Lazaro, M. & Figueiras-Vidal, A. R. (2021). A Bayes Risk Minimization Machine for Example-Dependent Cost Classification. *IEEE Transactions on Cybernetics, 51*(7), 3524–3534.

DOI: 10.1109/tcyb.2019.2913572

A Bayes risk minimization machine for example-dependent cost classification

Marcelino Lázaro, Aníbal R. Figueiras-Vidal, *Life Fellow, IEEE*

***Abstract*—A new method for example-dependent cost classification is proposed. The method constitues an extension of a recently introduced training algorithm for neural networks. The surrogate cost function is an estimate of the Bayesian risk, where the estimates of the conditional probabilities for each class are defined in terms of a one-dimensional Parzen window estimator of the output of (discriminative) neural networks. This probability density is modeled with the objective of allowing an easy minimization of a sampled version of the Bayes risk. The conditional probabilities included in the definition of the risk are not explicitly estimated, but the risk is minimized by a gradient-descent algorithm. The proposed method has been evaluated using linear classifiers and neural networks, with both shallow (a single hidden layer) and deep (multiple hidden layers) architectures. Experimental results show the potential and flexibility of the proposed method, which can handle example-dependent cost classification under imbalanced data situations that commonly appear in this kind of problems.**

I. Introduction

During decades, learning machines used for pattern classification [1] were designed to learn from labeled examples, using different algorithms, but with the general goal of obtaining designs with a good generalization capability, i.e. a good performance with patterns that were not seen during the training procedure. The performance was usually measured by the number of erroneous decisions, which is an appropriate figure of merit as long as it is possible to assume that the importance of the errors is the same for all patterns. However, there are many real problems where this assumption is not practical because different errors can have different costs. Problems with this characteristics are known as Cost-Sensitive (CS). There are two types of CS problems. In some cases, the cost naturally depends on the class of the pattern, for instance in medical diagnosis problems, and the name Class-Dependent Cost (CDC) is used. CS learning deals with these CDC problems, but the same formulation has also been applied to deal with imbalanced problems, i.e. problems where the number of examples of patterns is very different for each class [2]. In this kind of problems, to assign a higher cost to the errors of the minority classes can be a useful mechanism to compensate the effects of the underrepresentation of these classes in the training set [3], [4]. Another interesting application of CS learning with CDC is ordinal regression (also known as ordinal classification), where the classes exhibit a natural order, and therefore the cost of an erroneous classification depends on the difference between the order of the correct and wrong classes [5].

Marcelino Lázaro and Aníbal R. Figueiras-Vidal are with the Department of Signal Theory and Communications of University Carlos III of Madrid, Spain.

Example-Dependent Cost (EDC) problems are other class of CS problems. EDC classification problems are those in which the cost for attributing a class i item to class j is not only a function of i, j, as in CDC problems, but also of the observed item $\mathbf{x}$. They are frequent and relevant in many application areas, such as security and health, and particularly in business and finance. Fraud detection [6], [7], [8], [9], [10] and credit scoring [11], [12] are core activities in finance. Customer churn [13], [14] and direct marketing [15], [16], [17] are much relevant in business.

Obviously, EDC classification would not require special attention if the statistical information for the Bayesian formulation were available. In practice, keeping a statistical perspective is necessary. The initial research followed this line, such as in [18], which is one of the pioneer works on the subject. Some years later, machine learning approaches were considered. Training discriminative machines to solve EDC problems is far from being an easy task. Proportional re-sampling is proposed in [19], [15]. This approximate way of dealing with EDCs suffers the drawbacks of sampling techniques, which can modify the problem by reducing the influence of critical samples and/or emphasizing unimportant instances [20]. Decision trees have also been considered for EDC problems [21], [22], and perceptrons and piecewise linear classifiers were used in [23] with an hybrid learning algorithm that constructs separating hyperplanes for each pair of classes. In general, all these techniques suffer from the limitations of the constrained partition of the input space. After an analysis of an SVM-based formulation, the authors of [24] conjecture that, when the number of samples is high enough, the $\mathcal{L}_2$ version can be used to minimize the Bayes cost for EDCs. The problem is that, for finite training sets, SVMs merely establish a bound of the classification cost [25].

An interesting, partly principled approach was proposed by Bahnsen and his colleagues [8], [9], [12], [14], [26]. They introduce a two-step procedure: First, they estimate the posterior class probabilities using constant costs. Then, they use these estimates to solve the EDC classification by means of a Bayesian formulation. Unfortunately, they do not use in the first step learning schemes that provide consistent estimates of the posterior probabilities, whose characteristics will be discussed below (the only exception being the logistic regression when training minimizes appropriate surrogate costs). Consequently, the authors explored further steps to improve the first estimate. In any case, even these improved estimates of posterior probabilities are sensitive to imbalance situations.

In this paper, we propose an alternative method to solve binary EDC classification problems. It is based on our previous work [27], that introduced a novel method for CDC classification with application to imbalanced problems. Here, we extend the formulation to deal with EDC classification and discuss its advantages for this class of problems. Specifically, we propose a training algorithm for neural networks with sample-by-sample updating equations to minimize a cost function that is defined using the joint theoretical framework of Bayesian decision and Parzen windows [28] for probability density estimation.

The Parzen windows method is well known since the sixties, but its formulation still continues finding applications in different fields (as in [29], to cite a very recent example). In the field of classification machines, Parzen's formulation has been previously used in combination with the Bayesian theory, as in the probabilistic neural networks proposed by Specht [30] or in [31], where Parzen windows (or a weighted version of them) are used to estimate the multi-dimensional probability densities in the input space and this estimation is used to solve the classification problem by means of Bayesian decision rules.

Our approach is totally different: Parzen windows are not applied in the multi-dimensional input space but in the one-dimensional output of a leaning machine with an arbitrary architecture, the classifier does not require an explicit density estimation, neither the decision rule is based on such an estimation. The decision rule for a pattern is the result of comparing the output of the machine for this pattern with the "zero-threshold" that is commonly used in conventional learning machines for binary classification problems. Parzen windows are included in the cost function to be minimized during the training procedure, allowing a sample-by-sample updating algorithm, which depends on the window associated to a single sample. So, this approach gives freedom to train sample-by-sample or in mini-batches, according to the characteristics of the problem being solved and the available computational resources.

The rest of the paper is organized as follows. Section II presents the problem of EDC classification and proposes a surrogate cost function that is a direct estimation of the Bayes risk, and creates a training algorithm to optimize it. In Section III, some experiments serve to evaluate the performance of the method in both synthetic and real data sets. Extensive comparisons of performance against other methods in several selected synthetic and real problems support the effectiveness of the method that is proposed for EDC. Finally, Section IV summarizes the main conclusions and outlines some promising further research lines.

II. Bayesian neural network for example-dependent cost learning

This section presents a new surrogate cost function for example-dependent cost classification based on the Bayesian formulation.

A. General Bayesian formulation

In a binary classification problem, a D-dimensional input pattern

$$\mathbf{x} = [x_1, x_2, \cdots, x_D]^T \tag{1}$$

has to be assigned to one out of two possible classes, $C_i, \quad i \in \{0, 1\}$. Superindex ^{T} in (1) denotes the transpose operator.

In an EDC problem, some specific cost policy $c_{i,j}(\mathbf{x})$, $i, j \in \{0, 1\}$ is defined for each pattern, where $c_{i,j}(\mathbf{x})$ is the cost of assigning pattern $\mathbf{x}$ to class i when the true class is j. These EDCs are characteristic of each classification problem under analysis, and usually analytical forms of them are provided from expert knowledge on the corresponding field. However, in some problems the costs are obtained from observations and uncertainty can remain about the EDC for unseen samples: therefore, classification algorithms that implicitly learn the EDCs can provide advantage. This is the case of the procedure we propose in this paper, which trains a neural network to minimize a sampled version of the Bayes risk. The possibility of using powerful deep neural networks is one more advantage.

The optimal assignment rule in this kind of problems minimizes the decision cost, and this optimal rule is provided by the Bayesian theory. The Bayes risk function is defined for the decision on the class of pattern $\mathbf{x}$ as

$$\mathcal{R}(\mathbf{x}) = \sum_j \left(\sum_i c_{i,j}(\mathbf{x}) \Pr(D_i|C_j) \right) P_j, \tag{2}$$

where D_i, $i \in \{0, 1\}$ denotes the decision for class C_i, and P_j denotes the a priori probability of class C_j. When the posterior probabilities for each class, $\Pr(C_i|\mathbf{x})$, are known, the decision rule minimizing (2) for pattern $\mathbf{x}$ is to assign it to the class with minimum average decision cost

$$i^* = \arg\min_i \{\bar{c}(i|\mathbf{x})\}, \tag{3}$$

where

$$\bar{c}(i|\mathbf{x}) = \sum_j c_{i,j}(\mathbf{x}) \Pr(C_j|\mathbf{x}). \tag{4}$$

Taking into account the relationship of posterior probabilities $\Pr(C_i|\mathbf{x})$ and likelihoods $p(\mathbf{x}|C_i)$, this rule can be written as the well known Likelihood Ratio Test (LRT) [32]

$$\frac{p(\mathbf{x}|C_1)}{p(\mathbf{x}|C_0)} \underset{C_0}{\overset{C_1}{\gtrless}} \frac{c_{10}(\mathbf{x}) - c_{00}(\mathbf{x})}{c_{01}(\mathbf{x}) - c_{11}(\mathbf{x})} \frac{P_0}{P_1}. \tag{5}$$

B. Machine learning in the Bayesian framework

In many practical cases the required statistical knowledge for carrying out the classification, necessary to evaluate (4) or (5), is not explicitly available, and the information about the problem is enclosed in a set of labeled examples or training set, $\mathcal{T}$, consisting of L labeled examples

$$\mathcal{T} = \{(\mathbf{x}_\ell, y_\ell, c_{i,j}(\mathbf{x}_\ell)) \mid \ell \in \{1, \cdots, L\},\ i, j \in \{0, 1\}\} \tag{6}$$

that in an example-dependent cost problem must include the cost policy for each pattern, $c_{i,j}(\mathbf{x}_\ell)$, as well as labels $y_\ell \in \{0, 1\}$ indicating the class for each pattern $\mathbf{x}_\ell$. When the available information is constrained to this labeled set, one of

the usual approaches is to use machine learning methods to solve the classification problem.

In the framework of Bayesian theory, machine learning can be used for EDC classification in several different ways. One approach is to use a two-step methodology:

1) A machine is used to obtain estimates of the a posteriori probabilities $\Pr(C_i|\mathbf{x})$.
2) These estimates are used along with the decision costs $c_{i,j}(\mathbf{x})$ to implement the Bayesian decision rule, i.e., (3)-(4) or alternatively (5).

Although several machine learning methods have been used to provide the estimate of step 1), such as logistic regression or random forest [12], [8], theoretically a more interesting approach is to consider neural networks that make use of Bregman divergences [33] as surrogate costs, because networks trained to minimize this kind of costs provide consistent estimates of the a posteriori probabilites [34], [35].

Another approach is to introduce the Bayes risk into the cost function to be minimized during the training of the neural network, as proposed in [27], where the surrogate cost function is an estimate of the Bayesian risk with fixed cost per class. Here, we will follow this approach. Specifically, a new cost function is proposed, by averaging an estimate of Bayes risk (2), which is presented in the next sub-section.

C. Proposed surrogate cost function

A new surrogate cost function for EDC binary classification is presented to be used to train neural networks with a single output neuron and a threshold based decision rule. The output of the neural network for a pattern $\mathbf{x}$ is

$$z = f_{\mathbf{w}}(\mathbf{x}), \tag{7}$$

where $\mathbf{w}$ denotes the trainable parameters of the neural network and the transfer function $f_{\mathbf{w}}(\cdot)$ is given by the neural architecture. Any architecture can be considered, such as Multi-Layer Perceptrons (MLPs) or radial basis function networks, but also linear classifiers, $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T[1, \mathbf{x}^T]^T$, can be included. From the network output, the decision rule of the classifier is obtained by comparing z with a zero threshold, i.e.,

$$\hat{y} = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0 \end{cases}. \tag{8}$$

Note that, contrary to rules (3) or (5), the decision rule in this case does not require the knowledge of the costs $c_{i,j}(\mathbf{x})$, because it only depends on the input pattern $\mathbf{x}$[1]. This is an important advantage for those problems in which functional cost expressions are not available for unseen samples.

For this kind of networks, the proposed cost function is the average of an estimate of the Bayes risk (2). For the sake of simplicity, in the following the costs for correct decisions will be considered null, $c_{0,0}(\mathbf{x}) = c_{1,1}(\mathbf{x}) = 0$, because this is a very common situation and to include non-null values is straightforward. Moreover, we define $c_0(\mathbf{x}) = c_{1,0}(\mathbf{x})$ and $c_1(\mathbf{x}) = c_{0,1}(\mathbf{x})$. Using this notation, the proposed cost function is

$$J_{\mathcal{R}}(\mathbf{w}) = E\left[c_0(\mathbf{x})\hat{\Pr}(D_1|C_0)\hat{P}_0 + c_1(\mathbf{x})\hat{\Pr}(D_0|C_1)\hat{P}_1\right], \tag{9}$$

where $\hat{\cdot}$ denotes an estimate, and $E[\cdot]$ is the statistical expectation operator. When not known from side information, estimates for the a priori probabilities, $\hat{P}_i$, can be easily obtained by the proportion of samples per class in the training set. In a network with a one-dimensional output (7) and decision rule (8)

$$\Pr(D_1|C_0) = \int_0^{\infty} p(z|C_0)\, dz \tag{10}$$

and

$$\Pr(D_0|C_1) = \int_{-\infty}^{0} p(z|C_1)\, dz, \tag{11}$$

$p(z|C_i)$ being the conditional probability density of the output z under class C_i. Estimates $\hat{\Pr}(D_i|C_j)$ in (9) can be obtained by integrating estimates of $p(z|C_i)$ computed using the Parzen window estimator [28]

$$\hat{p}(z|C_i) = \frac{1}{L_i} \sum_{n \in S_i} k_i(z - z_n) \tag{12}$$

with

$$S_i = \{n \ : \ y_n = i\}, \tag{13}$$

L_i is the number of samples in set S_i, i.e., $L_i = |S_i|$, and $k_i(z)$ is the Parzen window used in the estimator for class C_i. Each window $k_i(z)$ has to be a valid probability density function, i.e., it has to be a non-negative function $k_i(z) \geq 0$ with unit area. When modeling with Parzen windows we combine the representation power of the machine with the Parzen classical approach to deal with probability densities. This means that better results are expected, because we can train the neural network towards an easy representation of the classes at the output level (unidimensional), instead of working directly with the (multidimensional) samples in the input space, as usual when Parzen windows are used for classification [31], [36]. The use of Parzen windows to model z allows to build a sample-by-sample training algorithm in a direct and easy manner, while other non-parametric models, such as k nearest neighbours, or more compact representations, such as Gaussian mixtures, would not allow it. Moreover, non-parametric density estimators do not make any assumption about the probabilistic structure of z. This is appropriate because z values are obtained by projecting input samples via training the neural network. Finally, since z is unidimensional, it is easy to select different profiles of windows in order to get good classification results. These are additional advantages of our proposed method.

Since $\hat{\Pr}(D_i|C_j)$ is obtained by integrating $\hat{p}(z|C_i)$, it is useful to define the integrals of the Parzen windows

$$K_i(z) = \int_{-\infty}^{z} k_i(a)\, da, \tag{14}$$

because with this definition

$$\int_0^{\infty} k_i(z - z_n)\, dz = 1 - K_i(-z_n) \tag{15}$$

[1]Obviously, the parameters $\mathbf{w}$ will be obtained during the training of the network taking into account the costs, but these costs are not required in the evaluation phase.

and

$$\int_{-\infty}^{0} k_i(z - z_n)\, dz = K_i(-z_n). \tag{16}$$

By replacing $p(z|C_i)$ in (10) and (11) by their Parzen window estimates given by (12) and taking into account (15) and (16), the proposed cost function becomes

$$J_{\mathcal{R}}(\mathbf{w}) = \frac{\hat{P}_0}{L_0} \sum_{n \in S_0} c_0(\mathbf{x}_n)\ (1 - K_0(-z_n)) + \frac{\hat{P}_1}{L_1} \sum_{n \in S_1} c_1(\mathbf{x})\ K_1(-z_n). \tag{17}$$

It is common to estimate

$$\hat{P}_0 = \frac{L_0}{L_0 + L_1} = \frac{L_0}{L}, \quad \hat{P}_1 = \frac{L_1}{L_0 + L_1} = \frac{L_1}{L}. \tag{18}$$

In this case, the normalized costs are defined as

$$\bar{c}_i(\mathbf{x}) = \frac{c_i(\mathbf{x})}{L}, \tag{19}$$

and the proposed cost function can be written as

$$J_{\mathcal{R}}(\mathbf{w}) = \sum_{n \in S_0} \bar{c}_0(\mathbf{x}_n)\ (1 - K_0(-z_n)) + \sum_{n \in S_1} \bar{c}_1(\mathbf{x})\ K_1(-z_n). \tag{20}$$

The problem of consistency of convex risks minimizations is classical in the statistical literature [37], [38]. Here, we are applying a consistent probability density estimator, the Parzen windows model, which shows a reasonable performance. But our objective is not exactly to get good EDC Bayes risk estimates, but to design classifiers that offer high performance when applied to the corresponding classification problem. Further studies could lead to a better understanding of the statistical characteristics of the method, but, in any case, we remark that we deal with an easy situation: The estimator is applied to a one-dimensional variable, the output z. Moreover, conditional distributions $\hat{p}(z|C_i)$ will not be explicitly estimated, but the proposed cost in whose definition these estimates are included will be minimized using an iterative procedure, as it is shown below. This procedure leads to an updating expression that depends on the one-dimensional Parzen windows, $k_i(z)$, included in the theoretical definition of the estimates, (12). This approach is notably different from using Parzen windows to obtain estimates of the conditional distributions of the input, $p(\mathbf{x}|C_i)$, or the joint input-output distributions, $p(\mathbf{x}, y)$, such as in [30], [39].

D. Training algorithm and computational load

The proposed cost function can be minimized by using any appropriate optimization procedure. Here we present the procedure to minimize the cost function by means of an iterative gradient descent algorithm, but its extension to gradient plus momentum, Adam optimizer [40], or similar procedures is straightforward. The updating equation for gradient descent is

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \mu\ \frac{\partial J_{\mathcal{R}}(\mathbf{w})}{\partial \mathbf{w}}\bigg|_{\mathbf{w}=\mathbf{w}(n)}, \tag{21}$$

where μ is the step-size parameter. The gradient expresion can be computed sample-by-sample, batch (using the whole training set) or mini-batch (using a sub-set of the training set in each step). The sample-by-sample expression of the gradient is

$$\frac{\partial J_{\mathcal{R}}(\mathbf{w})}{\partial \mathbf{w}}\bigg|_{\mathbf{w}=\mathbf{w}(n)} = \frac{\partial J_{\mathcal{R}}(\mathbf{w})}{\partial z_n}\ \frac{\partial z_n}{\partial \mathbf{w}}\bigg|_{\mathbf{w}=\mathbf{w}(n)}. \tag{22}$$

The first factor in the right side is[2]

$$\frac{\partial J_{\mathcal{R}}(\mathbf{w})}{\partial z_n} = \begin{cases} -\bar{c}_1(\mathbf{x}) k_1(-z_n) & \text{if } y_n = 1 \\ +\bar{c}_0(\mathbf{x}) k_0(-z_n) & \text{if } y_n = 0 \end{cases}. \tag{23}$$

The second term in the right side of (22) is independent of the cost function and it only depends on the network architecture.

To briefly analyze the computational burden of the training algorithm, we compare the updating expressions with those associated to the well-known minimum mean squared error (MMSE) cost function. For this cost function the second term in the right side of (22) is the same, because it does not depend on the cost. The first term is

$$\frac{\partial J_{MMSE}(\mathbf{w})}{\partial z_n} = -2(y_n - z_n) \tag{24}$$

By comparing (23) with (24), it can be concluded that the computational complexity using the proposed cost function is of the same order of magnitude than that required for the MMSE cost function.[3]

III. Experiments

The proposed method will be evaluated by means of some experiments.

A. Benchmark methods

Several methods designed for example-dependent cost classification will be used as benchmarks. In particular, a linear method: linear regression (LR); a generalized linear model: Cost-Sensitive Logistic Regression (CSLR) [12]; and several non-linear methods: random forest (RF), Cost-Sensitive Decision Tree (CSDT) [26], and Bayes Minimum Risk (BMR) method [8] with several different approaches to estimate the posterior probabilities: LR, RF, CSLR, and CSDT. The methods will be denoted BMR(LR), BMR(RF), BMR(CSLR) and BMR(CSDT), respectively. To improve the performance, these methods are sometimes combined with re-sampling techniques aiming at producing data distributions which are proportional to the costs. Therefore, the training phase of these methods will be performed from 3 different data sets: the training set, and two additional sets obtained using the cost-proportional over-sampling (OS) [19] and the cost-proportional rejection sampling (RS) [15] techniques. For LR and RF, the Python implementation provided in *Scikit-learn*[4] software [41] has

[2]This is the expression for (20). In the more general case (17), $\bar{c}_i(\mathbf{x})$ is replaced by $c_i(\mathbf{x})\hat{P}_i/L_i$.

[3]In a practical implementation, the difference is mainly dependent on the cost for evaluating the Parzen windows $k_i(\cdot)$, which basically depends on the implementation platform.

[4]http://scikit-learn.org

been employed. For CSLR, CSDT and BMR, the Python implementation of the *Costcla* module [42] has been used. The parameters for each method are those recommended in [12].

B. Proposed method

Complementary kernels have been considered, $k_0(z) = -k_1(z)$. We have added a normalized Gaussian kernel, with variance 0.15 (thus having 99% of the energy in $[-1, 1]$) to the four kernels proposed in [27], uniform, linear, triangle, and absolute value. In this way, the basic support of all kernels is constrained to this interval, which fits the range of the tanh activation function. The five kernels are plotted in Fig. 1. Taking into account the updating equation (23), where the gradient is proportional to $k_i(-z)$, the choice of a given kernel is related with how different patterns contribute to the gradient as a function of the distance between their output and the decision threshold at $z = 0$. Gaussian or triangle kernels enhance the contribution of samples that are close to the threshold. Absolute value enhances the contribution of samples that are far away from the threshold. Uniform kernel weights equally all patterns at a distance ≤ 1. The linear kernel enhances the contribution of samples in the wrong side of the threshold, because the contribution increases linearly with the distance to the desired extreme value ($+1$ for samples of Class 1, or -1 for samples of Class 0).

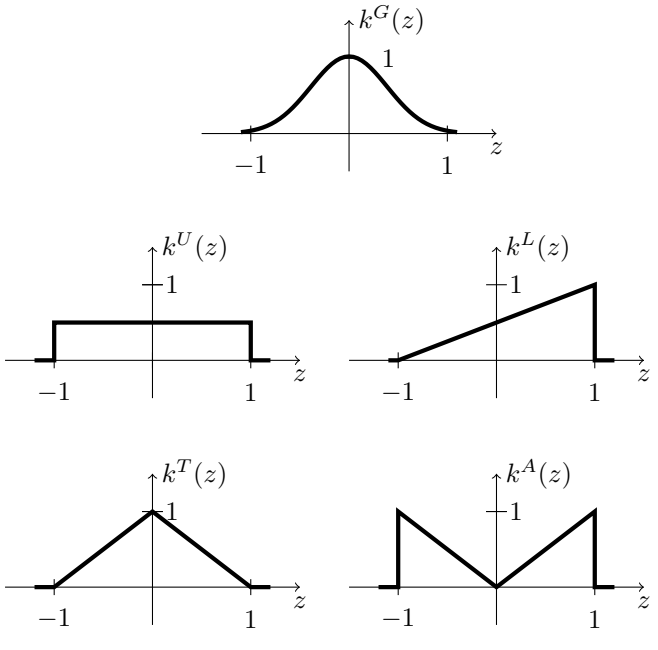

Fig. 1. Kernels used in the experiments, with $k_1(z) = k^X(z)$ and complementary kernels ($k_0(z) = k_1(-z)$). $X \in \{G, U, L, T, A\}$, with G: Gaussian, U: uniform, L: linear, T: triangle, A: absolute value.

The performance of the proposed cost function will be evaluated for two architectures: a linear classifier and a neural network classifier. In this case, MLPs with H_L hidden layers and N neurons at each layer are employed. A tanh activation function is used for the single neuron of the output layer, and tanh or rectified linear units (ReLU) activations are used for neurons of the hidden layers. For the linear classifier, Lin(X) will denote that kernel $k_1(z) = k^X(z)$ is used, with $X \in \{G, U, L, T, A\}$. For the MLP, MLP(H_L,N,X) will denote H_L hidden layers with N neurons and that kernel $k_1(z) = k^X(z)$ is employed. For N, several number of neurons are tried, and the best model size is obtained by cross-validation.

The proposed cost function is optimized using a mini-batch gradient descent algorithm. An adaptive step-size μ has been used in gradient updates. After each epoch the cost $J_{\mathcal{R}}(\mathbf{w})$ is computed and compared with the cost at the end of the previous epoch. Then

- If cost decreased: step size is increased, $\mu = c_I\ \mu$
- If cost increased: step size is decreased, $\mu = \mu / c_D$, and weights are re-computed with the new step size.

Parameters $c_I = 1.05$ and $c_D = 2$ have been used in all experiments, with initial step size $\mu = 10^{-3}$.

C. Figure of merit

The cost savings will be used as figure of merit to evaluate the proposed method performance. If $Cost(\{\hat{y}_\ell\})$ denotes the cost of a given decision rule $\{\hat{y}_\ell : \ell \in \{1, 2, \cdots, L\}\}$

$$Cost(\{\hat{y}_\ell\}) = \sum_{\ell \in S_0} \hat{y}_\ell\ c_0(\mathbf{x}_\ell) + \sum_{\ell \in S_1} (1 - \hat{y}_\ell)\ c_1(\mathbf{x}_\ell), \tag{25}$$

the savings for that decision are

$$Sav(\{\hat{y}_\ell\}) = \frac{Cost_T - Cost(\{\hat{y}_\ell\})}{Cost_T}, \tag{26}$$

where

$$Cost_T = \min\{Cost_{T0}, Cost_{T1}\} \tag{27}$$

and $Cost_{Ti}$ is the cost of the trivial rule $\hat{y}_\ell = i\ \forall \ell$. It can be seen that savings are defined as the percentage of the costs that are saved by using a given decision rule with respect to the costs of not using an intelligent rule, i.e., with respect to the best trivial decision (the best of the "all patterns of class 0" and "all patterns of class 1" rules). This is an appropriate metric for EDC problems that has been used in several works (see, for instance, [9], [26]) because it has the advantage, with respect to the raw cost measure (25), of being normalized.

D. Datasets

We have considered four datasets, two synthetic and two with real data. The two synthetic datasets are two-dimensional problems where data of both classes have Gaussian distributions, which allows to compute the optimal Bayesian solution. The first synthetic dataset is as follows:

- Class 0: A Gaussian distribution with mean vector and covariance matrix

$$\mu_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \Sigma_0 = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}$$

- Class 1: A Gaussian distribution with mean vector and covariance matrix

$$\mu_1 = \begin{bmatrix} 6 \\ 0 \end{bmatrix}, \quad \Sigma_1 = \begin{bmatrix} 4 & 0 \\ 0 & 8 \end{bmatrix}$$

The decision cost is constant, 1, for patterns of class 0, and is the exponential of the first coordinate for patterns of class 1

$$c_{1,0}(\mathbf{x}) = 1, \quad c_{0,1}(\mathbf{x}) = \exp(x_1)$$

Some examples of patterns of both classes and the Bayesian solution to this problem if $P_0 = P_1$ are shown in Fig. 2. The Bayes solution for this dataset provides savings of 75.95%.

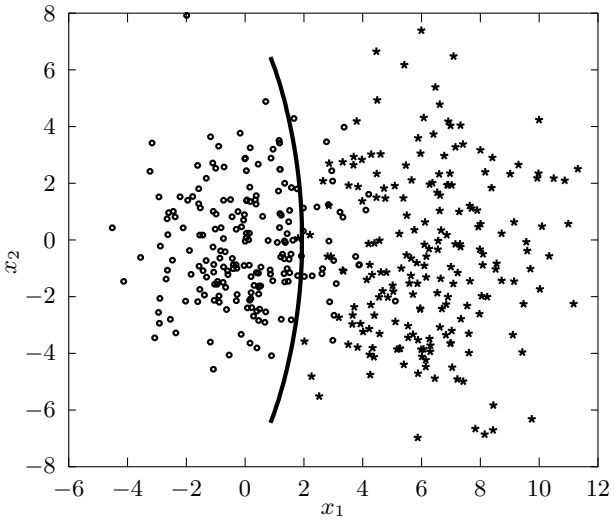

Fig. 2. Some examples of patterns for the first synthetic experiment and the Bayesian decision boundary (circles for class 0, stars for class 1).

The second synthetic dataset has the following distributions:

- Class 0: A mixture of two Gaussians with the following mean vectors and covariance matrices:

$$\mu_{0a} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}, \quad \mu_{0b} = \begin{bmatrix} 1 \\ -4 \end{bmatrix}, \quad \Sigma_{0a} = \Sigma_{0b} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

- Class 1: A Gaussian distribution with mean vector and covariance matrix

$$\mu_1 = \begin{bmatrix} 4 \\ 0 \end{bmatrix}, \quad \Sigma_1 = \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}$$

The decision costs are the same that in the previous dataset. Some examples of patterns of both classes and the Bayesian solution for equiprobable classes are shown in Fig. 3. It can be seen that now the boundary is clearly non-linear, unlike for the first dataset, where the decision boundary is almost linear. The Bayes solution for this dataset provides savings of 66.3%.

The third dataset is a real dataset, the 2009 Pacific-Asia Knowledge Discovery and Data-Mining (PAKDD) conference competition, a credit scoring problem, where the objective is to identify credit card applicants that are likely to be in default. It contains 38,578 patterns with 32 features for each pattern. It is an imbalanced dataset, with a 19.89% of patterns of Class 1. Costs for false negatives are proportional to the client's credit line, and costs for false positives are the addition of the

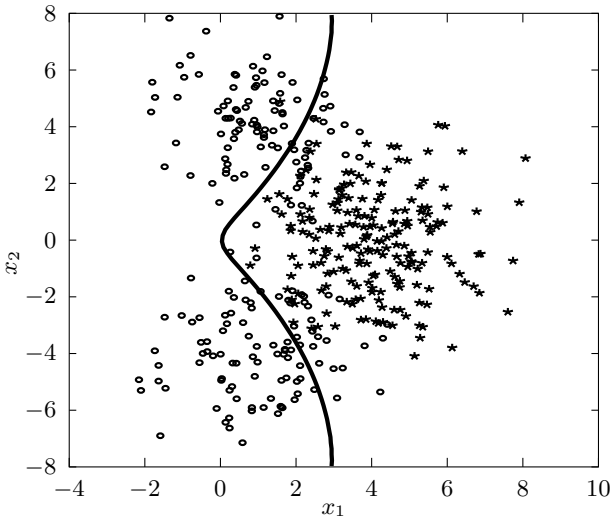

Fig. 3. Some examples of patterns the second synthetic experiment and the Bayesian decision boundary (circles for class 0, stars for class 1).

expected loss and the average loss of profit by rejecting a good customer (specific details are given in [12]). The dataset was downloaded from the *Costcla* Python module repository[5].

Finally, the home equity (HMEQ) loans data set [43][6] reports characteristics and delinquency information for 5,960 home equity loans, where the obligor uses the equity of a home as the underlying collateral. Each pattern has 12 characteristics, and the percentage of applicants that defaulted on loan is 19.95 %. For this dataset we have considered that cost for false negatives is proportional to the loan, in particular we fixed 75% of the loan amount. The cost for false positives is given by the loss of profit, which has also been considered proportional to the loan amount, in this case a 15% of this amount.

In the experiments with synthetic datasets, the sizes for train, validation and test sets will be specified at every case. For the two real datasets, data is randomly split in train, validation and test subsets in each realization according to the following percentages: 50%, 25% and 25%, respectively.

E. Experiments with the synthetic datasets

The first experiment is performed with the first synthetic dataset and balanced classes ($P_0 = P_1$). In this case, 4,000 samples are used for designing the solutions, with 2/3 and 1/3 proportions for train and validation sets, respectively. For the test set, 100,000 independent samples are generated. The benchmark methods are designed from 3 different sets: the train set, and two re-sampled sets, using both over-sampling (OS) and rejection-sampling (RS). The proposed method is not trained working with the re-sampled sets because the Bayesian formulation allows to deal directly with the costs and the a priori probabilities for each class. The mini-batch size is 1000 samples, and the proportion of samples for each class is kept in every mini-batch.

[5]https://pypi.org/project/costcla/

[6]Available at http://www.creditriskanalytics.net

Table I compares the results obtained with the benchmark methods and with the proposed method using both a linear classifier and an MLP with a single hidden layer with tanh activations. 2, 4, 10, 25 and 50 neurons were tried for this hidden layer. The table shows the average savings (in %) obtained in the test set in 100 independent realizations. To make easier the comparison, the best result[7] for the linear (and generalized linear) benchmark methods, non-linear benchmark methods, the proposed method with linear models and the proposed method with MLPs, are highlighted (boldface). The best size for the MLPs among the 5 sizes under test has been selected by cross-validation using the validation test. Training has also been stopped by cross-validation.

The linear model obtains better results than LR or CSLR. LR obtains better results when combined with re-sampling techniques, as expected (re-sampling produces a distribution proportional to the costs), especially with the RS method in this case, although the achieved performance is far from the optimal solution. It can be seen that several non-linear methods are able to obtain a performance close to the optimal Bayesian rule: CSDT, and all BMR methods working with the train set but BMR(RF), which has a lower performance (RF, by its own nature, does not work well in such a low-dimension space). The proposed method with MLPs obtains results that are very close to the optimal. The proposed cost function with a linear classifier obtains slightly lower results, as expected, taking into account that the optimal boundary is slightly non-linear. The best kernel for both linear and MLP models for this problem is the linear kernel. The sensitivity to the choice of the kernel is slightly lower for MLPs than for the linear classifiers.

Method	Train Set	Re-Sampling(OS)	Re-Sampling(RS)
LR	24.85 ± 6.44	25.23 ± 6.38	**51.90 ± 5.94**
CSLR	31.14 ± 0.35	31.22 ± 0.57	31.18 ± 0.70
RF	-143.93 ± 186.07	-115.58 ± 62.57	-12.29 ± 151.32
CSDT	75.07 ± 0.59	75.02 ± 0.73	-83.25 ± 399.15
BMR(LR)	**75.18 ± 0.45**	75.18 ± 0.46	72.54 ± 3.10
BMR(RF)	21.06 ± 14.45	21.17 ± 13.96	61.67 ± 49.00
BMR(CSLR)	75.04 ± 0.45	-349.94 ± 261.11	-180.28 ± 649.66
BMR(CSDT)	75.07 ± 0.58	75.02 ± 0.73	-38.85 ± 408.70
Lin(G)	73.18 ± 6.07		
Lin(U)	73.42 ± 5.54		
Lin(L)	**74.20 ± 4.50**		
Lin(T)	73.45 ± 5.37		
Lin(A)	73.11 ± 5.58		
MLP(1,10,G)	74.99 ± 0.74		
MLP(1,50,U)	75.01 ± 0.64		
MLP(1,10,L)	**75.89 ± 0.48**		
MLP(1,25,T)	75.13 ± 0.57		
MLP(1,50,A)	75.12 ± 0.66		

TABLE I
COST SAVINGS IN THE TEST SET (AVERAGE ± STANDARD DEVIATION, IN %) FOR THE FIRST DATASET WITH BALANCED CLASSES AND 4000 EXAMPLES FOR TRAINING AND VALIDATION (IN A 2/3 - 1/3 PROPORTION). AVERAGE RESULTS FOR 100 INDEPENDENT REALIZATIONS.

The next experiment is performed with the second synthetic dataset and balanced classes ($P_0 = P_1$) with 4,000 samples with 2/3 and 1/3 proportions for train and validation sets, respectively, and 100,000 independent samples for the test set. The mini-batch size is again 1000 samples. Table II compares the results obtained with the different methods. It can be seen that now, in this problem with a clearly non-linear solution, the proposed linear classifier cannot obtain an average performance close to the optimal value. Anyway, it obtains better results than LR and CSLR. Again, LR benefits clearly from re-sampling.

The non-linear benchmark methods, although in general obtain better results than their linear counterparts, are far from the optimal solution. In fact, their performance is close to the performance of the proposed linear method. However the proposed method with MLPs obtains much better results, very close to the optimal Bayesian solution, in this nolinear dataset. The best kernel in both cases, linear classifiers and MLPs, is the linear kernel, although in the case of the MLPs, as in the previous example, the sensitivity to the choice of the kernel is notably lower than for the linear classifiers.

Method	Train Set	Re-Sampling(OS)	Re-Sampling(RS)
LR	-11.84 ± 5.77	-3.01 ± 4.95	**32.37 ± 5.33**
CSLR	13.63 ± 5.45	15.32 ± 6.81	15.76 ± 7.10
RF	-148.23 ± 56.28	-126.20 ± 51.92	21.14 ± 31.50
CSDT	40.90 ± 17.48	49.54 ± 12.58	5.68 ± 27.80
BMR(LR)	52.65 ± 0.51	**52.66 ± 0.51**	49.38 ± 3.61
BMR(RF)	-10.18 ± 23.19	-9.20 ± 27.32	49.81 ± 9.00
BMR(CSLR)	49.51 ± 2.80	-199.30 ± 158.03	-235.19 ± 260.86
BMR(CSDT)	44.18 ± 15.05	49.55 ± 12.56	25.23 ± 24.07
Lin(G)	51.72 ± 3.33		
Lin(U)	48.51 ± 8.40		
Lin(L)	**52.59 ± 0.54**		
Lin(T)	51.78 ± 3.02		
Lin(A)	46.32 ± 9.37		
MLP(1,25,G)	65.03 ± 3.38		
MLP(1,25,U)	65.59 ± 0.70		
MLP(1,25,L)	**65.74 ± 0.70**		
MLP(1,25,T)	65.43 ± 0.84		
MLP(1,25,A)	65.14 ± 3.27		

TABLE II
COST SAVINGS IN THE TEST SET (AVERAGE ± STANDARD DEVIATION, IN %) FOR THE SECOND DATASET WITH BALANCED CLASSES AND 4000 EXAMPLES FOR TRAINING AND VALIDATION (IN A 2/3 - 1/3 PROPORTION). AVERAGE RESULTS FOR 100 INDEPENDENT REALIZATIONS.

F. Experiments with the PAKDD dataset

This section shows the results obtained with the PAKDD dataset, a real problem for credit scoring with imbalanced classes. Table III presents the savings obtained with the different methods. For the proposed method, the mini-batch size is 1000 samples, with a constant ratio between the number of samples of each class in each mini-batch (maintaining the original dataset imbalance ratio). The MLP has a single hidden layer with tanh activations, and 2, 4, 10, 25 or 50 neurons were validated. In this problem, the linear model again performs better than LR and CSLR (and again, LR works better with rejection sampling). The proposed method with MLPs behaves slightly better than the best non-linear benchmark method, which is the BMR using LR to estimate the posterior probabilities. This fact along with the relatively good performance of the linear classifiers (they provide results

[7]Strictly speaking, the apparently best result: That with the highest average (and the lower variance, if there are ties). Obviously, there are several statistically equivalent results.

close to the best ones, only 1% lower) suggest that the PAKDD problem is not highly nonlinear.

With respect to the kernels, for the linear classifiers now the best kernel is the absolute value kernel, very close to the Gaussian kernel, but notably better than the linear kernel for this problem. In the case of the MLPs, the best kernel is again the linear kernel, although the sensitivity to the kernel choice is relatively low.

Method	Train Set	Re-Sampling(OS)	Re-Sampling(RS)
LR	0.13 ± 0.12	24.02 ± 1.19	**25.72 ± 1.46**
CSLR	12.58 ± 1.43	8.17 ± 3.20	6.48 ± 3.41
RF	2.38 ± 0.96	20.12 ± 1.09	24.65 ± 1.59
CSDT	27.34 ± 0.96	16.68 ± 3.14	13.65 ± 3.35
BMR(LR)	**30.17 ± 1.29**	29.99 ± 1.03	29.93 ± 1.13
BMR(RF)	11.55 ± 2.83	18.61 ± 0.63	28.08 ± 1.08
BMR(CSLR)	26.80 ± 1.04	26.74 ± 1.03	26.91 ± 1.00
BMR(CSDT)	27.35 ± 0.96	26.53 ± 1.28	27.05 ± 1.03
Lin(G)	29.05 ± 1.09		
Lin(U)	28.94 ± 1.13		
Lin(L)	26.75 ± 1.47		
Lin(T)	28.79 ± 1.30		
Lin(A)	**29.10 ± 1.13**		
MLP(4,G)	29.29 ± 1.14		
MLP(50,U)	29.80 ± 1.14		
MLP(2,L)	**30.23 ± 1.51**		
MLP(25,T)	29.21 ± 1.25		
MLP(50,A)	29.47 ± 1.30		

TABLE III
COST SAVINGS IN THE TEST SET (AVERAGE ± STANDARD DEVIATION, IN %) FOR THE PAKDD DATASET. AVERAGE RESULTS FOR 100 INDEPENDENT REALIZATIONS.

The choice of the kernel has an interesting effect in the size of the model, given by the number of neurons in the hidden layer. Fig. 4 shows the evolution of the savings as a function of the number of neurons in the hidden layer for the five kernels under test. Different trends are seen for each kernel. In this case, the linear kernel requires just 2 neurons to obtain the best performance, and the performance degrades for a higher size, which is a typical example of over-fitting. With the other kernels, the sensitivity to over-fitting is reduced. In the more extreme cases, with the uniform or the absolute value kernels, performance increases up to the size of 50 neurons, the highest under evaluation. The linear kernel enhances the influence of the patterns whose output is far away from their target value, and it reduces the influence of patterns with outputs close to the target values. This effect makes this kernel sensitive to outliers and to over-fitting to these outliers. Kernels enhancing also the effect of samples with outputs close to the target values can reduce the sensitivity to over-fitting.

We have also tested the benchmark and the proposed methods with the other two data sets provided with the *Costcla* package, the 2011 Kaggle competition "*Give Me Some Credit*"[8] and the "*Bank Marketing*" [44] datasets, as well as with IBM's "*Telco Customer Churn*"[9] dataset, where costs were defined to be proportional to the yearly bill of the customer. In all these cases, the behavior was similar to

[8]https://www.kaggle.com/c/GiveMeSomeCredit

[9]Available at https://www.ibm.com/communities/analytics/watson-analytics-blog/predictive-insights-in-the-telco-customer-churn-data-set/

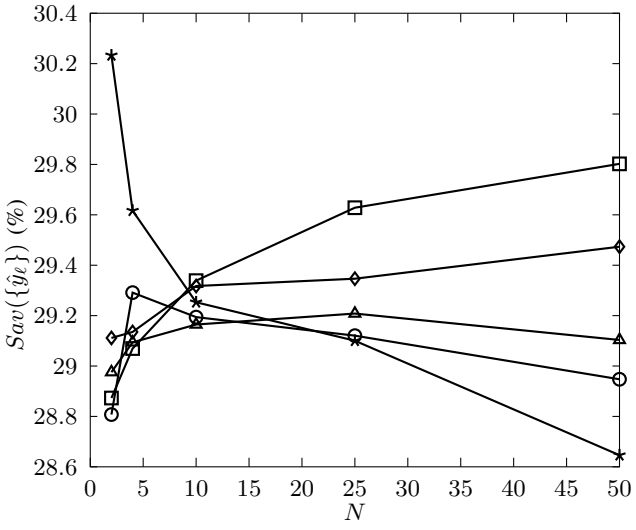

Fig. 4. Evolution of average savings in the PAKDD dataset as a function of the number of neurons in the hidden layer for each kernel function: Gaussian (circle), Uniform (square), Linear (star), Triangle (triangle), Absolute value (diamond).

the obtained in the PAKDD dataset[10]: the proposed cost using linear classifiers was very close to the best result of benchmark methods, which was

- BMR(CDST) for "*Give Me Some Credit*".
- BMR(LR) with OS re-samplig for "*Bank Marketing*" and "*Telco Customer Churn*".

The linear kernel provided the best results for linear classifiers in "*Bank Marketing*" and "*Telco Customer Churn*", and the uniform kernel for "*Give Me Some Credit*". The proposed cost function with a single hidden layer MLP obtained slightly better results than those methods, with linear kernel providing the best results in all datasets. As in PAKDD, these results point to a relatively low non-linearity in all these three problems.

G. Experiments with the HMEQ dataset

The final dataset under test was the HMEQ dataset. Table IV presents the savings obtained with all methods. For the proposed method, the mini-batch size is 1000 samples, with a constant ratio between the number of samples of each class in each mini-batch (maintaining the original dataset imbalance ratio). MLPs with a single hidden layer and 2, 4, 10, 25, 50, 100 and 150 neurons were validated. In this dataset, unlike in the previous ones, ReLU activations in the hidden layer provided better results than tanh activations in the case of a shallow network with a single hidden layer. As in all the previous datasets, the linear model obtains better results than LR and CSLR, with LR with rejection sampling being the best linear benchmark method. In this problem the difference

[10]For this reason detailed results have not been included in the manuscript, but they are available as supplementary material at http:ieeexplore.ieee.org. Summary of results for *Best Benchmark* / *Linear* / *MLP*: 47.61 ± 1.42 / 47.51 ± 1.17 / 48.35 ± 1.08 for "*Give Me Some Credit*"; 49.48 ± 0.93 / 49.13±1.39 / 49.51±1.28 for "*Bank Marketing*"; 38.31±2.97 / 38.79±2.64 / 38.40 ± 3.35 for "*Telco Customer Churn*".

between the linear classifier and the MLP with a single hidden layer can be clearly perceived, which suggest that the solution to this problem is clearly non-linear. The MLP provides better results than all the non-linear benchmark methods except BMR(RF), which provides the best results.

Method	Train Set	Re-Sampling(OS)	Re-Sampling(RS)
LR	25.69 ± 9.45	48.12 ± 4.96	**50.82 ± 4.69**
CSLR	4.48 ± 5.36	-0.51 ± 1.79	-0.10 ± 0.44
RF	53.68 ± 5.19	56.80 ± 5.62	63.32 ± 4.04
CSDT	64.99 ± 3.06	1.78 ± 2.36	1.65 ± 1.89
BMR(LR)	-0.51 ± 1.79	-0.51 ± 1.79	-0.15 ± 0.65
BMR(RF)	**72.67 ± 3.89**	52.06 ± 12.59	5.56 ± 9.56
BMR(CSLR)	-0.51 ± 1.79	-0.51 ± 1.79	-0.09 ± 0.43
BMR(CSDT)	-0.27 ± 1.94	1.78 ± 2.30	1.31 ± 1.98
Lin(G)	53.02 ± 3.96		
Lin(U)	**53.69 ± 3.88**		
Lin(L)	53.21 ± 3.95		
Lin(T)	53.29 ± 3.90		
Lin(A)	52.85 ± 4.08		
MLP(1,50,G)	58.58 ± 5.50		
MLP(1,150,U)	59.88 ± 5.94		
MLP(1,150,L)	**69.09 ± 3.75**		
MLP(1,50,T)	58.05 ± 5.65		
MLP(1,100,A)	60.45 ± 5.49		

TABLE IV
COST SAVINGS IN THE TEST SET (AVERAGE ± STANDARD DEVIATION, IN %) FOR THE HMEQ DATASET. AVERAGE RESULTS FOR 100 INDEPENDENT REALIZATIONS.

Given the nonlinearity of the problem, deep networks have been considered, in particular, MLPs with several hidden layers. It is interesting to see that the benchmark method providing the best results in this dataset, and the only one with a performance close to the proposed method, is BMR with random forest providing the estimates of the a posteriori probabilities for each class. Given that random forest is not guaranteed to provide consistent estimates for these probabilities, as seen in the previous experiments (BMR(RF) shows a poor performance in the synthetic datasets and in the PAKDD dataset), the good behavior of this method in the HMEQ dataset lead us to think that ensembles can provide a good performance in HMEQ. For this reason, we also evaluated the performance of ensembles made of deep neural classifiers trained with the proposed cost function. Two aggregation rules have been evaluated: comparison of the addition of the outputs of all neural classifiers with a threshold at zero, denoted as TA (thresholded addition), and majority voting rule, based on the majority of the decisions and denoted as MVR.

Table V shows the average savings obtained with ensembles of 5 MLPs with from one to seven hidden layers, and 150 neurons with ReLU activations at each hidden layer, as well as the average savings obtained with a single MLP. The linear kernel is used in all individual classifiers. To avoid overfitting, dropout [45] is used during training, with dropout probability of 0.5 for MLPs with one[11] and two hidden layers, and 0.75 for MLPs with more than two hidden layers.

Considering individual classifiers, i.e., using a single MLP, the best performance for this dataset is provided by the network with $H_L = 4$ hidden layers, although with all sizes the performance is now comparable with the best benchmark method. The performance of the MLP with a single hidden layer is remarkable: it notably increases when dropout is used to avoid overfitting (as compared with results in Table IV, where dropout was not used). As expected, the use of ensembles of 5 MLPs slightly improves the performance for all network sizes, with the TA rule having a slightly better performance than MVR in all cases.

[11]When comparing the results of MLP(1,150,L) with the results of Table IV, it is necessary to take into account that the results in this table were obtained for MLPs trained without dropout.

	Individual	Ensemble (TA)	Ensemble (MVR)
MLP(1,150,L)	72.59 ± 3.23	73.57 ± 3.35	73.42 ± 3.23
MLP(2,150,L)	72.66 ± 3.16	74.10 ± 3.11	73.63 ± 3.19
MLP(3,150,L)	73.19 ± 2.87	74.15 ± 2.67	73.97 ± 2.70
MLP(4,150,L)	**73.49 ± 2.83**	**74.57 ± 2.94**	74.32 ± 3.02
MLP(5,150,L)	73.16 ± 2.80	74.07 ± 3.01	73.93 ± 3.06
MLP(6,150,L)	73.21 ± 2.79	74.05 ± 2.68	73.87 ± 2.83
MLP(7,150,L)	73.20 ± 2.98	74.04 ± 3.05	73.91 ± 3.12

TABLE V
COST SAVINGS IN THE TEST SET (AVERAGE ± STANDARD DEVIATION, IN %) FOR THE HEMQ DATASET USING DEEP NETWORKS. AVERAGE RESULTS FOR 100 INDEPENDENT REALIZATIONS.

IV. CONCLUSIONS

A new method for example dependent cost-sensitive machine classification has been presented. The main contribution is a new surrogate cost function and the corresponding training algorithm to optimize it. The cost is an estimate of the Bayesian risk for example-dependent decision costs. The proposed method has several advantages when compared with current state of the art methods:

- In can be used with different architectures: from linear models to machine learning architectures such as MLPs, in both shallow or deep networks, and does not exclude the use of ensembles, thus having clear possibilities of getting good results.
- The Bayesian formulation of the problem allows to deal with the example-dependent costs in both balanced and imbalanced situations, because the a priori probabilities of the classes as well as the example-dependent costs are included in the proposed cost function. This feature makes unnecessary to re-sample the available data set to compensate imbalance or to obtain distributions that are proportional to the costs.
- Costs for each pattern must be known in the training phase but, unlike in most of the methods used for EDC problems, they are not needed in the evaluation phase because the network implicitly learns them during training. This feature can be helpful in applications were the costs are not directly available for unseen samples.
- The use of Parzen windows in the definition of the surrogate cost function not only allows to construct easily a sample-by-sample training algorithm, but it provides also an additional flexibility with the choice of the window, that can be fitted to the characteristics of the problem (a discussion about the role of the window with illustrative examples can be found in [27]). An appropriate window can be selected by cross-validation, as seen in the experiments, where the results showed a

higher sensitivity in linear models than in MLPs, where the hidden layers introduce the flexibility of projecting the input data into a different space, which can be implicitly forced to match the characteristics of the selected kernel.

The proposed method has been evaluated in two synthetic experiments, where the optimal Bayesian solution is known, and in several publicly available real datasets. In all cases it has shown a good performance when compared with several benchmark methods. The proposed method with a linear model obtained better results than linear regresion and cost sensitive logistic regresion in all datasets, without using re-sampling techniques. With MLPs, the proposed method obtained the best results also in all datasets.

There is another remarkable aspect. It is well known that a classification method can be very well fitted to solve a specific problem but can also have poor performance in some other problems. The proposed method showed a consistent good performance in all datasets. However, the benchmark methods exhibited a greater variability, working well in some problems, but working very poorly in some cases. Considering only the PAKDD and the HMEQ datasets to simplify the analysis, in the PAKDD dataset BMR(LR), with the 3 training options, is the only method that provides results of the same quality than the proposed method (the next method, BMR(RF)+RS provides only 28.08% and the next one is BMR(CSDT) with 27.35%). But all these methods have catastrophic results in the HMEQ dataset. In the HMEQ dataset the best benchmark method is BMR(RF), but this method has very poor results in PAKDD dataset. Therefore, in a general comparison, the proposed method provides much better results than any other benchmark method. A more detailed comparison can be found in the supplementary materials that are available at http:ieeexplore.ieee.org.

The proposed cost function has been presented here for binary problems, but it can be extended to multiclass problems in several ways, from the simplest approach of combining binary classifiers, to the direct definition of a Bayesian risk for a multiclass problem using a multiple output neural architecture.

Currently, we are actively working in the application of the proposed methodology to different deep network architectures and to different example-dependent cost problems, both in binary and multi-class situations.

ACKNOWLEDGMENT

This work has been partly supported by grants CASI-CAM-CM (S2013/ICE-2845, Madrid C/ FEDER, EUSF) and MacroADOBE (TEC2015-67719-P, MINECO/FEDER, UE).

REFERENCES

[1] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY: Springer, 2006.

[2] N. Japkowicz, "The class imbalance problem: Significance and strategies," in *Proc. of the 2000 Intl Conf. on Artificial Intelligence (ICAI)*, Las Vegas, USA, 2000, pp. 111–117.

[3] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, Sep. 2009.

[4] S. H. Khan, M. Hayat, M. Bennamoun, and F. A. Sohel, "Cost-sensitive learning of deep feature representations from imbalanced data," *IEEE Trans. on Neural Networks and Learning Systems*, vol. 29, no. 8, pp. 3573–3587, Aug. 2018.

[5] P. A. Gutierrez, M. Pérez-Ortiz, J. Sánchez-Monedero, F. Fernández-Navarro, and C. Hervás-Martínez, "Ordinal regression methods: Survey and experimental study," *IEEE Trans. on Knowledge and Data Engineering*, vol. 28, no. 1, pp. 127–146, Jan. 2016.

[6] S. Panigrahi, A. Kundu, S. Surai, and A. K. Majumdar, "Credit card fraud detection: A fusion approach using Dempster-Shafer theory and Bayesian learning," *Information Fusion*, vol. 10, no. 4, pp. 354–363, Oct. 2009.

[7] S. Bhattacharyya, S. Jha, K. Tharakunnel, and J. C. Westland, "Data mining for credit card fraud: A comparative study," *Decision Support Systems*, vol. 50, no. 3, pp. 602–613, Feb. 2011.

[8] A. C. Bahnsen, A. Stojanovic, D. Aouada, and B. Ottersten, "Cost sensitive credit card fraud detection using Bayes minimization risk," in *Proc. of the 12th Intl. Conf. on Machine Learning and Applications*. IEEE Computer Society, 2013, pp. 333–338.

[9] ——, "Improving credit card fraud detection with calibrated probabilities," in *Proc. of the 14th Intl Conf. on Data Mining*. Philadelphia, USA: SIAM, Apr. 2014, pp. 677–685.

[10] A. Dal Pozzolo, O. Caelen, Y. Le Borgne, S. Waterschoot, and G. Bontempi, "Learned lessons in credit card detection from a practitioner perspective," *Expert Systems with Applications*, vol. 41, no. 10, pp. 4915–4928, Aug. 2014.

[11] T. Verbraken, C. Bravo, R. Webber, and B. Baesens, "Development and application of consumer credit scoring models using profit-based classification measures," *European Journal of Operational Research*, vol. 238, no. 2, pp. 505–513, Oct. 2014.

[12] A. C. Bahnsen, D. Aouada, and B. Ottersten, "Example-dependent cost-sensitive logistic regression for credit scoring," in *Proc. of the 13th Intl. Conf. Machine Learning and Applications*. IEEE Computer Society, 2014, pp. 263–269.

[13] T. Verbraken, W. Verbeke, and B. Baesens, "A novel profit maximizing metric for measuring performance of customer churn prediction models," *IEEE Trans. on Knowledge and Data Engineering*, vol. 25, no. 5, pp. 961–973, May 2013.

[14] A. C. Bahnsen, D. Aouada, and B. Ottersten, "A novel cost-sensitive framework for customer churn predictive modeling," *Decision Analytics*, vol. 2, no. 5, pp. 1–15, 2015.

[15] B. Zadrozny, J. Langford, and N. Abe, "Cost-sensitive learning by cost-proportionate example weighting," in *Proc. of the Third Intl. Conf. on Data Mining*, Dec. 2003, pp. 435–442.

[16] E. W. T. Ngai, L. Xiu, and D. C. K. Chau, "Application of data mining techniques in customer relationship management: A literature review and classification," *Expert System with Applications*, vol. 36, no. 2, pp. 2592–2602, Mar. 2009.

[17] S. Moro, R. M. S. Laureano, and P. Cortez, "Using data mining for bank direct marketing: An application of the CRISP-DM methodology," in *Proc. of the European Simulation and Modeling Conf.*, Guimaraes (Portugal), 2011, pp. 117–121.

[18] A. Lenarcik and Z. Piasta, "Rough classifiers sensitive to costs varying from object to object," in *Proc. of the 1st Intl. Conf. on Rough Sets and Current Trends in Computing (LNAI 1424)*. Warsaw, Poland: Springer, Jun. 1998, pp. 222–230.

[19] C. Elkan, "The foundations of cost-sensitive learning," in *Proc. of the 17th Intl Joint Conf. on Artificial Intelligence*, vol. 2, 2001, pp. 973–978.

[20] P. Branco, L. Torgo, and R. P. Ribeiro, "A survey of predictive modeling on imbalanced domains," *ACM Computer Surveys*, vol. 49, no. 2, pp. 31:1–31:50, Aug. 2016.

[21] K. M. Ting, "An instance-weighting method to induce cost-sensitive trees," *IEEE Trans. on Knowledge and Data Engineering*, vol. 14, no. 3, pp. 659–665, 2002.

[22] F. Wysotzki and P. Geibel, "A new information measure based on example-dependent misclassification cost and its application in decision tree learning," *Advances in Artificial Intelligence*, pp. 3:1–3:13, Aug. 2009.

[23] P. Geibel and F. Wysotzki, "Learning perceptrons and piecewise linear classifiers sensitive to example dependent costs," *Applied Intelligence*, vol. 21, pp. 45–56, 2004.

[24] U. Brefeld, P. Geibel, and F. Wysotzki, "Support vector machines with example dependent costs," in *Proc. of the European Conf. on Machine Learning: ECML 2003 (LNCS 2837)*. Springer, 2003, pp. 23–34.

[25] P. González, E. Álvarez, J. Díez, R. González-Quinteros, E. Nogueira, A. López-Urrutia, and J. J. del Coz, "Multiclass support vector machines with example dependent costs applied to plankton biomass estimation,"

IEEE Trans. on Neural Networks and Learning Systems, vol. 24, no. 11, pp. 1901–1905, Jul. 2013.
[26] A. C. Bahnsen, D. Aouada, and B. Ottersten, "Example-dependent cost-sensitive decision trees," *Expert Systems with Applications*, vol. 42, no. 19, pp. 6609–6619, Nov. 2015.
[27] M. Lázaro, M. H. Hayes, and A. R. Figueiras-Vidal, "Training neural network classifiers through Bayes risk minimization applying unidimensional Parzen windows," *Pattern Recognition*, vol. 77, pp. 204–215, May 2018.
[28] E. Parzen, "On the estimation of a probability density function and the mode," *Annals of Mathematical Statistics*, vol. 33, pp. 1065–76, 1962.
[29] P. Duda, L. Rutkowski, M. Jaworski, and D. Rutkowska, "On the Parzen kernel-based probability density function learning procedures over time-varying streaming data with applications to pattern classification," *IEEE Trans. on Cybernetics*, pp. 1–14, Nov. 2018, (Early Access at http://ieeexplore.ieee.org).
[30] D. F. Specht, "Probabilistic neural networks," *Neural Networks*, vol. 3, pp. 109–118, 1990.
[31] G. A. Babich and O. I. Camps, "Weighted Parzen windows for pattern classification," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 18, no. 5, pp. 567–570, May 1996.
[32] H. L. Van Trees, *Detection, Estimation, and Modulation Theory: Part I*. New York: John Wiley and Sons, 1968.
[33] 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 Computational Mathematics and Mathematical Physics*, vol. 7, pp. 200–217, 1967.
[34] J. Cid-Sueiro, J. I. Arribas, S. Urbán-Muñoz, and A. R. Figueiras-Vidal, "Cost functions to estimate a posteriori probabilities in multiclass problems," *IEEE Trans. on Neural Networks*, vol. 10, no. 3, pp. 645–656, May 1999.
[35] J. Cid-Sueiro and A. R. Figueiras-Vidal, "On the structure of strict sense Bayesian cost functions and its applications," *IEEE Trans. on Neural Networks*, vol. 12, no. 3, pp. 445–455, May 2001.
[36] D.-Y. Yeung and C. Chow, "Parzen-window network intrusion detection," in *Proc. of the Intl Conf. on Pattern Recognition*, vol. 4, no. 4, 2002.
[37] T. Zhang, "Statistical behavior and consistency of classification methods based on convex risk minimization," *Annals of Statistics*, vol. 32, no. 1, pp. 56–85, Feb. 2004.
[38] J. Zhang, T. Liu, and D. Tao, "On the rates of convergence from surrogate risk minimizers to the Bayes optimal classifier," arXiv preprint, arXiv:1802:03688, 2018.
[39] D. F. Specht, "A general regression neural network," *IEEE Trans. on Neural Networks*, vol. 2, no. 6, pp. 568–576, Nov. 1991.
[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv:1412.6980 [cs.LG], Dec. 2014.
[41] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[42] A. C. Bahnsen, "Cost Sensitive Classification (costcla) Python module for cost-sensitive machine learning (classification)," https://pypi.org/project/costcla/, Jan. 1996, version 0.5.
[43] B. Baesens, D. Roesch, and H. Scheule, *Credit Risk Analytics: Measurement Techniques, Applications, and Examples in SAS*. John Wiley & Sons, 2016.
[44] S. Moro, P. Cortez, and P. Rita, "A data-driven approach to predict the success of bank telemarketing," *Decision Support Systems*, vol. 62, pp. 22–31, Jun. 2014.
[45] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, Jun. 2014.

Marcelino Lázaro was born in Carriazo, Cantabria, Spain, in 1972. He received the Ingeniero de Telecomunicación and Doctor Ingeniero de Telecomunicación degrees from the Universidad de Cantabria, Spain, in 1996 and 2001, respectively.

From 1996 to 2002, he was with the Departamento de Ingeniería de Comunicaciones, Universidad de Cantabria. In 2003, he joined the Departamento de Teoría de la Señal y Comunicaciones, Universidad Carlos III de Madrid, Spain. His research interest includes digital signal processing, detection, estimation, and statistical learning methods.

Aníbal R. Figueiras-Vidal (M'76, SM'84, F'12, LF'15) received the Telecommunication Engineer (Hons.) degree from Universidad Politécnica de Madrid, Madrid, Spain, in 1973, and the Doctor (Hons.) degree from Universidad Politécnica de Barcelona, Barcelona, Spain, in 1976. He also received Honoris Causa Doctor degrees from Universidad de Vigo, Vigo, Spain, in 1999, and Universidad San Pablo, Arequipa, Peru, in 2011.

He is a Professor of signal theory and communications with Universidad Carlos III de Madrid, Leganés (Madrid), Spain. He has authored or co-authored more than 300 journal and conference papers, and he has been the principal researcher in almost 100 research projects and contracts. His current research interests are digital signal processing and machine learning, including their applications.

Dr. Figueiras-Vidal has been Chair of IEEE Spain Section and IEEE Spain Joint Chapter on Signal Processing and Communications. Dr. Figueiras-Vidal is a member of the Royal Academy of Engineering of Spain. He was its President from 2007 to 2011. He is also Corresponding Member of the Mexico Academy of Engineering.