Available online at www.sciencedirect.com
SCIENCE DIRECT®
Pattern Recognition 39 (2006) 180–188
PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY
www.elsevier.com/locate/patcog

Learning prototypes and distances: A prototype reduction technique based on nearest neighbor error minimization☆

Roberto Paredes*, Enrique Vidal

Universidad Politecnica de Valencia, DSIC, Camino de Vera S/N, 46022 Valencia, Spain

Abstract

A prototype reduction algorithm is proposed, which simultaneously trains both a reduced set of prototypes and a suitable local metric for these prototypes. Starting with an initial selection of a small number of prototypes, it iteratively adjusts both the position (features) of these prototypes and the corresponding local-metric weights. The resulting prototypes/metric combination minimizes a suitable estimation of the classification error probability. Good performance of this algorithm is assessed through experiments with a number of benchmark data sets and with a real task consisting in the verification of images of human faces.

Keywords: Nearest neighbor; Condensing; Weighted dissimilarity distances

1. Introduction

The nearest neighbor (NN) classification rule has successfully been used in many pattern recognition applications. The good behavior of the NN rule with unbounded number of prototypes is well known [1]. However, in many practical pattern recognition applications only a small number of prototypes is usually available. Also, in order to reduce computational requirements of the classifier, it is often necessary to reduce the number of prototypes even further. Typically, under such a scarce-prototype framework, the ideal asymptotical behavior of NN classifiers degrades dramatically.

NN performance can be improved by using appropriately trained distance measures or metrics. Improvements can be particularly significant for small prototype sets. Trained metrics can be *global* (the same for all the prototypes) [2–4], *class-dependent*, (shared by all the prototypes of the same class) [5–7], and/or *locally dependent* (the distance measure depends on the particular position of the prototypes) in the feature space [8–11,15].

In Refs. [6,7,12–15], we explored most of these frameworks. In all the cases, the metric weight learning was based on minimizing a suitable criterion index, closely related with the (leaving-one-out) estimated probability of misclassification. While good results were achieved using these techniques for many classification tasks, it was observed that additional improvements were possible by further reducing the number of parameters to be learnt. In particular, adequately reducing the number of prototypes for which distance weights have to be learnt was considered as a promising source of improvements.

In Refs. [12–14], we developed a prototype *editing* technique based on trained local weights that only depends on the prototypes, but not on the features. This proved very effective to reduce the number of prototypes, while retaining or improving the original NN classification accuracy. Here we consider again prototype reduction, but following a different direction; that is, we aim at simultaneously training both a reduced (or "*condensed*") set of prototypes and a suitable *local metric* for *these* prototypes.

☆ Work partially supported by the Spanish "Ministerio de Ciencia y Tecnología" under Grant TIC2003-08496-C04-02.
* Corresponding author. Tel.: +34 963877000.
E-mail addresses: rparedes@iti.upv.es (R. Paredes), evidal@iti.upv.es (E. Vidal).

0031-3203/$30.00
doi:10.1016/j.patcog.2005.06.001

The proposed approach is called *learning prototypes and distances* (LPD). It starts with an initial selection of a small number of randomly selected prototypes from the training set. Then it iteratively adjusts both the position (features) of the prototypes themselves and the corresponding local-metric weights, so that the resulting combination of prototypes and metric minimizes a suitable estimation of the probability of classification error. The adjustment rules are derived by solving the minimization problem through gradient descent.

It is interesting to note that the prototype position update rules formally derived in this way are similar (though more general and accurate) to the so-called "reward–punishment" rules heuristically introduced in a number of popular techniques such as learning vector quantization (LVQ) and decision surface mapping (DSM) [16–18]. On the other hand, the results provided by LPD are reminiscent of those of support vectors machine (SVM) classifiers [19] in that both are obtained by optimizing a well-founded criterion function and in that the learned prototypes are "discriminative" as they generally lay close to the class-decision boundaries. Finally, another interesting relation comes from the fact that LPD not only learns prototype positions but also optimizes feature-dependent distance weights associated to each prototype. This reminds us of the well-known EM estimation of Gaussian mixtures [20]: LPD prototype positions and feature weights would be equivalent to the mixture component mean vectors and (diagonal) covariance matrices, respectively. The difference here is that, while EM Gaussian mixture learning is a parametric technique seeking to maximize training sample likelihood, LPD is non-parametric and explicitly aims at minimizing an empirical training sample error estimation.

The LPD technique proposed here has been tested on a number of benchmark data sets from the UCI/Statlog repositories, as well as on a real task of human face verification. The results are comparable to or better than those recently reported for other techniques in the same tasks. They also confirm that with a substantial reduction in the prototype-set size, significant accuracy improvements over conventional, all-prototype NN approaches can be obtained.

In Section 2 the proposed approach is presented and the LPD algorithm is formally derived. Section 3 describes different types of experiments and reports on the corresponding results. In Section 4, finally, we present our conclusions and discuss future work.

2. Approach

Let $T = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ be a *training set*; i.e., a collection of *training vectors* or *class-labeled* points $\mathbf{x}_i \in E$, $1 \leqslant i \leqslant N$ in a suitable *representation space* $E = \mathbb{R}^m$. A generic training vector in T will be denoted either "$\mathbf{x} \in T$" or "$\mathbf{x}_i$, $1 \leqslant i \leqslant N$". The index of a training vector $\mathbf{x}$ in T is denoted as $index(\mathbf{x})$, defined as $index(\mathbf{x}) = i$ *iff* $\mathbf{x} = \mathbf{x}_i$. Let C be the number of classes.

We seek to use T to obtain a reduced set of *prototypes*, $P = \{\mathbf{y}_1, \ldots, \mathbf{y}_n\} \subset E$, $n \ll N$, and a suitable *weighted distance* $d : E \times P \to \mathbb{R}$ associated to P, which optimize the NN classification performance.

The *weighted distance* from an arbitrary vector $\mathbf{x} \in E$ to a prototype $\mathbf{y}_i \in P$ is defined as

$$d_W(\mathbf{x}, \mathbf{y}_i) = \sqrt{\sum_{j=1}^{m} w_{ij}^2 (x_j - y_{ij})^2}, \tag{1}$$

where w_{ij} is a weight associated with the feature j of the prototype $\mathbf{y}_i$. These weights can be represented as an $n \times m$ weight matrix $W = \{w_{ij}, 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m\}$. In what follows, whenever the weight matrix W can be understood from the context, $d_W(\mathbf{x}, \mathbf{y}_i)$ will be simply denoted as $d(\mathbf{x}, \mathbf{y}_i)$.

Note that this definition assigns separate weights to the different dimensions or *features* of the representation space. Note also that it is asymmetric and *local* in that it depends on the particular position of each $\mathbf{y}_i$.

2.1. Learning the prototypes and their weights

In order to find both a matrix W and a suitable reduced set of prototypes P that results in a low error rate of the NN classifier, we propose to minimize a criterion index which is an approximation to the NN classification error of T using P and $d(\cdot, \cdot)$.

This NN error estimate can be written as

$$J(P, W) = \frac{1}{n} \sum_{\mathbf{x} \in T} step\left(\frac{d(\mathbf{x}, \mathbf{y}_{\mathbf{x}}^{=})}{d(\mathbf{x}, \mathbf{y}_{\mathbf{x}}^{\neq})} \right), \tag{2}$$

where the *step* function is defined as

$$step(z) = \begin{cases} 0 & \text{if } z \leqslant 1, \\ 1 & \text{if } z > 1 \end{cases}$$

and $\mathbf{y}_{\mathbf{x}}^{=}$, $\mathbf{y}_{\mathbf{x}}^{\neq} \in P$ are, respectively, the *same-class* and *different-class* NNs of $\mathbf{x}$. Note that each term of the above sum (2) involves two different prototypes $\mathbf{y}_{\mathbf{x}}^{=}$, $\mathbf{y}_{\mathbf{x}}^{\neq}$, and their associated weights.

As in previous work [7,12–14], a gradient descent procedure is proposed to minimize this index. This requires J to be differentiable with respect to all the parameters to be optimized; i.e., y_{ij} and w_{ij}, $1 \leqslant i \leqslant k$, $1 \leqslant j \leqslant m$. Therefore, some approximations are needed. First, the *step* function will be approximated by using a *sigmoid* function, defined as

$$\mathscr{S}_\beta(z) = \frac{1}{1 + e^{\beta(1-z)}}. \tag{3}$$

With this approximation, the proposed index becomes

$$J(P, W) \approx \frac{1}{n} \sum_{\mathbf{x} \in T} \mathcal{S}_\beta(r(\mathbf{x})), \tag{4}$$

where

$$r(\mathbf{x}) = \frac{d(\mathbf{x}, \mathbf{y}_\mathbf{x}^=)}{d(\mathbf{x}, \mathbf{y}_\mathbf{x}^\neq)}. \tag{5}$$

Clearly, if β is large $\mathcal{S}_\beta(z) \approx step(z)$, $\forall z \in \mathbb{R}$, and this approximation is very accurate. On the other hand, if β is not so large, the contribution of each NN classification error (or success) to the index J depends upon the corresponding quotient of the distances responsible for the error (or the success). As will be discussed later, in some cases this can be a desirable property which may make the *sigmoid* approximation preferable to the exact *step* function.

The derivative of $\mathcal{S}_\beta(\cdot)$ will be needed throughout the paper:

$$\mathcal{S}'_\beta(z) = \frac{\mathrm{d}\,\mathcal{S}_\beta(z)}{\mathrm{d}\,z} = \frac{\beta \mathrm{e}^{\beta(1-z)}}{(1+\mathrm{e}^{\beta(1-z)})^2}, \tag{6}$$

where $\mathcal{S}'_\beta(z)$ is a "windowing" function which has its maximum for $z = 1$ and vanishes for $|z-1| \gg 0$. Note that if β is large $\mathcal{S}'_\beta(z)$ approaches the Dirac delta function, while if it is small $\mathcal{S}'_\beta(z)$ is approximately constant for a wide range of values of z.

To obtain the partial derivatives from Eqs. (4) to (5), required for gradient descent, it should be noted that J depends on P and W through the distances $d(\cdot,\cdot)$ in two different ways. First, it depends directly through the prototypes and weights involved in the definition of $d(\cdot,\cdot)$ (1). The second, more subtle dependence is due to the fact that, for some $\mathbf{x} \in T$, $\mathbf{y}_\mathbf{x}^=$ and $\mathbf{y}_\mathbf{x}^\neq$ may be different as prototype positions and their associated weights are varied.

While the derivatives due to the first, direct dependence can be developed from Eqs. (4) and (5), the secondary dependence is non-linear and is thus more problematic. Therefore, a simple approximation will be followed here by assuming that the secondary dependence is not significant compared to the direct one. In other words, we will assume that, for sufficiently small variations of the positions and weights, the prototype neighborhoods remain unchanged. Correspondingly, we can derive from Eqs. (1), (4) and (5):

$$\frac{\partial J}{\partial y_{ij}} \approx \tag{7}$$
$$\frac{1}{n} \sum_{\substack{\forall \mathbf{x} \in T: \\ index(\mathbf{y}_\mathbf{x}^=)=i}} \mathcal{S}'_\beta(r(\mathbf{x})) r(\mathbf{x}) \frac{(y^=_{\mathbf{x}j} - x_j)}{d^2(\mathbf{x}, \mathbf{y}_\mathbf{x}^=)} w_{ij}^2$$
$$- \frac{1}{n} \sum_{\substack{\forall \mathbf{x} \in T: \\ index(\mathbf{y}_\mathbf{x}^\neq)=i}} \mathcal{S}'_\beta(r(\mathbf{x})) r(\mathbf{x}) \frac{(y^\neq_{\mathbf{x}j} - x_j)}{d^2(\mathbf{x}, \mathbf{y}_\mathbf{x}^\neq)} w_{ij}^2,$$

Algorithm *LPD* $(T, P, W, \beta, \boldsymbol{\mu}, \boldsymbol{\nu}, \varepsilon)$ {
  // T: training set; P, W: initial prototypes and weights;
  // β: sigmoid slope; $\boldsymbol{\mu}, \boldsymbol{\nu}$: learning factors; ε: small constant
  $\lambda' = \infty$; $\lambda = J(P,W)$; $P' = P$; $W' = W$
  while$(|\lambda' - \lambda| > \varepsilon)$ {
    $\lambda' = \lambda$
    for all $\mathbf{x} \in T$ {
      $\mathbf{y}_\mathbf{x}^= =$ FINDNNSAMECLASS$(P, W, \mathbf{x})$
      $\mathbf{y}_\mathbf{x}^\neq =$ FINDNNDIFFCLASS$(P, W, \mathbf{x})$
      $i = index(\mathbf{y}_\mathbf{x}^=)$
      $k = index(\mathbf{y}_\mathbf{x}^\neq)$
      $R1 = \mathcal{S}'_\beta(r(\mathbf{x})) \cdot r(\mathbf{x}) \,/\, d^2(\mathbf{x}, \mathbf{y}_i)$
      $R2 = \mathcal{S}'_\beta(r(\mathbf{x})) \cdot r(\mathbf{x}) \,/\, d^2(\mathbf{x}, \mathbf{y}_k)$
      for $j = 1 \ldots m$ {
        $y'_{ij} = y'_{ij} - \nu_{ij} \cdot w_{ij}^2 \cdot (y_{ij} - x_j) \cdot R1$
        $y'_{kj} = y'_{kj} + \nu_{kj} \cdot w_{kj}^2 \cdot (y_{kj} - x_j) \cdot R2$
        $w'_{ij} = w'_{ij} - \mu_{ij} \cdot w_{ij} \cdot (y_{ij} - x_j)^2 \cdot R1$
        $w'_{kj} = w'_{kj} + \mu_{kj} \cdot w_{kj} \cdot (y_{kj} - x_j)^2 \cdot R2$
      }
    }
    $P = P'$; $W = W'$; $\lambda = J(P,W)$
  }
  return(P, W)
}

Fig. 1. Learning prototypes and distances algorithm.

$$\frac{\partial J}{\partial w_{ij}} \approx \tag{8}$$
$$\frac{1}{n} \sum_{\substack{\forall \mathbf{x} \in T: \\ index(\mathbf{y}_\mathbf{x}^=)=i}} \mathcal{S}'_\beta(r(\mathbf{x})) r(\mathbf{x}) \frac{(y^=_{\mathbf{x}j} - x_j)^2}{d^2(\mathbf{x}, \mathbf{y}_\mathbf{x}^=)} w_{ij}$$
$$- \frac{1}{n} \sum_{\substack{\forall \mathbf{x} \in T: \\ index(\mathbf{y}_\mathbf{x}^\neq)=i}} \mathcal{S}'_\beta(r(\mathbf{x})) r(\mathbf{x}) \frac{(y^\neq_{\mathbf{x}j} - x_j)^2}{d^2(\mathbf{x}, \mathbf{y}_\mathbf{x}^\neq)} w_{ij},$$

where $r(\mathbf{x})$ and $\mathcal{S}'_\beta(\cdot)$ are as in Eqs. (5) and (6), respectively. Using these derivatives leads to the corresponding gradient descent update equations. A simple manner to implement these equations is by visiting each prototype $\mathbf{x}$ in T and updating the positions and the weights associated with the *same-class* and *different-class* NNs of $\mathbf{x}$. This is shown in the procedure presented in Fig. 1.

Two sets of learning step factors, μ_{ij}, v_{ij}, are needed by this algorithm. They can take just a fixed value for all i, j or may depend on i, j following simple rules; for instance, μ_{ij} may be inversely proportional to the variance of each feature j. In addition, for smoother (but slower) convergence, these values may be decreased along the successive iterations of the LPD *while* loop. Large values of v_{ij} give more importance to the learning of the prototypes themselves while large values of μ_{ij} emphasize the learning of the distance associated to these prototypes. In general, the values of these factors may have a significant impact on the learning speed but, as long as they are not too large, they should

not have an important influence on the learning results. Suitable values of the step factors depend on the characteristics of the training data and, typically, it is advisable that $\mu_{ij} < \nu_{ij}$.

The effects of the update equations in the LPD algorithm are intuitively clear. For each training vector $\mathbf{x}$, its *same-class* NN, $\mathbf{y}_i = \mathbf{y}_{\mathbf{x}}^{=}$, is moved towards $\mathbf{x}$, while its *different -class* NN, $\mathbf{y}_k = \mathbf{y}_{\mathbf{x}}^{\neq}$, is moved away from $\mathbf{x}$. Similarly, the feature-dependent weights associated with $\mathbf{y}_{\mathbf{x}}^{=}$ are modified so as to make it appear closer to $\mathbf{x}$ in a feature-dependent manner, while those of $\mathbf{y}_{\mathbf{x}}^{\neq}$ are modified so that it will similarly appear farther from $\mathbf{x}$. Since these update steps are weighted by the distance ratio, $r(\mathbf{x})$, their importance depends upon the relative proximity of $\mathbf{x}$ to $\mathbf{y}_{\mathbf{x}}^{=}$ or $\mathbf{y}_{\mathbf{x}}^{\neq}$. This is further divided by the corresponding squared distance, thereby reducing the update importance for large distances. Finally, the resulting steps are *windowed* by the derivative of the sigmoid function applied to the distance ratio, $r(\mathbf{x})$. This way, only those prototypes (and their weights) which are sufficiently close to the decision boundaries are actually updated.

The attentive reader will find the above *prototype* update rules closely related to the so-called *reward–punishment* rules heuristically introduced in such popular procedures as LVQ1, LVQ2 and DSM [16–18]. For each $\mathbf{x} \in T$, LVQ1 updates *only* its unique NN prototype if it is misclassified. LVQ2 further requires that its two-NNs belong to different classes and their distance ratio ($r(\mathbf{x})$ in our formulation) falls within a specified range—a very crude approximation to the $r(\mathbf{x}) \cdot \mathscr{S}'_{\beta}(r(\mathbf{x}))$ smooth windowing term of LPD. DSM, on the other hand, also requires misclassification for updating but, as with LPD, it does update both the same-class and the different class NNs. However, it does not apply any smoothing or windowing to the updates, thereby failing to work properly for those (most, in practice) tasks with overlapping classes.

It is remarkable that an intuitive interpretation of the formally derived LPD prototype update rules is so closely related with popular heuristics which, without formal proof of their potential usefulness, have proved quite helpful to improve accuracy in many practical situations. Nevertheless, the advantages of LPD are clear: not only the update policy for prototype positions, but also for the associated metric weights, along with the corresponding smoothing and windowing terms, come from a mathematical derivation which guarantees convergence towards an (approximate) local minimum of the empirical NN error estimation.

While LPD as such only guarantees convergence to local minima of an approximation to the NN error estimation, slightly better behavior is possible in some cases. As a by-product of computing the update equations, a direct empirical estimation of the NN classification error can straightforwardly be obtained at each step of the descent procedure. Therefore, following the idea already used in [7,12–14], rather than selecting the prototypes and weights obtained at the convergence point, those prototypes and weights which achieved a minimum error estimation along the descent process are selected instead.

3. Experiments

Three different sets of experiments were carried out. In the first set, the LPD algorithm was applied to well controlled, artificial data in order to show the impact of different parameter settings. In the second, a number of tasks from the UCI/Statlog repositories were considered. The aim of these experiments was to show that the proposed technique is uniformly adequate for a variety of tasks involving different data conditions: large/small training sets, continuous/discrete features, large/small dimensionalities, etc. The last set corresponds to a real application of human face verification.

3.1. Artificial data

These experiments were intended to illustrate the behavior of the proposed technique and to show the effects of different parameter settings. The main parameters to adjust are the number of prototypes of the reduced set, n, and the slope of the sigmoid function, β. For these experiments the learning factors were set to $\mu = 0.001$ and $\nu = 0.1$.

An artificial task, similar to the well-known Hart's condensing task [21], and previously considered in Refs. [22,23], was used. It involves two classes in two dimensions, as shown in Fig. 2.

The error estimate obtained using the reduced set of prototypes, P, and the local-metric weights, W, yielded by the LPD algorithm, was computed for different values of the parameters mentioned above. The number of prototypes per class of the reduced set P was varied from 1 to 4, which corresponds to values of n from 2 to 8. The β parameter was exponentially varied from 0.1 to 200. For each of these pairs of parameters, the algorithm was run 100 times with a different random initialization of P. The average NN error rates obtained with the initial sets of prototypes and weights were 26.6%, 21.5%, 17.6%, and 15.1%, for $n = 2, 4, 6, 8$, respectively. The average NN error obtained with the prototypes and weights learned by LPD for the different parameter settings is shown in Fig. 3.

Two interesting conclusions can be drawn from these results. First, the estimated error decreases as the number of prototypes in the reduced set P increases. The reason is clear; if the number of prototypes in P increases, then P approaches T and, since T is the set used for error estimation, the estimated error tends to 0%. It should be pointed out, however, that the goal of LPD is *not* to produce this kind of overfitting. In order to avoid it, the aim is to achieve an *improvement* of the results with a *significantly smaller* prototype set.

The second conclusion is that LPD performance significantly depends on the values of β, in particular, for

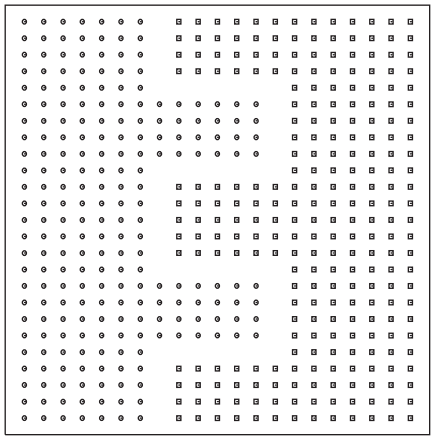

Fig. 2. A two-class artificial classification problem with $N = 476$ training points: 223 of class A (left) and 253 of class B (right).

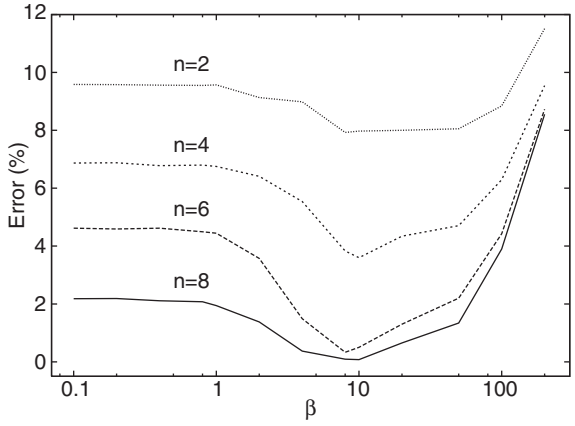

Fig. 3. Estimated error for different LPD parameter settings. For the largest values of n the LPD algorithm reaches a 0% estimated error. The average NN error rates obtained with the initial sets of prototypes and weights were 26.6%, 21.5%, 17.6%, and 15.1%, for $n = 2, 4, 6, 8$, respectively.

"intermediate" values of n. As shown in Fig. 3, good behavior is most generally achieved around $\beta = 10$. All the update equations used in the LPD algorithm are affected by the windowing factor $\mathscr{S}'_\beta(r(\mathbf{x}))$ (the derivative of the sigmoid function). The argument of $\mathscr{S}'_\beta$ is the distance ratio (5) between the $\mathbf{y}_{\mathbf{x}}^{=}$ and $\mathbf{y}_{\mathbf{x}}^{\neq}$ for each training vector $\mathbf{x} \in T$. For small values of β, the sigmoid derivative is almost constant and the algorithm learns almost the same regardless of the different values of $r(\mathbf{x})$. That is, the same importance is given to those training vectors $\mathbf{x}$ that are confidently classified (with $r(\mathbf{x}) \ll 1.0$) as to other vectors $\mathbf{x}'$ that lay close to the class-decision boundaries ($r(\mathbf{x}') \approx 1.0$), or to vectors $\mathbf{x}'$ which are plainly misclassified ($r(\mathbf{x}') > 1.0$). Usually, as the number of correctly classified vectors becomes much larger than the errors, after some iterations the algorithm becomes reluctant to learn more. On the other hand, for large values of β, learning only happens when the distance ratio is (very) close to 1, may be never, if β is very large. A suitable β value should allow LPD to learn from the prototypes that lay near the class-decision boundaries or are misclassified, but moreover, the windowing effect of the sigmoid derivative should prevent learning from outliers whose $r(\mathbf{x})$ value is too large.

The proposed algorithm often reaches an error estimation of 0% using only three prototypes per class and $\beta = 10$. Fig. 4(left) shows one of these random initializations of P (and homogeneous W; i.e., Euclidean distance (ED)). It constitutes a bad initialization for this task. The prototypes and corresponding local weighted distances learnt by the LPD algorithm are shown in Fig. 4(right). All the prototypes have migrated towards the class-decision boundaries, with local weights adequately adapted to their particular positions. The metric weights of each prototype are represented as the size of the horizontal and vertical bars associated with it. Particularly interesting is the leftmost prototype of class A. Because of the position learnt for this prototype, it only contributes to correct classification of points depending on the horizontal coordinate, which is clearly reflected by the small value of the vertical weight learnt for this prototype.

The results of these first experiments show that, for a reasonably wide range of values of β, the LPD algorithm can accurately approach the class-decision boundaries with the combination of a small number of prototypes and local weighted metric.

3.2. Benchmark data sets

Several standard benchmark corpora from the UCI repository of machine learning databases and domain theories [24] and the Statlog project [25] have been used. Most of these data sets involve both *numeric* and *categorical* features. In our experiments, each categorical feature has been replaced by b binary features, where b is the number of different values allowed for the categorical feature. Since many UCI and Statlog data sets are (very) small, *K-fold cross-validation* (K-CV) [26] has been applied to obtain the classification results. Each corpus is divided into K blocks using $K - 1$ blocks as a training set, T, and the remaining block as a test set. Here, the number of blocks is set to $K = 5$ and the size of each T is $N = \frac{4}{5}N'$, where N' is the total number of vectors in the corresponding whole data set.

Baseline results were obtained using a plain NN classifier, with each original training set, T, the Euclidean (L_2) metric and the class-dependent variance-weighted (CDVW) (diagonal Mahalanobis) distance. CDVW consists in weighting each feature by the inverse of the variance of this feature in each class. Both techniques were compared with the results obtained with the reduced set P and the weighted distance W obtained by the LPD algorithm.

In the LPD algorithm the sigmoid slope was set to $\beta = 10$ and the initial prototypes in P where obtained by randomly sampling each training set, T, approximately following the corresponding class-priors. In future work other initializa-

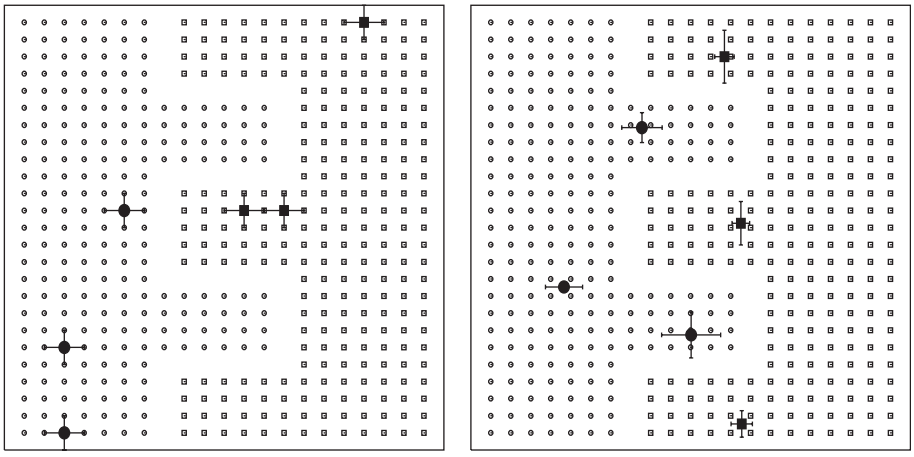

Fig. 4. (Left) Initial random selection of $n=6$ prototypes with Euclidean distance. (Right) Prototypes and distances learnt for $\beta=10$. The bars associated with each prototype show the distance weights of this prototype along each axis.

tion methods will be explored. The size of each P, n, was fixed to approximately[1] 5% of N, the number of vectors in the corresponding training set.

The distance weights of W were initialized following a simple heuristic, based on leaving-one-out NN performance of conventional methods on the training data: if raw L_2 outperforms CDVW, then set all initial $\omega_{ij}=1$; otherwise, set them to the inverse of the corresponding class-dependent training data standard deviations. Similarly, the step factors μ_{ij} and ν_{ij} were set to small constants, 0.001 and 0.01, respectively, in the former case and to these constants multiplying the inverse of the standard deviation in the latter.

Table 1 summarizes the results. Using only 5% randomly selected initial prototypes, the proposed algorithm improves the (plain L_2 or CDVW) NN accuracy for all the tasks considered except *cancer* and *wine*. The results are significantly better for most of the tasks: *Australian*, *balance*, *diabetes*, *DNA*, *heart*, and *vote*. Experiments with other values of n and β were also conducted. Using $\beta \in \{5, 10\}$ and $n \in \{2\%, 5\%, 10\%\}$ of N, yielded similar results for all the tasks except *wine*, for which significantly lower error rates were obtained (as low as 0.5% for $n=2\%$ of N and $\beta=10$). Overall, the LPD results are comparable to or better than those recently reported by other authors on the same data sets (see, e.g., [15,23,27]).

These results are generally better than those reported in Ref. [12] using other metric learning techniques and weights configurations.

The LPD technique was also explicitly compared with LVQ [17,18] and SVM [19]. DSM was not considered because it is only useful when classes do not overlapped, which very seldom happens with the UCI and Statlog data sets. LVQ1, LVQ2, and LVQ3 were tested with all the datasets presented before, while, for the sake of clarity, SVM was only tested for those tasks involving a two-class problem. The "*lvq-pak*" toolkit was used for the LVQ experiments. The number of prototypes for the LVQ technique was set following the same criteria as for LPD; i.e., 5% of the training vectors available, distributed among classes according to the class-priors. The experiments with SVM were performed with the "*svm-light*" toolkit using three different kernels: first-order polynomial (SVM_1), second-order polynomial (SVM_2), and radial basis functions (SVM_3). Table 2 shows the comparative results.

From these comparisons and the previously discussed results (Table 1) it can be seen that LPD exhibits a *uniformly good behavior* for all the tasks.

3.3. Face verification

In this section, LPD is used to train a face verification system. These experiments were carried out with the BANCA dataset [28].

This database was designed in order to test multimodal identity verification systems using different sensors under several scenarios. Different cameras and microphones where used within three different scenarios: controlled, degraded, and adverse. Each recording session involved 52 subjects, 26 male and 26 female. Each subject participated in 12 recording sessions in different conditions and with different cameras as mentioned above. Each session contains two recordings per subject, aimed at being considered a true client access and an impostor attack. Each recording is composed of five images, with a total of 120 images per subject. Each group of 26 gender-specific subjects was itself subdivided into two groups of 13 subjects, denoted by G1 and G2.

In the BANCA protocol, seven different experimental configurations were specified. In the present work, the experiments were carried using the Pooled (P) configuration. In this configuration, the training process is performed using only five true client images of each subject of the first

[1] This is only approximate because of the further need to approximate class-priors by appropriately dividing n among classes.

Table 1
Five-fold cross-validation error rate (in %) for different NN classifiers and several data sets

Task	N'	C	m	L_2	CDVW	LPD	n
Australian	690	2	42	30.5	18.4	**13.9**	27
Balance	625	3	4	41.2	32.1	**16.3**	26
Cancer	685	2	9	**3.1**	9.8	3.4	28
Diabetes	768	2	8	40.8	32.6	**26.0**	31
DNA	3186	3	180	29.0	12.8	**4.9**	128
German	1000	2	24	27.1	32.1	**26.0**	40
Glass	214	6	9	28.6	34.1	**28.0**	10
Heart	270	2	13	45.5	24.1	**18.6**	11
Letter	20000	26	16	4.2	7.0	**3.5**	799
Liver	347	2	6	35.6	38.0	**33.3**	14
Satimage	6435	6	36	11.2	15.5	**10.6**	256
Vehicle	846	4	18	29.1	31.6	**27.4**	35
Vote	435	2	16	9.9	7.1	**3.7**	18
Wine	180	3	13	17.5	**1.2**	5.0	7

N', C and m are, respectively, the total number of training vectors, the number of classes and the dimension of each data set. The overall number of prototypes used in the LPD method is shown under column n (this number is always $\approx 5\%$ of the number of vectors used by L_2 and CDVW). Results in boldface correspond to the best accuracy.

Table 2
Comparative results using the proposed LPD algorithm and the well-known LVQ and SVM algorithms

Task	LPD	LVQ1	LVQ2	LVQ3	SVM_1	SVM_2	SVM_3
Australian	**13.9**	33.3	34.0	31.1	35.0	36.1	44.5
Balance	16.3	16.3	**14.7**	16.3	—	—	—
Cancer	3.4	4.4	4.5	4.2	**3.1**	4.4	17.4
Diabetes	26.0	26.4	25.8	26.0	24.3	**24.1**	34.9
DNA	**4.9**	9.8	12.1	21.6	—	—	—
German	**26.0**	30.1	28.5	28.7	30.0	29.2	30.0
Glass	**28.0**	39.6	42.4	41.5	—	—	—
Heart	**18.6**	36.0	34.0	34.0	39.2	34.5	44.4
Letter	**3.5**	12.0	10.3	10.5	—	—	—
Liver	33.3	32.5	32.7	33.6	**31.9**	42.0	40.8
Satimage	**10.6**	11.7	11.1	11.1	—	—	—
Vehicle	**27.4**	38.2	32.0	34.4	—	—	—
Vote	**3.7**	7.6	6.2	5.7	5.0	5.3	16.1
Wine	**5.0**	29.7	25.8	29.2	—	—	—

session in the controlled scenario while images from the controlled, degraded, and adverse scenarios are used for the client and impostor tests. The P configuration is known to be a *difficult* test.

The experiments were performed with a relatively low resolution, 55×51 gray-level face images, obtained from the original 720×576 images. To train the system the mirror versions of the original training images were also used, thereby increasing each subject's training set to 10 images. The test set has 2990 images, 115 (mixed client and impostor) images per subject. Given the relatively small amount of training data per subject available, LPD was initialized using only one prototype per class, with the ED. The sigmoid slope was set to $\beta = 10$.

Following the standard BANCA protocol for the P configuration, to evaluate the face verification system the *false acceptance* (FA) and the *false rejection* (FR) need to be computed. The system is evaluated by means of the minimum of the so-called *half-total error rate* (HTER) which is the average of FR and FA.

In order to compute FA and FR an estimation of the posterior probability that a test image belongs to each class (subject) is needed. Using the prototypes and associated distances learnt by LPD, this is estimated as an adequately normalized inverse of the local distance to the nearest prototype of each class:

$$\hat{P}(c|\mathbf{x}) = \frac{1}{Q(\mathbf{x})} \frac{1}{d(\mathbf{x}, \mathbf{y}_{\mathbf{x}}^{c})^{\alpha}}, \quad Q(\mathbf{x}) = \sum_{c'=1}^{C} \frac{1}{d(\mathbf{x}, \mathbf{y}_{\mathbf{x}}^{c'})^{\alpha}}, \tag{9}$$

where $\mathbf{x}$ is a test image, $\mathbf{y}_{\mathbf{x}}^{c}$ is the NN of $\mathbf{x}$ in class c (the only prototype in c, in our case), C is the number of classes and α is a parameter to control the smoothing degree.

We compare the results obtained with the LPD algorithm with the results obtained in Ref. [29] and other results from research reports of the IDIAP [30,31]. These results were

Table 3
Results of face verification using the (hard) protocol P

Method	FA	FR	min HTER
ORG/ED	27.2	26.4	26.8
ORG/NC	26.6	26.6	26.6
ORG/SVM	4.9	27.7	16.3
PCA/SVM	4.4	35.3	19.8
LDA/SVM	1.7	46.6	24.2
LDA/MLP	18.9	10.5	14.7
ORG/LPD	10.3	16.5	**13.4**

The values FA, FR and min HTER are averaged for the two configurations, G1 and G2.

obtained with a similar (a bit higher) resolution images. The preprocessing used in these experiments was also similar to the preprocessing steps described above. In these works different results are reported using ED, normalized correlation (NC) distance, SVM and multi layer perceptron (MLP) using the original image (ORG), principant component analysis (PCA), and linear discriminant analysis (LDA) as features.

Table 3 shows the results averaged for the two different configurations of the P protocol, configurations G1 and G2. LPD performs better than the other methods. Among the approaches that directly use the gray-level images as features (those marked as ORG in the table), LPD performs much better in all the cases. It also outperforms other classifiers based on more elaborate feature spaces. In particular, LPD is better than SVM even when SVM is assisted by PCA or LDA. Only the result obtained using a MLP assisted by LDA is comparable to that of LPD.

4. Conclusions

A new gradient descent technique has been proposed to simultaneously train both a *reduced set of prototypes* and a corresponding *local metric* in order to optimize NN classification accuracy. The effectiveness of this technique has been assessed through a number of experiments involving standard benchmark data sets.

LPD exhibits a uniformly good behavior in a wide range of applications. The impact of the two only important parameters, the slope of the sigmoid function, β, and the number of prototypes n, has been studied both conceptually and empirically. The slope of the sigmoid function plays an important role to control the learning process. Adequate values (typically around 10) help reducing the impact of outliers and/or prototypes which are far away from the class-decision boundaries. The number of prototypes, on the other hand, has to be sufficiently small so as to avoid possible overfitting caused by a too optimistically biased error estimation if n is large. Overall, the tuning of these two parameters has proved not to be critical in any of the tasks we have considered so far.

Future work will aim at exploring better techniques to initialize the reduced prototype set, P. The currently used random selection entails using large prototype learning step factors, ν, (relative to those of the weights, μ) in order to allow the prototypes to "migrate" to adequate positions near the class-decision boundaries. Therefore, LPD might achieve faster and smoother convergence (and perhaps improved results) by selecting initial prototypes which are closer to the decision boundaries.

References

[1] T. Cover, P. Hart, Nearest neighbor pattern classification, IEEE Trans. Inf. Theory 13 (1) (1967) 21–27.
[2] R. Kohavi, P. Langley, Y. Yung, The utility of feature weighting in nearest-neighbor algorithms, in: Proceedings of the Ninth European Conference of Machine Learning, Prague, Springer, Berlin, 1997.
[3] I. Kononenko, Estimating attributes: analysis and extensions of RELIEF, Technical Report, University of Ljubjana, Faculty of Electrical Engineering and Computer Science, 1993.
[4] D. Wilson, T.R. Martinez, Value difference metrics for continuously valued attributes, in: Proceedings of the AAAI'96, 1996, pp. 11–14.
[5] N. Howe, C. Cardie, Examining locally varying weights for nearest neighbor algorithms, in: Second International Conference on Case-Based Reasoning, Lecture Notes in Artificial Intelligence, Springer, Berlin, 1997, pp. 455–466.
[6] R. Paredes, E. Vidal, A nearest neighbor weighted measure in classification problems, in: VIII Simposium Nacional de Reconocimiento de Formas y Análisis de Imágenes, Proc, vol. 1, Bilbao, Spain, May 1999, pp. 437–444.
[7] R. Paredes, E. Vidal, A class-dependent weighted dissimilarity measure for nearest neighbor classification problems, Pattern Recognition Lett. 21 (2000) 1027–1036.
[8] C. Domeniconi, J. Peng, D. Gunopulos, Locally adaptive metric nearest neighbor classification, IEEE Trans. Pattern Anal. Mach. Intell. 24 (9) (2002) 1281–1285.
[9] T. Hastie, R. Tibshirani, Discriminant adaptive nearest neighbor classification and regression, in: D.S. Touretzky, M.C. Mozer, M.E. Hasselmo (Eds.), Advances in Neural Information Processing Systems, vol. 8, MIT Press, Cambridge, MA, 1996, pp. 409–415.
[10] F. Ricci, P. Avesani, Data compression and local metrics for nearest neighbor classification, IEEE Trans. Pattern Anal. Mach. Intell. 21 (4) (1999) 380–384.
[11] R. Short, K. Fukunaga, A new nearest neighbor distance measure, in: Proceedings of the Fifth IEEE International Conference on Pattern Recognition, Miami Beach, FL, 1980, pp. 81–86.
[12] R. Paredes, Técnicas para la mejora de la clasificación por el vecino más cercano, Ph.D. Thesis, DSIC-UPV, 2003.
[13] R. Paredes, E. Vidal, Weighting prototypes,. A new editing approach, in: Proceedings of the 15th International Conference on Pattern Recognition, vol. 2, Barcelona, September 2000, pp. 25–28.
[14] R. Paredes, E. Vidal, D. Keysers, An evaluation of the WPE algorithm using tangent distance, in: ICPR 2002, International Conference on Pattern Recognition, 2002, pp. 48–51.
[15] J. Peng, D.R. Heisterkamp, H. Dai, Adaptive quasiconformal kernel nearest neighbor classification, IEEE Trans. Pattern Anal. Mach. Intell. 26 (5) (2004).
[16] S. Geva, J. Sitte, Adaptive nearest neighbor pattern classification, IEEE Trans. Neural Networks 2 (2) (1991) 318–322.
[17] T. Kohonen, The self-organizing map, in: Proceedings of the IEEE, vol. 78, 1990, pp. 1464–1480.
[18] T. Kohonen, G. Barna, R. Chrisley, Statistical pattern recognition with neural networks: benchmarking studies, in: IEEE International Conference on Neural Networks, vol. 1, San Diego, 1988, pp. 61–68.

[19] V. Vapnik, Statistical Learning Theory, Wiley-Interscience, New York, 1998.
[20] G.J. McLachlan, D. Peel, Finite Mixture Models, Wiley, New York, 2000.
[21] P.E. Hart, The condensed nearest neighbor rule, IEEE Trans. Inf. Theory 14 (1968) 515–516.
[22] C.-L. Chang, Finding prototypes for nearest neighbor classifiers, IEEE Trans. Comput. 23 (11) (1974) 1179–1184.
[23] R.A. Mollineda, F.J. Ferri, E. Vidal, An efficient prototype merging strategy for the condensed 1-NN rule through class-conditional hierarchical clustering, Pattern Recognition 35 (12) (2002) 2771–2782.
[24] C. Blake, E. Keogh, C. Merz, UCI Repository of machine learning databases, ⟨http://www.ics.uci.edu/mlearn/MLRepository.html⟩.
[25] D. Statistics and M.S.S.S. University, Statlog Corpora, ⟨ftp.strath.ac.uk⟩.
[26] S. Raudys, A. Jain, Small sample effects in statistical pattern recognition: recommendations for practitioners, IEEE Trans. Pattern Anal. Mach. Intell. 13 (3) (1991) 252–264.
[27] M. Titsias, A. Likas, Class conditional density estimation using mixtures with constrained component sharing, IEEE Trans. Pattern Anal. Mach. Intell. 25 (7) (2003) 924–928.
[28] E. Bailly-Bailliere, S. Bengio, F. Bimbot, M. Hamouz, J. Kittler, J. Mariethoz, J. Matas, K. Messer, V. Popovici, F. Poree, B. Ruiz, J. Thiran, The BANCA database and evaluation protocol, in: AVBPA03, 2003, pp. 625–638.
[29] A. Kostin, M. Sadeghi, J. Kittler, K. Messer, On representation space for SVM based face verification, in: Cost 275 Workshop The Advent of Biometrics on the Internet, 2002, pp. 9–15.
[30] S. Marcel, A symmetric transformation for LDA-based face verification, Technical Report, IDIAP, 2003.
[31] S. Marcel, Face verification using LDA and MLP on the BANCA database, Technical Report, IDIAP, 2003.