feature that uses the weights shown in the top-right box is proportional to the average of the input signal over an interval; the extremes of the interval are determined by the two parameters (A1 and A2 in the figure) encoded in genes. The weights in the top-left box produce a similar effect, but the samples at the center of the interval weight more. The functions in the middle row compute the differences between two adjacent intervals; again the extremes are encoded in genes. The functions in the bottom row compute the cross-correlation with a sine wave; genes encode frequency and phase of the sines. The interval where the bottom-right weight function is not zero is fixed, and it goes from 0 to 600 ms after the stimulus, i.e., it is centered around the P300. These last two functions permit to do a sort of frequency analysis of the signal.

\[\sum_{t=1}^{T} u(t)s(t)\]

C. Fitness and Selection

The fitness of a chromosome is determined by measuring the performance of a logistic classifier on the features it encodes. To have a fair estimate of the performance, a 4-fold cross-validation scheme on the training set is used, and the mean performance on the 4 folds is used as the fitness. The actual criterion used to evaluate the “performance” depends on the kind of data. For data recorded with a P300 speller, the number of correctly predicted letters is used, with a little bonus for letters that can be correctly predicted with less than the maximum number of repetitions, i.e., the number of times the whole grid is flashed for each letter. Let us call \(l\) the number of correctly predicted letters out of a total of \(n\), \(N\) the number of repetitions in the data set, and \(r_i, i = 1 \ldots n\), the number of repetitions needed for the prediction of the letter \(i\). The fitness \(f\) is then given by

\[
f = \frac{1}{n} \left( I + \frac{1}{l} \sum_{i \in I} \frac{N - r_i}{N + 1} \right),
\]

where \(I\) is the set of correctly predicted letters. The second term in the parentheses computes an index, averaged over the \(l\) correct letters, that grows with the decreasing of \(r_i\); this index is always strictly less than 1, and therefore it contributes to the fitness less than a single correctly predicted letter. In this way, a higher number of correct letters is always preferred to a lower number of repetitions needed for correct prediction.

Repetitions are taken in their natural order, and \(r_i\) is computed in way such that if a letter is correctly predicted by using the first \(r_i\) repetitions, then it must be correctly predicted also by using the first \(r_i + 1, \ldots, N\) repetitions. In other words, if a letter were predicted correctly after 3 repetitions, wrongly after 4, and again correctly when using 5 repetitions or more, then \(r_i\) would be 5, and not 3.

The fitness function of our genetic algorithm can be easily changed without modifying anything else in the algorithm. This permits to adapt the fitness computation to a different BCI task, and if there are no letters to spell, other measures like accuracy, recall, or mutual information can be used.

In a genetic algorithm, fitness is used to select the most promising individuals for the next generation. The selection mechanism employed is tournament selection with elitism, a standard setup in genetic algorithms, with no particular adaptation. Briefly, in tournament selection each individual of the new population is selected by setting up a tournament: a fixed number \(k\) of individuals are chosen at random from the old population, and the one with the highest fitness is declared the winner and will get in the next generation. Values for \(k\) are usually small, as large values would favor the fittest individuals too much, causing a loss of diversity in the population; in our work, \(k = 4\). Elitism is the practice of keeping the fittest individual or individuals in the new generation, even when selection discarded them (e.g., because they never participated to any tournament), or mutation and selection modified them.
D. Genetic Operators

After selection, the selected population undergoes crossover and mutation. These two operators have been slightly modified in order to adapt them to the non-standard chromosome structure we employed. Figure 5 shows how crossover works. Crossover is applied to pairs of chromosomes in the selected population (chosen at random) with a probability of 0.7; both chromosomes are split in two sections at a gene boundary in a random way, and then the four sections are recombined. Because the order of genes in a chromosome is not important, one section from one chromosome can be coupled with either section from the other one, and so there are two different ways of doing crossover. Which way to use is randomly chosen each time, and it is important to use both ways, as this choice increases the mixing of the genetic material.

Crossover may be applied to individuals with a common ancestor, and so they may share some genes. In this case, it is very likely that at least one of the new chromosomes contains duplicated genes, and many duplicates accumulate with time. These duplicates are ignored for fitness evaluation.

Mutation (see Figure 6) operates on gene elements; for each element in each gene, a random choice is taken whether to mutate it, independently from each other, but with the same (small) probability, which is 0.005 in this algorithm. Elements are modified differently accordingly to their type. For a discrete element (extractor, channel, and active flag), mutation modifies it by choosing one of the other admissible values for that kind of elements, at random. For a continuous element (the two extractor parameters), a perturbation is added according to a Gaussian distribution; if the result lies outside the admissible interval [0, 1], it is wrapped around, e.g., a value of 0.95 which is perturbed by 0.07 does not result in a new value of 1.02, which is not legal, but it is wrapped to 0.02.

The use of normalized extractor parameters is useful because the way mutation works. When mutation is applied to the gene element that encodes the feature extractor, the parameters are always legal also for the resulting new feature extractor; moreover, in some cases the old and the new weight functions are similar, and this helps the GA.

E. Population Size and Stop Criterion

The size of the population is constant throughout a GA run; for our experiments we used populations ranging from 70 to 120 individuals. Apart from size, the initial population is completely random; the length, i.e., the number of genes, for each chromosome is extracted from a geometric distribution with mean 20. The actual values for gene elements are taken from uniform distributions over the whole range of legal values for each element.

The last component to complete the GA description is the stop criterion. We relied only on the number of generations, after some initial experiments where we noticed that in all runs no improvements could be seen in both the fitness of the best individual and the mean fitness of the population after 10–15 generations. Figure 7 shows how the fitness of a population evolves in a typical GA run; it is evident that the maximum fitness reaches a plateau after only 7–8 generations, and population fitness tends to stabilize around the 12th generation. This kind of behavior has been observed for all runs, with numbers varying little; for this reason, we decided to stop GA after 15 generations, or a couple of generations before for the most time-consuming runs. In any case, a check on the fitness growth is made after each run, so as to be sure that evolution has actually stopped: if the maximum and mean fitness has been constant for the last 3–4 generations, evolution is considered finished.

F. Feature Set Validation

After the end of each GA run, the performance of the individuals with a high fitness is validated on a test set, never used before by the GA. This validation is done on the individuals with a fitness at least 99% of the fitness of the best individual in the last generation. Evaluating more than one individual and not just the best one results in a more robust assessment of the effectiveness of the method.

For each individual, the features encoded by its chromosome are extracted from all the training data (i.e., the data used for fitness evaluation), and a logistic classifier is trained.
on them. The same features are extracted from the test set, and the classifier is evaluated on them. The classifier can also be used online, together with the feature extractors it was trained on.

With very large data sets (more than 10,000 epochs) the evaluation of a population of about 100 individuals could take more than an hour on a modern 32-bit processor running at 2 GHz. Yet, the feature extractors and the trained logistic classifier are very fast to apply (especially with the trick shown in Equation (16) in Section IV), and can be used online in real-time.

A possible problem of GAs is overfitting, as they are optimization algorithms. However, we have not observed overfitting in any of our classification experiments, probably because of the cross-validation used in the fitness evaluation combined with the noisiness of the EEG data.

III. EXPERIMENTS AND RESULTS

We tested the genetic algorithm described in the previous section on two different data sets. We used the data set IIb from the BCI Competition 2003 [13], [14], originally provided by the group lead by Jonathan R. Wolpaw at Wadsworth Center, NYS Department of Health. These data come from three recording sessions with a P300 speller, with a grid of 6 × 6 letters. Matrix flashing lasts for 100 ms, 75 ms separates two consecutive flashes, and 15 repetitions of the stimulation rounds are done for each letter.

Data were acquired from 64 EEG electrodes, in all the positions of the 10-10 system, with a sample frequency of 240 Hz. In the first two recording sessions, 11 words were spelled, for a total of 42 letters; the actual words spelled were made available for the competition, so the first two sessions constitute the training set. The third session, 8 words and 31 letters, is the test set. The words spelled in the third session have been made available after the end of the competition, and we used this piece of information for the evaluation of the best individuals generated by the GA.

We also recorded some data at Airlab (Artificial Intelligence & Robotics Laboratory) [15] at the Department of Electronics and Information of Politecnico di Milano (Italy), using a P300 speller with matrix flashing of 100 ms, 100 ms separating two consecutive flashes, and 5 repetitions per letter. Seven young healthy subjects (2 female and 5 male) participated to one or three sessions, each one consisting in the spelling of some words, for a total of about 150–200 letters per session. Data were recorded continuously for four EOG channels, Fz, Cz, Pz, and Oz of the 10-20 system, and one EOG channel, all sampled at 512 Hz.

We applied the genetic algorithm first on the BCI Competition 2003 data set IIb, to compare it with other classification methods. In the procedure we used for this data set, data is first decimated by a factor of 3, so as to reduce memory usage, segmented in epochs, and detrended. We initially used epochs 1.5 s long; in later experiments we shortened them to 1 s, from 200 ms before the stimulus to 800 ms after it, to speed up the computation. Of the 64 channels, only 10 have been used, in three different combinations: those lying on the midline of the skull (Fpz, Afz, Fz, Fcz, Cz, Cpz, Pz, Poz, Oz, Iz); ‘k’ = Fz, Cz, Pz, Oz, C3, C4, P3, P4, Po7, Po8; ‘v’ = Fz, Fc3, Fc4, Cz, Cp3, Cp4, Pz, Po3, Po4, Oz.

Epoch is ‘long’ for the (−0.5, +1.0) interval, and ‘short’ for (−0.2, −0.8).

Feature functions indicates the subset of the functions of Figure 4 used: ‘f2’ = triangle and square wave; ‘f3’ = triangle, square, and sine wave; ‘f5’ = all the functions except for the short sine wave.

Runs indicates the number of times the GA has run.

Test perf. is the number of correct letters in the test set.

<table>
<thead>
<tr>
<th>Norm.</th>
<th>Channels</th>
<th>Epoch</th>
<th>Feature Funcs</th>
<th>Runs</th>
<th>Test perf.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>z</td>
<td>long</td>
<td>f3</td>
<td>3</td>
<td>29–30/31</td>
</tr>
<tr>
<td>1</td>
<td>z</td>
<td>long</td>
<td>f3</td>
<td>1</td>
<td>28–30/31</td>
</tr>
<tr>
<td>4</td>
<td>z</td>
<td>long</td>
<td>f3</td>
<td>1</td>
<td>30–31/31</td>
</tr>
<tr>
<td>3</td>
<td>z</td>
<td>long</td>
<td>f5</td>
<td>2</td>
<td>28–30/31</td>
</tr>
<tr>
<td>3</td>
<td>k</td>
<td>long</td>
<td>f5</td>
<td>1</td>
<td>31/31</td>
</tr>
<tr>
<td>3</td>
<td>k</td>
<td>long</td>
<td>f3</td>
<td>1</td>
<td>31/31</td>
</tr>
<tr>
<td>3</td>
<td>k</td>
<td>short</td>
<td>f2</td>
<td>4</td>
<td>31/31</td>
</tr>
<tr>
<td>-</td>
<td>k</td>
<td>short</td>
<td>f2</td>
<td>7</td>
<td>31/31</td>
</tr>
<tr>
<td>-</td>
<td>v</td>
<td>short</td>
<td>f2</td>
<td>2</td>
<td>30–31/31</td>
</tr>
<tr>
<td>3</td>
<td>v</td>
<td>short</td>
<td>f2</td>
<td>2</td>
<td>26–31/31</td>
</tr>
</tbody>
</table>

Norm. shows the type of normalization: ‘-’ = no normalization; ‘1’ = the same factor is used for all channels of all epochs; ‘3’ = each channel of each epoch is normalized independently of the others; ‘4’ = within each epoch, the same factor is used for all channels, but epochs are normalized independently of each other.

Channels shows which channel combination has been used: ‘z’ = Fpz, Afz, Fz, Fcz, Cz, Cpz, Pz, Poz, Oz, Iz; ‘k’ = Fz, Cz, Pz, Oz, C3, C4, P3, P4, Po7, Po8; ‘v’ = Fz, Fc3, Fc4, Cz, Cp3, Cp4, Pz, Po3, Po4, Oz.

Epoch is ‘long’ for the (−0.5, +1.0) interval, and ‘short’ for (−0.2, −0.8).

Feature Funcs indicates the subset of the functions of Figure 4 used: ‘f2’ = triangle and square wave; ‘f3’ = triangle, square, and sine wave; ‘f5’ = all the functions except for the short sine wave.

Runs indicates the number of times the GA has run.

Test perf. is the number of correct letters in the test set.
cross-correlation of the signal with a weight function \( u_j(\cdot) \):

\[
x_j = \sum_{t=1}^{T} u_j(t) s(t),
\]

where \( j \) means that \( x_j \) is the feature encoded by the \( j \)-th gene, and \( T \) is the number of time samples per epoch. Please notice that \( u_j(\cdot) \) is not a generic function, but the precise function encoded by all the parameters contained in a gene. A trained logistic classifier estimates the probability for the signal \( s(\cdot) \) to belong to a class according to Equation (1):

\[
P(y = +1 | x) = \frac{1}{1 + \exp(w_0 + \sum_{j=1}^{n} w_j x_j)},
\]

where \( x_j \) are the features given by Equation (6). Equation (7) gives the probability for the target class, which is enough because the probability for the other class can be computed by difference: \( P(y = -1 | x) = 1 - P(y = +1 | x) \).

It is possible to rewrite the argument of the exponential in Equation (7) by substituting the (6) in it:

\[
w_0 + \sum_{j=1}^{n} w_j x_j = w_0 + \sum_{j=1}^{n} w_j \left( \sum_{t=1}^{T} u_j(t) s(t) \right) = w_0 + \sum_{t=1}^{T} \left( \sum_{j=1}^{n} w_j u_j(t) \right) s(t).
\]

The term

\[
v(t) = \sum_{j=1}^{n} w_j u_j(t),
\]

with \( t = 1 \ldots T \), depends only on the feature set (through \( u_j(\cdot) \)) and on the classifier (through \( w \)), and therefore it is the same for all epochs. Equation (7) becomes

\[
P(y = +1 | s(\cdot)) = \frac{1}{1 + \exp(w_0 + \sum_{t=1}^{T} v(t) s(t))},
\]

or, by considering \( v(\cdot) \) and \( s(\cdot) \) as vectors, whose components are the time sample, and using the dot product notation:

\[
P(y = +1 | s) = \frac{1}{1 + \exp(w_0 + (v, s))}.
\]

Let us return to consider all the channels. Let \( s_{c}(\cdot) \) denote the signal for channel \( c \), with \( c = 1 \ldots C \), and \( C \) is total number of channels, and let also \( c(j) \) be the channel the \( j \)-th feature is to be extracted from. Equation (6) becomes

\[
x_j = \sum_{t=1}^{T} u_j(t) s_{c(j)}(t),
\]

and from Equation (8) we have

\[
w_0 + \sum_{j=1}^{n} w_j x_j = w_0 + \sum_{j=1}^{n} w_j \left( \sum_{t=1}^{T} u_j(t) s_{c(j)}(t) \right) = w_0 + \sum_{t=1}^{T} \left( \sum_{j=1}^{n} w_j u_j(t) s_{c(j)}(t) \right).
\]

TABLE II
RESULTS OF GA ON THE AIRLAB DATA SET.

<table>
<thead>
<tr>
<th>Subject</th>
<th>No. of sessions</th>
<th>Training set size</th>
<th>Test performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIRLAB S1</td>
<td>3</td>
<td>429</td>
<td>121–126/143 (85–88%)</td>
</tr>
<tr>
<td>AIRLAB S2</td>
<td>3</td>
<td>165</td>
<td>24–27/31 (80–90%)</td>
</tr>
<tr>
<td>AIRLAB S3</td>
<td>3</td>
<td>409</td>
<td>89–95/165 (54–58%)</td>
</tr>
<tr>
<td>AIRLAB S4</td>
<td>3</td>
<td>155</td>
<td>17–18/30 (57–60%)</td>
</tr>
<tr>
<td>AIRLAB S5</td>
<td>3</td>
<td>344</td>
<td>80–85/199 (40–45%)</td>
</tr>
<tr>
<td>AIRLAB S6</td>
<td>1</td>
<td>175</td>
<td>14–16/23 (61–70%)</td>
</tr>
<tr>
<td>AIRLAB S7</td>
<td>3</td>
<td>396</td>
<td>46–52/135 (34–39%)</td>
</tr>
</tbody>
</table>

IV. FEATURE INTERPRETATION

While analyzing the features found by the GA, we realized that it is possible to do better than a simple list or a graph where a 2-dimensional map highlights the most important time interval for every channel. The GA deals not only with features, it optimizes the features for the classifier used in the fitness function as well. So, features are strongly connected with the classifier, and with some algebra it is possible to explicit this connection and understand the real meaning of what the GA finds.

Let us call \( s(\cdot) \) an EEG signal from a single channel of an epoch, and consider only the features extracted by the GA for this channel. For feature extractors like the one described above, a feature is obtained by computing the speed up the computation of the fitness function and reduces the memory usage.

The number of the features found by the GA ranges from 100 to 150, more or less, but in general the chromosomes contain a number of genes about three times as bigger, because some genes are inactive, and duplicates crop up because of recombination.

Table II shows the results obtained on data from the P300 speller experiment run in our laboratory, where Training set size is the number of letters spelled in the training set, and the performance is given as the number of correctly predicted letters over the total numbers of letters in the test set. The test set was composed by the last session or the last words “spelled” by each subject. Data were decimated with a factor of 4, resulting in a frequency of 128 Hz, segmented in epochs from 200 ms before up to 800 ms after stimulus, and detrended; the EOG was not used. As before, the performance on the test set is given as the range of the values obtained for the GA individuals with at least 99% of the maximum fitness; the results come from two different runs of the GA. Two subjects achieved very good results, with more than 80% of correct letters; another subject achieved correct classification rates above 60%, which is still a good result, and the other four subjects obtained less impressive figures, although still much above the level of a random classifier, which is about 3%. Given that we used only 4 channels and 5 repetitions, the overall performance of the GA is pretty good.
If we split the inner summation by grouping features relative to the same channel, we get

\[ w_0 + \sum_{t=1}^{T} \left( \sum_{j=1}^{n} w_j u_j(t) s_{c(j)}(t) \right) \]

\[ = w_0 + \sum_{t=1}^{T} \left( \sum_{c=1}^{C} \sum_{j: c(j)=c} w_j u_j(t) s_{c}(t) \right) \]

\[ = w_0 + \sum_{c=1}^{C} \sum_{t=1}^{T} \left( \sum_{j: c(j)=c} w_j u_j(t) \right) s_{c}(t) . \]

Again, there is a term that is the same for all epochs, but now it depends on the channel:

\[ v_c(t) = \sum_{j: c(j)=c} w_j u_j(t) , \]

with \( t = 1 \ldots T \) and \( c = 1 \ldots C \).

Putting everything together, in vector notation:

\[ P(y = +1 | s_1, \ldots, s_C) = \frac{1}{1 + \exp(w_0 + \sum_{c=1}^{C} \langle v_c, s_c \rangle)} . \]

This formula estimates the probability for the target class directly from the epoch signals \( s_1, \ldots, s_C \). The dot product \( \langle v_c, s_c \rangle \) is actually a correlation, as elements of these vectors are time samples, and we can see Equation (16) as a classification based on the similarity of epoch signals with some templates \( (v_c) \). These templates can be considered as the real output of a GA run.

This explains the fact that after some experiments with or without using some of the feature extractors, we found that the first two functions in Figure 4 (the triangle and the rectangle) were sufficient. The reason is now clear: if templates are the real output of GA, what the individual features are is not important, as long as they allow the building of complex enough templates.

V. DISCUSSION AND CONCLUSIONS

Figure 8 shows the ten templates (green solid line) for the ten channels used for the BCI competition data set, obtained in two different runs of the GA with the same preprocessing: same channel selection, decimation by a factor of 3, and detrending. The averages for target and nontarget epochs are also shown, for reference. Templates have been multiplied by a scaling factor to show them in the same graph as the averages; the same factor has been used for all templates, so no distortion has been introduced.

Both template sets shown in the figure achieved 100% correct letters, yet they are very different. There are some common characteristics, like the negative deflection around 300 ms in Pz, and the positive peak followed by a negative one between 300 and 500 ms in Po7. The negative peak of Pz is present in all the templates that achieved the maximum fitness we have looked at, and is much in accord with the most important feature of the P300 found in the literature:
a positive peak about 300 ms after the stimulus, stronger in the parietal region. The sign of the template is negative because the logistic function in (16) reaches the maximum for negative values of the exponent.

There also some peculiarities in the templates. For example, there is a strong difference between the two averages in channels Fz, Cz, and C4, but it is ignored in all the examples shown. Yet, the template for C4 in left graph of Figure 8 shows a strong peak at 550 ms after the stimulus, where the averages almost coincide. This is probably caused by the presence in the training set of epochs for which the difference between target and nontarget signals is significant at 500 ms in C4, and also the features that are more discriminant for other epochs are less effective. Verifying such an explanation is not easy, though, and also difficult is to assess the real impact of a template on the classification, as neither the templates nor the EEG channels can be considered independent. Some hints may come from the comparison of the templates from many good-performing individuals; for example, the fact that the template signals for either channel P07 or P08 are rather strong suggests that the activity in the visual cortex is important for the correct classification.

The last consideration suggests that the interpretation of features and weights as templates could permit to select the most important channels for classification. An objective measure of the relative contribution of the various channel could be extracted from Equation (16).

The templates obtained by the GA show some similarities with the P300 averages in some time intervals, so we wondered if the P300 averages could be used as templates in Equation (16). Averages cannot be put in the formula directly, because templates are on a different scale, and therefore we considered the cross-correlation of epoch signals with P300 averages as features to be used in a logistic classifier. The results for such a system are shown in Table III, in the column Template/Difference. They have been obtained by using the very same training and test sets as for the results shown in the previous pages, and so a direct comparison with the GA results is possible (shown in the column GA). Using P300 averages as templates performs worse than the GA, and the use of the differences between the averages of P300 epochs and the averages of non-P300 epochs as templates (shown in column Template/Difference) does not improve the results.

The interpretation of the features extracted by the genetic algorithm shows how the parts of an EEG signal that are most useful for the detection of P300 are not always those expected from information found in the literature. The genetic algorithm works blindly, and it is not tailored for a particular potential: results indicate that it can be effective not only for P300, but also for error-potential detection.

### Acknowledgments

This work has been partially supported by the Italian Institute of Technology (IIT). We are thankful to EBNeuro S.p.A. [16] for the use of their BE Light EEG system and their support. The classification software uses the BioSig toolkit [17] to import EEG data, and Weka [18], a collection of machine learning algorithms in Java, for classification. We are thankful to Gianmaria Visconti for his help in EEG data acquisition, and to all the participants in our study.

### References


