University of Reading

Cybernetics Project

MODELLING AN EPILEPTIC BRAIN PATTERN CLASSIFICATION

Telecommunication Engineering, Blanca Florentino Liaño
from Carlos III University of Madrid, lzu07bf@rdg.ac.uk (2008)

ABSTRACT

Supervised training and classification are heavily used in biomedical sciences. This paper describes two different classifiers, Support Vector learning Machine (SVM) and Artificial Neural Network (ANN) for detecting epileptic seizures in humans. SVM and ANN are evaluated for accuracy, specificity and sensitivity on classification of each patient into the correct group categorization: epileptic seizure or non-epileptic seizure. It is discussed how the model selection should be in order to achieve an acceptable rate of correct classification.

INDEX

1. INTRODUCTION

Since the initial discovery of the electrical activity of the brain, the ability to measure this activity using EEGs has been perfected. EEG technology is very inexpensive and accurately measures brain wave activity in the outer layer of the brain.

Epilepsy is a chronic illness identified by irregular occasion of unconsciousness, sometimes associated with violent convulsions. It is a pathological condition of a group of nerve cells firing harmoniously. With the default location of the nerve cells in the cortex, this will manifest in a measurable EEG. It is estimated that approximately 1 out of 2000 people per year will develop some type of epilepsy [1].

Because the EEG recordings are digitally recorded, it is possible to do a previous automatic analysis in order to make easier the work of technicians.

The amount of data presented to the technician can be reduced by a first pass on the data with automatic seizure detection. That allows more patients to be treated. A second aspect of automatic seizure detection is to attempt to extract from the data information that may be clinically important, but which cannot readily be discerned from simple visual inspection, even by a professional neurophysiologist.

The four basic steps to follow in order to process EEG signals are:

- Data acquisition: The EEG acquisition system is responsible for gathering and digitizing the EEG signals measured at the scalp. EEG signals are composed of the signals measured at different electrodes placed on the scalp.
- Artifact rejection: Severe contamination of EEG activity by eye movements, blinks, muscles, heart and line noise is a serious problem for EEG interpretation and analysis. Many methods have been proposed to remove these artefacts. The preferable alternative is to apply ICA to multichannel EEG recordings and remove a wide variety of artifacts from EEG records by eliminating the contributions of artifactual sources onto the scalp sensors [2]
- Feature extraction: Extract relevant information from the data and present it to the classifier.
- Classification: Detect epileptic patterns.

Numerous techniques from the theory of signal analysis have been used to obtain representations and extract the features of interest for classification purposes [1][3].

One of the aims of this research study is to identify a set of features derived from EEG which can be used to distinguish seizures from non-seizure section of EEG. Each sample (section) is labelled either as non-epileptic or as epileptic; this is possible because it is known a priori when a seizure takes place in the EEG signal. A classifier is trained with the labelled examples to create a set of rules or a mathematical model that can then look at the features captured from a new set of EEG samples and label the case as non-epileptic or epileptic (test phase). Then, the prediction is compared with the true label of the case, and if the two labels match, the classifier is said to have learned the concept.

Traditional classification architectures rely on empirical risk minimization algorithms. Statistical learning theory proposes a structural risk minimization criterion that balances the trade-off between good empirical performance (classification accuracy on training data) and good generalization capacity (classification accuracy on unseen data). One popular application of SRM is the Support Vector Machine (SVM), first presented in 1992

[4] The main idea behind the SVM is to find an hyperplane in a feature space that separates two classes. Many other linear learning machines have been considered for this task, but SVM yields a unique solution that can be shown to minimize the expected risk of misclassifying unseen samples. Training algorithms involve the solution of an optimization problem, constrained quadratic programming, which is computationally efficient and yields global solutions [5].

In order to compare the performance of the SVM, it is implemented a Neural Network, a Multilayer Perceptron (MLP). There are several successful attempts at seizure detection using artificial neural network classifiers. In [6] it was achieved a false detection rate of 1.0/h using ANN for automatic, on-line, seizure detection. In [7] the FFT and AR spectrums are used as an input to an artificial neural network and achieved a 92% of accuracy.

Nowadays in the machine learning technologies for pattern classification problems, the support vector machines have been extensively adopted for machine learning tools. In contrast to traditional neural networks, SVM achieves higher generalization performance, due to the utilization of structural risk minimization principle. [8]

The classifiers are evaluated for accuracy, specificity, and sensitivity on classification of each patient into the correct two-group categorization: epileptic seizure or non-epileptic seizure.

With the extracted features, described in [9], the SVM achieves an accuracy greater than 92% and a sensitivity of 0.9. However, using MLP is achieved an accuracy of 80% and a poor sensitivity of 0.6.

In this research project it will be sought to understand the inner workings of each classifier, and the influence of the selection of free parameters, selected by the user, in its performance. Likewise, it will be tried to design the classifiers so that the choice of these parameters is done automatically getting good accuracy and good generalization.

As this work is related to another one, which deals in choosing the best feature vector that characterize an epileptic signal [9], we will focus on the design of classifiers (Section 3 and 4). However, to verify the performance of them, during the survey it will be used as input vector the one described in section 2.

2. FEATURE EXTRACTION

The process of features extraction is one of the most important stages in the process of epileptic EEG signals classification, because the effectiveness of the classifier design depends on the proper selection.

The electroencephalogram (EEG) is the most commonly used clinical measure in diagnostics of almost all types of neurological disorders. A typical use of the EEG is the diagnosis of epilepsy.

The frequency of epileptic EEG signal usually displays depolarization spikes at 3 Hz interval. The characteristic feature of an epileptic seizure is an abrupt onset with perfectly synchronized action potentials across all 20 electrodes. Therefore, the amplitude of the high-frequency epileptic EEG is much higher than a normal delta wave (0.5-4 Hz). The mechanism driving an epileptic seizure is still not completely understood [1].

EEG signals are very complex non-stationary signals, whose properties do not evolve in time, and it is very difficult to obtain all statistical characteristics on EEG signals in time and frequency domain. Therefore, various analysis and processing methods for non-stationary signals have been used to extract the feature information of EEG signals.
Some of the signal processing features most often used in biomedical signals are:

- Signal Power in Frequency bands: The main method of EEG feature extraction in the Fourier domain is the evaluation of the power of specific frequencies in the power spectra of the signal. This can show how strong these particular frequencies are in the signal.
- Wavelet Measures. The wavelet transform provides a number of coefficients that decompose a signal at different scales. The feature that are commonly used for classification of biomedical signal are the coefficients of the wavelet transform in each sub-band as well as the normalized power of coefficients in each sub-band.

The research [9] discusses which are the most proper features in order to recognize epileptic patterns.

In this section it will be extracted only two features of the EEG signal, since in two dimensions is easier to visualize the input space.

It is used a data set provided by Navarra University, Spain. It will be used two recordings of a patient named Tunsen (Patient nº 5). The scalp electrodes location is shown in Figure 1.

Through out the recording there is an epileptic seizure. The seizure was identified by a specialist in neurology.

The information that is known is the time where the seizure takes place and which electrodes are closest to the epileptic focus.

The EEG signal contains among the useful information, which allows scientists to view the cerebral activity, redundant or noise information. The sources of noise on EEG could be eyes movement, electrode motion, the temperature-depend nature of the signal, muscle contraction of the face caused by blinking, etc. In order to remove all these artifacts, the original channel is filtered by a low pass filter with a 20 Hz cut-off frequency, and an independent component analysis (ICA). From now on, the EEG signals obtained after this processing will be called channel_20ICA.

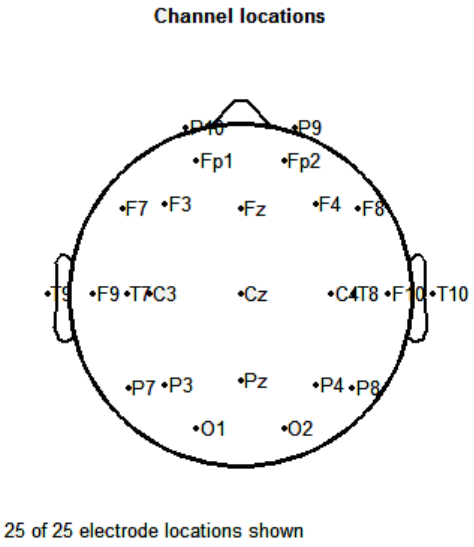


Figure 1, Channel locations.

It is used EEGLAB to display EEG signals. EEGLAB is an interactive MATLAB toolbox for processing continuous and event-related EEG using independent component analysis (ICA), time/frequency analysis, artifact rejection, and several modes of data visualization [10].

The channel_20ICA of the first recording of the patient nº 5 is divided into two segments. The first one is taken from the channel_20ICA when the epilepsy takes place (Figure 2), and the other one when an epileptic seizure is not present (Figure 3).

In Figure 4 is plotted the power spectrum of the epileptic EEG activity, and in Figure 5 is plotted the power spectrum of the non-epileptic EEG signal.

Each coloured trace represents the spectrum of the activity of one channel. The leftmost scalp map shows the scalp distribution of power at 1, 2, 4, 6, 8, 10, 20 Hz.

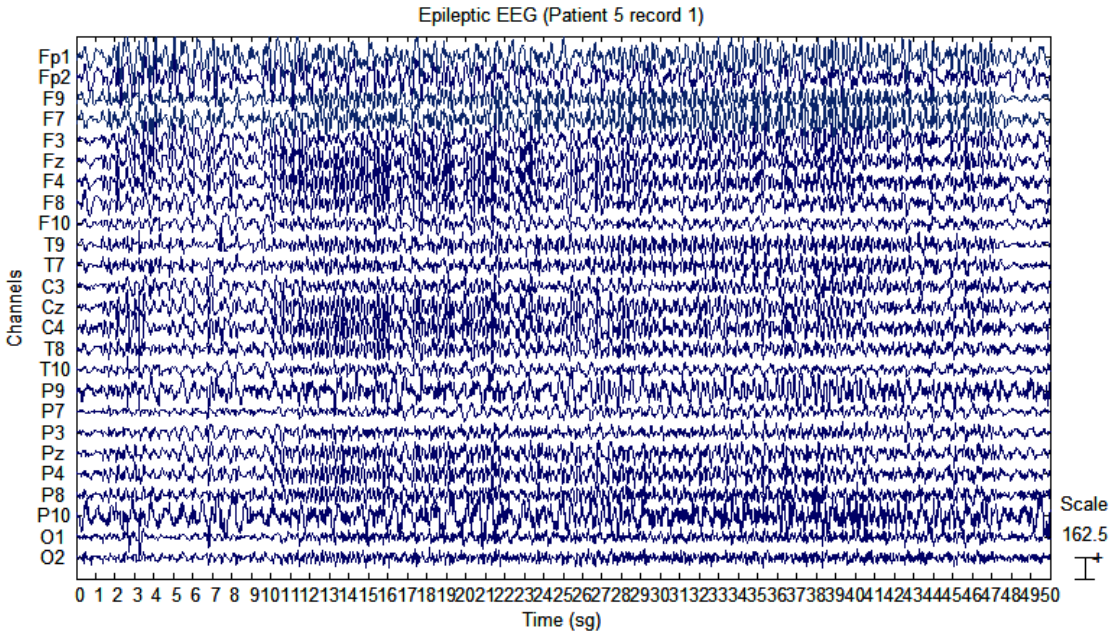


Figure 2, EEG signal of the patient number 5, when an epileptic seizure is present.

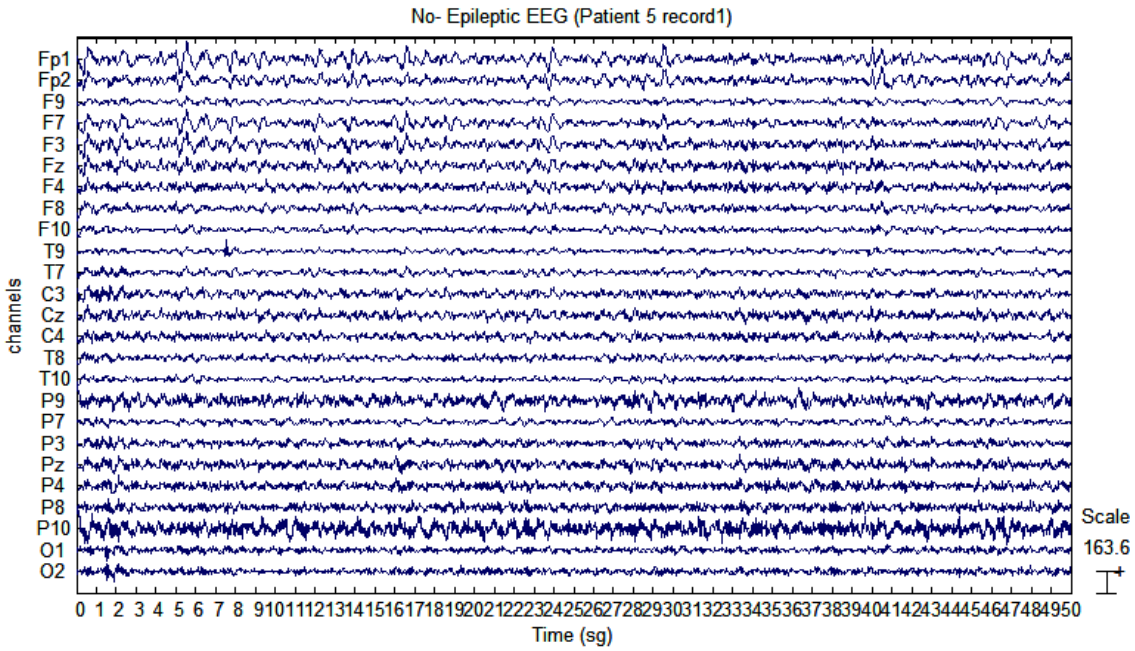


Figure 3, EEG signal of the patient number 5, when an epileptic seizure is not present.

.

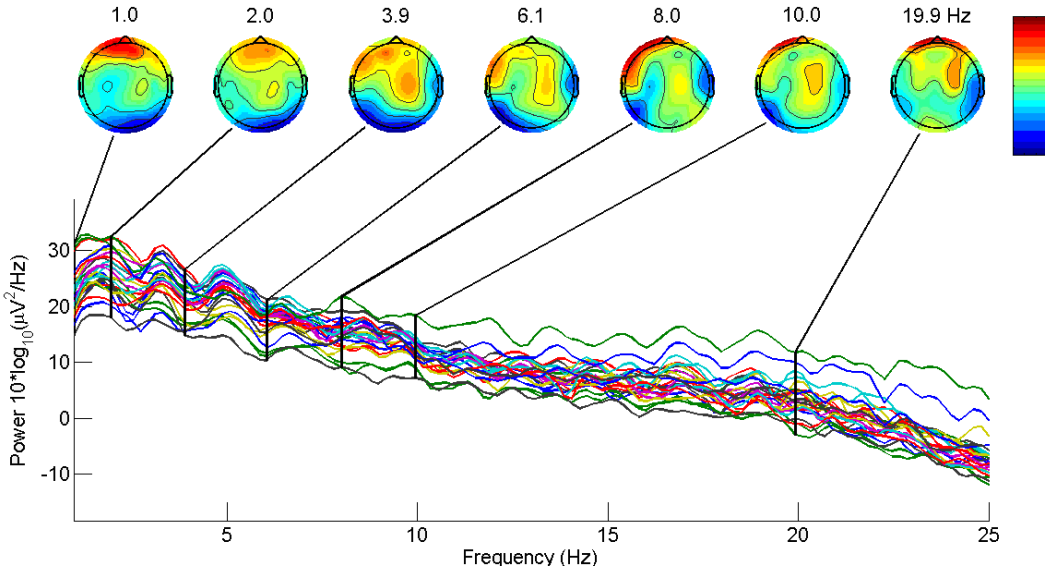


Figure 4, Power spectrum of the epileptic signal.

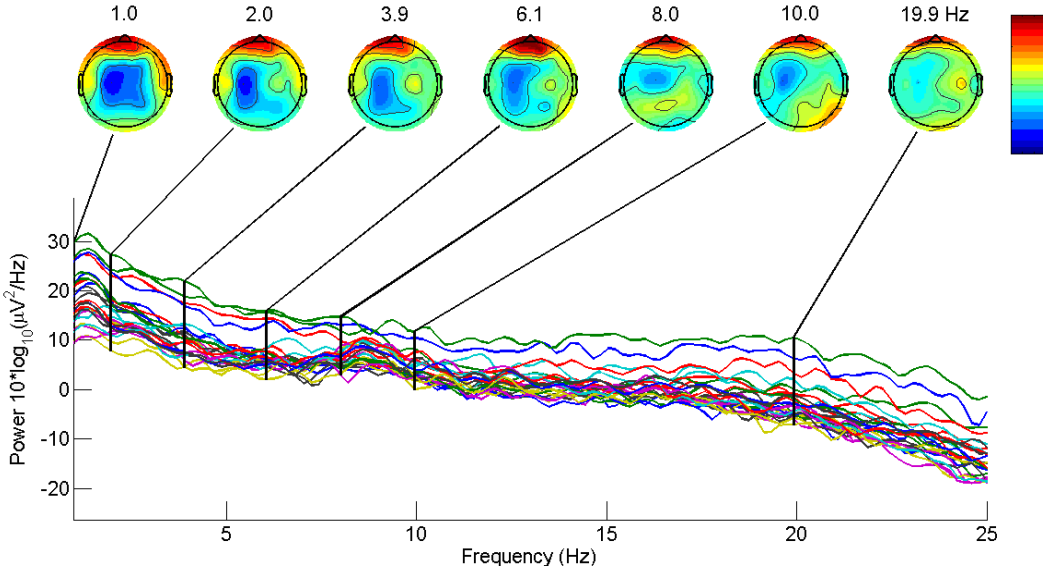


Figure 5, Power spectrum of the non-epileptic signal.

. From the information available, it is known that the seizure is most obvious in channels T9, F9, T7 and F7. As it can be seen from the Figure 4, in scalp power distribution, the power density on these electrodes is higher in the frequency range of 3 Hz to 8 Hz. When there is an epileptic seizure, the activity is picked up by most of the electrodes, and according to the figure, this occurs in the frequency range of 3-6 Hz. It should be emphasized that the maximum power is produced at a frequency of 1 Hz and collected in the electrodes closer to the eyes, so it can be concluded that it is due to eyes artifacts. By the same token, it is shown in the Figure 5, that the power collected by the electrodes of the eyes is always higher than in the other electrodes. In the Figure 5, the power at 3-6 Hz is lower than in Figure 4.

So, the power at 3-6 Hz could be a right feature for this patient in order to recognize an epileptic pattern.

Now, some estimated variables of the channel F7 are taken from the epileptic EEG (Figure 6) and from the non-epileptic EEG (Figure 7).

In Figures 6 and 7, some estimated variables of the statistics are printed as text in the lower panel to facilitate graphical analysis and interpretation. These variables are the signal mean, standard deviation, skewness, and kurtosis (technically called the 4 first cumulants of the distribution) as well as the median. The upper panel shows the data histogram (blue bars), a vertical red line indicating the data mean, a fitted normal distribution (light blue curve) and a normal distribution fitted to the trimmed data (yellow curve).

The biggest difference between the figures is the variance. In an epileptic signal, because of the big amplitude of the spikes, the magnitude of the variance is bigger.

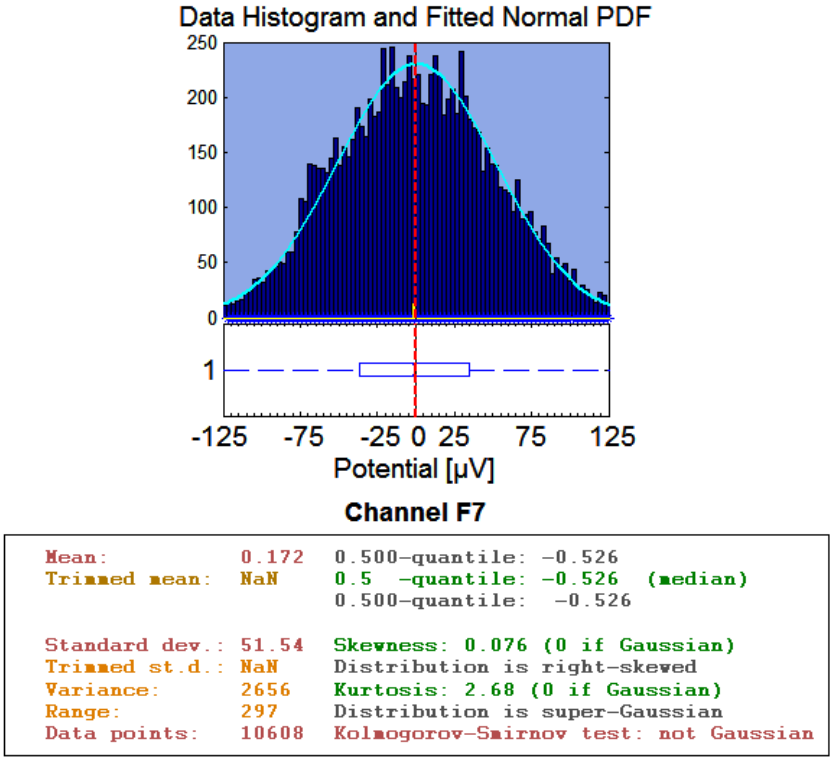


Figure 6, Estimated variables of the channel F7 of a epileptic signal.

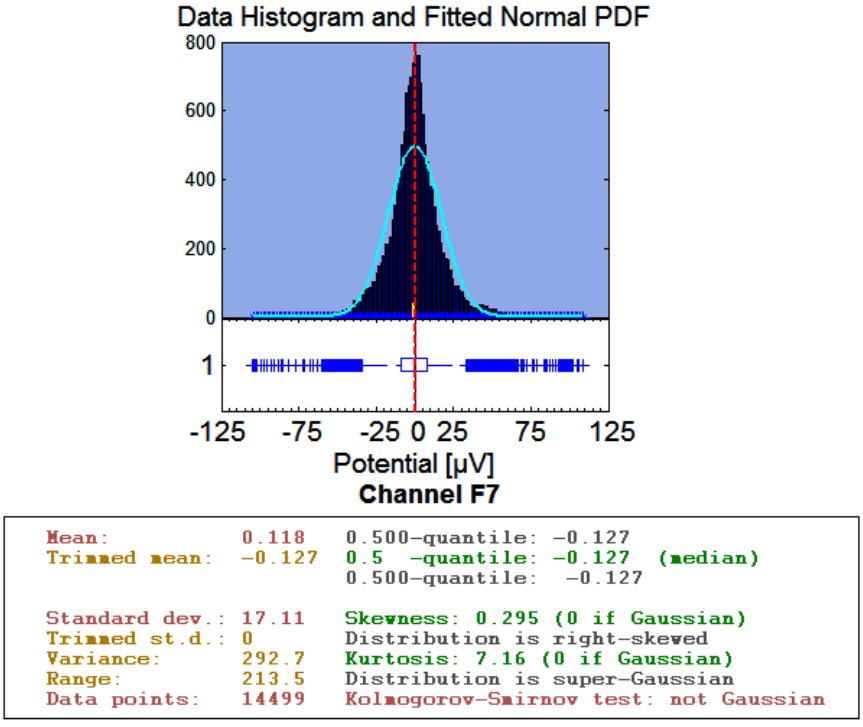


Figure 7, Estimated variables of the channel F7 of a non-epileptic signal.

With these results obtained with this particular patient, it will be shaped the input vector of the classifier taking the variance and the power at 3-6 Hz in windows of 1024 points (if the sampling frequency is 200 Hz, windows will be 5.12 sec.) of the four channels where the seizure is more obvious.

In Figure 8, it is shown the first channel of the EEG. The red line in the first plot shows when the epileptic seizure is. The other two plots draw the variance and the power at 3-6 Hz in windows of 1024 points.

It is important to normalize the input vector [11]. The main advantage is to avoid attributes in greater numeric ranges dominate those in smaller numeric ranges. Each attribute is normalized to the range [0,1] dividing by its maximum.

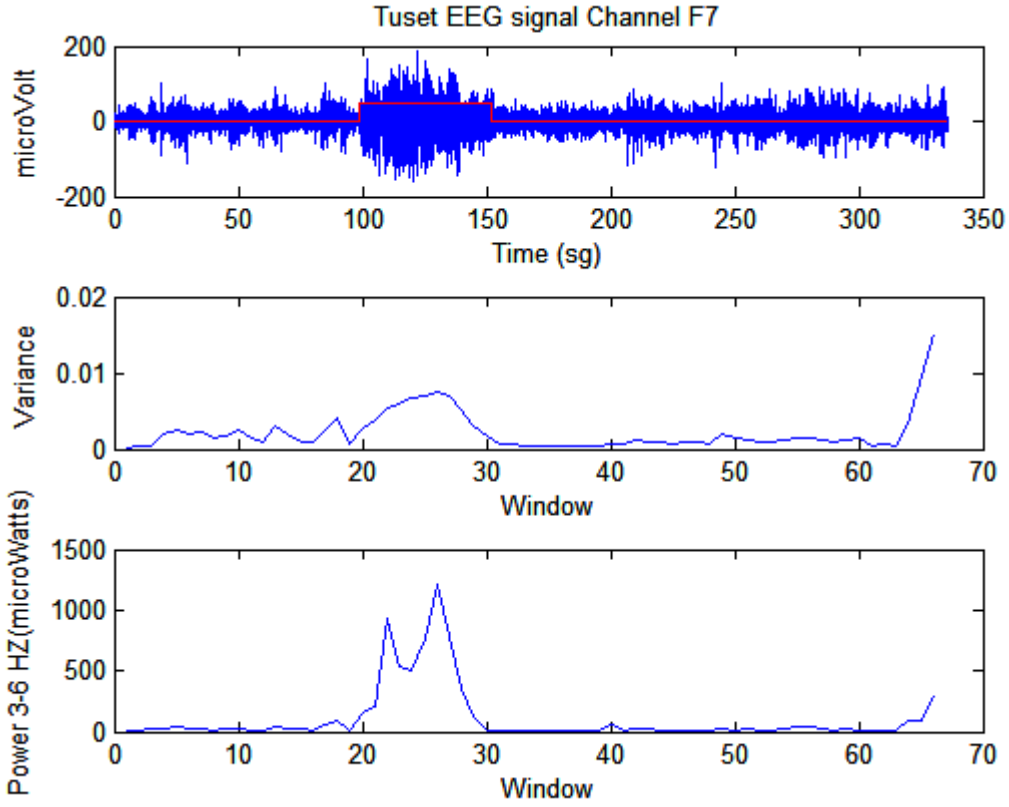


Figure 8, EEG signal of Patient 5, variance and power at 3-6 Hz in a window of 1024.

3. SUPPORT VECTOR MACHINE (SVM)

3.1 Introduction

The main idea of a support vector machine is to construct a hyperplane as the decision surface in such a way that the margin of separation between positive and negative examples is maximized. The support vector machine is an approximate implementation of the method of structural risk minimization. This induction principle is based on the fact that the error rate of a learning machine on test data is bounded by the sum of the training-error rate and a term that depends on the Vapnik-Chervonenkis (VC) dimension. Accordingly, the support vector machine can provide a good generalization performance on pattern classification problems despite the fact that it does not incorporate problem-domain knowledge. This attribute is unique to support vector machine.

A notion that is essential to the construction of the support vector learning algorithm is the inner product kernel between a support vector $\mathbf{x}_i$ and a vector $\mathbf{x}$ drawn from the input space. The support vector consists of a small subset of the training data extracted by the algorithm. Depending on how this inner-product kernel is generated, it might be constructed different learning machines characterized by nonlinear decision surface of their own [12].

3.2 The classification problem

Consider the training sample $\{(\mathbf{x}_i, d_i)\}_{i=1}^{N}$ where $\mathbf{x}_{i,}$ is the input pattern for the *i*th example and d_i is the corresponding desired response (target output) [5].

The classification problem is to find a classifier with the decision function, *f*(*x*) such that $f(x) = d$. The performance of the classifier is measured in terms of classification error which is defined in Equ. 3.1

$$E(d, f(x)) = \begin{cases} 0 & if \quad d = f(x) \\ 1 & otherwise \end{cases} \tag{3.1}$$

The task of the machine is to learn the mapping $\mathbf{x_i} \longrightarrow d_i$. The machine is actually defined by a set of possible mappings $\mathbf{x} \longrightarrow f(d, \alpha)$, where α is the adjustable parameters. The machine is assumed to be deterministic: for a given input **x**, and choice of α, it will always give the same output $f(d, \alpha)$. A particular choice of α generates a "trained machine."

The performance of this machine can be measured by the expected risk, shown in Equ. 3.2:

$$R(\alpha) = \int \frac{1}{2} |d - f(\mathbf{x}, \alpha)| dP(\mathbf{x}, d) \tag{3.2}$$

But unless it is known the estimation of $P(\mathbf{x}, d)$, it is not very useful.

The empirical risk (Equ. 3.3) is defined to be just the measured mean error rate on the training set:

$$R_{emp}(\alpha) = \frac{1}{2N}\sum_{i=1}^{N}\left|d_i - f(\mathbf{x}_i, \alpha)\right| \tag{3.3}$$

R_{emp} is a fixed number for a particular value of α and a training set.

Note that, a fixed α that minimizes the training error (R_{emp}) does not imply that it will minimize the probability of classification error $R(\alpha)$. So, if the task of the training algorithms is to minimize the empirical risk error and do not consider the capacity of the learning machine (the ability of the machine to learn any training set without error), this can result in over fitting.

The challenge in solving a supervised learning problem is, therefore, to realize the best generalization performance. To address this problem,[13] proposed the concept of `Vapnik Chervenenkis (VC) confidence. VC confidence is defined as the second term on the right hand side of the Equ. 3.4

$$R(\alpha) \le R_{emp}(\alpha) + \sqrt{\frac{h\left(\log(2N/h)+1-\log(\eta/4)\right)}{N}} \tag{3.4}$$

where $0 \le \eta \le 1$, N is the size of the training data, h is the VC dimension, and the bound in Equ. 3.4 will hold with probability 1- η.

The method of structural risk minimization provides an inductive procedure for achieving this goal by making the VC dimension of the learning machine a control variable.

The VC dimension of a learning machine determines the way in which a nested structure of approximation functions should be used. In order to apply the method of structural risk minimization, it is needed to construct a set of separating hyperplanes varying VC dimension such that the empirical risk and the VC dimension are both minimized at the same time.

It is important to know that VC dimension, and hence, the capacity does not depend on the number of free parameters directly (not larger number of free parameters means more capacity).

3.3 Building a support vector machine

The idea of a support vector machine hinges on two mathematical operations:

1. Non linear mapping of an input vector into a high-dimensional feature space that is hidden from both the input and the output.
2. Construction of an optimal hyperplane for separating the features discovered in the step 1.

Operation 1 is performed in accordance with Cover´s theorem on separability of patterns. Considering an input space made up of non-linearly separable patterns, Cover´s theorem states that a multidimensional space may be transformed into a new feature space where the patterns are linearly separable with high probability.

Operation 2 exploits the idea of building an optimal separating hyperplane and is performed in accordance with the principle of structural risk minimization that is rooted in VC dimension theory. The separating hyperplane is defined as a linear function of vectors

drawn from the feature space. The construction hinges on the evaluation of an inner- product kernel.

3.3.1 The optimal hyperplane. A Maximun Margin classifier.

Considering a two-class classifier and the training sample $\{(\mathbf{x}_i, d_i)\}_{i=1}^{N}$ where $\mathbf{x}_i$ is the input pattern for the *i*th example, and d_i (+1,-1) (+1 non-epileptic pattern, -1 epileptic pattern) is the corresponding desired response (target output). To begin, it is assumed that the patterns are linearly separable.

The issue at hand is to find a $\mathbf{w}$ (weight vector), and b (bias) that follows:

$$\begin{aligned} \mathbf{w}^T\mathbf{x}_i + b \geq 1 \quad & for\ d_i = +1 \\ \mathbf{w}^T\mathbf{x}_i + b \leq 1 \quad & for\ d_i = -1 \end{aligned} \quad \Rightarrow \quad d_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \tag{3.5}$$

As it is shown in Figure 9, margin 1 will give a lower expected risk than margin 2, so it will be chosen the pair of parameters $(\mathbf{w}, b)$ which maximises the margin of separation between the hyperplane and the closest data point.

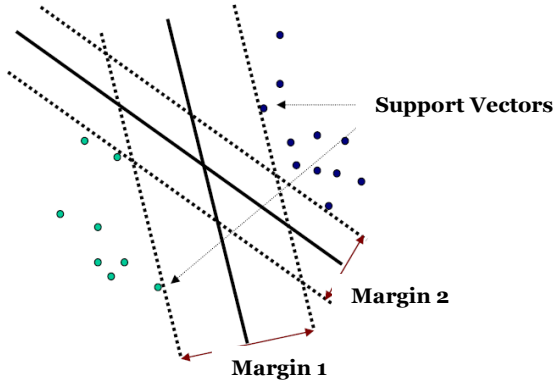


Figure 9, Different margins for a linear classifier.

Consequently, in order to maximize the margin of separation, that is $2/\|\mathbf{w}\|$ as it is shown in Figure 10, the optimum hyperplane is attained by minimizing the Euclidean norm of the weight vector with the constraint of Equ. 3.5

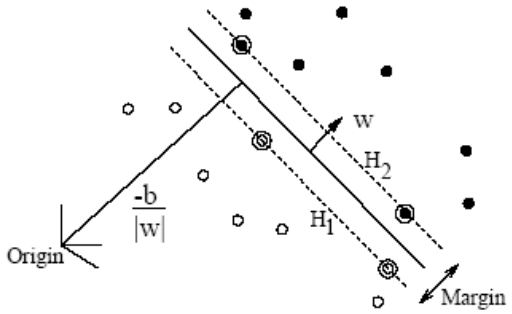


Figure 10, Geometric interpretation of distance of points to the optimal hyperplane.[5]

So, the constrained optimization problem may be stated as [12]:

Given the training sample $\{(\mathbf{x}_i, d_i)\}_{i=1}^{N}$, find the optimum values of the weight vector $\mathbf{w}$ and bias b such that they satisfy the constraint:

$$d_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad \text{for } i = 1,2,\ldots N$$

And the weight vector $\mathbf{w}$ minimizes the cost function:

$$\Phi(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

There is only one global minimum (convex problem) if the data are linearly separable. Using the method of Lagrange multipliers:

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N}\alpha\left[d_i(\mathbf{w}^T\mathbf{x}_i + b) - 1\right] \tag{3.6}$$

The saddle point of $J(\mathbf{w}, b, \alpha)$ is minimum with $\mathbf{w}$ and b, and maximum with α.

$$\frac{\partial J}{\partial \mathbf{w}} = 0 \longrightarrow \mathbf{w} = \sum_{i=1}^{N}\alpha_i d_i \mathbf{x}_i \tag{3.7}$$

$$\frac{\partial J}{\partial b} = 0 \longrightarrow \sum_{i=1}^{N}\alpha_i d_i = 0 \tag{3.8}$$

Replacing (3.7) and (3.8) in (3.6), it is obtained the dual problem,

$$Q(\alpha) = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j d_i d_j \mathbf{x}^T{}_i\mathbf{x}_j \tag{3.9}$$

subject to the constrains

(1) $\sum_{i=1}^{N}\alpha_i d_i = 0$

(2) $\alpha_i \geq 0$

where in Equ 3.9, $Q(\alpha)$, the objective function, has to be maximized by α. Note that the function $Q(\alpha)$ to be maximized, depends only on the input patterns in the form of a set of dot products $\{\mathbf{x}^T{}_i\mathbf{x}_j\}_{(i,j)=1}^{N}$

This is a Quadratic Programing problem (QP) and exist software packets to reach a efficient solution.

From the Karush-Kuhn-Tucker (KKT) conditions, this conclusion can be stated:

If $\alpha_i > 0$ then $d_i(\mathbf{w}^T\mathbf{x}_i + b) = 1$

If $\alpha_i = 0$ then $d_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

In the solution, those points for which $\alpha_i > 0$, are called support vectors and lie on one of the hyperplanes. This set of data contributes to defining the decision boundary, so, once the machine is trained, the data that is not a support vector, can be removed. The larger α_i of a support vector is, the stronger influence this SV has in its area.

In the previous section it was reached a solution when the training data is linearly separable.

For non separable patterns the margin of separation between classes is said to be soft if a data point violates the condition stated in Equ. 3.5. Now, the Equ 3.5 is reformulated by adding slack variables, which allow some points to be misclassified (Figure 11).

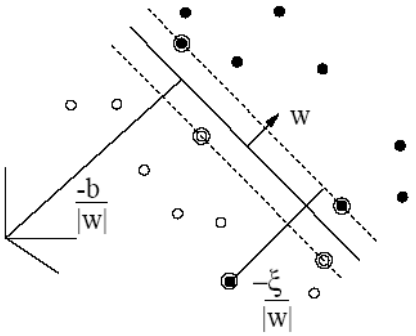


Figure 11, Linear separating hyperplanes for the non-separable case. Taken from [5]

In this case the cost function is:

$$\Phi(\mathbf{w},\xi) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_i \tag{3.10}$$

where minimizing the first term in Eq 3.10 is related to minimizing the VC dimension of the support vector machine. For the second term it is an upper bound on the number of test errors. The parameter C controls the trade-off between complexity of the machine and the number of non separable points.

Using a similar approach as in the separable case, the training results in a similar QP problem. As in linear case, the equation 3.9 has to be maximized by α , but the second constraint is replaced by:

(2) $0 \le \alpha_i \le C$

The value of C will be discussed in the next section. As it is stated, in the non separable case, α has an upper bound. This issue is explained with the following example. Imagine there is a misclassified data point. It is trying to assert a stronger influence on the boundary so that it can be classified correctly, by increasing its α . If the value of α has not been bounded, this data would force a hyperplane which computes a large amount of errors. So, the parameter C controls the trade-off between maximum margin and classification error.

3.3.2 Non linear mapping

As the support vector machine just computes a linear hyperplane as a decision surface, in order to be able to apply all the formulation discussed in the previous section to nonlinearly separable data, it is discuss in this section how to transform the data in a high dimension space (feature space) , where, in this dimension, the data is linearly separable.

Let denote $\mathbf{x}$ as a vector drawn from the input space of a dimension m_0 . Let $\varphi(\mathbf{x}) = [\varphi_0(\mathbf{x}), \varphi_1(\mathbf{x}), ..., \varphi_{m1}(\mathbf{x})]^T$ denote a set of non-linear transformation from the input space to the feature space of dimension m_1 . It is defined the decision surface as:

$$\mathbf{w}^T\varphi(\mathbf{x}) = 0 \tag{3.11}$$

Adapting Equ. 3.7 to the current situation, it is obtained Equ. 3.12

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i d_i \varphi(\mathbf{x}_i) \tag{3.12}$$

where the feature vector $\varphi(\mathbf{x}_i)$ corresponds to the input pattern $\mathbf{x}_i$ in the *i*th example. Therefore, substituting Equ. 3.12 in Equ. 3.11 the decision surface computed in the feature space is defined as:

$$\sum_{i=1}^{N} \alpha_i d_i \varphi^T(\mathbf{x}_i)\varphi(\mathbf{x}) = 0 \tag{3.13}$$

The term $\varphi^T(\mathbf{x}_i)\varphi(\mathbf{x})$ represents the inner product of two vectors induced in the feature space by the input vector $\mathbf{x}$ and the input pattern $\mathbf{x}_i$ pertaining to the *i*th example. It is introduced the inner product kernel denoted by $K(\mathbf{x}, \mathbf{x}_i)$ and defined by:

$$K(\mathbf{x}, \mathbf{x}_i) = \varphi^T(\mathbf{x})\varphi(\mathbf{x}_i) \tag{3.14}$$

Now, the optimal hyperplane is defined by

$$\sum_{i=1}^{N} \alpha_i d_i K(\mathbf{x}, \mathbf{x}_i) = 0 \tag{3.15}$$

It is important to note that it is constructed the optimal hyperplane in the feature space without having to consider the feature space itself in explicit form.

The expansion of the inner product kernel, $K(\mathbf{x}, \mathbf{x}_i)$ in Equ. 3.14, permits to construct a decision surface that is non-linear in the input space, but its image in the feature space is linear. With this expansion at hand, it might be now stated the dual form for the constrained optimization of a support vector machine as follows [12]:

Given the training data sample $\{(\mathbf{x}_i, d_i)\}_{i=1}^{N}$ find the Lagrange multipliers $\{\alpha_i\}_{i=1}^{N}$ that maximize the objective function

$$Q(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{3.16}$$

Subject to the constraints:

$$(1)\ \sum_{i=1}^{N} \alpha_i d_i = 0 \tag{3.17}$$

$$(2)\ 0 \le \alpha_i \le C \tag{3.18}$$

Once the optimization problem is stated, the question is how has to be a kernel in order to be suitable for using a support vector machine. Mercer´s theorem tells whether or not a candidate kernel is actually an inner product kernel in some space, and therefore, admissible for using in a SVM. It can be proved that the kernels in Table 1 satisfy Mercer's theorem [13].

Type of SVM	**Inner product kernel ($K(\mathbf{x}, \mathbf{x}_i)$)**
Polynomial learning machine	$(\mathbf{x}^T\mathbf{x}_i + 1)^p$
Radial-basis function network	$\exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x} - \mathbf{x}_i\|^2\right)$
Two-layer perceptron	$\tanh\left(\beta_0 \mathbf{x}^T\mathbf{x}_i + \beta_1\right)$

Table 1, Types of SVM.

For all three machine types, the dimensionality of the feature space is determined by the number of support vectors extracted from the training data by the solution to the constrained optimization problem.

In the radial-basis function type of a support vector machine, the number of radial-basis functions and their centres are determined automatically by the number of support vectors and their values, respectively.

3.4 Modelling a SVM

The architecture of the SVM just depends on the parameter C and the kernel function, which in the case of RBF only requires the parameter sigma, thus avoiding requirements on unique architectural parameters, such as number of nodes and layers, connection type between layers, and so on.

This research project is not focused on the implementation of SVM. In fact, this section will discuss how and why, the selection of the free parameters affect in the training of SVM.

A SVM classifier is implemented using LIBSVM, a freely-available library of SVM tools. Implementation details of LIBSVM in [14]

The major free parameters of the SVM are C of Equ. 3.10 and σ of the RBF kernel.

3.4.1 Selection of the Kernel

As shown in Table 1, depending on the selected kernel, there will be different types of SVM. The type of SVM is selected by the user depending on the problem to solve.

The kernel used is a radial-basis function. The RBF kernel performs a non-linear mapping in a higher dimension of input space, which, unlike the linear kernel, can handle the case in which classes are not linearly separable. Another reason for using RBF kernel is that the number of hyperparameters, which influence in the complexity of the model selection, is smaller than using a polynomial kernel. Moreover, RBF kernel has less numerical difficulties because the values resulting from the mapping (range between 0 and 1) while the polynomial kernel mapping values may go to infinite when the degree is large.

This section will discuss Radial basis-function network and its parameters.

The Radial basis-function network uses, as an inner product kernel, a Gaussian function.

$$K(\mathbf{x}, \mathbf{x}_i) = \exp\left(-\frac{1}{2\sigma^2} \|\mathbf{x} - \mathbf{x}_i\|^2 \right) \tag{3.19}$$

The centre of the function is a support vector (In Figure 12 the SV is placed at (0,0)). The output of the kernel depends on the Euclidean distance between the support vector and the points of the space. This means that a point closer to the SV will have a higher output. σ^2 is the variance of the Gaussian function. It determines the influence of the SV over data space.

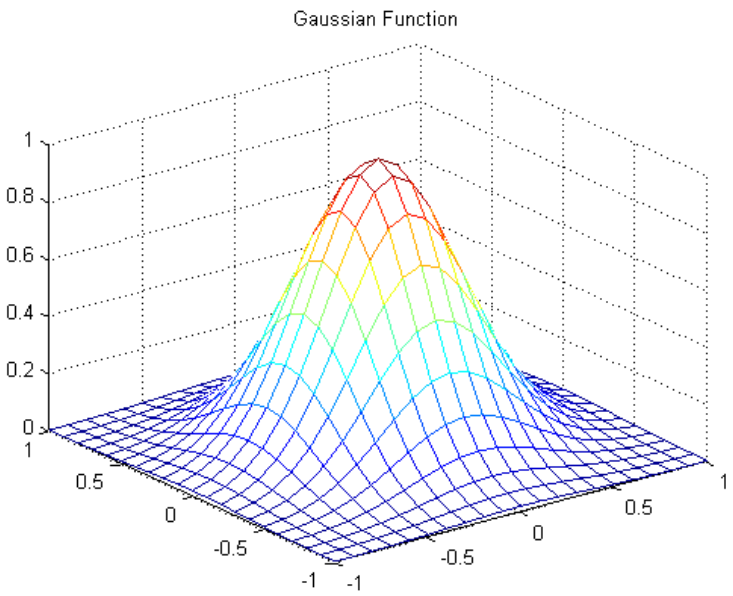


Figure 12, Gaussian function

Once the support vectors are determined by the training algorithm to find the optimal hyperplane of separation, the training data is mapped into a higher dimension of a size equal to the number of the support vectors. It is called feature dimension. In this dimension, the data are linearly separable. It is not possible to visualize the plane of separation graphically, as the number of vectors is usually high. But it is possible to draw the surface of separation in the input space.

To understand it in a more intuitive way, imagine that over each support vector is placed a Gaussian function (Figure 12). Each point in the input space has a value that depends on the Euclidean distance to a SV. Each SV will provide a transformation of the input space. Thus, the feature space is defined, and its size is equal to the number of support vectors.

In this new space, it is computed the optimization problem. As a result it is obtained an optimal hyperplane.

In theory outlined in the previous section, it was explained how it is obtained the hyperplane of separation. Equ. 3.15 defines the hyperplane as:

$$\sum_{i=1}^{N} \alpha_i d_i K(\mathbf{x}, \mathbf{x}_i) = 0$$

Taking a point of the feature space, and inserting it into the equation above, if the result is greater than 0, that point would be in the non-epileptic class area. If the result is, in contrast, smaller than 0 it would be in the epileptic class area.

The Figure 13, in the z-axis is represented the output of the previous equation, but depending on the input space.

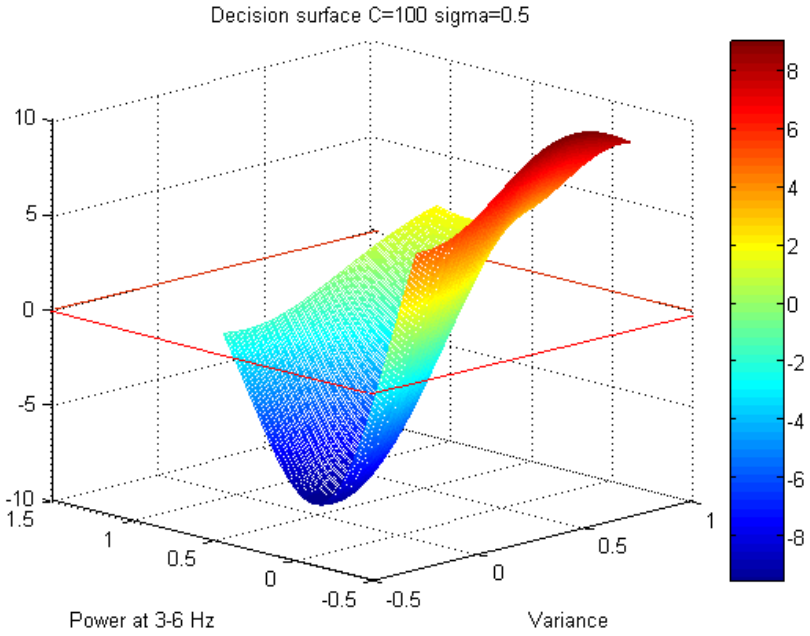


Figure 13, Decision surface C=100 sigma=0.5

In the Figure 13 has been marked with a red square the plane of z = 0. This plane contains the input space. The decision contour on the input space will be defined by the intersection of the decision surface with the plane marked in red.

The Figure 14 shows the decision contour and the margin of separation resulting from the intersection of the decision surface with the plane z = 1 (margin of non-epileptic class) and z =- 1 (margin of epileptic class).

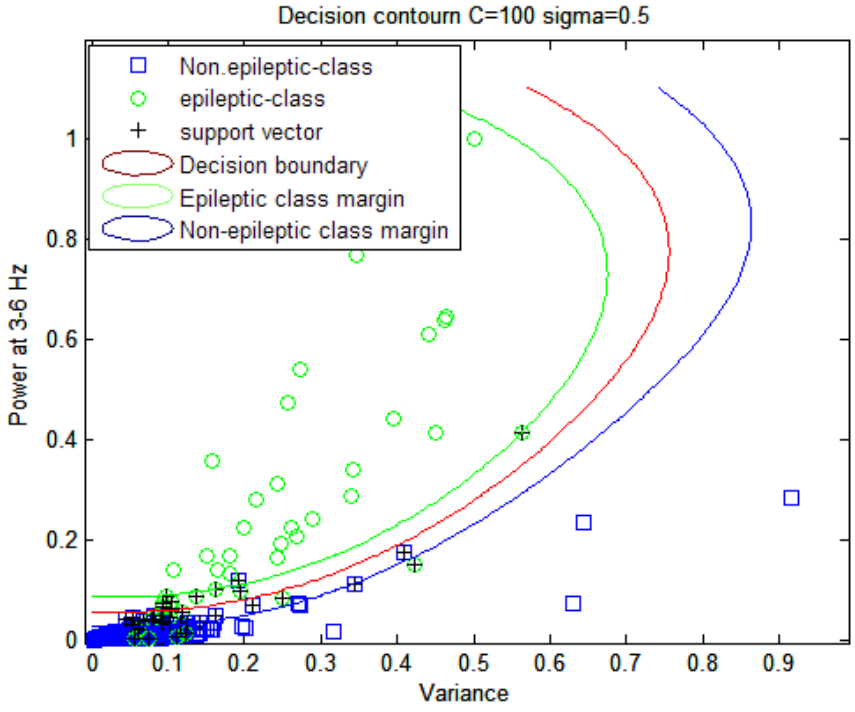


Figure 14, Decision contour with C=100 sigma=0.5

As expected, the decision contour in the input space is not a straight line.

Depending on the parameters C (Equ. 3.10), and σ (Equ. 3.19), the decision surface will be different.

In this section it will be discussed the influence of the choice of the parameter σ. The SVM will be trained with different values of σ.

Since data set is normalized, and its values are between 0 and 1, the values of gamma to be selected to check its influence in the performance of SVM, would be between 0.022 and 1. The value of the parameter C will be fixed to 100.

σ	Nº SV	Accuracy (%)
0.022	70	97.727%
0.031	56	97.727%
0.0442	51	96.97%
0.0884	44	96.591%
0.125	40	96.591%
0.1768	37	95.833%
0.25	39	95.833%
0.3536	39	95.833%
0.5	38	95.833%
0.707	39	95.076%
1	41	93.561%

Table 2, Results with different values of sigma

As it was explained, the centre of the Gaussian will be a support vector and the parameter σ will be the area of influence in the data space. A large value of σ will determine a smoother decision surface and more regular decision contour, as it is shown in the Figure 15 and 16 compared with the Figure 17 and 18.

A large σ also increases the value of the Lagrange multipliers. When a support vector covers a large area, the other SVs have to increase its multiplier to counter its influence.

As it is shown in the Table 2, for a smaller value of σ it is needed more support vectors to compute the optimal hyperplane, because one support vector cannot cover a wide area. Moreover, it can be seen from the Table 2, that a smaller value of sigma provides better accuracy.

It is recalled that this accuracy is achieved on the training data set. This means that when the test data are introduced in order to check how the classifier works, it will not be reached this accuracy. Moreover, when σ is very small, the decision surface fits the training data, hence the high accuracy, but with test data, it will be shown that a smaller accuracy will be reached, due to the overfitting.

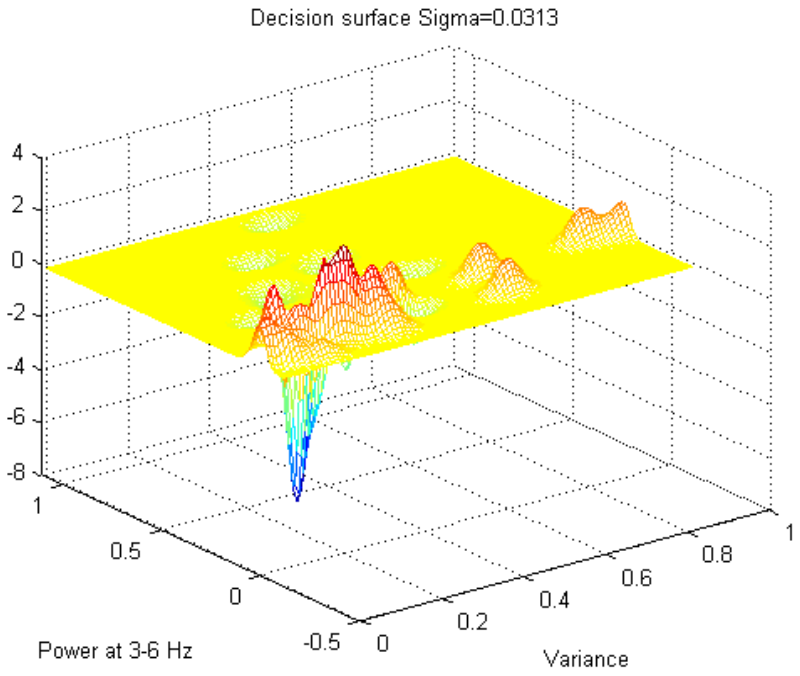


Figure 15, Decision surface C=100 sigma=0.0313

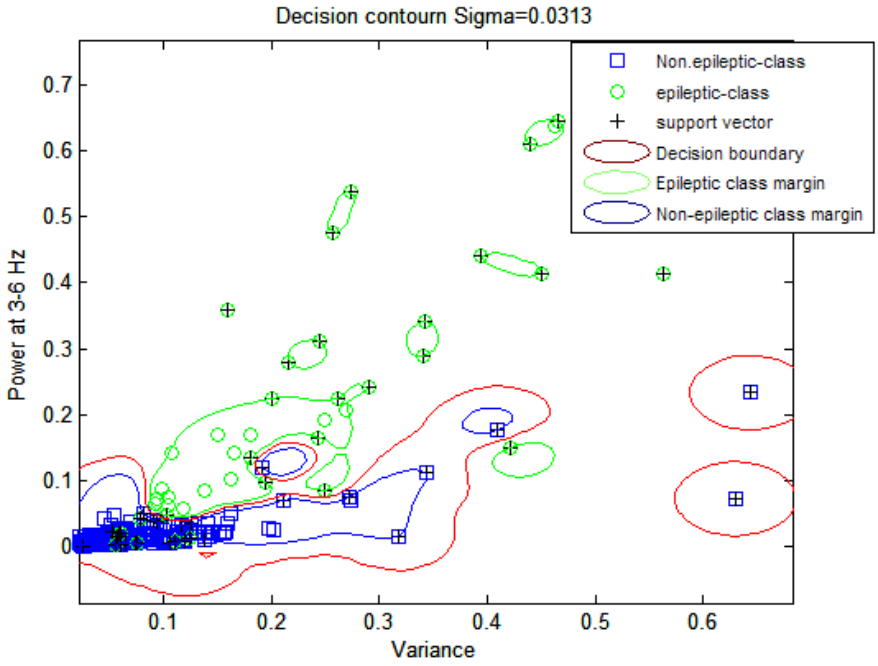


Figure 16, Decision contour C=100 sigma=0.0313

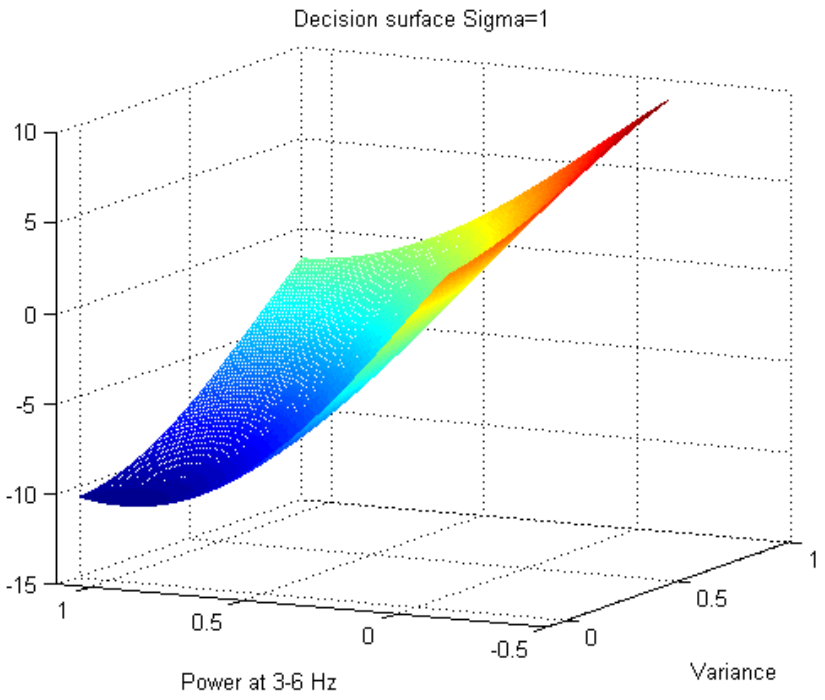


Figure 17, Decision surface C=100 sigma=1

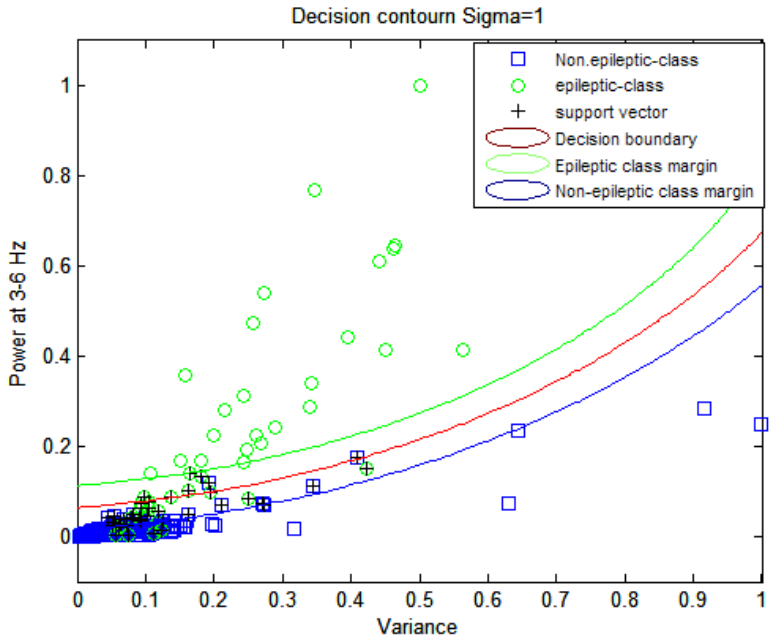


Figure 18, Decision contour C=100 sigma=1

3.4.2 Selection of C (Error penalty)

As it was discussed in section 3.3, the parameter C is a penalty constant, which determines the relative significance given to maximizing the distance, as opposed to minimizing errors. The parameter C is called Error Penalty.

To bring into the context, it is reiterated that the goal is to find a hyperplane defined by $\mathbf{w}$ that minimizes the following cost function.

$$\Phi(\mathbf{w},\xi) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_i$$

The first term is related to minimizing the VC dimension maximizing the margin of separation between the two classes. The second term is related to avoid classification error. For a large value of C, that means a large error penalty, in order to minimize the cost function it will be allowed fewer errors at the expense of making smaller the margin of separation.

It will verify experimentally how the value of the parameter C affects the performance of SVM.

The support vector machine will be trained with different values of the parameter C. To discuss its influence, it will be shown after the training the number of support vectors used to compute the optimal hyperplane, the time spent in training, the accuracy achieved and the norm (L_2) of the weight vector, $\|\mathbf{w}\| = \mathbf{w}^T\mathbf{w} = \sqrt{\sum_{i=1}^{m}|w_i|^2}$.

The other free parameter is fixed to σ =0.5. The results are shown in Table 3.

Cost	$\|\mathbf{w}\|$	Nº SV	Accuracy (%)	Time (sg)
1	7.8824	66	91.667%	0.015
10	66.259	46	93.182%	0.016
100	589.51	38	95.833%	0.016
1000	5612.7	37	95.833%	0.016
10000	55216	38	95.833%	0.016
100000	$5.2968 \cdot 10^5$	36	95.833%	0.156
1000000	$4.9208 \cdot 10^6$	35	96.591%	2.453

Table 3, Results with different value of the parameter C, sigma=0.5

As it is shown in Table 3, the smaller the value of the parameter C is, the smaller is the norm of $\mathbf{w}$, therefore, larger margin of separation since its values is $2/\|\mathbf{w}\|$.

That was expected, because if errors are allowed, the training algorithm can find a separation hyperplane with a greater margin between classes. For a higher value of C, Lagrange multipliers will have a higher upper bound (Equ. 3.18). In the case in which a data is in the wrong side of the hyperplane, it will try its utmost to increase its multiplier in order to have more influence in its area, reducing the margin of separation. To verify this, it is shown the margin of separation for two different values of C.

The Figure 19 and 20 show the contour decision reached after the training. The red line represents the boundary decision. The green and blue lines represent the margin of the non-epileptic class and the epileptic class, respectively.

In the margin of each class there is a support vector, as it was shown in Figure 19. It is also seen that the margin of separation is soft, and it allows data misclassified. These data are also support vectors, seeking to influence their area to be well classified, as it was said, increasing its Lagrange multiplier.

It is noted that for a small value of C (the decision contour is shown in Figure 19 for C = 1) the margin of separation between the classes is higher than when the value of C is large (Figure 20 for C = 100000).

The Data 1 in figure 19 is located at the margin of separation but in the wrong side of the decision boundary. If the value of C is increasing, Data 1 may increase its multiplier, having greater influence in its area. This will change the boundary decision and Data 1 will be placed on the correct side of hyperplane (Figure 20). In this case, Data 1 will be correctly classified, at the expense of making smaller the margin between the two classes.

In Table 3 it is shown also these results. A larger value of the parameter C, greater accuracy, less data incorrectly classified, but lower margin of separation.

Moreover, it is shown in Table 3 that the greater value of C is, more complex is the quadratic problem solution and it is required more time to resolve it.

The number of support vectors also changes when is increased the value of C. It can be concluded that when the value of C is small, more data are misclassified. These data, when is constructed the separation plane, try to influence in the calculation of the plane like support vectors to be classified correctly.

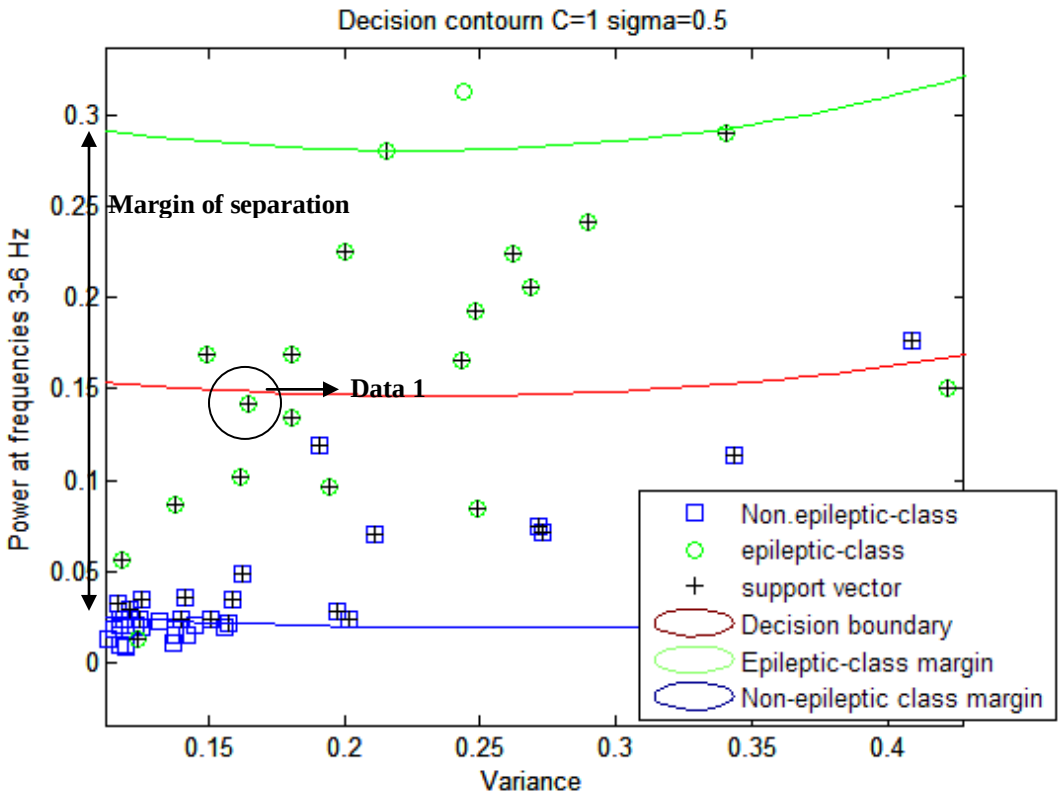


Figure 19, Decision contour with parameter C=1

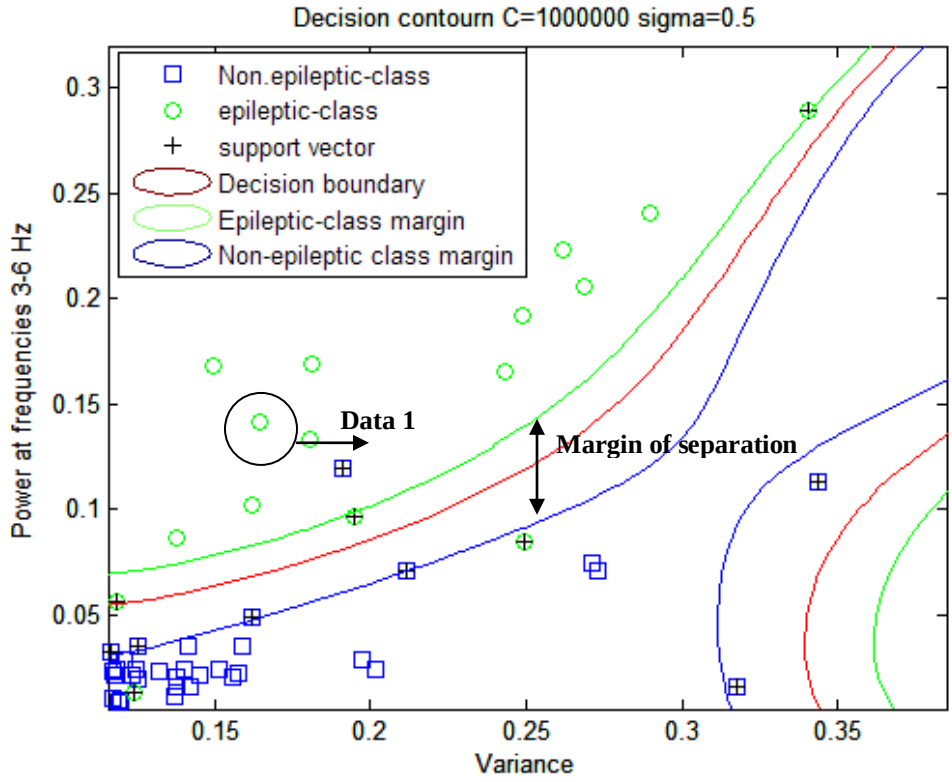


Figure 20, Decision contour with parameter C=100000

3.4.3 Choosing the parameters to get a good generalization

The evaluation criterion is based on the abilities of both, learning and generalization of architectures.

The goal is to identify good (C,σ) so that the classifier can accurately predict unknown data. It may not be useful to achieve high training accuracy because it can appear the overfitting problem. The overfitting is a problem of statistical models where there are too many parameters. This is a bad situation because instead of learning to approximate the function present in the data, the network can simply memorize each instance of training. The noise in the data training is learned as part of the function, often destroying the ability of the machine to generalize.

As it is not known beforehand what values of C and σ^2 are the best for one problem, consequently a model selection must be done.

In this context, a standard tool in statistics known as cross-validation provides an appealing guiding principle. First, the available data set is partitioned into a training set and a test set. The training set is further partitioned into two disjoint subsets; estimation subset, used to select the model, and validation subset, used to test or validate the model.

The aim is to validate the model on a data set different from the one used for parameter estimation. So it may be used the training set to assess the performance of various candidate models, and thereby choose the best one. Cross-validation procedure can prevent overfitting problems. To identify good parameters with the 60% of the data set, is implemented a search grid on (C, σ^2) using cross-validation. The best pair is picked up in order to train SVM. It was tried with exponential growing sequence of C and σ^2.
Figure 21 shows the search grid.

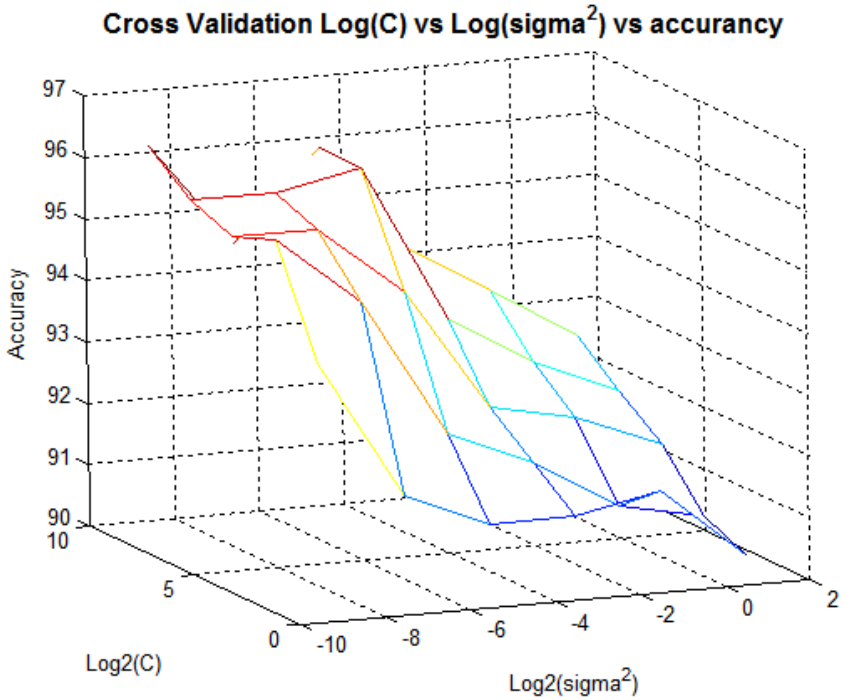


Figure 21, Cross-validation.

3.5 Summary and results

In Figure 22 it is shown the SVM scheme with some graphics from each part of the process. The first one shows how non-epileptic and epileptic samples are spread over the input space. It is shown that in the input space the data are non-linearly separable. The second graphic shows the cross validation surface over the parameter C and σ^2. The graphic number 3 shows the decision surface. The graphic shows the prediction over the decision boundary. It can be checked that the predicted labels agree with the decision boundary. It is also shown the MATLAB code used (APPENDIX A).

Table 4 shows the results achieved with the feature extraction of patient 5 described in section 2. As the data is presented to the algorithm randomly, the results are for 3 different trials.

Trial	Accuracy	Sensitivity	Specificity	C	σ^2
1	93.56%	0.703	0.9778	256	0.078
2	93.99%	0.805	0.9731	128	0.0156
3	93.99%	0.735	1	128	0.0039

Table 4, Results using a SVM

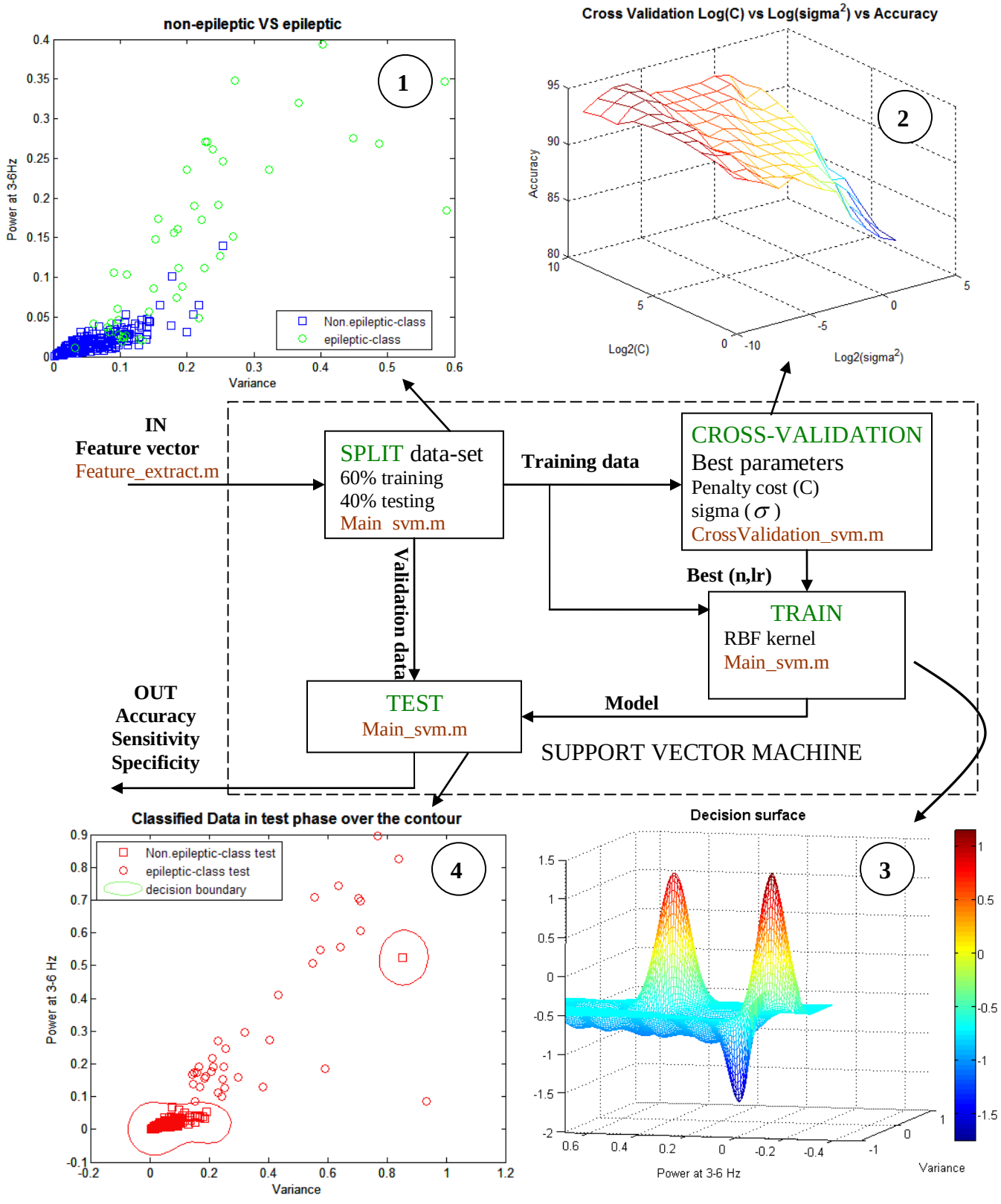


Figure 22, SVM scheme.

4. NEURAL NETWORK (MULTILAYER PERCEPTRON)

4.1 Introduction

Neural networks are computational method directly inspired by the formation and function of biological neural structures. Just like biological neural structures, artificial neural networks are simply formation of artificial neurons that learn patterns directly from examples. In addition, because of the nonlinear nature of the method, neural networks are capable of solving complex nonlinear classification problems that cannot be addressed by simple statistical methods.

Multilayer perceptrons (MLPs) is an important class of neural networks which consists of a set of sensory units that constitute the input layer, one or more hidden layers of computation nodes, and an out-put layer of computation nodes (Figure 23). The input signal propagates through the network in a forward direction, on a layer-by-layer basis [12].

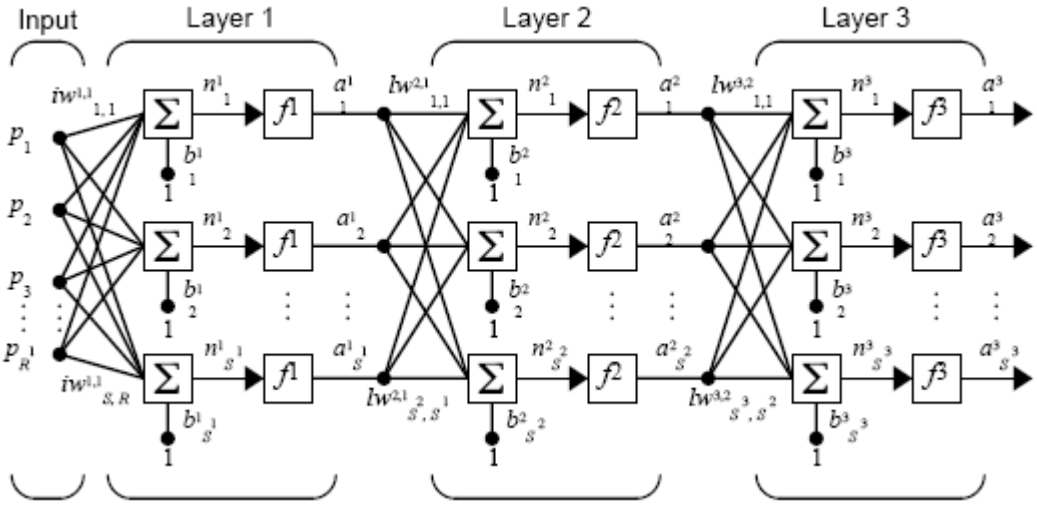


Figure 23, MLP network.

4.2 Multilayer Perceptron background (Back-propagation algorithm)

The MLP is training with the error back-propagation algorithm. It may be viewed as a generalization of the Least-Mean-Square adaptive filtering algorithm.

Error back-propagation learning consists of two passes through the different layers of the network: a forward pass and a backward pass.

In the forward, an input vector is applied to the sensory nodes and its effect propagates through the network layer by layer. Finally, a set of outputs is produced as the actual response of the network. During the forward pass the synaptic weights of the network are all fixed.

The function signal appearing at the output of the neuron j is computed as

$$a_j(n) = \varphi(n_j(n)) \tag{4.1}$$

where $\varphi(.)$ is a non-linear activation function of neuron j and $n_j(n)$ is the induced local field of neuron j, defined by:

$$n_j(n) = \sum_{i=0}^{m} w_{ji}(n) a_i(n) \tag{4.2}$$

where *m* is the total number of inputs applied to neuron j and $w_{ji}(n)$ is the synaptic weight connection neuron *i* to neuron *j*, and $a_i(n)$ is the input signal of neuron *j* or equivalently, the function signal appearing at the output of neuron i. If neuron *j* is in the first hidden layer *m=m0*, $a_i = p_i$ and the index *i* refers to the *i*th input terminal of the network.

The backward pass, in the other hand, starts at the output layer by passing the error signal leftward through the network, layer by layer, and recursively computing the local gradient for each neuron. This recursive process permits the synaptic weights of the network to undergo changes in accordance with the delta rule.

The objective of the learning process is to adjust the free parameters of the network to minimize the average squared error energy:

$$\xi_{av} = \frac{1}{2N} \sum_{n=1}^{N} \sum_{j \in C} \left(d_j(n) - a_j(n)\right)^2 \tag{4.3}$$

where $d_j(n)$ is the desired response, and *C* are the neurons of the out-put layer. In this way, the correction $\Delta w_{ji}(n)$ applied to $w_{ji}(n)$ is defined by the delta rule:

$$\Delta w_{ji}(n) = -\eta \frac{\partial \xi(n)}{\partial w_{ji}(n)} \tag{4.4}$$

where η is the learning-rate parameter. The use of the minus sign accounts for gradient descent in weight space, seeking a direction for weight change that reduces the value of $\xi(n)$.

4.3 Selection of the Optimal MLP

The implementation is done using MATLAB Neural Network toolbox [15]. The data set using in this section was described in section 2.

In this section, it is going to be discussed the influence of the parameters selected by the user in order to achieve a good generalization.

- Activation function.

To the computation of the delta rule for each neuron of the MLP, the only requirement that an activation function has to satisfy is to be differentiability. The activation function used is the hyperbolic tangent function (Equ. 4.5)(Figure 24.b). It is an antisimetric function which allows learning faster, because the output values of the activation function are closer to 1 and -1.

$$\varphi(n) = a \tanh(bn) \tag{4.5}$$

The more suitable values of *a* and *b* in Equ. 4.5 are [12].

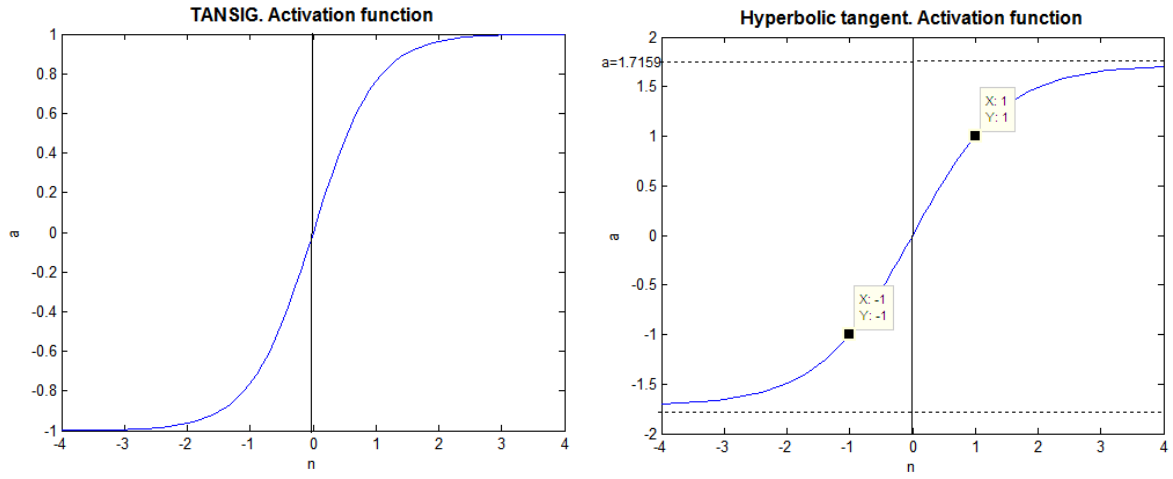
$$a = 1.7159 \quad b = 2/3$$

The Table 5 shows the results obtained using in the hidden and output layer the function of the Equ. 4.5 as an activation function and using one of the functions in the MATLAB Toolbox shown in Figure 24.a.

HYPERBOLIC TANGENT	
Mean-squared Error	**Accuracy**
0.3219	92,275%
0.31912	87,983%
0.3792	89,27%

TANSIG (MATLAB)	
Mean-squared Error	**Accuracy**
0.49982	83.262%
0.36188	87,124%
0.40926	85,837%

Table 5, Results using different activation functions.

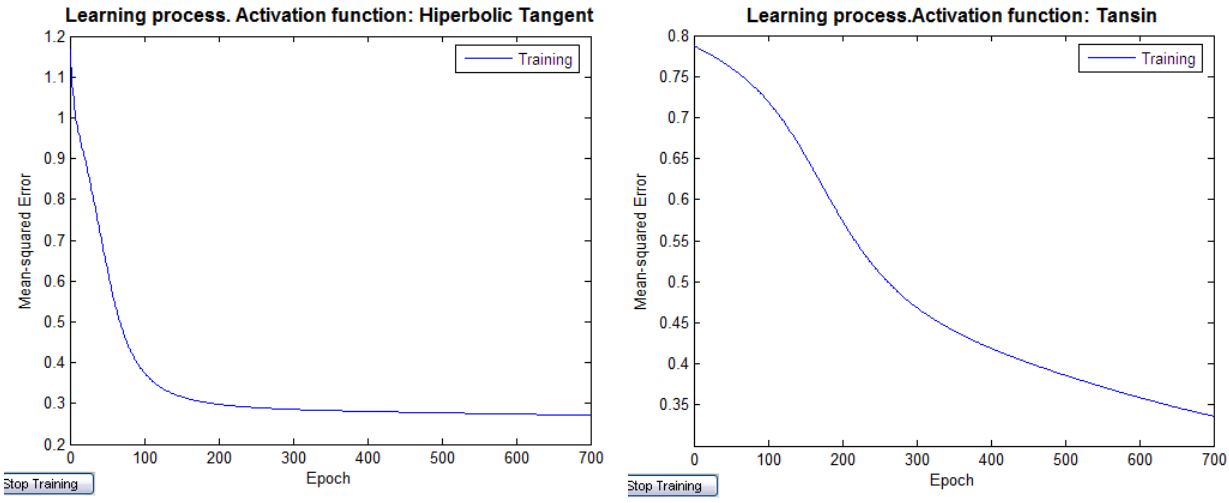


(a) **(b)**

Figure 24, (a) Tansig from MATLAB toolbox. (b) Tuned hyperbolic tangent function

The learning algorithm used was a batch steepest descent with a learning rate of 0.01 during 300 epochs. The training set was the 60% of the data set. The number of neurons in the hidden layer was fixed to 6. The results show the Mean-squared error achieved in the training and the accuracy obtained in the test of three different trials.

Figure 25 shows how decreases the Mean-square Error in each iteration (epoch) in the learning process.



(a) **(b)**

Figure 25, (a) Learning process using Tuned hyperbolic tangent function. (b) Learning process using Tansig from MATLAB toolbox

As it is shown in the Figure 25, with the hyperbolic tangent function the learning is faster than with a tansig function.

It is emphasized that a small Mean-square Error does not necessarily imply good generalization, as it is shown in Table 5.

- Sequential versus batch update

One complete presentation of the entire training set during the learning process is called epoch. In the sequential mode of back propagation learning, weight updating is performed after each training example, while in the batch mode weight updating is performed after each epoch.

It is used the batch mode since it provides an accurate estimate of the gradient vector; convergence to local minimum is thereby guaranteed under simple condition.

- Learning rate (η)

While, in general, a small learning-rate parameter η results in slower convergence, it can locate deeper local minima in the error surface than a larger η, because it should cover more of the error surface that would be the case for a larger η. But if the learning rate is too large in order to speed up the learning, the network may become unstable.

In Figure 26 it is shown the learning process for different learning rates. The learning algorithm used was a batch steepest descent with a learning rate between 0.01, 0.05, 0.5 0.9 during 300 epochs. The training set was the 60% of the data set. The number of neurons in the hidden layer was fixed to 6. The activation function is a hyperbolic tangent function.

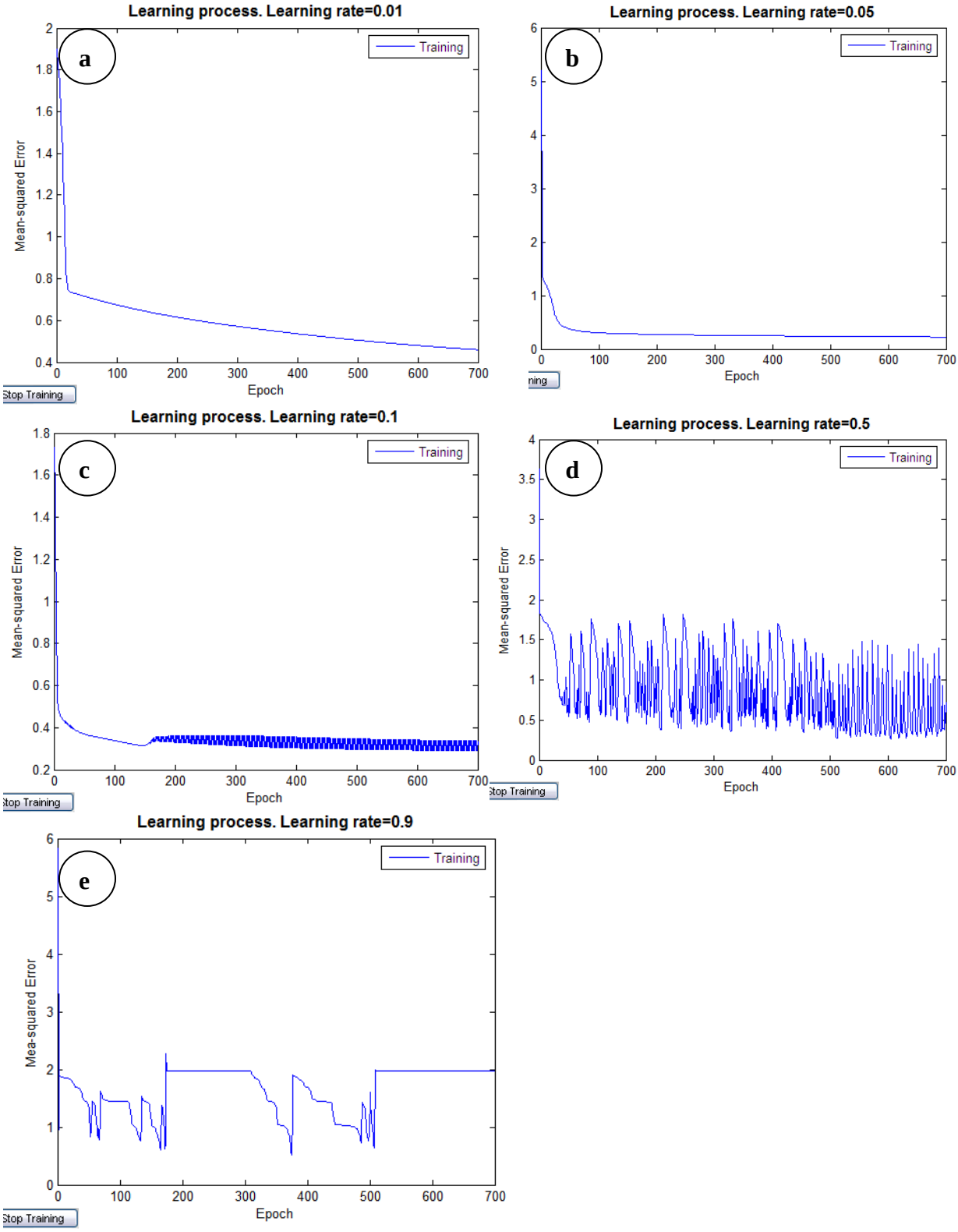


Figure 26, Processing learning with different learning rates.

As it is shown in the Figure 26.b, the learning process using a rate of 0.05 reaches faster the same Mean-squared Error than using a learning rate of 0.01. However, if it is increased the learning rate in order to converge earlier, the network can became unstable (Figure 26.c and 26.d), even more it could not converge (Figure 26.e).

- Number of neurons in the hidden layer

A large number of neurons in the hidden layer give the network more flexibility because the network has more parameters it can optimize. If the hidden layer is too large, contributes to overfitting, in which all training points are well fitted, but the fitting curve oscillates wildly between these points. That means that the network does not generalize well.

In this case, it will be measured the accuracy of the net, trained with different number of neurons in the hidden layer, in the validation phase.

The output of the neural network is a soft output, because the activation function has to be derivable by the synaptic weights in order to compute the gradient descent algorithm in the backward pass of the back-propagation algorithm. For this reason, to compute the accuracy in the testing phase, if the soft output of the network is greater than 0 it will be compute the output like 1 (not seizure), and if it is smaller than 0 like -1 (seizure). With these values, the accuracy is calculated as the difference of the output of the network and the desired response.

So, the learning algorithm used will be a batch steepest descent with a learning rate of 0.01 during 3000 epochs. It is randomly divided the data into two subsets. The training set (60% of the data set) to train the network and the testing set (40% of the data set) to validate it. The activation function is a hyperbolic tangent function.

2 NEURONS		
Trial	MSE (training)	Accuracy (testing)
1	0.363	80.6%
2	0.3023	85.5%
3	0.412	83.7%

4 NEURONS		
Trial	MSE (training)	Accuracy (testing)
1	0.413	85.8%
2	0.332	87.5%
3	0.447	80.6%

8 NEURONS		
Trial	MSE (training)	Accuracy (testing)
1	0.2784	90.1%
2	0.2839	93.9%
3	0.2843	90.5%

16 NEURONS		
Trial	MSE (training)	Accuracy (testing)
1	0.313	90.1%
2	0.3527	94.8%
3	0.2535	85.4%

32 NEURONS		
Trial	MSE (training)	Accuracy (testing)
1	0.2510	88.3%
2	0.2819	89%
3	0.2576	87.9%

50 NEURONS		
Trial	MSE (training)	Accuracy (testing)
1	0.168	87.4%
2	0.215	89.3%
3	0.283	88.7%

Table 6, Results with different number of neurons in the hidden layer

As it is shown in the Table 26, although it was set a large number of epochs to give to the algorithm enough time to converge (it was used the early stopping method to not overtrain), using 2 or 4 neurons, in the hidden layer, the network is not enough powerful to achieve a low value of MSE. However, as it can be checked after increasing the complexity of the network, using 8 or 16 neurons it is achieved a lower training MSE and a good generalization (it is reached a higher accuracy). But if the complexity of the network is raised to 32 or 50 neurons, the training MSE values are lower but, in contrast with the previous case (8-16 neurons), the accuracy decreases. So, increasing the complexity of the neural network allows to do a better mapping between inputs and outputs, but this increase must be limited because it could became in overfitting.

- Early Stopping Method of training

The *train* function of the MATLAB neural network toolbox, has the option of the early stopping method [15]. The early stopping method is used to avoid overfitting by using cross-validation for which the training data are split into an estimation subset and a validation subset. The estimation subset is used to train the network in the usual way, except for a minor modification: the training session is stopped periodically and the network is tested on the validation subset after each period of training.

For this test, the number of neurons in the hidden layer will be fixed to 20, in this way, having a higher number of free parameters, it will be observed better the overfitting. Another issue is the size of the estimation and validation set. [12] states the optimum value of parameter *r* that determines the split of the training data set between estimation and validation subset. If *W* is the number of free parameters, *r* is defined by:

$$r_{opt} = 1 - \frac{\sqrt{2W-1}-1}{2(W-1)} \tag{4.6}$$

In our case if there are two neurons in the input layer (the dimension of the input vector is two), the first neuron in the hidden layer will compute:

$$n_1(n) = w_{11}(n)x_1(n) + w_{12}(n)x_2(n) + b_1$$

where in our case x_1 is the variance and x_2 is the power at the frequency range of 3-6 Hz, as it was explained. So the number of free parameters is 3 (w_{11}, w_{12}, b_1). If there is just one neuron in the output layer (the dimension of the label vector is 1) and 20 neurons in the hidden layer, there will be 21 free parameters to be optimized in the connections of the hidden and output layer. In total, there will be 91 free parameters to be optimized by the back propagation algorithm. So, following the Equ. 4.6, the validation subset will be the 7% of the training data set, and the 93% of the training data are allotted to the estimation subset.

The function *train*, in MATLAB neural network toolbox, offers different ways to stop the training.

The training will be stopped when any of these conditions are met:

- The maximum number of epochs (repetitions) is reached.
- Performance has been minimized to the goal.
- The maximum amount of time has been exceeded.
- Validation performance has increase more than max_fail times since the last time it decreased (when using validation).

The value of max_fail will be set to 50, and the number maximum of epoch to 3000. The Figure 27 shows the performance of the training algorithm with validation.

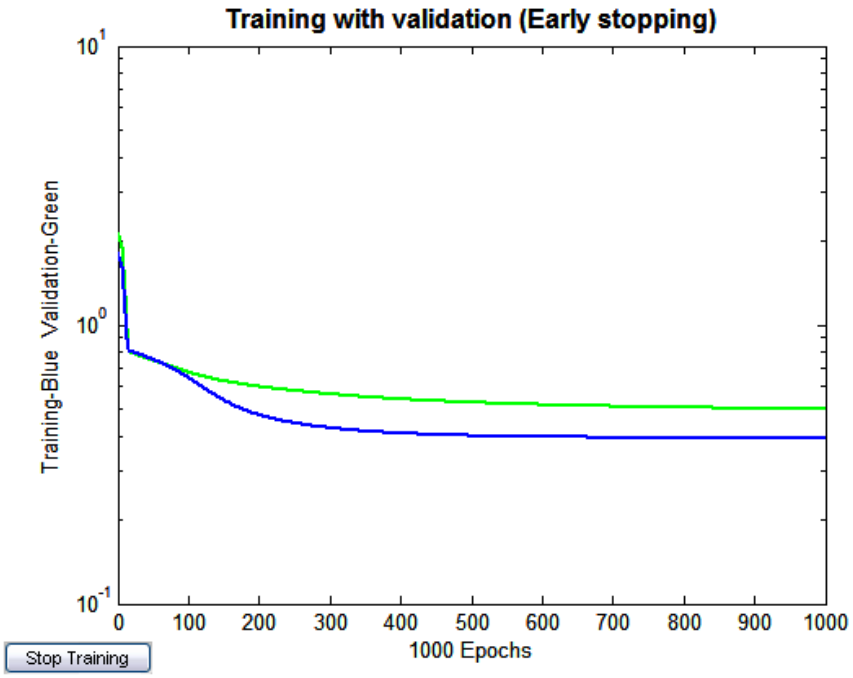


Figure 27, Training with early stopping

- Best parameters

The aim of this research project is to detect epileptic seizures. Even so, the only link with EEGs has been to perform tests to better understand how the classifiers work. This project is linked to another [9], which tries to find the best features that characterize an epileptic signal. Attempts are being made to implement a classifier that does not have any of its parameters fixed, in order to be prepared for any kind of input, and is capable of finding the best parameters, which are selected by the user, in each problem.

In the case of the MLP, the parameters to select depending on the problem are: the number of neurons in the hidden layer, the learning rate and training time. We have done a review of the influence of each parameter and it has been proposed a solution to find the optimum number of training epochs. In order to find the optimal number of neurons in the hidden layer, and the optimal learning rate, it is proposed, again, the cross validation method.

When it is referred to the optimum parameter, it means the one which generalizes well because, as it has been mentioned many times, the hope is that the network becomes well training so that it learns enough to generalize about the past to the future.

It is used a *v*-fold cross-validation by dividing the training set into *v* subsets. The model is trained on all the subsets except for one, and the validation error is measured by testing it on the subset left out. This procedure is repeated for a total of *v* trials, each time using a different subset for validation. The performance of the model is assessed by the averaging the square error under validation over all the trials of the experiment.

The major disadvantage is that the method requires an excessive amount of computation. On the other hand, as the samples are presented to the training algorithm randomly, the mean-squared error is different between one and another execution of the algorithm, so, in this sense, to repeat the process several trial and computing the average, the selection of the optimal parameters makes more accurate.

It was used 3-fold cross-validation. The optimal value of the number of neurons in the hidden layer is selected from the range of 15 to 25. The values of the learning rate will be 0.01, 0.015, 0.02, 0.05, to avoid the risk that the algorithm becomes unstable.
Figure 28 shows the mesh obtained after performing cross validation.

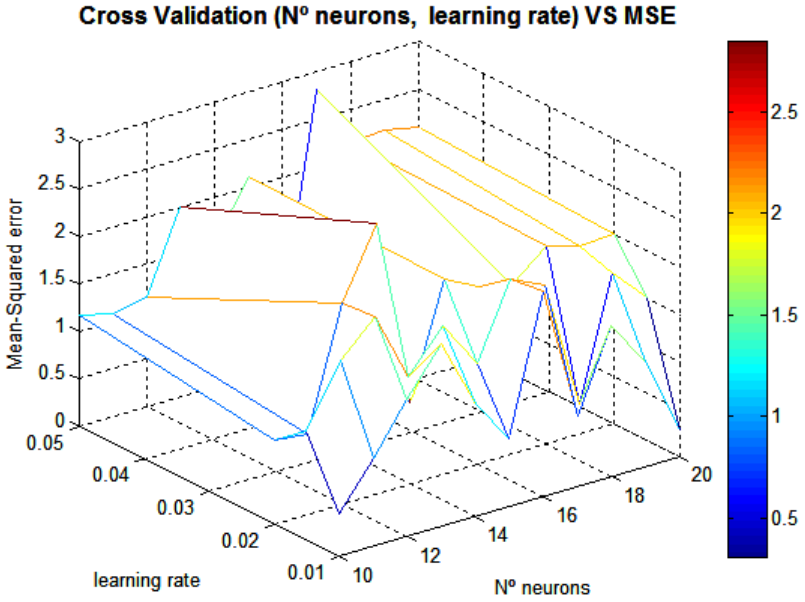


Figure 28, Cross-validation

In this case the selected parameters were learning_rate= 0.01 and nª_neurons=20. With this parameter the network was trained and it was achieved an accuracy of 91%.

4.4 Summary and results

In Figure 29 it is shown the MLP scheme with some graphics from each part of the process. The first one shows the cross-validation in order to obtain the best number of neuron in the hidden layer and the best learning rate. The second graphic shows the network created with these best parameters (in this case 15 was the best number of neurons). The graphic number 3 shows the training phase using early stopping. It is also shown the MATLAB code used (APPENDIX B).

Table 7 shows the results achieved with the feature extraction of patient 5 described in section 2. As the data is presented to the algorithm randomly, the results are for 3 different trials.

Trial	Accuracy	Sensitivity	Specificity	Nº hidden neurons	Learning rate
1	95,7%	0.98	0.873	25	0.01
2	93,991%	0.694	0.98	15	0.015
3	92%	0.74	0.978	15	0.02

Table 7, Results using a MLP

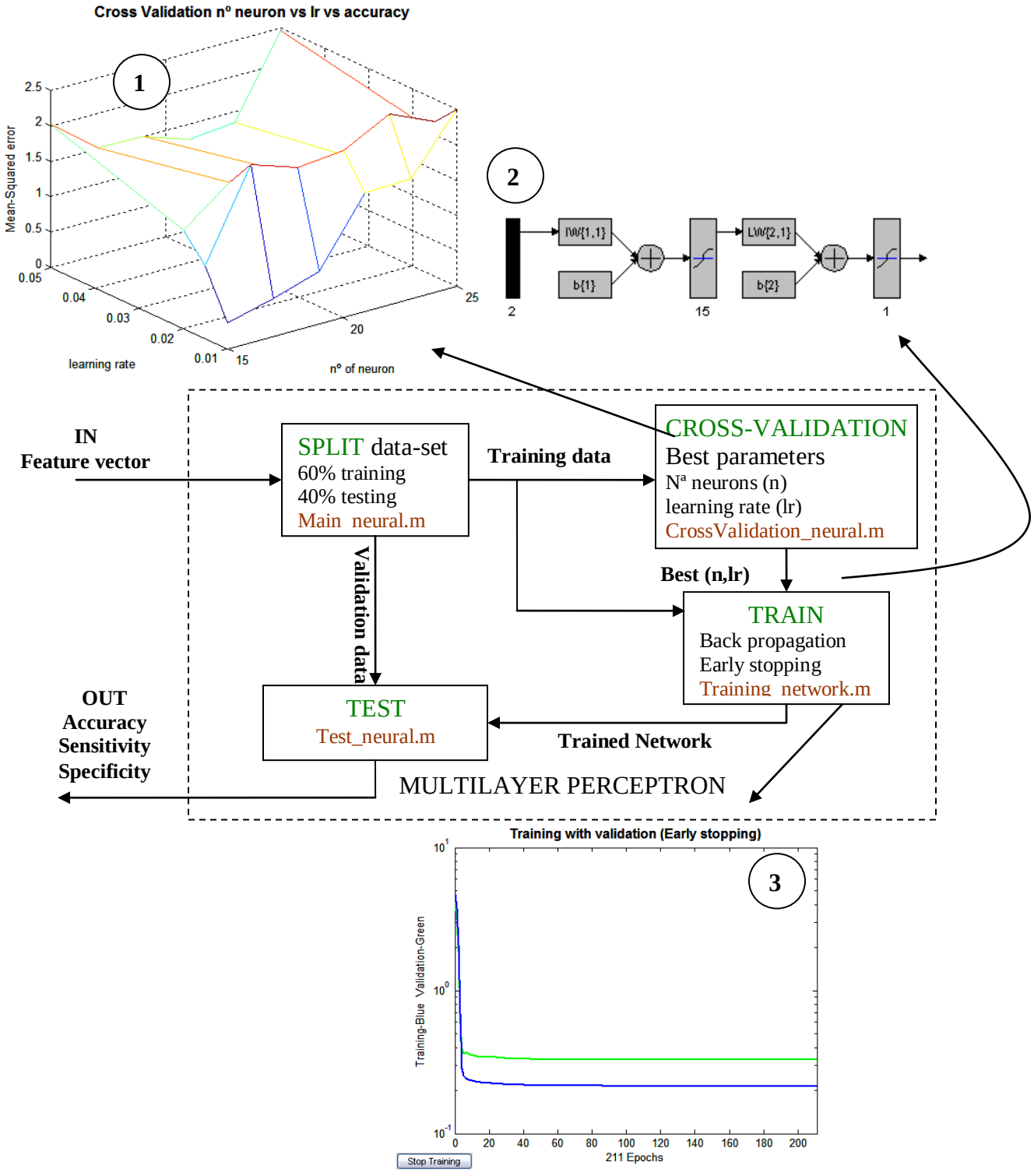


Figure 29, MLP scheme

5. RESULTS

The results of this section are related to the assessment of classifiers for the input data described in [9]. Using the STFT with a windows sized of 3600, it is extracted the power in the bands of 0-2 Hz, 2-5 Hz , 5-10 Hz, 10-20 Hz , 20-100 Hz and the variance and maximum amplitude in the window. In this case, this feature extraction is done over the 25 scalp channels in order to have a bigger data set size.

Table 8 shows the results achieved with the Multilayer Perceptron, while Table 9 shows the results achieved using a SVM.

Pacient	Accuracy	Sensitivity	Specificity	Nº hidden neurons	Learning rate
1	78.03%	0.718	0.806	18	0.01
2	77.6%	0.61	0.871	18	0.02
3	85.45%	0.786	0.903	20	0.015
4	67.169%	0.556	0.783	18	0.02
5	86.11%	0.604	0.98	16	0.015
6	79.15%	0.2664	0.95	15	0.05
All	65.991%	0.202	0.9204	15	0.01

Table 8, Results using a MLP.

Pacient	Accuracy	Sensitivity	Specificity	C	σ^2
1	92.35%	0.94	0.924	384	0.0019
2	92.06%	0.886	0.936	6144	0.025
3	96.96%	0.986	0.944	216	0.0052
4	93.34%	0.938	0.926	1322	0.0033
5	95.15%	0.85	0.982	11136	0.089
6	93.5%	0.845	0.968	4112	0.0019
All	80.63%	0.645	0.889	5504	0.0027

Table 9, Results using a SVM.

6. CONCLUSIONS

Irrespective of how support vector machine is implemented, it differs from the design of a MLP in a fundamental way. In MLP, model complexity is controlled by keeping the number of adjustable parameters small, while SVM offers a solution to the design of the learning machine by controlling model complexity independently of the dimensionality.

About the results achieved with both classifiers, it is clearly shown that the SVM accomplish better results than MLP.

It is also true that in this study, we have focused on designing a SVM, and MLP was just developed to compare results. Even so, it has been sought the best possible to tune the MLP, in spite of using gradient descent as a training function, which is the most simple.

So, for a multilayer perceptron trained on the back propagation algorithm, to attain a classification performance comparable to that of the support vector machine for the same pattern-classification tasks, it has to be tuned a multitude of design parameters.

For epileptic pattern recognition, the MLP does not accomplish a small mean square error during the training although it was achieved an acceptable accuracy. The SVM model allows a good understanding of its theoretical details, as shown in Section 3, and this can be used to identify the important parameters for the classifier.

For both classifiers it has been tried to create a model for, irrespective of the type of input data, classifiers to be able to tune their parameters by themselves to get better performance. In the case of support vector machine this process is simpler because it has only two parameters to be adjusted. On the other hand, using MLP the number of parameters to be set is larger and requires to be trained more times to obtain the best parameter. This produces a higher processing time than with a SVM.

In the task of detecting epileptic patterns, in the context of an initial processing of EEG, the most important parameter for evaluating the performance of a classifier is the sensitivity. It is preferable that the classifier determines that a seizure has occurred when it has not really happened than predicting an EEG without seizure when really there has been one. In this sense, SVM achieves better sensitivity than MLP.

.

7. REFERENCES

[1] K. Najarian, R Splinter, (ed) (2006) *Biomedical Signal and Image Processing*. U.S of America: Taylor & Francis Group.

[2] T. Jung, S Makeig, C. Humphriesa T. Lee. *Removing electroencephalographic artifactsby blind source separation.* Psychophysiology, 37 (2000) 163–178. Cambridge University Press. Printed in the USA.

[3] J. Gotman, *Automatic recognition of epileptic seizure in the EEG*, Electroencephalogrphy and Clinical Neurophysiology 54, 530-540, 1982.

[4] Bernard E. Boser, Isabelle M. Guyon, Vladimir N. Vapnik,. *A Training Algorithm for Optimal Margin Classifiers* . Proceedings of the fifth annual workshop on Computational learning theory. Pittsburgh, Pennsylvania, United States Pages: 144 - 152

[5] Burges, C. J. C. (1998). *A Tutorial on Support Vector Machines for Pattern Recognition*. Kluwer Academic Publishers, Boston

[6] W.R.S. Webber, Ronald P. Lesser, Russell T. Richardson, Kerry Wilson, *An approach to seizure detection using an artificial neural network (ANN).* Electroencephalography and clinical Neurophysiology 98 (1996) 250-272

[7] A. Subasi, M. Kermal Kiymik, a.Alkan, E. Koklukaya, *Neural Network classification of EEG signals by using AR with MLE preprocessing for Epileptic seizure detection* Mathematical and Computacional Application, Vol. 10 No 1, pp. 57-70, 2005

[8] Chandaka, S. et al., *Cross-correlation aided support vector machine classifier for classification of EEG signals*, Expert Systems with Applications (2008).

[9] L. Boto Espada, *Modeling an epileptic brain. Feature extraction.* University of Reading from Carlos III de Madrid (2008)

[10] EEGLAB. MATLAB toobox.Swartz Center for Computational Neuroscience (SCCN) of the Institute for Neural Computation at the University of California San Diego in La Jolla.

[11] Sarle, W. S. (1997). Neural Network FAQ. Periodic posting to Unset news-group comp.ai.neural-nets.

[12] S. Haykin. (ed) (1999) *Neural Networks.* New Jersey: Prentice-Hall

[13] Vapnik, V. (1995). *The Nature of Statistical Learning Theory*, chapter 5. Springer-Verlag, New Cork

[14] LIBSVM: a Library for Support Vector Machines Chih-Chung Chang and Chih-Jen Lin_ Last updated: June 14, 2007 http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf

[15] H. Demuth, M. Beale, M. Hagan. (ed) (2007) *Neural Networks Toolbox 5, User's Guide.* The MathWorks, Inc.

APPENDIX A

- **Feature_extraction.m**

```
function [x_train,y_train]=feature_extraction(patients)
%          USAGE: MAIN_SVM
% [x_train,y_train]=feature_extraction(patients)
% Output:
%       - x_train:Input vector
%       - y_train: Desired response
%
%
% Input
%      - Patients: Matrix of patients
%
%-------------------------------------------------------------------------
close all;
len_pat=length(patients(:,1));
eeg=[];
y_eeg=[];
phrase=['processing patient'];
for i=1:len_pat
   pat=patients(i,:);
   phrase=['Processing patient',' ',pat];
   disp(phrase);
load (pat)
for h=1:length(canales_crisis)
 eeg_channel= double(canal_20ICA(:,canales_crisis(h)));
Lx = length(eeg_channel);
%window size
M = 1024;
Nw = ceil(Lx/M);
Npad = Nw*M - Lx;
eeg_channel_norm = eeg_channel-mean(eeg_channel);
eeg_channel_norm = eeg_channel_norm/var(eeg_channel_norm);
eeg_channel_norm = [eeg_channel_norm; zeros(Npad,1)];
eeg_epile_channel=ones(Nw,1);
eeg_epile_channel([ceil(crisis_min/M):ceil(crisis_max/M)])=-1;
y_eeg=[y_eeg; eeg_epile_channel];
eeg = [eeg;eeg_channel_norm];

if h==1
paint(eeg_channel_norm,pat,Nw,M)
end
clear('eeg_channel','eeg_channel_norm','eeg_epile_channel');
end
```

```
end

Nw_total = ceil(length(eeg)/M);
fs=200;
   for j=1:Nw_total
      x_var(j)=var(eeg([(1+M*(j-1)):j*M]));
      Y = fft(eeg([(1+M*(j-1)):j*M]));

x_max(j)=sum(abs(Y([ceil(3*M/fs):ceil(6*M/fs)])).^2)/length(Y([ceil(3*M/fs):ceil(6*M/fs)
]));

   end
x_train=[(x_var)' (x_max)'];
y_train=y_eeg;
```

- **Svm_main.m**

```
function [accurancy,sensitivity,specificity]=main_svm(x,y)
%          USAGE: MAIN_SVM
%[accurancy,sensitivity,specificity]=main_neural(x,y)
%Output:
%       - accuracy: accuracy achieved in the testing phase
%       - Sensitivity
%       - Specificity
%
% Input
%        - x: Input vector
%        - y: Desired response
%---------------------------------------------------------------------------
% Training percentage
training_porc=0.6;
% the input vector is normalized by its maximun value
for i=1:length(x(1,:))
   x(:,i)=x(:,i)./max(x(:,i));
end
% the training and testing set are taken randomly from input vector
random=ceil(length(x(:,1))*rand(1,length(x(:,1))));
train_values=ceil(training_porc*length(x(:,1)));
x_train=x(random(1:train_values),:);
y_train=y(random(1:train_values),:);
x_test=x(random((train_values+1):length(x(:,1))),:);
y_test=y(random((train_values+1):length(x(:,1))),:);

% Displaying training data
figure;
fprintf(1,'displaying data...\n');
```

```
plot(x_train(find(y_train==1),1)',x_train(find(y_train==1),2)','bs',x_train(find(y_train==-
1),1)',x_train(find(y_train==-1),2)','go')
legend('Non.epileptic-class','epileptic-class',0);
title('non-epileptic VS epileptic');
xlabel('Variance');
ylabel('Power at 3-6Hz')
drawnow;

%       CROSS VALIDATION
% Crossvalidation_neural is called, it returns the optimum values of
% Cost and gamma=1/(2*sigma^2) in LIBSVM

[cost,gamm]=crossValidation_svm(x_train,y_train);
phrase=['The parameters chosen by cross validation are: Costs= ',num2str(cost,10),'
gamma= ',num2str(gamm,10)]

% It is trained with the best parameters from cross-validation with a
% Radial basis-function (default)
model=svmtrain(y_train,x_train, ['-c',' ',num2str(cost,10),' ','-g',' ',num2str(gamm,10)]);

% The model is tested
[predict_label,accurancy,dec_values]=svmpredict(y_test,x_test,model);

% The accuracy is computed
equal=0;
for(i=1:length(y_test))
   if (y_test(i)==predict_label(i))
    equal=equal+1;
   end
end
acurancy_mine=equal*100/length(y_test);
% is displayed the support vectors
fprintf(1,'displaying support vectors...\n');
figure;
plot(x_train(find(y_train==1),1)',x_train(find(y_train==1),2)','bs',x_train(find(y_train==-
1),1)',x_train(find(y_train==-1),2)','go')
axis(axis + [-0.1 +0.1 -0.1 +0.1]);
legend('Non.epileptic-class','epileptic-class',0);
drawnow;
hold on
plot(model.SVs(:,1)',model.SVs(:,2)','k+');
legend('Non.epileptic-class','epileptic-class','support vector',0);
hold off

% is displayed the decision surface and contour
fprintf(1,'displaying decision boundary...\n');
a     = axis;
```

```
[X,Y] = meshgrid(a(1):0.01:a(2),a(3):0.01:a(4));
X2    = [reshape(X,prod(size(X)),1) reshape(Y,prod(size(X)),1)];
size(X2,1);
cof=model.SVs(:,[1:2]);
n = size(cof,1);
m = size(X2,1);
K = zeros(m,n);

if (m <= n)
  for p = 1:m
    K(p,:) = sum((ones(n,1)*X2(p,:) - cof).^2,2).';
  end
else
  for p = 1:n
    K(:,p) = sum((X2 - ones(m,1)*cof(p,:)).^2,2);
  end
end
K = exp(-gamm*K);
w = repmat(model.sv_coef', size(X2,1), 1);
y = sum(((w).*K)')- model.rho;
z     = y';
z     = reshape(z,size(X));
hold on

contour(X,Y,z,[+1,+1],'b');
contour(X,Y,z,[0,0],'r');
contour(X,Y,z,[-1,-1],'g');
legend('Non.epileptic-class','epileptic-class','support vector','non-epileptic class
margin','decision boundary','Non-epileptic class margin',0);
title('Decision contour');
xlabel('Variance');
ylabel('Power at 3-6 Hz')
hold off
figure;
mesh(X,Y,z)
title('Decision surface');
xlabel('Variance');
ylabel('Power at 3-6 Hz')
% The predicted data is displayed over the decision bundary
figure;
plot(x_test(find(predict_label==1),1)',x_test(find(predict_label==1),2)','rs',x_test(find(predic
t_label==-1),1)',x_test(find(predict_label==-1),2)','ro')
legend('Non.epileptic-class test','epileptic-class test',0);
hold on;
contour(X,Y,z,[0,0],'r');
legend('Non.epileptic-class test','epileptic-class test','decision boundary',0);
title('Test data and contourn');
```

```
xlabel('Variance');
ylabel('Power at 3-6 Hz')
% the sensitivity and the specficity are computed
TP=0;
FN=0;
FP=0;
TN=0;
for(i=1:length(y_test))
if(y_test(i)==predict_label(i))
   if(predict_label(i)==-1)
   %True positive
   TP=TP+1;
   else
      TN=TN+1;
   end
else
   if(predict_label(i)==1)
      FN=FN+1;
   else
      FP=FP+1;
   end

end
end
clear cof
sensitivity= TP/(TP+FN)
specificity= TN/(TN+FP)
```

- **CrossValidation_svm.m**

```
function [c,g]=crossValidation_svm(x_train,y_train)
%                 USAGE CROSSVALIDATION_SVM
% [accur]=crossValidation(x_train,y_train)
% %Output:
%       - c: Best Penalty Cost selected
%       - g: Best gamma selected
%
%
% Input
%        - x: Training vector
%        - y: Training Desired response
%---------------------------------------------------------------------
%Exponential growing secuence for C and sigma^2
exp_cost=[1:1:10];
cost=2.^exp_cost;
```

```
exp_sigma=[-9:1:2];
sigma=2.^exp_sigma;
leng_cost=length(cost);
leng_sig=length(sigma);
accur=ones(leng_cost,leng_sig);
% svmtrain has the option of cross validation. Is selected a 3-fold cross
% validation
for c=1:leng_cost
   for g=1:leng_sig
      phrase=['-c',' ',num2str(cost(c),10),' ','-g','  ',num2str((1/(2*sigma(g))),10),' ','-v 3'];
      model=svmtrain(y_train,x_train,phrase);
      accur(c,g)=model;
   end
end

figure;
mesh(exp_sigma,exp_cost,accur);
title('Cross Validation Log(C) vs Log(sigma^2) vs accurancy' );
ylabel('log2(C)')
xlabel('log2(sigma^2)')
% It is picked the pair with achieves best accuracy over estimation set
maxim=max(max(accur));
[c,g]=find(accur==maxim);
hold on;
c=cost(c(1));
g=1/(2*sigma(g(1)));
```

APPENDIX B

- **Main_neural.m**

```
function [accuracy,sensitivity,specificity,n_neuron,lr]=main_neural(x,y)
%          USAGE: MAIN_NEURAL
%[accurany,sensitivity,specificity,n_neuron,lr]=main_neural(x,y)
%Output:
%       - accuracy: accuracy achieved in the testing phase
%       - Sensitivity
%       - Specificity
%       - n_neuron: number of neuron in hidden layer
%       - lr: learning rate
% Input
%         - x: Input vector
%         - y: Desired response
%---------------------------------------------------------------------
% Training percentage
training_porc=0.6;
% the input vector is normalized by its maximun value
for i=1:length(x(1,:))
   x(:,i)=x(:,i)./max(x(:,1));
end
% the training and testing set are taken randomly from input vector
random=ceil(length(x(:,1))*rand(1,length(x(:,1))));
train_values=ceil(training_porc*length(x(:,1)));
x_train=x(random(1:train_values),:);
y_train=y(random(1:train_values),:);
x_test=x(random((train_values+1):length(x(:,1))),:);
y_test=y(random((train_values+1):length(x(:,1))),:);

%       CROSS VALIDATION
% Crossvalidation_neural is called, it returns the optimum values of
% the number of neuron in the hidden layer and teh learning rate
[n_neuron,lr]=crossValidation_neural(x_train,y_train)

%       NETWORK CREATION
% The neural networ is created with:
%  - n neuron in hidden layer
%  - The activation function in hidden and output layer is mytf (see mytf.m
%    and mydtf.m
%  - Training function--> traingd (gradiente descent backpropagation)
net=newff(minmax(x_train'),[n_neuron,1],{'mytf','mytf'},'traingd')
net.trainParam.show=50;
%The learning rate of the networ is set
net.trainParam.lr=lr;
```

```
% Maximum number of epochs to train
net.trainParam.epochs=2000;
% Maximum number of times the the validation curve increases in a row
net.trainParam.max_fail=100;

%       TRAINING
net=train_network(net,x_train,y_train,n_neuron)

%      VALIDATION
[accuracy,sensitivity,specificity]=test_neural(net,x_test,y_test);
```

- **CrossValidation_neural.m**

```
function [n,l]=crossValidation_neural(x_train,y_train)
%             USAGE CROSSVALIDATION_SVM
% [accur]=crossValidation_neural(x_train,y_train)
% %Output:
%       - n: Best number of neuron in the hidden layer
%       - l: Best learning rate
%
%
% Input
%         - x: Training vector
%         - y: Training Desired response
%---------------------------------------------------------------------

n_neuron=[15:2:25];
lr=[0.01 0.015 0.02 0.05];
%It is done a 3-fold cross validation
div=length(x_train)/3;
mse_val=zeros(length(n_neuron),length(lr));
for i=1:3
x_train_cross=[x_train([1:(i-1)*floor(div)],:);x_train([i*ceil(div):3*floor(div)],:)];
y_train_cross=[y_train([1:(i-1)*floor(div)],:);y_train([i*ceil(div):3*floor(div)],:)];
x_test_cross=[x_train([(i-1)*floor(div)+1:i*floor(div)],:)];
y_test_cross=[y_train([(i-1)*floor(div)+1:i*floor(div)],:)];
for n=1:length(n_neuron)
   for l=1:length(lr)
      % the networ is created and it is set some parameters
   net=newff(minmax(x_train_cross'),[n,1],{'mytf','mytf'},'traingd');
   net.trainParam.show=NaN;
   net.trainParam.lr=l;
   net.trainParam.epochs=2000;
   net.trainParam.max_fail=100;
   % it is choosen the size of the estimation set depending on the number
   % of free parameters
   free_parameters=n*(length(x_train(1,:)))+n+1;
```

```
    r=((sqrt(2*free_parameters-1)-1)/(2*(free_parameters-1)));
    size_validation_subset=ceil(r*length(x_train(:,1)));
    x_validation=x_train([1:size_validation_subset],:);
    x_estimation=x_train([size_validation_subset+1:length(x_train(:,1))],:);
    y_validation=y_train([1:size_validation_subset],:);
    y_estimation=y_train([size_validation_subset+1:length(x_train(:,1))],:);
    val.P=x_validation';
    val.T=y_validation';
    % The net is trainin with early stopping method
    [net,tr]=train(net,x_estimation',y_estimation',[],[],val);
    [y_result,Pf,Af,E,perf]=sim(net,x_test_cross');
    % is picked the MSE over the validation set
    mse_val(n,l)=mse_val(n,l)+perf;
    end
end
end
figure;
mse_val=mse_val./3;
mesh(n_neuron,lr,mse_val');
title('Cross Validation n_neuron vs lr) vs accuracy' );
ylabel('lr')
xlabel('n_neuron')
zlabel('Mean-Squared error')
% is taken the pair with achieved smaller MSE
mini=min(min(mse_val));
[n,l]=find(mse_val==mini);
hold on;
n=n_neuron(n(1));
l=lr(l(1));
```

- **Train_network.m**

```
function [net]=train_network(net,x_train,y_train,n)
%       USAGE: TRAIN_NETWORK
%  [net]=train_network(net,x_train,y_train,n)
%  Outputs:
%     - net: trained net
%  Inputs:
%     - net: Untrained net to be trained
%     - x_train
%     - y_train
%     - n: number of neuron in the hidden layer
%-------------------------------------------------------------------------

% It is implemented a EARLY STOPPING METHOD
% The training set is divided in estimation subset and validation subset
```

```
% To compute size of the estimation subset
free_parameters=n*(length(x_train(1,:)))+n+1;
r=((sqrt(2*free_parameters-1)-1)/(2*(free_parameters-1)));
size_validation_subset=ceil(r*length(x_train(:,1)));
x_validation=x_train([1:size_validation_subset],:)
x_estimation=x_train([size_validation_subset+1:length(x_train(:,1))],:);
y_validation=y_train([1:size_validation_subset],:);
y_estimation=y_train([size_validation_subset+1:length(x_train(:,1))],:);
val.P=x_validation';
val.T=y_validation';
% The network is training, and it is obtained the trained network, and the
% MSE obatined in each epoch
[net,tr]=train(net,x_estimation',y_estimation',[],[],val);

figure;
plot(tr.epoch,tr.perf)
Title('Training MSE')
legend('Training');
ylabel('Squared Error');
xlabel('Epoch');
```

- **Test_neural.m**

```
function [accuracy,sensitivity,specificity]=test_neural(net,x_test,y_test)
%          USAGE: MAIN_NEURAL
%[accurancy,sensitivity,specificity]=main_neural(net,x_test,y_test)
%Output:
%       - accuracy: accuracy achieved in the testing phase
%       - Sensitivity
%       - Specificity
%
% Input
%         - x_test: Validation vector
%         - y_train: Validation Desired response
%         - net: Trained network
%-------------------------------------------------------------------------
% Simulation
[y_result,Pf,Af,E,perf]=sim(net,x_test');
y_test=y_test';
y_result=y_result(:);
error=0;
% The soft output of the network is transformed to a hard output
for i=1:length(y_test)
    if y_result(i)<0
        y_result(i)=-1;
    else
```

```
        y_result(i)=1;
    end
    if(y_test(i)==y_result(i))
    else
        error=error+1;
    end
end
errorTotal=error/length(y_test);
accuracy=(length(y_test)-error)*100/length(y_test);
TP=0;
FN=0;
FP=0;
TN=0;
for(i=1:length(y_test))
if(y_test(i)==y_result(i))
    if(y_result(i)==-1)
    TP=TP+1;
    else
        TN=TN+1;
    end
else
    if(y_result(i)==1)
        FN=FN+1;
    else
        FP=FP+1;
    end

end
end

sensitivity= TP/(TP+FN)
specificity= TN/(TN+FP)
```

- **Myft.m**

```
function a = mytf(n)

%MYTF Example custom transfer function.
%
%  Use this function as a template to write your own function.
%
%  Calculation Syntax
%
%    A = mytf(N)
%      N - SxQ matrix of Q net input (column) vectors.
%      A - SxQ matrix of Q output (column) vectors.
%
```

```
%  Information Syntax
%
%    info = mytf(code) returns useful information for each CODE string:
%      'version' - Returns the Neural Network Toolbox version (3.0).
%      'deriv'   - Returns the name of the associated derivative function.
%      'output'  - Returns the output range.
%      'active'  - Returns the active input range.
%
%  Example
%
%    n = -5:.1:5;
%    a = mytf(n);
%    plot(n,a)

% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.2.2.1 $

if nargin < 1, error('Not enough arguments.'); end

if isstr(n)
  switch (n)
    case 'version'
    a = 3.0;       % <-- Must be 3.0.

    case 'deriv'
    a = 'mydtf';   % <-- Replace with the name of your
                   %     associated function or "
    case 'output'
    a = [-1 1];    % <-- Replace with the minimum and maximum
                   %     output values of your transfer function
    case 'active'
    a = [-2 2];    % <-- Replace with the range of inputs where
                   %     the outputs are most sensative to changes.

    otherwise, error('Unrecognized code.')
  end

else

  a = 1.7159*tanh((2/3)*n);   % <-- Replace with your calculation

end
```

- **Mydtf.m**

```
function d = mydtf(n,a)
%MYDTF Example custom transfer derivative function of MYTF.
```

```
%
%  Use this function as a template to write your own function.
%
%  Syntax
%
%    dA_dN = mydtf(N,A)
%      N - SxQ matrix of Q net input (column) vectors.
%      A - SxQ matrix of Q output (column) vectors.
%      dA_dN - SxQ derivative dA/dN.
%
%  Example
%
%    n = -5:.1:5;
%    a = mytf(n);
%    da_dn = mydtf(n,a);
%    subplot(2,1,1), plot(n,a)
%    subplot(2,1,2), plot(n,da_dn)

% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.2.2.1 $

% **  Replace the following calculation with your
% **  derivative calculation.

d = 1.7159*(2/3).*(sech((2/3)*n)).^2;
```