Lázaro, M., Hayes, M.H. y Figueiras Vidal, A.R. (2018). Training neural network classifiers through Bayes risk minimization applying unidimensional Parzen windows. *Pattern Recognition*, 77, pp. 204-215.

DOI: https://doi.org/10.1016/j.patcog.2017.12.018

Training neural network classifiers through Bayes risk minimization applying unidimensional Parzen windows

Marcelino Lázaro [a,*], Monson H. Hayes [b], Aníbal R. Figueiras-Vidal [a]

[a] Signal Theory and Communications Department Universidad Carlos III de Madrid, Spain
[b] Department of Electrical and Computer Enginering of George Mason University, Fairfax, VA, USA

A B S T R A C T

A new training algorithm for neural networks in binary classification problems is presented. It is based on the minimization of an estimate of the Bayes risk by using Parzen windows applied to the final one- dimensional nonlinear transformation of the samples to estimate the probability of classification error. This leads to a very general approach to error minimization and training, where the risk that is to be minimized is defined in terms of integrated one-dimensional Parzen windows, and the gradient descent algorithm used to minimize this risk is a function of the window that is used. By relaxing the constraints that are typically applied to Parzen windows when used for probability density function estimation, for example by allowing them to be non-symmetric or possibly infinite in duration, an entirely new set of training algorithms emerge. In particular, different Parzen windows lead to different cost functions, and some interesting relationships with classical training methods are discovered. Experiments with synthetic and real benchmark datasets show that with the appropriate choice of window, fitted to the specific prob- lem, it is possible to improve the performance of neural network classifiers over those that are trained using classical methods.

Keywords: Bayes risk Parzen windows Binary classification

1. Introduction

Rosenblatt proposed the Perceptron Rule to train a two-class linear discriminant in the late 1950s [1,2]. It can be considered as the first Learning Machine (LM). In the field of Statistics, soft activations - called link functions - appeared as the result of considering different classes of likelihoods or probabilistic solutions (see [3], Ch. 4). This is the case of logistic and probit regressions, that use a sigmoid or a Gaussian distribution as activation functions for the linear combinations. Although several works presented the chain rule approach to train multi-layer networks [4–6] that include other activations to build non-linear transformations of the input samples, it was not until the Back-Propagation (BP) algorithm was introduced in 1986 [7,8] that Multi-Layer Perceptrons (MLPs) received a great deal of attention and found many practical applications, including ensemble forms [9–11] to increase their expressive capabilities. The appearance of Support Vector Machines [12,13] that employ the kernel trick [14,15] and impose a hinge cost diminished the interest in MLPs in the late 1990s and early 2000s, but the introduction of Deep Learning (DL) architectures and algorithms [16–18] put them again in the focus of current research.

Bayesian formulations [19] play a central role in analytical studies of decision and classification [20,21]. However, for the important class of discriminative (non-generative) LM classifiers, the only well-studied connection with Bayesian theory is to get estimates of the "a posteriori" probabilities of the hypotheses at the output of a LM trained by means of Bregman divergences [22]. Overviews of this subject from the perspective of Machine Learning may be found in [23–25].

In this paper, we will establish a general and direct correspondence between Bayesian risk minimization and LM classifier training for binary classification, via modeling the one-dimensional output of the neural network by means of the Parzen windows method [26] to estimate probability densities. Addressing just binary cases is not a serious limitation, because binarizing multi-class problems provides better (ensemble) machine designs than using classical soft-max forms [10,27]. Using single machines for multi-class problems would impose a multi-threshold decision, which is difficult to design and will degrade performance, or it will require multi-dimensional kernels, creating serious difficulties in their design. The direct connection between the windows that are applied in the Parzen estimator and the cost or risk function that is minimized emerges immediately, showing that several well known cost functions are particular cases of the general framework

* Corresponding author.
E-mail address: mlazaro@tsc.uc3m.es (M. Lázaro).

that is proposed. Some experiments show that this perspective can help to improve the performance of classical LM classifiers.

We want to emphasize that this approach merges discriminative training with generative approaches concepts: The overall training process is carried out according to discriminative principles, but the last step consists of modeling the non-linear transformation from input patterns to the output of the network by means of unidimensional Parzen windows. This permits to combine some advantages of both families of techniques, such as high performance and robustness against imbalanced situations. To avoid any kind of confusion, we want to strongly remark that we are proposing to apply Parzen windowing just at the output level of the LM classifier, in the definition of the cost to be minimized during training. This definition is independent of the classifier architecture, whose internal layers can use any form of activations, even including radial basis functions.

The rest of the paper is organized as follows. The basic formulation of our approach and the gradient-type algorithms that serve to optimize the estimated Bayesian objective are introduced in Section 2. Some characteristics of Parzen windows, focusing on their role in the training algorithm, are discussed in Section 3. In Section 4, the learning rules obtained using some particular windows are presented, and the equivalence of some of them with classical methods such as the perceptron rule is proved. A series of experiments that illustrate the benefits of adopting some forms of windows are presented in Section 5. The main conclusions of this work and some avenues for further research close our contribution.

2. Training of neural network classifiers minimizing Bayesian risk by Parzen windows

The basic problem of binary classification may be described simply as follows. Given a training set $\mathcal{T}$,

$$\mathcal{T} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\} \tag{1}$$

consisting of N pairs of labeled patterns, $(\mathbf{x}_i,\ y_i)$, where $\mathbf{x}_i$ are vectors and $y \in \{\pm 1\}$ are target values that represent one of two classes, it is assumed that there is some unknown target function,

$$f : \mathcal{X} \longrightarrow \mathcal{Y} \tag{2}$$

mapping $\mathbf{x}$ to y that is to be learned from the training data. The goal is then to find a function $g(\mathbf{x})$ within a set of functions, $\mathcal{F}$, for predicting y from $\mathbf{x}$, where the function minimizes some error, or is optimal according to some criterion. In this work, the set of functions (classifiers) that are considered are those that correspond to a neural network with a single output and a threshold-based decision. Thus, each function in $\mathcal{F}$ is the soft output of the network that is a nonlinear function,

$$z = g(\mathbf{x}, \mathbf{w}) \tag{3}$$

where $\mathbf{w}$ is a set of trainable parameters, and the decision rule of the classifier is

$$\widehat{y} = \operatorname{sgn}(z) \tag{4}$$

The analytical expression of $g(\mathbf{x}, \mathbf{w})$ in terms of the parameters $\mathbf{w}$ depends on the architecture of the neural network. The proposed training method is valid for every possible architecture, such as an MLP with one or several hidden layers, or a Radial Basis Function (RBF) network, just to mention the most common architectures; and with every possible activation function in the neurons of the network (hyperbolic tangent, rectified linear units, Gaussian units for RBF's, etc.). But it can also be applied to a linear classifier, i.e., $z = g(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\mathbf{x}$.

Once that the neural network architecture is fixed, and thus the analytical expression of $g(\mathbf{x}, \mathbf{w})$ is fixed, the neural network parameters that are to be found are those that minimize the following simplified Bayes' risk[1]

$$\mathcal{R} = c_{-1}\Pr(\widehat{y}=1|y=-1)\Pr(y=-1) + c_1\Pr(\widehat{y}=-1|y=1)\Pr(y=1) \tag{5}$$

where c_i is the cost of making an error when the correct class is i. The probabilities $\Pr\{y = i\}$ may be estimated from the relative number of samples of each class in the training set, but estimating the conditional probabilities can be more difficult. However, since $\mathbf{x}$ is classified according to the decision rule $\widehat{y} = \operatorname{sgn}(z)$, where the output of the neural network, z, is one-dimensional, then the conditional probabilities are

$$\Pr(\widehat{y} = 1|y = -1) = \int_0^\infty p(z|y = -1)\ dz \tag{6}$$

$$\Pr(\widehat{y} = -1|y = 1) = \int_{-\infty}^0 p(z|y = 1)\ dz \tag{7}$$

Because the conditional densities $p(z|y)$ are unknown, a large number of training samples in each class may be necessary in order to estimate them accurately. However, z is a one-dimensional variable, and all that is required are estimates of the integrals of those conditional densities, and not the densities themselves, so it may not be as critical to have a large training set. Therefore, we consider to use Parzen window estimates of the conditional densities $p(z|y = i)$ from the set of outputs $\{z_n\}$ associated to the labeled training set $\{(\mathbf{x}_n,\ y_n)\}$ to obtain an estimate of the Bayes risk (5). Note that this approach is notably different of using Parzen windows to obtain estimates of the conditional distributions of the input, $p(\mathbf{x}|y = i)$ or the joint input-output distributions, $p(\mathbf{x}, y)$, such as in [28,29]. These distributions related with the input patterns are multi-dimensional, while here Parzen method is applied to estimate conditional densities at the output of the neural network, $p(z|y = i)$, which are one-dimensional. Parzen window estimates of these distributions are as follows

$$\widehat{p}(z|y = i) = \frac{1}{N_i}\sum_{n\in S_i} k_i(z - z_n) \quad ; \quad i \in \{\pm 1\} \tag{8}$$

where

$$S_1 = \{n : y_n = 1\} \quad \text{and} \quad S_{-1} = \{n : y_n = -1\} \tag{9}$$

and where N_i is the number of samples in S_i and $k_i(z)$ is the Parzen window used to estimate $p(z|y = i)$. Note that, in order to be a valid window, it is necessary that $k_i(z) \geq 0$ and has unit area. Substituting the Parzen estimate of the conditional densities into the conditional probabilities in Bayes' risk gives

$$\Pr(\widehat{y} = 1|y = -1) = \frac{1}{N_{-1}}\sum_{n\in S_{-1}}\int_0^\infty k_{-1}(z - z_n)dz \tag{10}$$

$$\Pr(\widehat{y} = -1|y = 1) = \frac{1}{N_1}\sum_{n\in S_1}\int_{-\infty}^0 k_1(z - z_n)dz \tag{11}$$

Since these probabilities involve integrals of the Parzen windows, define $K_i(z)$ to be the integral of the window,

$$K_i(z) = \int_{-\infty}^z k_i(\alpha)d\alpha \tag{12}$$

An example is given in Fig. 1, where the Parzen window is a rectangular pulse. Note that since $k_i(z)$ has the form of a probability

[1] Note that in this definition of the risk, the costs of taking correct decisions in the classical Bayesian formulation have been neglected, which is a common assumption that does not limit the validity of the formulation.

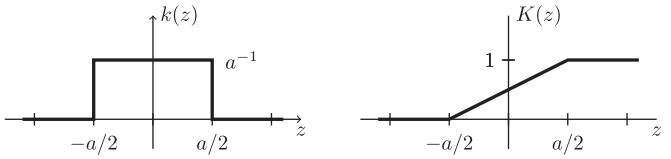

Fig. 1. A Parzen window $k(z)$ and the integrated window $K(z)$.

density function, then $K_i(z)$ is a valid probability distribution function.

The integrals in the conditional density estimates now become

$$\int_0^\infty k_{-1}(z-z_n)dz = \int_{-z_n}^\infty k_{-1}(\alpha)d\alpha = 1 - K_{-1}(-z_n) \tag{13}$$

$$\int_{-\infty}^0 k_1(z-z_n)dz = \int_{-\infty}^{-z_n} k_1(\alpha)d\alpha = K_1(-z_n) \tag{14}$$

and considering $\Pr(y=i) = N_i/N$, the estimate of the Bayes risk may now be expressed as

$$\hat{\mathcal{R}} = \sum_{n\in S_{-1}} \bar{c}_{-1}[1 - K_{-1}(-z_n)] + \sum_{n\in S_1} \bar{c}_1 K_1(-z_n) \tag{15}$$

where $\bar{c}_i = c_i/N$ for $i = \pm 1$.

Now, the goal is to minimize this estimate of the Bayes risk. Since $K_i(z)$ is the integral of a Parzen window, then $K_i(z)$ is differentiable along with $\hat{\mathcal{R}}$. Therefore, a gradient descent algorithm may be used to minimize (15), according to

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \mu \frac{\partial \hat{\mathcal{R}}}{\partial \mathbf{w}}\bigg|_{\mathbf{w}(n)} \tag{16}$$

where μ is the step-size parameter and the batch expression of gradient is

$$\frac{\partial \hat{\mathcal{R}}}{\partial \mathbf{w}} = \sum_{n=1}^{N} \frac{\partial \hat{\mathcal{R}}}{\partial z_n}\frac{\partial z_n}{\partial \mathbf{w}} \tag{17}$$

The partial derivatives of z_n with respect to $\mathbf{w}$ depend on the network architecture, and the partial derivatives of $\hat{\mathcal{R}}$ with respect to z_n depend on the windows. Differentiating $\hat{\mathcal{R}}$ with respect to z_n it follows that

$$\frac{\partial \hat{\mathcal{R}}}{\partial z_n} = -y_n \bar{c}_{y_n} k_{y_n}(-z_n) \tag{18}$$

The update equation for instantaneous (sample-by-sample) gradient takes the form

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu y_n \bar{c}_{y_n} k_{y_n}(-z_n)\frac{\partial z_n}{\partial \mathbf{w}}\bigg|_{\mathbf{w}(n)} \tag{19}$$

Note that the form of the update, i.e., how much a given weight vector is changed by a pattern $\mathbf{x}_n$ of class i, depends explicitly on the form of the window $k_i(z)$ being used. In terms of computational complexity, note that the difference with conventional cost functions, such as MMSE, is (18), which only requires to evaluate the kernel function. Therefore, computational complexity is of the same order than in conventional techniques.

3. On window characteristics

In this section we will discuss some characteristics of the Parzen windows and we will introduce a new related function, the projection function, that makes easier to understand the role of Parzen windows in the risk function and thus in the training algorithm.

The first thing to note is that there are (conceptually) no constraints on the Parzen windows that may be used other than that they are valid windows. Each window $k_i(z)$ may be different and designed independently.

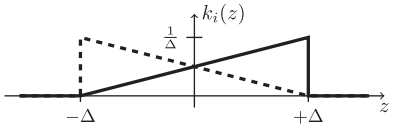

Fig. 2. A linear window $k_1(z)$ (solid line) and its complement $k_{-1}(z)$ (dashed line).

3.1. Complementary windows

Consider the special case where the windows $k_i(z)$ are complementary in the sense that

$$k_{-1}(z) = k_1(-z) \tag{20}$$

With complementary windows, it is only necessary to design one window, say $k_1(z)$, since the other is defined from the first by mirror symmetry. In this case, we design a single window, $k(z)$, which we call the Parzen kernel, and the pair of Parzen windows generated from this kernel is

$$k_t(z) = k(tz), \qquad t = \pm 1 \tag{21}$$

With complementary Parzen windows, the estimate of the Bayes risk may be expressed more concisely as

$$\hat{\mathcal{R}} = \sum_n \bar{c}_{y_n} K(-y_n z_n) \tag{22}$$

where $K(z)$ is the integrated Parzen kernel, and the update equation to minimize the risk using instantaneous gradient descent becomes

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu y_n \bar{c}_{y_n} k(-y_n z_n)\frac{\partial z_n}{\partial \mathbf{w}}\bigg|_{\mathbf{w}(n)} \tag{23}$$

In addition, note that in the special case where we have a linear classifier, $z = g(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\mathbf{x}$, the update equation is

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu y_n \bar{c}_{y_n} k(-y_n z_n)\mathbf{x}_n \tag{24}$$

3.2. Asymmetrical windows

When using Parzen windows to estimate a probability density function, it is customary to use a window that is symmetric since there is no reason to bias it away from the sample. However, in minimizing the Bayes risk, the purpose of the window is to establish a classification boundary that places the output z_n for input pattern $\mathbf{x}_n$ on the correct side of the decision threshold, $z = 0$. Therefore, it may be appropriate to have a larger penalty assigned to samples that are on the wrong side. This can be done with an asymmetrical kernel, such as

$$k(z) = \begin{cases} \dfrac{z+\Delta}{2\Delta^2}, & |z| \le \Delta \\ 0, & |z| > \Delta \end{cases} \tag{25}$$

The two complementary windows obtained from this kernel, $k_1(z) = k(z)$ and $k_{-1}(z) = k(-z)$, are shown in Fig. 2. If the kernel has a finite support as in this case, note that when a sample is located within a distance Δ of the boundary $z = 0$, the contribution to the Bayes risk increases as the sample moves in the direction of the wrong side of the decision boundary and continues to increase as it crosses the decision boundary where it becomes misclassified. Moreover, since the gradient is proportional to $k_{y_n}(-z_n)$, samples that are far away from the decision boundary, whether they are correctly or incorrectly classified, do not contribute to the adjustment of the weights $\mathbf{w}$. Therefore, this window results in an adjustment of the decision surface (a change in $\mathbf{w}$) only for samples whose output z_n is within a distance Δ of $z = 0$, and the stochastic gradient descent algorithm for this kernel works to push these

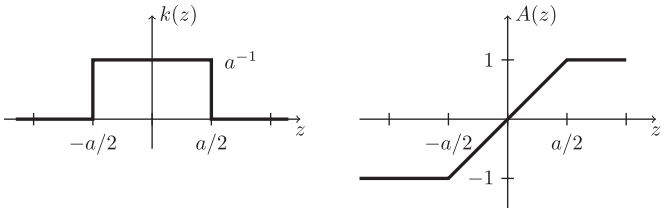

Fig. 3. The projection function $A(z)$ for a kernel that is a pulse.

samples to the correct side of the decision boundary until they are at a distance of Δ or greater away from the boundary.

It is worthwhile commenting on the role of symmetry in the kernel. Note that the shape of the kernel function determines the size of the update used in the weight update, which changes the classification boundary so that the sample is closer to or deeper inside the correct decision region. With symmetric kernels, the size of this correction depends only on the distance of the output z_n from zero, whether or not the sample is correctly classified. But the use of asymmetric kernels allows for the size of the update to be different for samples that are correctly compared to those that are incorrectly classified. For example, using the kernel in Fig. 2, the correction will be larger for those samples that are incorrectly classified, and the size of the correction increases linearly as the sample moves away from the decision boundary. This capability to penalize samples differently depending upon whether they are correctly or incorrectly classified can help to obtain a better classifier.

3.3. Projection functions

Since the integrated windows $K_i(z)$ are non-decreasing functions of z and $0 \le K_i(z) \le 1$, then they may be used to define a projection function $A_i(z)$ which is helpful to understand the role of Parzen windows in the proposed risk function (15). Specifically, if we define

$$A_i(z) = 1 - 2K_i(-z) \tag{26}$$

then $A_i(z)$ is a monotonically non-decreasing function and $-1 \le A_i(z) \le 1$. An example is given in Fig. 3, which shows a kernel that is a pulse and the corresponding projection function. Note that as the width of $k(z)$ in this example decreases and $k(z)$ approaches an impulse, $K(z)$ approaches a step function, and the projection function becomes the sign function, $A(z) = \mathrm{sgn}(z)$.

When the kernel is symmetric, then the associate projection function is odd

$$A(z) = -A(-z) \tag{27}$$

and with complementary windows, these functions have the property:

$$A_1(z) = -A_{-1}(-z) \tag{28}$$

It is interesting to mention that, because of their properties, these projection functions are analogous to some activation functions used in the neurons of many neural networks. However, the role of these projection functions is completely different, because they are not used in the neural network itself but in the proposed risk function associated to the training algorithm. The estimate of Bayes' risk $\hat{\mathcal{R}}$ may be expressed in terms of $A_i(z)$. Specifically, since

$$K_i(-z) = \frac{1}{2}[1 - A_i(z)] \tag{29}$$

then (15) may be written as

$$\hat{\mathcal{R}} = \sum_n \bar{c}_{y_n} |y_n - A_{y_n}(z_n)| \tag{30}$$

Therefore, we find that minimizing $\hat{\mathcal{R}}$ is equivalent to minimizing the absolute error between y_n and the projection function evaluated at network output z_n, $A_{y_n}(z_n)$. Thus, $A_i(z)$ can be seen as a function projecting the output of the neural network towards the desired label for the class of each pattern, and this projection defines an error with respect to the label. Clearly, different projection functions lead to different errors and, in fact, one may define an error to minimize and then determine the corresponding associate projection. Some examples will be given in Section 4.4.

4. Some particular windows

This section is dedicated to analyze the proposed risk function, which has to be minimized during training, for some specific Parzen windows. This analysis shows that the proposed framework expands a general family of training algorithms, with a different risk $\hat{\mathcal{R}}$ for every possible choice of the pair of Parzen windows $k_i(z)$ used to estimate likelihoods $p(z|y=i)$, for $i = \pm 1$. And, more interesting, that this general framework includes as particular cases several well known training rules when some specific windows are selected.

4.1. An impulse

Consider the special and simple case of complementary windows with a Parzen kernel that is an impulse, $k(z) = \delta(z)$.[2] In this case, the estimate of the likelihood $p(z|y=i)$ is a probability density function of the form

$$\hat{p}(z|y=i) = \frac{1}{N_i} \sum_{n \in S_i} \delta(z - z_n) \tag{31}$$

The integrated kernel $K(z)$ is a step function, $K(z) = u(z)$, and the risk is

$$\hat{\mathcal{R}} = \sum_n \bar{c}_{y_n} u(-y_n z_n) \tag{32}$$

Note that for a misclassified sample $y_n z_n < 0$, and the misclassified sample contributes a value $\bar{c}_{y_n}$ to the risk. On the other hand, if a sample is correctly classified, then $y_n z_n > 0$ and it contributes nothing. Particular interest has the case when the costs are equal, $\bar{c}_1 = \bar{c}_{-1} = 1/N$. Minimizing the risk $\hat{\mathcal{R}}$ is then equivalent to minimizing the average misclassification error on the training samples,

$$\hat{\mathcal{R}} = \frac{1}{N} \sum_n I[\mathrm{sgn}(z_n) \ne y_n] \tag{33}$$

If the kernel is an impulse that is shifted away from the origin by an amount a,

$$k(z) = \delta(z + a) \tag{34}$$

then this results in an integrated kernel that is a shifted step function, $K(z) = u(z + a)$, and the risk becomes

$$\hat{\mathcal{R}} = \sum_n \bar{c}_{y_n} u(-y_n z_n + a) \tag{35}$$

Note that shifting the impulse by $a > 0$ has the effect of shifting the estimated conditional density functions $p(z|y)$ for each class towards each other. The effect is to impose a form of margin in the sense that correctly classified samples that are within a distance a of the decision boundary $z = 0$ will contribute to the risk. The same effect can be obtained with other windows.

[2] Although this window is useless for gradient descent minimization of the risk function, given its special support, we analyze this case because of some interesting properties of the risk function.

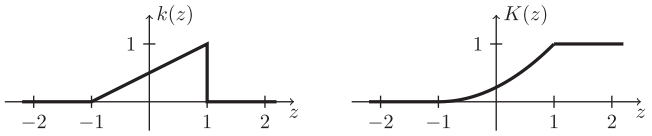

Fig. 4. Linear kernel with finite support $[-1, 1]$ and corresponding integrated kernel.

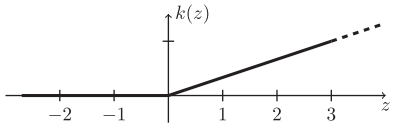

Fig. 5. A linear ramp kernel.

4.2. Step function

Now consider the case in which the kernel is a unit step, $k(z) = u(z)$ (although not a valid Parzen window for probability density function estimation, we will ignore this). The integrated kernel is a ramp, $K(z) = zu(z)$, and if the costs are $\bar{c}_1 = \bar{c}_{-1} = 1$ then the estimated risk is

$$\hat{\mathcal{R}} = \sum_n |z_n| u(-y_n z_n) \tag{36}$$

Note that with the decision rule $\hat{y}_n = \operatorname{sgn}(z_n)$, when $y_n z_n < 0$ the pattern is misclassified and its contribution to the risk is $|z_n|$, which is the distance of the output from the decision threshold, $z = 0$. On the other hand, when $y_n z_n > 0$ the pattern is correctly classified and contributes nothing to the risk. If an instantaneous gradient descent algorithm is used to minimize the risk with $\mu = 1$, and for a linear classifier, i.e. $z = g(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\mathbf{x}$, then

$$\mathbf{w}(n+1) = \mathbf{w}(n) + y_n \mathbf{x}_n; \quad \text{when} \quad y_n z_n < 0 \tag{37}$$

which we see is the well-known Perceptron Algorithm.

4.3. Linear kernel

When the kernel is linear over a finite interval and zero otherwise, such as the one illustrated in Fig. 4 where $k(z)$ is linear over the finite interval $[-1, 1]$,

$$k(z) = \frac{1}{2}(z+1), \quad z \in [-1, 1] \tag{38}$$

then the integrated kernel is a quadratic function

$$K(z) = \frac{1}{4}(z+1)^2, \quad z \in [-1, 1] \tag{39}$$

that increases from zero to one along the interval $[-1, 1]$.

For this kernel, it follows that the risk is

$$\hat{\mathcal{R}} = \sum_{n: y_n z_n < 1} \bar{c}_{y_n} \min\{1, e^2(n)\} \tag{40}$$

where the sum is taken over all samples for which $y_n z_n < 1$, and where

$$e(n) = \frac{1}{2}\big(y_n - z_n\big) \tag{41}$$

Thus, it follows that a finite linear kernel leads to a clipped least squares minimization problem. If an instantaneous gradient descent algorithm is used to minimize the risk, and for a linear classifier, the weight update equation has the form

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \mu \bar{c}_{y_n} e(n) \mathbf{x}_n; \quad \text{when } |z_n| \le 1 \tag{42}$$

Note that the weights are only updated for those patterns whose output is within a unitary distance of the decision threshold, $z = 0$.

For training a neural network, note that if the support of the kernel function is $[-1, 1]$ as in this example, and if the activation function of the neuron in the output layer saturates over this interval as the common hyperbolic tangent function does, then this kernel leads to a conventional least squares minimization.

Other kernels may be formed by shifting and/or scaling the linear kernel, and these lead to different error minimization problems. For example, suppose the linear kernel is shifted to the origin and allowed to increase linearly,

$$k(z) = \max\{0, z\} \tag{43}$$

This kernel, illustrated in Fig. 5, is a ramp function and, again, as with the step, although not a valid kernel, from a practical point of view this is not a serious issue since the samples z_n are bounded. With this kernel, one has a quadratic error for all misclassified samples.

4.4. Minimization of $\mathcal{L}_p$ norm

The definition of projection functions $A_i(z)$ and the corresponding risk equivalence (30) provide the proposed method with flexibility to define many different cost functions by selecting an appropriate projection function and, therefore, an appropriate kernel. For example, suppose that it is desired to minimize an $\mathcal{L}_p$ error of the form

$$\mathcal{L}_p = \sum_n \bar{c}_{y_n} |y_n - z_n|^p \tag{44}$$

If the neural network output is saturated, i.e. $|z| \le 1$, and the projection functions are defined to be

$$A_{y_n}(z) = y_n[1 - (1 - y_n z)^p] \tag{45}$$

substituting these projections into the risk (30) results in an $\mathcal{L}_p$ error. To determine the form of the corresponding windows, $k_t(z)$, and assuming that they have support in $-1 \le z \le 1$, it follows that in this support

$$K_t(z) = \frac{1}{2^p}\big(1 - t\big) + \frac{1}{2^p} t(1 + tz)^p \tag{46}$$

and the windows are then found by differentiating with respect to z,

$$k_t(z) = \frac{d}{dz} K_t(z) = \frac{p}{2^p}(1 + tz)^{p-1} \quad ; \quad \text{if } |z| \le 1 \tag{47}$$

Clearly, these windows are complementary.

It is now interesting and worthwhile to look at two specific cases: $p = 1$ and $p = 2$. In the first case, when $p = 1$, the kernel is a pulse as shown in Fig. 3 with $a = 2$, and the projection function is $A(z) = z$ for $z \in [-1, 1]$ as also shown in Fig. 3. In the second case, when $p = 2$, the kernel is linear over the interval $[-1, 1]$, which is the kernel shown in Fig. 4. Finally, the kernels that produce $\mathcal{L}_p$ error minimization in neural networks with output saturated in $[-1, 1]$ for $p \in \{1, 2, 3, 4\}$ appear in Fig. 6.

5. Some experiments and their discussion

A number of experiments have been designed to illustrate the potential advantages in the approach of training neural net classifiers using specially designed kernel functions. The main purpose of this section is to illustrate the flexibility of the proposed method. The proposed cost function is independent of the architecture of the neural network. It has been implemented in a multilayer perceptron with a single hidden layer, in a generalized radial

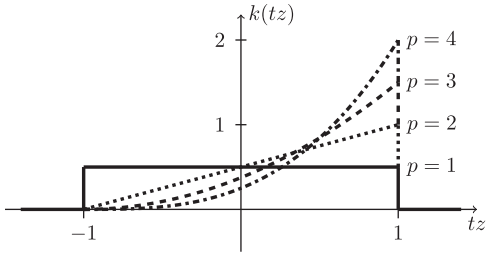

Fig. 6. Kernels to minimize $\mathcal{L}_p$ error in neural networks with saturation of output in $[-1, 1]$.

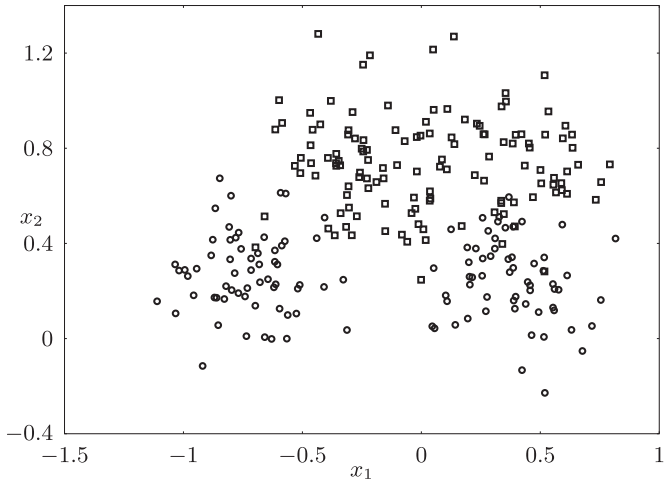

Fig. 7. Training set of the synthetic problem (circles for class -1, squares for class $+1$).

basis function network, allowing different variances for each input dimension [30], where centers, variances and weights are randomly initialized and updated by gradient descent, and in a convolutional neural network. Moreover, several kernel functions have been tested on the same data sets to demonstrate that an appropriate choice for this kernel, given the characteristics of the specific problem at hand, can improve the performance of the method.

5.1. A synthetic toy problem

The first block of experiments deals with a synthetic problem that allows to illustrate the advantage of selecting an appropriate kernel function. In particular, a two-dimensional dataset composed of a mixture of two Gaussians for each class is considered. Centers for class -1 are $[-0.7, 0.3]$ and [0.4, 0.25], with variance 0.03. Centers for class $+1$ are $[-0.25, 0.7]$ and [0.4, 0.75], with variance 0.04. Bayes rule has an accuracy rate of 92%. Training set consists of 250 patterns, which are shown in Fig. 7.

The performance of two neural networks has been tested:

- A multilayer perceptron with a single hidden layer with 10 neurons, and a single neuron in the output layer, with hyperbolic tangent activation functions in the neurons of both the hidden and the output layers.
- A generalized radial basis function network with 10 Gaussian neurons in the hidden layer and a single neuron in the output layer with a hyperbolic tangent activation to saturate the network output.

For the proposed cost function, complementary windows are considered, and the four kernels shown in Fig. 8 are tested. Superscripts denote uniform (U), linear (L), triangle (T), and absolute value (A), respectively. All of them have support $[-1, 1]$, which is

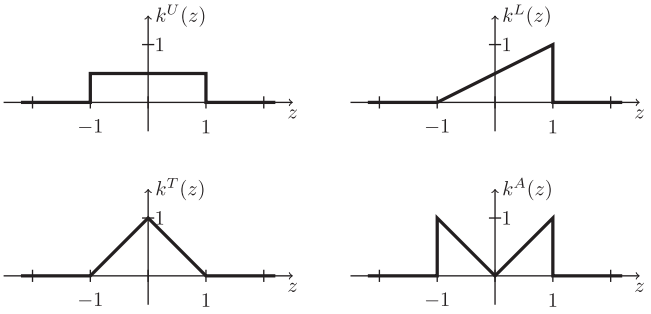

Fig. 8. Kernels under test for experiments with complementary windows.

Table 1
Average results obtained in the synthetic dataset (accuracy rate in %, and standard deviation).

	$k^U(z)$	$k^L(z)$	$k^T(z)$	$k^A(z)$
MLP	87.69 ($\pm$0.041)	91.53 ($\pm$0.005)	87.75 ($\pm$0.065)	89.70 ($\pm$0.110)
RBF	91.79 ($\pm$0.019)	91.81 ($\pm$0.025)	91.64 ($\pm$0.029)	91.55 ($\pm$0.013)

the range of the output z if a hyperbolic tangent activation function is used to saturate the output, as in this case. It is interesting to remind that for networks with such a saturated output and with $c_{-1} = c_1$, estimated risk (15) is equivalent to the conventional $\mathcal{L}_1$ cost function if $k^U(z)$ is used, and to $\mathcal{L}_2$ cost function, or Minimum Mean Squared Error (MMSE), if $k^L(z)$ is used.

A brief discussion about the effect of a given kernel in the training algorithm is appropriate. From equation (18), the intensity in the gradient that is associated to a given pattern $\mathbf{x}_n$ is proportional to $k_{y_n}(-z_n)$. For the linear kernel, as it was already discussed in Section 3.2, this intensity is proportional to the difference between the desired label, $y_n = \pm 1$, and the network output for this pattern, z_n. Therefore, this kernel emphasizes the role in the update of misclassified patterns, proportionally to their distance to y_n. Choosing the uniform kernel, the intensity is the same for all patterns, independently of their output. The triangle kernel will emphasize the contribution of patterns whose output is close to the decision threshold, and the absolute value kernel will do the contrary, increasing the contribution of patterns that are very well fitted (with an output close to the desired label) or very poorly fitted (with an output far away on the wrong side of the threshold).

Results obtained averaging 200 independent runs, with random initialization of the network parameters, are shown in Table 1. For each run, the same initial parameters were used with the four kernels under test. In this ideal scenario, where samples of the training set represent accurately the underlying distribution of the problem at hand, the best results are obtained using the linear kernel. This was expected because it is well known that, under these circumstances, MMSE asymptotically converges to the Bayesian solution [31]. Therefore, with this kernel both the MLP ad the RBF obtain results that are close to the Bayesian limits. The difference in performance using different kernels is higher for MLP than for RBF in this case. The different performance of MLP and RBF is explained by the different projection that these networks perform from input patterns to network outputs, which is where the cost function is computed. Obviously, for this specific example the RBF is better suited than the MLP.

To illustrate that in some cases other kernels can be more adequate, 40 outliers were introduced in the training set by changing the label y_n of 20 patterns of each class. The average results for 200 independent runs in this new scenario are shown in Table 2.

In this case, performance is obviously lower than above and the linear kernel exhibits the worst performance for both networks, MLP and RBF. The outliers, whose output tends to be on the wrong

Table 2
Average results obtained in the synthetic dataset with outliers in the training set (accuracy rate in % along with standard deviation).

	$k^U(z)$	$k^L(z)$	$k^T(z)$	$k^A(z)$
MLP	87.41 (±0.054)	79.77 (±0.412)	87.74 (±0.089)	87.40 (±0.642)
RBF	82.70 (±0.769)	80.46 (±0.184)	90.24 (±0.740)	81.04 (±1.071)

Table 3
Characteristics of the benchmark databases.

Database	Dimension	N_0/N_1 train	N_0/N_1 test
Abalone	8	1269/1238	827/843
Hand	62	1900/1923	891/906
Image	18	1027/821	293/169
Kwok	2	200/300	4080/6120
Ripley	2	125/125	500/500
Wave	21	1306/2694	341/659

side of the decision threshold with a high distance with respect to the desired label, can be pushing the gradient towards the wrong side with a relatively high intensity. The uniform kernel is less sensitive to the outliers than the linear one, because the intensity in the gradient is the same for all patterns, independently of the difference between the desired label and the network output for a given pattern. This better behavior of the $\mathcal{L}_1$ norm than the $\mathcal{L}_2$ norm in the presence of outliers was also expected. But, interestingly, there is even a better option. The triangle kernel is that providing the best results in this scenario. Using this kernel, the training algorithm is emphasizing patterns whose output is close to the decision threshold. Taking into account the overlapping between the samples of both classes, this is the option that is less sensitive to the presence of outliers far away from the threshold in this scenario. With respect to the absolute value kernel, it would be appropriate for instance for problems where outliers are concentrated in the region which is close to the boundary, when focusing on samples far away to this boundary can be the best option. This is not the case in this example, but later we will show that in some real problems this kernel can be useful.

5.2. Balanced datasets

The second block of experiments are carried out for a number of well known datasets where conventional designs have shown a relatively good performance. The reason for this selection is to show that even in this scenario, the proposed cost function along with an appropriate choice of the kernel function can improve the classification performance using the same network architecture. In particular, the proposed method has been tested on six binary problem databases. Two synthetic, *Kwok* and *Ripley*, from [32] to [33], respectively, and four real data sets, *Abalone, Hand, Image*, and *Wave*, from the UCI repository [34]. Table 3 shows the main characteristics of the benchmark databases. The number of patterns of each class in all these datasets is relatively balanced.

For this set of experiments, non complementary windows are considered, to analyze also non-symmetric situations. In particular, windows used in Parzen estimators for both classes are derived by shifting the clipped linear kernel of Eq. (25) with $\Delta = 1$, which is shown in Fig. 2. More specifically, in these experiments the windows are

$$k_1(z) = k(z - \delta_1), \qquad k_{-1}(z) = k(-z + \delta_{-1}) \tag{48}$$

Including a shift of δ_1 in $k_1(z)$ means that once that a sample of class 1 is at a distance $1 - \delta_1$ of the threshold (on the right side), it does not contribute to update weights. Equivalently, samples of class -1 that are at a distance of at least equal to $1 - \delta_{-1}$ (on the right side) produce no updates in the weights.

Experiments were performed using different shifts for kernels of Parzen estimators, with the best shifting parameters δ_{-1} and δ_1 selected by cross-validation (CV) among the following values:

$$\delta_{-1}, \delta_1 \in \{0.1, 0.2, 0.3, 0.4, 0.5\}. \tag{49}$$

The figure of merit is the probability of error, so equal costs are considered for errors in both classes, i.e., $c_{-1} = c_1$. The proposed method has been tested with MLPs with a single hidden layer of H neurons and a single neuron in the output layer. The activation function for the neurons in the hidden layer was the hyperbolic tangent function, and for the neuron in the output layer, two different options were tested:

- A linear activation function
- A hyperbolic tangent activation function

The difference between these two architectures, which are denoted by "MLP-L" and "MLP-T", respectively, is that the second includes saturation of its output, making $-1 \leq z_k \leq +1$.

Table 4 presents the results (accuracy rate in percentage) obtained using the proposed cost function and the conventional error cost functions for $\mathcal{L}_1$ and $\mathcal{L}_2$ norms. In all cases, the cost function is minimized by means of gradient descent. The result for the best combination of shifts δ_{-1} and δ_1 is provided in the case of the algorithms we propose, MLP-L(P) and MLP-T(P), which include Parzen windowing. Results are obtained averaging 100 independent realizations (networks trained from different initial parameters). The initial parameters are identical for all methods in each realization. MLPs with 2, 4, 8, 12, 16, 30, 40, 50 and 75 hidden neurons have been trained. The best network size for each method and each database has been selected by cross-validation.

First of all, note that the proposed algorithms never are worse than the conventional methods, and in some cases they are slightly

Table 4
Accuracy rates obtained using multilayer perceptron with linear activation (MLP-L) and hyperbolic tangent activation function (MLP-T), and with three different cost functions: $\mathcal{L}_1$ norm, $\mathcal{L}_2$ norm (MMSE), and proposed method. Cross-validated values for H, δ_{-1} and δ_1 are shown.

Database	MLP-L ($\mathcal{L}_1$) (H)	MLP-T ($\mathcal{L}_1$) (H)	MLP-L ($\mathcal{L}_2$) (H)	MLP-T ($\mathcal{L}_2$) (H)	MLP-L (P) (H, δ_{-1}, δ_1)	MLP-T (P) (H, δ_{-1}, δ_1)
Abalone	77.05 (±0.32) (75)	80.53 (±0.60) (50)	81.10 (±0.09) (75)	81.85 (±0.30) (16)	81.63 (±0.18) (8, 0, 0.1)	82.07 (±0.39) (16, 0, 0.1)
Hand	90.72 (±0.09) (2)	95.05 (±0.17) (30)	97.46 (±0.29) (50)	97.53 (±0.28) (50)	97.05 (±0.24) (30, 0.1, 0.2)	97.54 (±0.28) (50, 0, 0)
Image	92.13 (±1.98) (8)	97.19 (±0.26) (30)	97.00 (±0.49) (8)	97.56 (±0.36) (16)	97.82 (±0.31) (16, 0.3, 0.1)	97.88 (±0.28) (16, 0.4, 0.2)
Kwok	80.27 (±0.02) (4)	80.22 (±0.01) (30)	88.50 (±0.04) (8)	88.19 (±0.05) (16)	88.52 (±0.09) (4, 0.2, 0.2)	88.51 (±0.09) (4, 0.5, 0.5)
Ripley	89.35 (±0.31) (50)	88.60 (±0.81) (50)	90.65 (±0.54) (8)	90.29 (±0.08) (8)	93.30 (±0.17) (75, 0.1, 0.5)	93.24 (±0.26) (4, 0, 0.5)
Wave	87.94 (±0.26) (4)	92.39 (±0.12) (8)	92.75 (±0.19) (8)	92.55 (±0.17) (4)	92.88 (±0.11) (16, 0.3, 0.2)	92.90 (±0.11) (8, 0.5, 0.4)

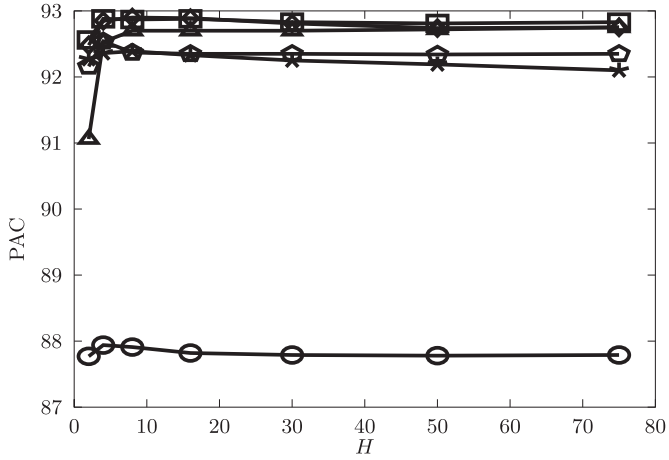

Fig. 9. Evolution of percentage of accuracy (PAC) as a function of the number of neurons in the hidden layer for dataset Wave. The following markers are used for each method: MLP-L ($\mathcal{L}_1$) circle; MLP-T ($\mathcal{L}_1$) star; MLP-L ($\mathcal{L}_2$) triangle; MLP-T ($\mathcal{L}_2$) pentagon; MLP-L (P) square; MLP-T (P) diamond.

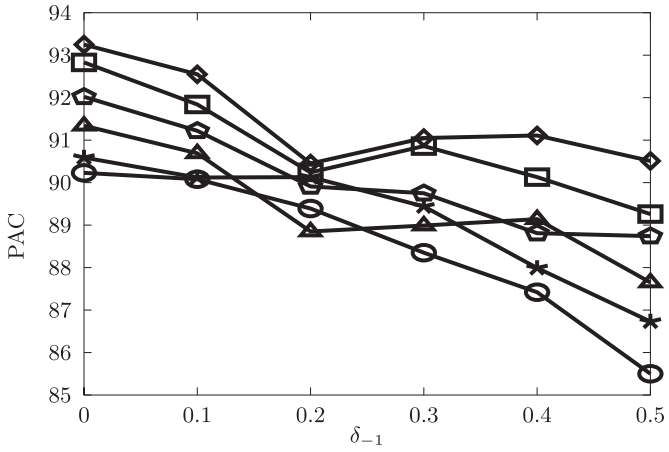

Fig. 10. PAC as a funcion of shifts for Ripley as a function of δ_{-1} for $\delta_1 = 0$ (circle), $\delta_1 = 0.1$ (star), $\delta_1 = 0.2$ (triangle), $\delta_1 = 0.3$ (pentagon), $\delta_1 = 0.4$ (square), and $\delta_1 = 0.5$ (diamond).

better (Image, Wave) or even clearly better (Ripley).[3] This supports the potential usefulness of our conceptual proposal, with no risk of reducing classification performance. With respect to the displacements δ_{-1} and δ_1, and the advantage of asymmetry, it can be seen that, in general, their values are similar, except in Image and Ripley. This seems to be related with the intrinsic asymmetry of these databases. More analysis is needed to fully understanding this issue. The sensitivity of all the designs with respect to the number of hidden units in the MLPs is relatively low in all cases, and it shows well-known variations. Fig. 9 shows the effects of selecting different values of H for the six methods when applied to the Wave dataset. There are underfitting effects for very low values of H, and overfitting effects when H is higher than their CV values. Overfitting is higher for the conventional $\mathcal{L}_1$ methods.

The sensitivity with respect to values of δ_{-1} and δ_1 for the proposed methods is in general higher and problem-dependent. Fig. 10 shows the PAC curves for MLP-T(P) with the Ripley database. The algorithm performance decreases with increasing values of δ_{-1}, and, in general, higher values of δ_1 are better. This is exactly true for $\delta_{-1} = 0$. Things are different for Abalone. Low

[3] It is worth to mention that the accuracy figures for Ripley can be higher than their theoretical limit because the sampling effects.

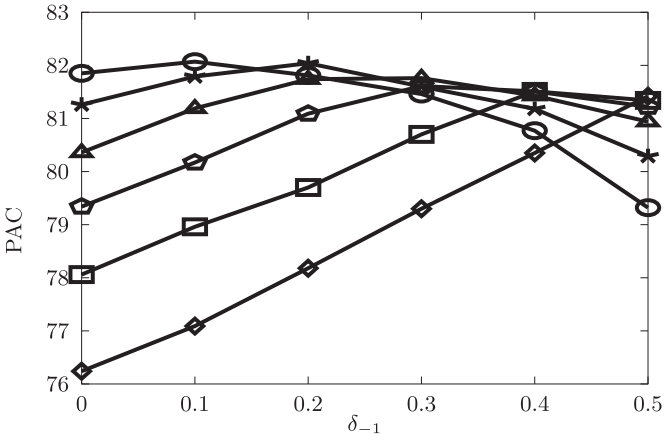

Fig. 11. PAC as a funcion of shifts for Abalone as a function of δ_{-1} for $\delta_1 = 0$ (circle), $\delta_1 = 0.1$ (star), $\delta_1 = 0.2$ (triangle), $\delta_1 = 0.3$ (pentagon), $\delta_1 = 0.4$ (square), and $\delta_1 = 0.5$ (diamond).

Table 5
Description of the databases used for experiments.

Database	Number of patterns	Dimensionality	Imbalance Ratio
Ecoli067vs5	220	6	10.00
Glass2	214	9	10.39
Led7digit	443	7	10.97
Cleveland0vs4	177	13	12.62
Shuttlec2vsc4	129	9	20.5
Yeast5	1484	8	32.78
Yeast6	1484	8	39.15
Abalone19	4174	8	128.87

and similar values for δ_{-1} and δ_1 are more appropriate, as it can be seen in Fig. 11.

Overall, we can conclude that the proposed framework can offer performance advantages if the window selection, including its parameters, is appropriate: A simple cross-validation process seems to be enough for this purpose.

5.3. Imbalanced datasets

The classification of imbalanced data has attracted a great attention in the last years. A problem is said to be imbalanced when the number of patterns of each class are very different. In a binary classification problem, learning from imbalanced data has the difficulty of representing the minority class, which can be shadowed in the thicker cloud of samples of the majority class. Moreover, there are many applications where the importance of detecting the minority class is even higher than the importance of detecting the majority class. Medical diagnosis or fraud detection are typical examples.

Different techniques have been used to deal with classification of imbalanced data: standard algorithms modified to compensate imbalance; preprocessing techniques that modify the data set to balance it (removing samples of the majority class, or introducing synthetic samples of the minority class); or ensembles of classifiers to improve the accuracy of individual classifiers. A detailed review of several of these methods can be found in [35,36].

The Bayesian formulation of the proposed method provides a simple mechanism to deal with imbalanced classification problems. It allows to specify different costs for errors classifying each class, c_{-1} and c_1, and at the same time to take into account the number of patterns of each class that are available in the training set.

We have tested the proposed method with several imbalanced real-world databases obtained from KEEL data-set repository [37].

Table 6
Average success probability p_S for the best method in [36] and the proposed method using four different kernels.

Database	Best in [36]	$k^U(z)$	$k^L(z)$	$k^T(z)$	$k^A(z)$
Ecoli067vs5	**89.00**	86.14 (±0.44)	88.36 (±1.59)	86.34 (±0.64)	86.93 (±0.32)
Glass2	80.45	82.08 (±3.82)	81.78 (±5.26)	81.49 (±4.17)	**83.43** (±1.25)
Led7digit	**88.80**	85.86 (±0.80)	87.85 (±0.90)	85.83 (±1.39)	85.12 (±1.62)
Cleveland0vs4	82.80	82.83(±0.68)	79.40 (±6.33)	82.62 (±2.47)	**83.03** (±1.45)
Shuttlec2vsc4	100	99.68 (±0.16)	100 (±0)	100 (±0)	99.51 (±0.19)
Yeast5	96.61	97.60 (±0.22)	**97.75** (±0.18)	97.64 (±0.02)	97.64 (±0.02)
Yeast6	86.78	86.23 (±0.54)	**87.36** (±0.90)	86.37 (±0.90)	85.61 (± 1.08)
Abalone19	70.81	79.50 (±0.64)	78.36 (±1.02)	79.33 (±0.84)	**80.10** (±2.38)

Data and information about these data sets can be found in the website http://www.keel.es/dataset.php. Data sets in [37] are organized in different k-fold partitions for training and test data. Here, we have worked with the 5-fold partition provided in KEEL-dataset repository, thus making easier to compare results. Results obtained for these 5 folds will be averaged. From this repository we have chosen several datasets with different values of imbalance ratio (defined as the ratio between the number of samples of the majority/minority classes). Table 5 shows the main characteristics of the tested databases, which are sorted according to the imbalance ratio.

The figure of merit for these imbalanced problems will be the average probability of a successful classification of samples of both classes, measured as

$$p_S = \frac{\Pr(\widehat{y}=1|y=1)+\Pr(\widehat{y}=-1|y=-1)}{2} \tag{50}$$

This figure of merit assigns the same importance to errors in both classes, independently of the number of patterns of each class. In the proposed Bayesian risk, this situation corresponds to select costs satisfying the relationship

$$c_1 = c_{-1}\frac{\Pr(y=-1)}{\Pr(y=1)} \tag{51}$$

Table 6 compares the average results obtained with the proposed method using complementary windows and the four kernels shown in Fig. 8. The neural network architecture is an MLP network with a hidden layer of 4 neurons, and an output layer with a single neuron, using hyperbolic tangent nonlinearities in both layers. For each one of the 5 folds available for each dataset, 100 independent runs, with random parameter initialization, have been averaged. As a baseline result, Table 6 also includes the best result provided by all the methods compared in [36]. This work compares the performance obtained using ensembles of classifiers (10–40 classifiers), which is one of the methodologies providing better results in imbalanced problems [35], along with several preprocessing techniques used to balance data sets before constructing the ensemble, like random undersampling of the majority class or Synthetic Minority Over-sampling Technique (SMOTE) [38].

The proposed method obtains the best results in 5 of the 8 datasets, and ties in another one. It can be seen that for each dataset the best kernel is different. Interestingly, the greatest advantage is obtained in the dataset in [36] with the highest imbalance ratio, which is Abalone19. In the comparison, it is important to consider that the results provided by the proposed method have been obtained using a single classifier, instead of an ensemble of classifiers, and without any previous data preprocessing to balance the training set, while, in [36], up to 8 different approaches using different preprocessing techniques were tested. In our approach, problems are balanced by choosing appropriate Bayesian cost parameters c_{-1} and c_1.

Table 7
Probability of error for each class: positive, negative, average between positive and negative. Average results for the 10 digits.

	Conv	$k^U(z)$	$k^L(z)$	$k^T(z)$	$k^A(z)$
$P_{e\|+1}$ (%)	1.698	1.098	1.157	1.263	1.061
$P_{e\|-1}$ (%)	0.115	0.386	0.349	0.336	0.377
$(P_{e\|+1}+P_{e\|+1})/2$ (%)	0.906	0.742	0.753	0.800	0.719

5.4. Deep neural networks

The proposed risk function can be used with any kind of neural network, including deep networks, such as convolutional neural networks (CNNs). In balanced problems, the conventional training algorithms for CNNs provide state of the art results, specially for image classification, and the proposed method is only able of providing similar results. However, in imbalanced problems, the proposed Bayesian formulation can be helpful. To show the possible advantages, we have tested the proposed method in the well known MNIST dataset [39], where the one against the rest problem provides 10 binary imbalanced problem that can be used as an illustrative example. The positive class is given by a digit, and the negative class by the remaining 9 digits, thus giving 10 different problems, one per reference digit. Average imbalance ratio is therefore 9/1.

A CNN with two convolutional layers followed by max pooling, the first one with 32 features in 5×5 patches, and the second one with 64 features in patches of the same size. One additional hidden layer of 1024 neurons with rectifier linear units (ReLU) is used. Dropout has been used to reduce the risk of overfitting, with dropping probability of 20%. Two linear neurons (one per class) in the output layer were used for conventional training based on the cross-entropy cost function, which is the usual set up for this kind of networks. With the proposed method, a single linear neuron is used in the output layer.

Table 7 compares the probability of error for each class along with the average of these two probabilities. Probabilities are given in %, and the average results obtained in the identification of the 10 digits are shown. Costs are chosen according to (51).

It can be seen that the traditional training method tends to overfit the majority class in detriment of the minority class. However, the Bayesian formulation of the proposed method allows to compensate relatively the different number of samples of each class by introducing appropriate costs. In this problem, $k^A(z)$ and k^U provide the best results. Table 8 shows results for each individual digit, and it can be seen that the same behavior happens for all digits.

The degradation of performance to detect the minority class increases as the imbalance ratio increases. Table 9 shows the average results obtained when half of the samples of the reference digit are removed from the training set, thus leading to an average imbalance ratio of 18/1, and Table 10 when, in the same manner, the average imbalance ratio is increased to 90/1.

Table 8
Probability of error for each class: positive, negative, average between positive and negative. Results for each individual digit.

Method	0	1	2	3	4	5	6	7	8	9
	0.848	0.776	1.631	1.491	1.137	1.465	1.701	2.177	2.289	3.463
Conv	0.096	0.092	0.118	0.118	0.137	0.081	0.090	0.116	0.159	0.139
	0.472	0.434	0.874	0.805	0.637	0.773	0.895	1.146	1.224	1.801
	0.617	0.423	1.129	0.757	0.703	1.020	1.263	1.469	1.335	2.265
$k^U(z)$	0.230	0.174	0.370	0.497	0.320	0.379	0.233	0.367	0.741	0.547
	0.424	0.298	0.750	0.627	0.511	0.699	0.748	0.918	1.038	1.406
	0.704	0.471	1.216	0.891	0.794	1.076	1.315	1.420	1.386	2.294
$k^L(z)$	0.174	0.182	0.328	0.433	0.282	0.342	0.213	0.351	0.566	0.622
	0.439	0.326	0.772	0.662	0.538	0.709	0.764	0.886	0.976	1.458
	0.663	0.520	1.308	0.901	0.983	1.093	1.341	1.654	1.561	2.607
$k^T(z)$	0.215	0.167	0.322	0.415	0.321	0.295	0.210	0.353	0.624	0.442
	0.439	0.343	0.815	0.658	0.652	0.694	0.775	1.003	1.092	1.524
	0.577	0.414	1.153	0.713	0.723	0.897	1.253	1.391	1.258	2.235
$k^A(z)$	0.227	0.187	0.363	0.421	0.383	0.384	0.234	0.409	0.614	0.543
	0.402	0.300	0.758	0.567	0.553	0.641	0.743	0.900	0.936	1.389

Table 9
Probability of error for each class: positive, negative, average between positive and negative. Average results for the 10 digits when average imbalance ratio increases to 18/1.

	Conv	$k^U(z)$	$k^L(z)$	$k^T(z)$	$k^A(z)$
$P_{e\|+1}$	2.833	1.365	1.458	1.583	1.354
$P_{e\|-1}$	0.083	0.474	0.452	0.409	0.452
$(P_{e\|+1} + P_{e\|+1})/2$	1.458	0.919	0.955	0.996	0.903

Table 10
Probability of error for each class: positive, negative, average between positive and negative. Average results for the 10 digits when average imbalance ratio increases to 90/1.

	Conv	$k^U(z)$	$k^L(z)$	$k^T(z)$	$k^A(z)$
$P_{e\|+1}$	7.785	2.804	3.050	3.241	2.632
$P_{e\|-1}$	0.029	0.544	0.530	0.457	0.466
$(P_{e\|+1} + P_{e\|+1})/2$	3.907	1.674	1.790	1.849	1.549

6. Conclusions

A new principled procedure to train neural networks for solving binary classification problems has been proposed. The method is based on minimizing Bayes' risk estimates that are constructed by means of using Parzen windows to model the distributions of the LM outputs under each hypotheses. There are many alternatives to select these windows and their characteristics. Some of these alternatives are equivalent to classical training methods, such as the perceptron rule.

Experimental evidence permits to conclude that an appropriate design of windows can allow performance improvements, with a controllable risk of degradation with respect to conventional training algorithms. No difficulties appear for selecting window parameters, and very simple cross-validation methods are enough for it. Additionally, the hybrid character of the proposed approach, which combines discriminative and generative procedures, provides them superiority when dealing with imbalanced problems, without needing any re-balancing technique.

There are several research lines to extend this work, such as looking for rules to select an appropriate window, or to revise the activation functions of conventionally trained MLs from the same perspective we use here with Parzen windows. We are actively working along these directions.

Acknowledgments

This work was partly supported by Grant TEC-2015-67719-P "Macro-ADOBE" (Spain MINECO/EU FSE, FEDER), and network TIN 2015-70808-REDT, "DAMA" (MINECO) (M. Lázaro and A.R. Figueiras-Vidal), and by Prof. Monson Hayes' Banco de Santander-UC3M Chair of Excellence, 2015.

References

[1] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, Psychol. Rev. 65 (1958) 386–408.
[2] F. Rosenblatt, Principles of Neurodynamics, Washington, DC: Spartan, 1962.
[3] C.M. Bishop, Pattern Recognition and Machine Learning, New York, NY: Springer, 2006.
[4] P.J. Werbos, Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences, Harvard University, Cambrigde, MA, 1974 Ph.d. thesis.
[5] D.B. Parker, Learning Logic, Stanford University Office of Technology Licensing, 1982 Technical report. S81-64, File 1.
[6] Y. LeCun, A learning scheme for asymmetric threshold network, in: Proceed- ings of Cognitiva'85, 1985, pp. 599–604. Paris, France.
[7] D.E. Rumelhart, G.E. Hinton, R.J. Willians, Learning representations by back-
-propagating errors, Nature (Lond.) 323 (1986) 533–536.
[8] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning internal representations by error propagation, in: D.E. Rumelhart et al. (Ed.), Parallel Distributed Process- ing: Explorations in the Microstructure of Cognition, 1, MIT Press, Cambridge, MA, 1986, pp. 318–362. Foundations.
[9] L.I. Kuncheva, Combining Pattern Classifiers: Methods and Algorithms, Hobo- ken, NJ: Wiley, 2004.
[10] L. Rokach, Pattern Classification Using Ensemble Methods, Singapore: World Scientific, 2010.
[11] R.E. Schapire, Y. Freund, Boosting: Foundations and Algorithms, Cambridge, MA: MIT Press, 2012.
[12] B.E. Boser, I.M. Guyon, V.N. Vapnik, A training algorithm for optimal margin classifiers, in: D. Haussler (Ed.), 5th Annual ACM Workshop on COLT, ACM Press, Pittsburgh, PA, 1992, pp. 144–152. http://www.clopinet.com/isabelle/Papers/colt92.ps.
[13] V.N. Vapnik, Statistical Learning Theory, John Wiley & Sons, New York, NY, 1998.
[14] M.A. Aizerman, E.M. Braverman, L.I. Rozonoer, The probability problem of pat- tern recognition learning and the method of potential functions, Autom. Re- mote Control 25 (1965) 1175–1190.
[15] G. Kimeldorf, G. Wahba, Some results on Tchebycheffian spline functions, J. Math. Anal. Appl. 33 (1971) 82–95.
[16] Y. Bengio, Learning deep architectures for AI, Found. Trends Mach. Learn. 2 (2009) 1–127.
[17] J. Schmidhuber, Deep Learning in Neural Networks: An Overview, Technical report, University of Lugano, 2014 arXiv:1404.7828v4. IDSIA-03-14.
[18] L. Deng, D. Yu, Deep learning: methods and applications, Found. Trends Signal Process. 7 (2014) 197–387.
[19] T. Bayes, An essay towards solving a problem in the doctrine of chances, Philos. Trans. R. Soc. (Lond.) 53 (1763) 370–418.
[20] J.O. Berger, Statistical Decision Theory and Bayesian Analysis, Springer Series in Statistics, 2nd ed., Springer, 1985.
[21] K. Fukunaga, Introduction to Statistical Pattern Recognition (2nd ed.), New York, NY: Academic, 1990.
[22] L.M. Bregman, The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming, USSR Comput. Math. Math. Phys. 7 (1967) 200–217.

[23] J. Cid-Sueiro, J.I. Arribas, S.U.-M. noz, A.R. Figueiras-Vidal, Cost functions to estimate a posteriori probabilities in multiclass problems, IEEE Trans. Neural Netw. 10 (1999) 645–656.
[24] J. Cid-Sueiro, A.R. Figueiras-Vidal, On the structure of strict sense Bayesian cost functions and its applications, IEEE Trans. Neural Netw. 12 (2001) 445–455.
[25] A. Guerrero-Curieses, J. Cid-Sueiro, R. Alaiz-Rodríguez, A.R. Figueiras-Vidal , Lo- cal estimation of posterior class probabilities to minimize classification errors, IEEE Trans. Neural Netw. 15 (2004) 309–317.
[26] E. Parzen, On the estimation of a probability density function and the mode, Ann. Math. Stat. 33 (1962) 1065–1076.
[27] T.G. Dietterich, G. Bakiri, Solving multiclass learning problems via error-correcting output codes, J. Artif. Intell. Res. 2 (1995) 263–286.
[28] D.F. Specht, Probabilistic neural networks, Neural Netw. 3 (1990) 109–118.
[29] D.F. Specht, A general regression neural network, IEEE Trans. Neural Netw. 2 (1991) 568–576.
[30] I. Santamaría, M. Lázaro, C.J. Pantaleón, J.A. García, A. Tazón, A. Mediavilla, A nonlinear MESFET model for intermodulation analysis using a generalized radial basis function network, Neurocomputing 25 (1999) 1–18.
[31] G.P. Zhang, Neural networks for classification: a survey, IEEE Trans. Syst. Man Cybern. 30 (20 0 0) 451–462.
[32] J.T. Kwok, Moderating the output of support vector classifiers, IEEE Trans. Neu- ral Netw. 10 (1999) 1018–1031.
[33] B.D. Ripley, Neural networks and related methods for classification, J. R. Stat. Soc. Ser. B 56 (1994) 409–456.
[34] M. Lichman, UCI machine learning repository, University of California, Irvine, School of Information and Computer Sciences, 2013 https://archive.ics.uci.edu/ml/citation_policy.html.
[35] H. He, E.A. Garcia, Learning from imbalanced data, IEEE Trans. Knowl. Data Eng. 21 (2009) 1263–1284.
[36] M. Galar, A. Fernández, E. Barrenechea, F. Herrera, EUSBoost: enhancing en- sembles for highly imbalanced data-sets by evolutionary undersampling, Pat- tern Recognit. (2013) 3460–3471.
[37] A. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Her- rera, KEEL data-mining software tool: data set repository, integration of algo- rithms and experimental analysis framework, J. Mult.Valued Logic Soft Com- put. 17 (2011) 255–287.
[38] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, SMOTE: synthetic mi- nority over-sampling technique, J. Artif. Intell. Res. 16 (2002) 321–357.
[39] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. of the IEEE 86 (1998) 2278–2324.