A Comparison of Three Methods with Implicit Features for Automatic Identification of P300s in a BCI

Luigi Sportiello, Bernardo Dal Seno, and Matteo Matteucci

Politecnico di Milano, Department of Electronics and Information, IIT-Unit, Piazza Leonardo da Vinci 32, 20133 Milano, Italy

Abstract. When using a pattern recognition technique to classify signals, a common practice is to define a set of features to be extracted and, possibly after feature selection/projection, to use a learning machine in the resulting feature space for classification. However it is not always easy to devise the "right" set of features for a given problem, and the resulting classifier might turn out to be suboptimal because of this, especially in presence of noise or incomplete knowledge of the phenomenon. In this paper we present an off-line comparison of three methods (genetic algorithm, time-delay neural network, support vector machines) that leverage different ideas to handle features; we apply them to the recognition of the P300 potential in an EEG-based brain-computer interface. They all performed good, with the genetic algorithm being slightly better.

Keywords: Brain-Computer Interface, P300 Potential, Time-Delay Neural Networks, Genetic Algorithms, Support Vector Machines.

1 Introduction

A brain-computer interface (BCI) is an interface that does not entail muscle movements, but it bypasses any muscle or nerve mediation to connect a computer directly with the brain through the signals generated by the brain activity. Among the different kinds of brain activity that can be used in a BCI, the *P300* phenomenon has been known [1] and investigated for many years. It is an event-related potential (ERP), visible in an EEG recording as a positive peak at approximately 300 ms from the event. It follows unexpected, rare, or particularly informative stimuli, and it is stronger in the parietal area. The shape of the P300 depends on the characteristics of the stimuli and their presentation.

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 (although the counting is not mandatory). As the P300 is an innate response, it does not require training on part of the user, but algorithms must adapt to the particular shape of each user's P300 in order to detect it. Detecting a P300 in a single trial is very difficult

C. Alippi et al. (Eds.): ICANN 2009, Part II, LNCS 5769, pp. 515–524, 2009.

and, therefore, repeated stimuli are normally used to facilitate the detection of the correct stimulus. The number of repetitions can be predetermined for each user to get the best trade-off between speed and accuracy.

In [2], Donchin and colleagues presented the first P300-based BCI, called also P300 speller, which permits to spell words. A 6×6 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 Fig. 1 for an example). When the column/row containing the desired letter is flashed, a P300 is elicited. 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. In [2], 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).

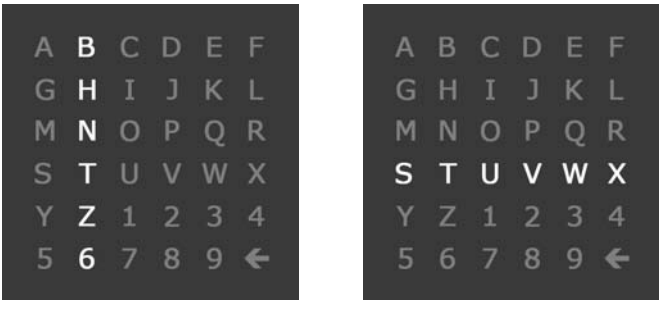

Fig. 1. The visual stimulator for a P300-based BCI speller. Lines and rows are flashed one at a time in random order, and the intersection between the row and the column that elicit a P300 indicates the selected letter.

P300 has been used in other BCI experiments, with different classification techniques. In [3], a virtual-reality system is presented where subjects operate objects selected through the P300. Classification is made by comparing the correlation of single responses with the averages of all target and non-target responses. In [4] a wheelchair driven by a BCI is described, and P300 is recognized through a genetic algorithm. In [5], the subjects (healthy and impaired ones) 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 signals, a fuzzy classifier selects one of the components 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.

Often in the literature (and in the works cited in the previous review) when a pattern recognition technique is used to recognize the P300 signal, a two-stage process is employed: feature extraction, followed by classification in this new feature space. This approach starts with the definition of a suitable set of features to be extracted from the signal and, possibly after feature selection/projection, it applies a learning machine on those features in order to generate a classifier. Especially when the signal to noise ratio is very low and the a-priori information/knowledge about the observed phenomenon is only partial, it is not always easy to define the "right" set of features for a given signal or problem, and the resulting classifier might turn out to be suboptimal due to the suboptimal choice of the feature space. In these situations, the use of methods able to automatically find the best features for classification is of uttermost importance.

In this paper we present a comparison of three methods for implicit-feature classification applied to the recognition of the P300 event-related potential in an EEG based brain-computer interface. These methods either use the raw signals for classification or implement an implicit, data driven, feature extraction that is not based on any knowledge about the phenomenon, but it is automatically optimized for the classification task. These methods are genetic algorithms, time-delay neural networks and support vector machines, and they are described in details in the next section. In Sect. 3, we present and briefly discuss experimental results on real data coming from an international BCI competition and from recordings at the AIRLab of Politecnico di Milano.

2 Models and Methods

In this section the three methods for implicit-feature classification of the P300 are presented. Two methods, one based on a genetic algorithm and one based on time-delay neural networks, have been developed by our group; the third, based on support vector machines, has been replicated from the literature. Our method to recognize P300s with a time-delay neural networks is presented here for the first time, while the other methods have been already published. All methods can be applied in real time on recent processors.

2.1 Genetic Algorithm

We applied the genetic algorithm described in [7] to data recorded from people using a P300 speller in an offline fashion. In this section, only a very brief description of the algorithm is given; details are given for the fitness function, as it differs from the one used in the cited work.

Genetic algorithms are a class of optimization algorithms that mimic the way natural evolution works. In a genetic algorithm, a set of possible solutions to an optimization problem are coded in strings called *chromosomes*; solutions are evaluated, and the best ones (those with highest *fitness*) are selected and combined together to form new possible solutions. After enough repetitions of this procedure, good solutions emerge.

In the genetic algorithm used in this work, each individual (chromosome) represents a set of possible features for discriminating the presence of a P300 in epochs of EEG recordings. Each gene encodes a feature and an EEG channel from which to extract it; a feature is obtained by multiplying the EEG channel by a weight function (see Fig. 2 for two examples), whose exact shape is encoded by parameters in the genes. Genetic operators are variants of 1-point crossover and uniform mutation, and tournament selection with elitism is used.

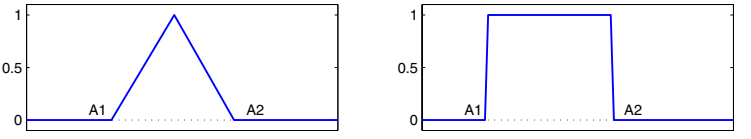

Fig. 2. Weight functions used for feature extraction

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 performance is measured as the number of correctly predicted letters, 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, $i = 1 \dots n$, the number of repetitions needed for the prediction of the letter i. The fitness f, used by the genetic algorithm, is then given by

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

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 occurring 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 $r_i + 1, \dots, N$ 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.

Typically a chromosome contains about 100–150 different unique features at the end of a run of the genetic algorithm; an analysis of the combination of the encoded features and the classifier trained on the training set allows to compute weights assigned to individual EEG samples (see Fig. 3).

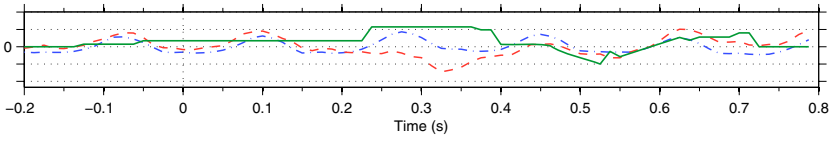

Fig. 3. The green solid line is an example of a template found by the GA for one EEG channel. For comparison, also the average of target (dashed red line) and non-target (dot-dashed blue line) responses are shown.

2.2 Time-Delay Neural Networks

Neural networks can be applied for the detection of patterns in signals. In case the signals to be processed are affected by translations and distorsions, the position of the patterns in the input may be subject to variation. In this case a neural network of relevant size may be required for the recognition task, with overfitting problem that may occur if the training data is scarce.

Time-delay neural networks (TDNNs) are multilayer backpropagation neural networks that present a modular architecture with replicated weights [8]. TDNNs use sliding windows to process input signals, with windows represented by sets of units in the input layer. The windows are partially overlapped and are connected to a set of neurons in the next layer, which use the windows as local receptive fields to look for local features in the input. All the windows share the same weights for the connections to the relative neurons, enabling the network to process several parts of the input in the same way. This replicated architecture allow to be invariant to shifts and distortions of the input, as local feature detectors scan the input. The structure is replicated in the following layers combining the extracted features of previous layers, with the network trained through a backpropagation algorithm. The reduced number of weights due to replication reduces the capacity of the network, improves its generalization ability, and tends to result in a reduced amount of data required for the training.

The idea is to feed a TDNN with raw EEG data, with the first layers that act as feature extractors, while the remaining part of the network performs the classification. As all the network parameters are set through the learning procedure, it would not be necessary to explicitly design a feature extractor.

In our application a TDNN has been designed to discern between presence or absence of a P300 in an EEG. For this classification task the window length in the input layer has been set to 400 ms, a time interval sufficient to contain a single elicited P300. Considering that a P300 peak is elicited roughly 300 ms after a stimulation, we use 1 s epochs derived from the EEG recordings in correspondence of each stimulus, starting from 200 ms before the stimulation. The derived patterns are classified by a 3-layer TDNN with a $[S \times C]$ unit matrix as input layer, where C represents the number of EEG channels, and S the number of EEG samples contained in the length of an epoch (see Fig. 4). According to the 400 ms time interval set to process the input, windows of size $[0.4S \times C]$ are applied in the input layer, with windows sliding one sample each time. Each

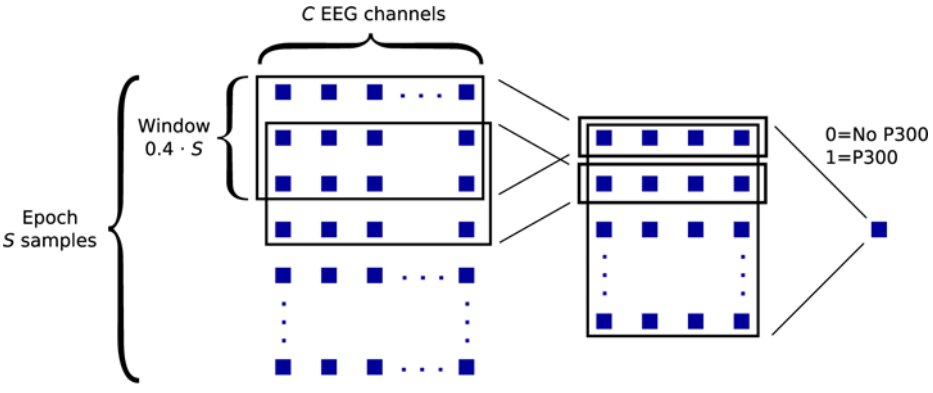

Fig. 4. The TDNN used to process EEG patterns

window is fully linked to 4 neurons in the hidden layer forming a $[(0.6S+1)\times 4]$ matrix. The hidden layer and the single output unit adopt the logistic function and are fully interconnected to perform the final classification.

As most of the examples in the data sets are relative to non-target stimuli, a balancing of the training and validation sets is necessary to avoid undesired bias towards non-target patterns. We have balanced the sets by discarding some non-target recordings so that the number of non-targets is the same as targets. For the definition and evaluation of the designed TDNNs, the Stuttgart neural network simulator (SNNS) tool and its Java counterpart, the JavaNNS [14], were adopted.

2.3 Support Vector Machines

We replicated also the method used by the winners of the BCI Competition 2003 [9,10], data set *IIb*, Kaper and colleagues from the University of Bielefeld, Germany [11]. Their method relies more on the power of the classifier employed, an SVM, than on signal processing, and the feature extraction is done implicitly by the SVM; in other words, this is a blind algorithm, which does not rely on specific knowledge about the P300.

A *support vector machine* (SVM) [12,13] is a supervised learning method used for classification and regression developed by Vladimir Vapnik in the late 1970s. In the simplest case, an SVM is a hyperplane in the space X of the samples $\boldsymbol{x}$. This hyperplane separates the space in two regions, one for each of the possible labels. Samples are assigned labels depending on which side of the hyperplane they lie (see Fig. 5). In formulas:

$$f(\boldsymbol{x}) = \operatorname{sign}(f^*(\boldsymbol{x})) \tag{2}$$

$$f^*(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b\,. \tag{3}$$

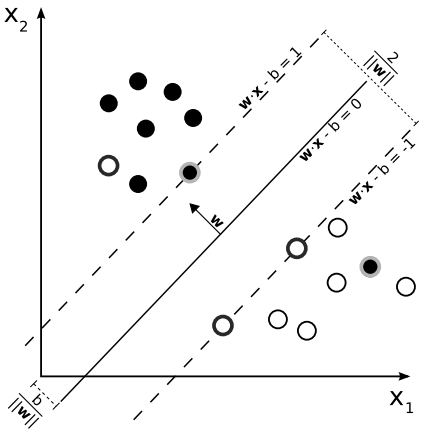

Fig. 5. SVM: the maximum-margin (optimal) hyperplane in the non-separable case. Support vectors are surrounded by a circle.

The hyperplane is found through an optimization algorithm in such a way to maximize the distance of the plane from the nearest samples (*margin* in SVM terminology) and minimize the number of misclassified samples. In mathematical terms, this goal can be written by introducing *slack variables* ξ_i, $i = 1 \dots N$ so that the SVM minimizes

$$\|\boldsymbol{w}\|^2 + C \sum_i \xi_i \tag{4}$$

subject to

$$y_i \left(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b \right) \geq 1 - \xi_i \tag{5}$$

$$\xi_i \geq 0 \tag{6}$$

The parameter C can be varied to shift the trade-off between margin and errors.

It is possible to extend the idea to non-linear SVMs by mapping samples $\boldsymbol{x}$ in a higher-dimensional space $\mathcal{H}$ by means of non-linear function $\boldsymbol{\Phi} : X \to \mathcal{H}$. The separating hyperplane is now to be found in $\mathcal{H}$. By using a $\boldsymbol{\Phi}$ such that $\mathcal{H}$ and $\boldsymbol{\Phi}$ satisfy Mercer's condition [13], it is possible to find a *kernel function* K such that $\langle \boldsymbol{\Phi}(\boldsymbol{x}_i), \boldsymbol{\Phi}(\boldsymbol{x}) \rangle = K(\boldsymbol{x}_i, \boldsymbol{x})$, where $K(\cdot, \cdot)$ is much easier to compute than the inner product in $\mathcal{H}$. The discriminating function becomes

$$f^*(\boldsymbol{x}) = \sum_i \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b \,, \tag{7}$$

where $\boldsymbol{x}_i$ are support vectors, i.e., the training samples closest to separating hyperplane and all the misclassified training samples, and α_i are coefficients found by the training process.

In Kaper's method, an epoch begins at the time of stimulus, and ends 600 ms after. Epochs are bandpass filtered between 0.5 and 30 Hz, and then normalized

to the $[-1, +1]$ interval. The training set is balanced by taking only two non-target examples from each repetition, which already contains exactly two target examples, and an SVM is trained directly on the balanced training set. In this case, the normalized EEG samples are used as features for the classification.

In order to find out the unknown letter in the test set, several repetitions are combined together. Starting from the first repetition, a score is assigned to each row and column. Scores from different repetitions are added together, and the row and the column with the maximum total score after the last repetition is selected and the corresponding letter is chosen. This procedure is repeated for each letter in the test set.

The score mentioned above is the SVM discriminant function $f^*(\cdot)$ in equation (7). In this case, the kernel function $K(\cdot)$ is Gaussian. The parameter σ of the kernel and the penalization coefficient C in the objective function of the SVM are found with a cross-validation scheme on the training set.

3 Experiments and Results

We applied the methods described above to two different data sets: the data set *IIb* from the BCI Competition 2003 [10], and recordings made at AIRLab, our laboratory. Both data sets consist in EEG recordings of subjects using a P300 speller like the one described in [2], made in three sessions; in our classification experiments, the first two sessions were used for training, and the last one was used for testing. For the competition data set, EEG was recorded from one subject with 64 electrodes in all positions of the 10-10 system and a sampling frequency of 240 Hz; stimuli were repeated 15 times for each letter. We used only channels Fz, Cz, Pz, Oz, C3, C4, P3, P4, Po7, and Po8, as they gave good results and they are the same selected by the competition winners [11]. Also, we recorded data from five subjects in our laboratory using four channels (Fz, Cz, Pz, Oz) at 512 Hz; stimuli were repeated only 5 times per letter. The two data sets differ also for the timing: a stimulus was given every 175 ms in the competition data set, and every 200 ms in AIRLab recordings. Care has been taken to avoid the presence strong 50 Hz noise in the data; spectral analyses confirmed this.

For GA and TDNN epochs started from 200 ms before each stimuli and were 1 s long, while for SVMs they started from the stimulus time and were 600 ms long. A decimation factor of 4 was used always for AIRLab data, resulting in a 128 Hz frequency, while data from the competition were decimated with different factors because of memory constraints; a factor of 3 (resulting in 80 Hz) was used for GA, no decimation was applied for SVMs, and a factor of 2 (down to 120 Hz) was used for TDNNs.

For all three classification methods, the test results are computed in terms of correctly classified letters in the test set, the same criterion used in the competition. Letters are selected by considering all repetitions for each letter: Classification results related to the same stimulus (i.e., same column or row) are added together, and the row and the column with the highest score are selected. Table 1 shows the test results of the three methods applied to data sets described

Table 1. Test results of the three methods applied to different data from P300 speller tasks

Data set / Subj.	GA	TDNN	SVM
Competition	31/31 (100%)	30/31 (97%)	31/31 (100%)
AIRLab S1	121–126/143 (85–88%)	118/143 (83%)	119/143 (83%)
AIRLab S2	89–95/165 (54–58%)	92/165 (56%)	89/165 (54%)
AIRLab S3	91–94/144 (63–65%)	65/144 (45%)	64/144 (44%)
AIRLab S4	80–85/199 (40–43%)	64/199 (32%)	92/199 (46%)
AIRLab S5	46-52/135 (34–39%)	41/135 (30%)	51/135 (38%)

above as the ratio (and the equivalent percentage) of correct letters over the total number of letters in the test sets. For the GA, a range of values is shown instead of a single number, as we have decided to show the entire range of values obtained for all chromosomes with a fitness that is at least 99% of the top fitness. In fact, a single GA run returns many different chromosomes that reaches the top fitness and all these chromosomes are closely related, due to the mixing of the genetic material. For this reason, it is not possible to summarize their performance in one number, as by averaging, for example.

The results of the three methods are similar, though their relative strength vary a little depending on the data set. Overall, the GA obtains the best results, while the results of SVMs are slightly better than those of TDNNs. The performances on some subjects are very low for all methods, but that is expected, because for the AIRLab data set only 5 repetitions and 4 EEG channels are used; such a "hard" data set makes it easier to compare the various methods.

The three methods implement the idea of implicit or automatically optimized features in different ways. The genetic algorithm uses a fixed classifier family and features are in a way created so as give the optimal classification results. Although features are involved, they are not chosen in advance, and when combined with the logistic classifier they naturally lead to a featureless template. In the TDNN model, features can be thought of as if learned implicitly in the weights of the neurons. In the case of an SVM, features cannot be pinpointed in any particular aspect of the model.

Beside the architecture presented in Sect. 2.2, we examined other network topologies for TDNNs, varying the number and the sizes of the hidden layers, increasing or decreasing the epoch length and the window size, and using neurons in the hidden layer connected to input-layer neurons relative to different numbers of channels. The results were always worse than for the proposed TDNN, or at most comparable with it.

The fact that the GA performed better than TDNNs and SVMs may seem strange, as the discriminant function found by the GA is linear, while the other two methods can find a nonlinear separation. There are two possible explanations for this: Either the training of TDNNs and SVMs makes a bad use of the available degrees of freedom because it needs a better tuning, or the highly noisy nature

of the problem calls for simpler classifier. Possibly, further experiments could tell which is the best explanation.

References

1. Sutton, S., Braren, M., Zubin, J., John, E.R.: Evoked-potential correlates of stimulus uncertainty. Science 1187, 1187–1188 (1965)
2. Donchin, E., Spencer, K.M., Wijesinghe, R.: The mental prosthesis: Assessing the speed of a P300-based brain-computer interface. IEEE Trans. Rehabil. Eng. 8(2), 174–179 (2000)
3. Bayliss, J.D., Inverso, S.A., Tentler, A.: Changing the P300 brain computer interface. Cyberpsychol. & Behav. 7(6), 694–704 (2004)
4. Blatt, R., Ceriani, S., Dal Seno, B., Fontana, G., Matteucci, M., Migliore, D.: Brain control of a smart wheelchair. In: Burgard, W., Dillmann, R., Plagemann, C., Vahrenkamp, N. (eds.) Intelligent Autonomous Systems 10 – IAS-10, Baden, Germany, pp. 221–228. IOS Press, Amsterdam (2008)
5. Piccione, F., Giorgi, F., Tonin, P., Priftis, K., Giove, S., Silvoni, S., Palmas, G., Beverina, F.: P300-based brain computer interface: reliability and performance in healthy and paralysed participants. Clin. Neurophysiol. 117(3), 531–537 (2006)
6. Vaughan, T.M., Mcfarland, D.J., Schalk, G., Sarnacki, W.A., Krusienski, D.J., Sellers, E.W., Wolpaw, J.R.: The Wadsworth BCI research and development program:At home with BCI. IEEE Trans. Neural Syst. Rehabil. Eng. 14(2), 229–233 (2006)
7. Dal Seno, B., Matteucci, M., Mainardi, L.: A genetic algorithm for automatic feature extraction in P300 detection. In: 2008 International Joint Conference on Neural Networks (IJCNN), Hong Kong, China, June 2008, pp. 3145–3152 (2008)
8. Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., Lang, K.J.: Phoneme recognition using time-delay neural networks. IEEE Trans. Acoust., Speech, Signal Process. 37(3), 328–339 (1989)
9. Blankertz, B.: BCI competition 2003 (2003), `http://ida.first.fraunhofer.de/projects/bci/competition_ii/`
10. Blankertz, B., Müller, K.R., Curio, G., Vaughan, T.M., Schalk, G., Wolpaw, J.R., Schlögl, A., Neuper, C., Pfurtscheller, G., Hinterberger, T., Schröder, M., Birbaumer, N.: The BCI competition 2003: Progress and perspectives in detection and discrimination of EEG single trials. IEEE Trans. Biomed. Eng. 51(6), 1044–1051 (2004)
11. Kaper, M., Meinicke, P., Grossekathoefer, U., Lingner, T., Ritter, H.: BCI competition 2003 data set IIb: support vector machines for the P300 speller paradigm. IEEE Trans. Biomed. Eng. 51(6), 1073–1076 (2004)
12. Vapnik, V.N.: An overview of statistical learning theory. IEEE Trans. Neural Netw. 10(5), 988–999 (1999)
13. Vapnik, V.N.: Statistical learning theory. Wiley, Chichester (1998)
14. Zell, A.: Stuttgart neural network simulator (SNNS) and Java neural network simulator (JavaNNS) (2009), `http://www.ra.cs.uni-tuebingen.de/SNNS/`