Large Margin Nearest Neighbor Classifiers

Carlotta Domeniconi, Dimitrios Gunopulos, and Jing Peng, *Member, IEEE*

***Abstract*—The nearest neighbor technique is a simple and appealing approach to addressing classification problems. It relies on the assumption of locally constant class conditional probabilities. This assumption becomes invalid in high dimensions with a finite number of examples due to the curse of dimensionality. Severe bias can be introduced under these conditions when using the nearest neighbor rule. The employment of a locally adaptive metric becomes crucial in order to keep class conditional probabilities close to uniform, thereby minimizing the bias of estimates. We propose a technique that computes a locally flexible metric by means of support vector machines (SVMs). The decision function constructed by SVMs is used to determine the most discriminant direction in a neighborhood around the query. Such a direction provides a local feature weighting scheme. We formally show that our method increases the margin in the weighted space where classification takes place. Moreover, our method has the important advantage of online computational efficiency over competing locally adaptive techniques for nearest neighbor classification. We demonstrate the efficacy of our method using both real and simulated data.**

***Index Terms*—Feature relevance, margin, nearest neighbor classification, support vector machines (SVMs).**

I. Introduction

In A classification problem, we are given J classes and l training observations. The training observations consist of n feature measurements $\mathbf{x} = (x_1, \ldots, x_n)^T \in \Re^n$ and the known class labels $y = 1, \ldots, J$. The goal is to predict the class label of a given query $\mathbf{q}$.

The K-nearest neighbor (K-NN) classification method [7], [15], [18], [19], [24], [26] is a simple and appealing approach to this problem: it finds the K-NNs of $\mathbf{q}$ in the training set, and then predicts the class label of $\mathbf{q}$ as the most frequent one occurring in the K neighbors. Such a method produces continuous and overlapping, rather than fixed, neighborhoods and uses a different neighborhood for each individual query so that all points in the neighborhood are close to the query, to the extent possible. In addition, it has been shown [8], [11] that the one nearest neighbor rule has asymptotic error rate that is at most twice the Bayes error rate, independent of the distance metric used.

Manuscript received March 21, 2003; revised December 21, 2004. The work of C. Domeniconi was supported in part by the 2004 R. E. Powe Junior Faculty Award 220950. The work of D. Gunopulos was supported in part by the National Science Foundation under Grants NSF CAREER Award 9984729 and NSF 0330481, in part by the US Department of Defense, and in part by a research award from AT&T. The work of J. Peng was supported in part by Grants from ARO DAAD19-03-C-0111 and Louisiana BOR LEQSF(2002–05)-RD-A-29.

C. Domeniconi is with the Information and Software Engineering Department, George Mason University, Fairfax, VA 22030 USA (e-mail: carlotta@ise.gmu.edu).

D. Gunopulos is with the Computer Science Department, University of California, Riverside, CA 92521 USA (e-mail: dg@cs.ucr.edu).

J. Peng is with the Electrical Engineering and Computer Science Department, Tulane University, New Orleans, LA 70118 USA (e-mail: jp@eecs.tulane.edu).

Digital Object Identifier 10.1109/TNN.2005.849821

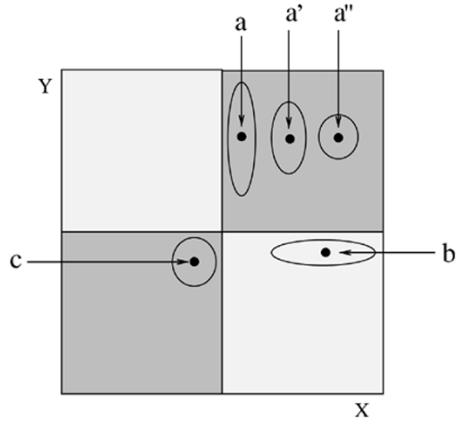

Fig. 1. Feature relevance varies with query locations.

The nearest neighbor rule becomes less appealing with finite training samples, however. This is due to the curse of dimensionality [5]. Severe bias can be introduced in the nearest neighbor rule in a high-dimensional input feature space with finite samples. As such, the choice of a distance measure becomes crucial in determining the outcome of nearest neighbor classification. The commonly used Euclidean distance measure, while simple computationally, implies that the input space is isotropic or homogeneous. However, the assumption for isotropy is often invalid and generally undesirable in many practical applications. Fig. 1 illustrates a case in point, where class boundaries are parallel to the coordinate axes. For query $\mathbf{a}$, dimension X is more relevant, because a slight move along the X axis may change the class label, while for query $\mathbf{b}$, dimension Y is more relevant. For query $\mathbf{c}$, however, both dimensions are equally relevant. This implies that distance computation does not vary with equal strength or in the same proportion in all directions in the feature space emanating from the input query. Capturing such information, therefore, is of great importance to any classification procedure in high-dimensional settings.

Several techniques [10], [12], [14], [21] have been proposed to try to minimize bias in high dimensions by using locally adaptive mechanisms. The "lazy learning" approach [1] used by these methods, while appealing in many ways, requires a considerable amount of online computation, which makes it difficult for such techniques to scale up to large data sets. The feature weighting schemes they introduce, in fact, are query based and applied online when the test point is presented to the "lazy learner."

In this paper we propose a locally adaptive metric classification method that, while still resting on a query based weighting mechanism, computes offline the information relevant to defining local weights. Specifically, our technique uses support vector machines (SVMs) to guide the process for estimating a

local flexible metric. SVMs have been successfully used as a classification tool in a variety of areas [6], [16], [20], and the maximum margin boundary they provide has been proved to be optimal in a structural risk minimization sense. The solid theoretical foundations that have inspired SVMs convey desirable computational and learning theoretic properties to the SVMs learning algorithm, and therefore SVMs are a natural choice for seeking local discriminant directions between classes.

While the solution provided by SVMs is theoretically sound, SVMs maximize the margin in feature space. However, the feature space does not always capture the structure of the input space. As noted in [2], the large margin in the feature space does not necessarily translate into a large margin in the input space. In fact, it is argued that sometimes SVMs give a very small margin in the input space, because the metric of the feature space is usually quite different from that of the input space [2]. Such a situation is undesirable. In this paper, we show that our approach overcomes this limitation. In fact, we formally prove that our weighting scheme increases the margin, and therefore the separability of classes, in the transformed space where classification takes place.

The solution provided by SVMs guides the extraction of local information in a neighborhood around the query. This process produces highly stretched neighborhoods along boundary directions when the query is close to the boundary. As a result, the class conditional probabilities tend to be constant in the modified neighborhood, whereby better classification performance can be achieved. The amount of elongation-constriction decays as the query moves farther from the vicinity of the decision boundary. This phenomenon is exemplified in Fig. 1 by queries a, a' and a''. In this paper, we present both theoretical and experimental evidence of the accuracy achieved by means of this local weighting scheme.

We avoid cross validation by using a principled technique for setting the procedural parameters of our method. Our approach to automatic parameter selection leverages the sparse solution provided by SVMs. As a result, our technique avoids expensive cross validation and has only one adjustable tuning parameter, namely the number K of neighbors in the final nearest neighbor rule. This parameter is common to all nearest neighbor techniques. On the other hand, the competing techniques have multiple parameters whose values must be determined through cross validation. (adaptive metric nearest neighbor (ADAMENN) [10] has six parameters; Machete/Scythe [12] each has four parameters; discriminant adaptive nearest neighbor (DANN) [14] has two parameters.)

Furthermore, the technique proposed here speeds up the online computation process since it computes offline local weighting information and applies the nearest neighbor rule only once. In contrast, ADAMENN, for example, applies it to each point within a region centered on the query. Indeed, our technique is capable of estimating local feature relevance using a global decision scheme such as SVMs. As such, the bulk of computation is done offline, leaving only local refinements online. This results in a method that is much more efficient computationally than current locally adaptive techniques for nearest neighbor classification [10], [12], [14], [21] that perform feature relevance estimates on the fly. We provide complexity analysis for each of the adaptive nearest neighbor methods in Section VIII. The analysis shows that our technique is indeed more efficient than the competing nearest neighbor methods.

The rest of the paper is organized as follows. In Section II, we briefly discuss the objectives that motivate adaptive metric nearest neighbor techniques. In Section III, we introduce the main concepts of SVMs. In Section IV, we present our approach to measuring local feature relevance based on SVMs. Section V describes how to estimate the quantities involved in our local feature relevance measure, and formally presents our algorithm. In Section VI, we show that our weighting scheme increases the margin in the transformed space. Section VII presents the methods we consider for comparison in our experiments. Section VII-A compares the methods through simulated examples while Section VII-B uses real data examples. In Section VIII, we provide complexity analysis of online computation for each of the competing locally adaptive methods. Section IX is a discussion of related work and a concluding summary is given in Section X.

II. Adaptive Metric Nearest Neighbor Classification

In a classification problem we are given l observations $\mathbf{x} \in \Re^n$, each coupled with the corresponding class label y, with $y = 1, \ldots, J$. It is assumed that there exists an unknown probability distribution $P(\mathbf{x}, y)$ from which data are drawn. To predict the class label of a given query $\mathbf{q}$, we need to estimate the class posterior probabilities $\{P(j|\mathbf{q})\}_{j=1}^{J}$.

K nearest neighbor methods are based on the assumption of smoothness of the target functions, which translates to locally constant class posterior probabilities $P(j|\mathbf{q})$. That is: $P(j|(\mathbf{q} + \delta\mathbf{q})) \simeq P(j|\mathbf{q})$, for $\|\delta\mathbf{q}\|$ small enough. Then, $P(j|\mathbf{q}) \simeq ((\sum_{\mathbf{x}\in N(\mathbf{q})} P(j|\mathbf{x}))/|N(\mathbf{q})|)$, where $N(\mathbf{q})$ is a neighborhood of $\mathbf{q}$ that contains points $\mathbf{x}$ that are "close" to $\mathbf{q}$, and $|N(\mathbf{q})|$ denotes the number of points in $N(\mathbf{q})$. This motivates the estimate

$$\hat{P}(j|\mathbf{q}) = \frac{\sum_{i=1}^{l} 1(\mathbf{x}_i \in N(\mathbf{q}))1(y_i = j)}{\sum_{i=1}^{l} 1(\mathbf{x}_i \in N(\mathbf{q}))}$$

where $1()$ is an indicator function such that it returns 1 when its argument is true, and 0 otherwise.

The assumption of smoothness, however, becomes invalid for any fixed distance metric when the input observation approaches class boundaries. The objective of locally adaptive metric techniques for nearest neighbor classification is then to produce a modified local neighborhood in which the posterior probabilities are approximately constant.

The techniques proposed in [10], [12], and [14] are based on different principles and assumptions for the purpose of estimating feature relevance locally at query points, and therefore weighting accordingly distances in input space. The idea common to these techniques is that the weight assigned to a feature, locally at query $\mathbf{q}$, reflects its estimated relevance to predict the class label of $\mathbf{q}$: larger weights correspond to larger capabilities in predicting class posterior probabilities. As a result, neighborhoods get constricted along the most relevant dimensions

and elongated along the less important ones. The class conditional probabilities tend to be constant in the resulting neighborhoods, whereby better classification performance can be obtained.

III. SVMs

In this section, we introduce the main concepts and properties of SVMs. Again, we are given $\{\mathbf{x}_i, y_i\}_{i=1}^{l}$, where class label $y_i \in \{-1, +1\}$. The task is to learn a set of parameters α in $f(\mathbf{x}, \alpha)$ so that f realizes the mapping $\mathbf{x}_i \rightarrow y_i$. A particular choice of α defines the corresponding trained machine $f(\mathbf{x}, \alpha)$.

The expectation of the test error for a trained machine is

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

The quantity $R(\alpha)$ is called the expected risk, or just the risk. It gives a nice way of writing the true mean error, but unless we have an estimate of what $P(\mathbf{x}, y)$ is, it is not very useful. The empirical risk $R_{\text{emp}}(\alpha)$ is then defined as the mean error rate measured over the training set

$$R_{\text{emp}}(\alpha) = \frac{1}{2l} \sum_{i=1}^{l} |y_i - f(\mathbf{x}_i, \alpha)|.$$

The following bound holds (with high probability over the random draw of the training sample) [27]:

$$R(\alpha) \leq R_{\text{emp}}(\alpha) + \text{Conf}(h)$$

where h is the Vapnik Chervonenkis (VC) dimension that measures the ability of the machine to learn any training set without error. The term $\text{Conf}(h)$ is called the VC confidence. Given a family of functions $f(\mathbf{x}, \alpha)$, it is desirable to choose a machine that gives the lowest upper bound on the risk. The first term, $R_{\text{emp}}(\alpha)$, represents the accuracy attained on a particular training set, whereas the second term $\text{Conf}(h)$ represents the ability of the machine to learn any training set without error. $R_{\text{emp}}(\alpha)$ and $\text{Conf}(h)$ drive the bias and variance of the generalization error, respectively. The best generalization error is achieved when the right balance between these two terms is attained. This gives a principled method for choosing a learning machine for a specific task, and is the essential idea of structural risk minimization.

Unlike traditional methods that minimize the empirical risk, SVMs aim at minimizing an upper bound of the generalization error. The previous bound does not directly apply to SVMs, since the VC dimension depends on the location of the examples. The bounds in [25] account for this data dependency. SVMs achieve this goal by learning the αs in $f(\mathbf{x}, \alpha)$ so that the resulting trained machine satifies the maximum margin property, i.e., the decision boundary it represents has the maximum minimum distance from the closest training point.

Another appealing property of SVMs is the sparseness representation of the decision function they compute. The location of the separating hyperplane in feature space is specified via real-valued weights on the training examples. In general, those training examples that lie far away from the hyperplane do not participate in its specification and therefore receive zero weight. Training examples that lie close to the decision boundary receive nonzero weights. These training examples are called support vectors, since their removal would change the location of the separating hyperplane. The design of SVMs, in general, allows the number of the support vectors to be small compared to the total number of training examples. This property allows SVMs to classify new examples efficiently, since the majority of the training examples will be safely ignored.

A. Learning With SVMs

In the simple case of two linearly separable classes, a SVM selects, among the infinite number of linear classifiers that separate the data, the classifier that minimizes an upper bound on the generalization error. The SVM achieves this goal by computing the hyperplane $\mathbf{w} \cdot \mathbf{x} - b = 0$ that has the maximum minimum distance from the closest training point, i.e., the maximum margin property [9], [28].

If the two classes are nonseparable, the SVM looks for the hyperplane that maximizes the margin and that, at the same time, minimizes an upper bound of the error. The tradeoff between margin and upper bound of the misclassification error is driven by a positive constant C that has to be chosen beforehand. The corresponding decision function is then obtained by considering the sign$(f(\mathbf{x}))$, where $f(\mathbf{x}) = \sum_i \alpha_i y_i \mathbf{x}_i^T \mathbf{x} - b$, and the α_is, defined over the hypercube $[0, C]^l$, are the Lagrange coefficients that maximize $L_D = \sum_i \alpha_i - (1/2) \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$. The parameter b is also computed from the data. In general, the solution will have a number of coefficients α_i equal to zero, and since there is a coefficient α_i associated to each data point, only the data points corresponding to nonzero α_i will influence the solution. These points are the support vectors. Intuitively, the support vectors are the data points that lie at the border between the two classes, and a small number of support vectors indicates that the two classes can be well separated.

This technique can be extended to allow for nonlinear decision surfaces. This is done by mapping the input vectors into a higher dimensional feature space: $\phi : \Re^n \rightarrow \Re^N$, and by formulating the linear classification problem in the feature space. Therefore, $f(\mathbf{x})$ can be expressed as $f(\mathbf{x}) = \sum_i \alpha_i y_i \phi^T(\mathbf{x}_i)\phi(\mathbf{x}) - b$.

If one were given a function $K(\mathbf{x}, \mathbf{y}) = \phi^T(\mathbf{x})\phi(\mathbf{y})$, one could learn and use the maximum margin hyperplane in feature space without having to compute explicitly the image of points in $\Re^N$. It has been proved (Mercer's Theorem) [9] that for each continuous positive definite function $K(\mathbf{x}, \mathbf{y})$ there exists a mapping ϕ such that $K(\mathbf{x}, \mathbf{y}) = \phi^T(\mathbf{x})\phi(\mathbf{y}), \forall \mathbf{x}, \mathbf{y} \in \Re^n$. By making use of such function K (*kernel function*), the equation for $f(\mathbf{x})$ can be rewritten as

$$f(\mathbf{x}) = \sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) - b. \tag{1}$$

IV. Feature Weighting

The maximum margin decision boundary found by SVMs is used here to determine local discriminant directions in the neighborhood around the query. The normal direction to local decision boundaries identifies the orientation along which data

points between classes are well separated. The gradient vector computed at points on the boundary allows us to capture such information, and to use it for measuring local feature relevance and weighting features accordingly. The resulting weighting scheme improves upon the solution computed by SVMs by increasing the margin in the space transformed by the weights. Here are the major thrusts of our proposed method.

SVMs classify patterns according to the sign$(f(\mathbf{x}))$. Clearly, in the case of a nonlinear feature mapping ϕ, the SVM classifier gives a nonlinear boundary $f(\mathbf{x}) = 0$ in the input space. The gradient vector $\mathbf{n_d} = \nabla_{\mathbf{d}} f$, computed at any point $\mathbf{d}$ on the level curve $f(\mathbf{x}) = 0$, points to the direction perpendicular to the decision boundary in the input space at $\mathbf{d}$. As such, the vector $\mathbf{n_d}$ identifies the orientation in the input space along which the projected training data are well separated in the neighborhood around $\mathbf{d}$. Therefore, the orientation given by $\mathbf{n_d}$, and any orientation close to it, carries highly discriminant information for classification. As a result this information can be used to define a local measure of feature relevance.

Let $\mathbf{q}$ be a query point whose class label we want to predict. Suppose $\mathbf{q}$ is close to the boundary, which is where class conditional probabilities become locally nonuniform, and therefore estimating local feature relevance becomes crucial. Let $\mathbf{d}$ be the closest point to $\mathbf{q}$ on the boundary $f(\mathbf{x}) = 0$

$$\mathbf{d} = \arg\min_{\mathbf{p}} \|\mathbf{q} - \mathbf{p}\|, \quad \text{subject to} f(\mathbf{p}) = 0. \tag{2}$$

Then we know that the gradient $\mathbf{n_d}$ identifies a discriminant direction.

As a consequence, the subspace spanned by the orientation $\mathbf{n_d}$ intersects the decision boundary and contains changes in class labels. Therefore, when applying the nearest neighbor rule to $\mathbf{q}$, we desire to stay close to $\mathbf{q}$ along the $\mathbf{n_d}$ direction, because that is where it is likely to find points similar to $\mathbf{q}$ in terms of the class conditional probabilities. Distances should be increased (due to larger weight) along $\mathbf{n_d}$ and directions close to it, thus, excluding points along $\mathbf{n_d}$ that are away from $\mathbf{q}$. The farther we move from the $\mathbf{n_d}$ direction, the less discriminant the correspondending orientation. This means that class labels are unlikely to change along those orientations, and distances should be reduced (due to smaller weight), thus, including points which are likely to be similar to $\mathbf{q}$ in terms of the class conditional probabilities.

This principle is in direct analogy to linear discriminant analysis. In fact, the orientation of the gradient vector identifies the direction, locally at the query point, along which the projected training data are well separated. This property guides the process of creating modified neighborhoods that tend to have homogeneous class conditional probabilities.

Formally, we can measure how close a direction $\mathbf{t}$ is to $\mathbf{n_d}$ by considering the dot product $\mathbf{n_d^T t}$. In particular, denoting $\mathbf{e}_j$ the canonical unit vector along input feature j, for $j = 1, \ldots, n$, we can define a measure of relevance for feature j, locally at $\mathbf{q}$ (and therefore at $\mathbf{d}$), as

$$R_j(\mathbf{q}) \equiv \left|\mathbf{e}_j^T \mathbf{n_d}\right| = |n_{\mathbf{d},j}| \tag{3}$$

where $\mathbf{n_d} = (n_{\mathbf{d},1}, \ldots, n_{\mathbf{d},n})^T$.

The measure of relative feature relevance, as a weighting scheme, can then be given by

$$w_j(\mathbf{q}) = \frac{(R_j(\mathbf{q}))^t}{\sum_{i=1}^{n} (R_i(\mathbf{q}))^t} \tag{4}$$

where t is a positive integer, giving rise to polynomial weightings. We propose the following exponential weighting scheme:

$$w_j(\mathbf{q}) = \frac{\exp(AR_j(\mathbf{q}))}{\sum_{i=1}^{n} \exp(AR_i(\mathbf{q}))} \tag{5}$$

where A is a parameter that can be chosen to maximize (minimize) the influence of R_j on w_j. When $A = 0$ we have $w_j = 1/n$, thereby ignoring any difference between the R_j's. On the other hand, when A is large a change in R_j will be exponentially reflected in w_j. The exponential weighting is more sensitive to changes in local feature relevance (3), and in general gives rise to better performance improvement. In fact, the exponential weighting scheme conveys stability to the method by preventing neighborhoods from extending infinitely in any direction. This is achieved by avoiding zero weights, which is instead allowed by the polynomial weightings.

Thus, (5) can be used as weights associated with features for weighted distance computation

$$D(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} w_i (x_i - y_i)^2}. \tag{6}$$

These weights enable the neighborhood to elongate less important feature dimensions, and, at the same time, to constrict the most influential ones. Note that the technique is *query-based* because weightings depend on the query [1], [4].

One may be tempted to use the weights $w_j(\mathbf{q})$ directly in the SVM classification, by applying the weighted distance measure (6) in (1). By doing so, we would compute the weighted distances of the query point $\mathbf{q}$ from all support vectors, and therefore we would employ the weights $w_j(\mathbf{q})$ for global distance computation over the whole input space. On the other hand, our weighting schemes $w_j(\mathbf{q})$ (4) and (5) are based on the local (to $\mathbf{q}$) orientation of the decision boundary, and therefore meaningful for local distance computation.

The weights $w_j(\mathbf{q})$, in fact, carry information regarding how the shape of a neighborhood should be constricted or enlongated locally at $\mathbf{q}$: we desire to contract the neighborhood along directions aligned with the gradient direction, and dilate it along dimensions that are orthogonal to the gradient direction. Accordingly, a locally adaptive nearest neighbor technique allows us to take into consideration only the closest neighbors (according to the learned weighted metric) in the classification process.

V. Large Margin Nearest Neighbor Classification

We desire that the parameter A in the exponential weighting scheme (5) increases as the distance of $\mathbf{q}$ from the boundary decreases. By using the knowledge that support vectors are mostly located around the boundary surface, we can estimate how close

Input: Decision boundary $f(\mathbf{x}) = 0$ produced by an SVM; query point $\mathbf{q}$ and parameter K.

1. Compute the closest point $\mathbf{d}$ to $\mathbf{q}$ on the boundary (2);
2. Compute the gradient vector $\mathbf{n_d} = \nabla_{\mathbf{d}} f$;
3. Set feature relevance values $R_j(\mathbf{q}) = |n_{\mathbf{d},j}|$ for $j = 1, \ldots, n$;
4. Estimate the distance of $\mathbf{q}$ from the boundary as: $B_{\mathbf{q}} = \min_{\mathbf{s}_i} \|\mathbf{q} - \mathbf{s}_i\|$;
5. Set $A = D - B_{\mathbf{q}}$, where D is defined as in equation (9);
6. Set $\mathbf{w}$ according to (4) or (5);
7. Use the resulting $\mathbf{w}$ for K-nearest neighbor classification at the query point $\mathbf{q}$.

Fig. 2. LAMANNA algorithm.

a query point $\mathbf{q}$ is to the boundary by computing its distance from the closest nonbounded support vector

$$B_{\mathbf{q}} = \min_{\mathbf{s}_i} \|\mathbf{q} - \mathbf{s}_i\| \tag{7}$$

where the minimum is taken over the nonbounded $(0 < \alpha_i < C)$ support vectors $\mathbf{s}_i$. Following the same principle described in [3] the spatial resolution around the boundary is increased by enlarging volume elements locally in neighborhoods of support vectors.

Then, we can achieve our goal by setting

$$A = D - B_{\mathbf{q}} \tag{8}$$

where D is a constant ("meta") parameter input to the algorithm. In our experiments we set D equal to the approximated average distance between the training points $\mathbf{x}_k$ and the boundary

$$D = \frac{1}{l} \sum_{\mathbf{x}_k} \left\{ \min_{\mathbf{s}_i} \|\mathbf{x}_k - \mathbf{s}_i\| \right\}. \tag{9}$$

By doing so the value of A nicely adapts to each query point according to its location with respect to the boundary. The closer $\mathbf{q}$ is to the decision boundary, the greater impact R_j will have on distance computation.

We observe that this principled technique for setting the parameters of our method takes advantage of the sparse representation of the solution provided by SVMs. In fact, for each query point $\mathbf{q}$, in order to compute $B_{\mathbf{q}}$ we only need to consider the support vectors, whose number is typically small compared to the total number of training examples. Furthermore, D can be computed offline and used in subsequent online classification.

The resulting locally flexible metric nearest classification algorithm based on SVMs is summarized in Fig. 2. We call our algorithm large margin nearest neighbor algorithm (LAMANNA) to highlight the fact that the algorithm operates in a space with enlarged margin, as we shall see in Section VI. The algorithm has only one adjustable tuning parameter, namely the number K of neighbors in the final nearest neighbor rule. This parameter is common to all nearest neighbor classification techniques.

VI. Weighting Features Increases the Margin

We define the input space margin as the minimal distance from the training points to the classification boundary in the input space [2]. More specifically, let $\mathbf{s} \in \Re^n$ be a sample point, and $\mathbf{d}$ (defined in (2)) the (nearest) foot of the perpendicular on the separating surface $f(\mathbf{x}) = 0$ from $\mathbf{s}$ (see Fig. 3). We define the input space margin as

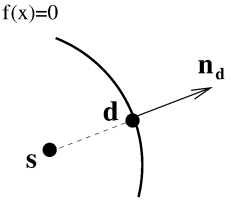

Fig. 3. Perpendicular distance and gradient vector.

$$M = \min_{\mathbf{s}} D(\mathbf{s}, \mathbf{d}) = \min_{\mathbf{s}} \sqrt{\frac{1}{n} \sum_{i=1}^{n} (s_i - d_i)^2} \tag{10}$$

where $\mathbf{s}$ is in the training set, and equal weights are assigned to the feature dimensions. In the following, we show formally that our weighting schemes increase the margin in the space transformed by the weights.

Consider the gradient vector $\mathbf{n_d} = \nabla_{\mathbf{d}} f = ((\partial/\partial x_1) f_{\mathbf{d}}, \ldots, (\partial/\partial x_n) f_{\mathbf{d}})$ computed with respect to $\mathbf{x}$ at point $\mathbf{d}$. Our local measure of relevance for feature j is then given by

$$R_j(\mathbf{s}) = \left| \mathbf{e}_j^T n_{\mathbf{d}} \right| = |n_{\mathbf{d},j}|$$

and $w_j(\mathbf{s})$ is defined as in (4) or (5), with $\sum_{j=1}^{n} w_j(\mathbf{s}) = 1$.

Let

$$D_{\mathbf{w}}^2(\mathbf{s}, \mathbf{d}) = \sum_{i=1}^{n} w_i(\mathbf{s})(s_i - d_i)^2 \tag{11}$$

be the squared weighted Euclidean distance between $\mathbf{s}$ and $\mathbf{d}$. We first prove the following main result.

Theorem 1: Let $\mathbf{s} \in \Re^n$ be a sample point and $\mathbf{d} \in \Re^n$ the nearest foot of the perpendicular on the separating surface $f(\mathbf{x}) = 0$. Define $D^2(\mathbf{s}, \mathbf{d}) = (1/n) \sum_{i=1}^{n} (s_i - d_i)^2$ and $D_{\mathbf{w}}^2(\mathbf{s}, \mathbf{d}) = \sum_{i=1}^{n} w_i(\mathbf{s})(s_i - d_i)^2$, where $w_i(\mathbf{q})$ are the weights computed according to (4) or (5). Then

$$D^2(\mathbf{s}, \mathbf{d}) \leq D_{\mathbf{w}}^2(\mathbf{s}, \mathbf{d}).$$

Proof: Let $\mathbf{s} = (s_1, s_2, \ldots, s_n)^T$, $\mathbf{d} = (d_1, d_2, \ldots, d_n)^T$, and $\mathbf{w}(\mathbf{s}) = (w_1, w_2, \ldots, w_n)^T$, where we drop the dependency of the w_is on $\mathbf{s}$ for clarity.

Since $(\mathbf{s} - \mathbf{d})$ is along the gradient direction $\mathbf{n_d}$, or opposite of it, we have $(\mathbf{s} - \mathbf{d}) = \beta \mathbf{n_d}$, for some scalar β. It follows $(|s_i - d_i|/R_i) = |\beta| \; \forall \, i$, and therefore:

$$\frac{|s_1 - d_1|}{R_1} = \frac{|s_2 - d_2|}{R_2} = \cdots = \frac{|s_n - d_n|}{R_n}.$$

If $w_i = (R_i^t / \sum_{j=1}^{n} R_j^t)$, for t positive integer (polynomial weighting)

$$|s_i - d_i| = \frac{R_1^{t-1} |s_1 - d_1|}{w_1} \frac{w_i}{R_i^{t-1}} = \left(\frac{R_1^{t-1} |s_1 - d_1|}{w_1 \sum_j R_j^t} \right) R_i, \quad \forall \, i = 1, \ldots, n \tag{12}$$

where, without loss of generality, we assume that $w_1 \neq 0$, since a positive w_i must exist, given that $\sum_{i=1}^{n} w_i = 1$ and $w_i \geq 0 \, \forall i$.

If $w_i = \left(e^{AR_i} / \sum_{j=1}^{n} e^{AR_j} \right)$ (exponential weighting)

$$|s_i - d_i| = \left(\frac{e^{AR_1}|s_1 - d_1|}{R_1 w_1} \right) \frac{R_i}{e^{AR_i}} w_i = \left(\frac{e^{AR_1}|s_1 - d_1|}{R_1 w_1 \sum_j e^{AR_j}} \right) R_i, \quad \forall \, i = 1, \ldots, n \tag{13}$$

where $A \geq 0$, and again without loss of generality we assume $R_1 \neq 0$, since a positive R_i must exist by definition (3).

For polynomial weighting, by (12), we obtain

$$D^2(\mathbf{s}, \mathbf{d}) = \frac{1}{n} a \sum_{i=1}^{n} R_i^2 \tag{14}$$

$$D_{\mathbf{w}}^2(\mathbf{s}, \mathbf{d}) = a \sum_{i=1}^{n} R_i^2 w_i \tag{15}$$

where $a = \left((R_1^{t-1}|s_1 - d_1|) / (w_1 \sum_j R_j^t) \right)^2$.

For exponential weighting, by (13), we obtain

$$D^2(\mathbf{s}, \mathbf{d}) = \frac{1}{n} b \sum_{i=1}^{n} R_i^2 \tag{16}$$

$$D_{\mathbf{w}}^2(\mathbf{s}, \mathbf{d}) = b \sum_{i=1}^{n} R_i^2 w_i \tag{17}$$

where $b = \left((e^{AR_1}|s_1 - d_1|) / (R_1 w_1 \sum_j e^{AR_j}) \right)^2$.

In both cases, to show that $D^2(\mathbf{s}, \mathbf{d}) \leq D_{\mathbf{w}}^2(\mathbf{s}, \mathbf{d})$, we need to show

$$\sum_{i=1}^{n} R_i^2 \leq n \sum_{i=1}^{n} R_i^2 w_i. \tag{18}$$

Consider *Chebychev's inequality* [13]. If $x_1 \geq \cdots x_n \geq 0$ and $y_1 \geq \cdots y_n \geq 0$ then

$$\left(\frac{x_1 + \cdots + x_n}{n} \right) \left(\frac{y_1 + \cdots + y_n}{n} \right) \leq \frac{x_1 y_1 + \cdots + x_n y_n}{n} \tag{19}$$

with equality if and only if all x_i or all y_i are equal.

Let $x_i = R_i^2$ and $y_i = w_i, \forall \, i$. Suppose, without loss of generality, that the R_is are in nondecreasing order: $R_1 \geq R_2 \geq \cdots \geq R_n \geq 0$. Then $R_1^2 \geq \cdots \geq R_n^2 \geq 0$, and $e^{AR_1} \geq \cdots \geq e^{AR_n} \geq 0$. Therefore, $w_1 \geq \cdots \geq w_n \geq 0$ for all weighting schemes. Chebychev's inequality gives

$$\frac{1}{n^2} \sum_{i=1}^{n} R_i^2 \leq \frac{1}{n} \sum_{i=1}^{n} R_i^2 w_i. \tag{20}$$

Multiplying both sides of (20) by n^2, we obtain (18). This concludes the proof. ■

Now we can show that our weighting schemes increase the margin in the transformed space. Let

$$\mathbf{s}^* = \arg\min_{\mathbf{s}} D(\mathbf{s}, \mathbf{d}).$$

Then

$$M = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (s_i^* - d_i^*)^2}.$$

We have the following result.

Corollary 2: $M \leq D_{\mathbf{w}}(\mathbf{s}^*, \mathbf{d}^*)$

Proof: By Theorem 1, $D^2(\mathbf{s}, \mathbf{d}) \leq D_{\mathbf{w}}^2(\mathbf{s}, \mathbf{d})$. Hence

$$D^2(\mathbf{s}^*, \mathbf{d}^*) \leq D_{\mathbf{w}}^2(\mathbf{s}^*, \mathbf{d}^*)$$

and therefore

$$M = D(\mathbf{s}^*, \mathbf{d}^*) \leq D_{\mathbf{w}}(\mathbf{s}^*, \mathbf{d}^*).$$

■

Theorem 1 shows that $D^2(\mathbf{s}, \mathbf{d}) \leq D_{\mathbf{w}}^2(\mathbf{s}, \mathbf{d})$. Now we show that the equality holds only when $w_i = (1/n) \, \forall \, i$. This result guarantees an effective increase of the margin in the transformed space whenever differential weights are credited to features (according to our weighting schemes), as stated in Corollary 3.

Corollary 3: $D^2(\mathbf{s}, \mathbf{d}) = D_{\mathbf{w}}^2(\mathbf{s}, \mathbf{d})$ if and only if $w_i = (1/n) \, \forall \, i$.

Proof: If $w_i = (1/n) \, \forall \, i$, then the equality $D^2(\mathbf{s}, \mathbf{d}) = D_{\mathbf{w}}^2(\mathbf{s}, \mathbf{d})$ is trivially satisfied.

Suppose now

$$D^2(\mathbf{s}, \mathbf{d}) = D_{\mathbf{w}}^2(\mathbf{s}, \mathbf{d}). \tag{21}$$

Theorem 1 shows that, when equality (21) holds, then

$$\sum_{i=1}^{n} R_i^2 = n \sum_{i=1}^{n} R_i^2 w_i. \tag{22}$$

This means that the equality in (19) is satisfied, with $x_i = R_i^2$ and $y_i = w_i, \forall \, i$. Then, for the equality conditions of (19) [13], all R_is or all w_is are equal. In either case, $w_i = (1/n) \, \forall \, i$. ■

And finally we have

Corollary 4: $M = D_{\mathbf{w}}(\mathbf{s}^*, \mathbf{d}^*)$ if and only if $w_i = (1/n) \, \forall \, i$.

Proof: This is a direct consequence of Corollary 1 and Corollary 2. ■

VII. Experimental Results

In the following we compare several classification methods using both simulated and real data. All problems considered here involve two classes, because SVMs are well suited for such cases. Multiclass classification problems can also be solved using binary classifiers. A good discussion on the subject can be found in [23]. We compare the following classification approaches.

- LAMANNA algorithm described in Fig. 2. SVM$^{\text{light}}$ [17] with radial basis kernels is used to build the SVM classifier.
- Radial basis function (RBF)-SVM classifier with radial basis kernels. We used SVM$^{\text{light}}$ [17], and set the value of γ in $K(\mathbf{x}_i, \mathbf{x}) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}\|^2}$ equal to the optimal one determined via cross validation. Also the value of C for the soft-margin classifier is optimized via cross validation. The output of this classifier is the input of LAMANNA.
- ADAMENN-adaptive metric nearest neighbor technique [10]. It uses the *Chi-squared* distance in order to estimate to which extent each dimension can be relied on to predict class posterior probabilities.
- Machete [12]. It is a recursive partitioning procedure, in which the input variable used for splitting at each step is the one that maximizes the estimated local relevance. Such relevance is measured in terms of the improvement in squared prediction error each feature is capable to provide.
- Scythe [12]. It is a generalization of the Machete algorithm, in which the input variables influence each split in proportion to their estimated local relevance, rather than applying the winner-take-all strategy of the Machete.
- DANN-discriminant adaptive nearest neighbor classification [14]. It is an adaptive nearest neighbor classification method based on linear discriminant analysis. It computes a distance metric as a product of properly weighted within and between sum of squares matrices.
- Simple K-NN method using the Euclidean distance measure.
- C4.5 decision tree method [22].

In all the experiments, the features are first normalized over the training data to have zero mean and unit variance, and the test data features are normalized using the corresponding training mean and variance. Procedural parameters (including K) for each method were determined empirically through cross validation over training data.

To avoid solving the nonlinear program $\mathbf{d} = \arg\min_{\mathbf{p}} \|\mathbf{q} - \mathbf{p}\|$, subject to the constraint $f(\mathbf{p}) = 0$, in the implementation of the LAMANNA algorithm we can estimate the closest point to the query on the boundary as follows. We move from the query point along the input axes (in both directions) at distances proportional to a given small step (whose initial value can be arbitrarily small, and doubled at each iteration till the boundary is crossed). We stop as soon as the boundary is crossed along one of the input axes, say axis i, i.e., when a point $\mathbf{p}_i$ is reached that satisfies the condition $\text{sign}(f(\mathbf{q})) \times \text{sign}(f(\mathbf{p}_i)) = -1$. Given $\mathbf{p}_i$, we can get arbitrarily close to the boundary by moving at (arbitrarily) small steps along the segment that joins $\mathbf{p}_i$ to $\mathbf{q}$. Let us denote with $\mathbf{d}_i$ the intercepted point on the boundary along direction i. We then approximate $\mathbf{n}_{\mathbf{d}}$ with the gradient vector $\mathbf{n}_{\mathbf{d}_i} = \nabla_{\mathbf{d}_i} f$, computed at $\mathbf{d}_i$. We have experimented with both the exact solution of the nonlinear program and the approximated one. We have observed no significant difference in performance. Results reported here are based on the approximated gradient computation.

A. Experiments on Simulated Data

For all simulated data, 10 independent training samples of size 200 were generated. For each of these, an additional independent test sample consisting of 200 observations was generated. These test data were classified by each competing method using the respective training data set. Error rates computed over all 2,000 such classifications are reported in Table I.

TABLE I
AVERAGE CLASSIFICATION ERROR RATES FOR SIMULATED DATA

	MultiGauss	NoisyGauss
LaMaNNa	**3.3** ± 0.17	**3.4** ±0.18
RBF-SVM	**3.3** ± 0.01	5.3 ±0.01
ADAMENN	3.4 ± 0.01	4.1 ±0.02
Machete	3.4 ±0.01	4.3 ±0.01
Scythe	3.4 ±0.01	4.8 ±0.01
DANN	3.7 ±0.01	4.7 ±0.01
K-NN	**3.3** ±0.01	7.0 ±0.01
C4.5	5.0 ±1.50	5.1 ±1.60

1) Problems: **Multi-Gaussians.** The data set consists of $n = 2$ input features, $l = 200$ training data, and $J = 2$ classes. Each class contains two spherical bivariate normal subclasses, having standard deviation 1. The mean vectors for one class are $(-3/4, -3)$ and (3/4,3); whereas for the other class are $(3, -3)$ and $(-3, 3)$. For each class, data are evenly drawn from each of the two normal subclasses. The first column of Table I shows the results for this problem.

Noisy-Gaussians. The data set consists of $n = 6$ input features, $l = 200$ training data, and $J = 2$ classes. The data for this problem are generated as in the previous example, but augmented with four predictors having independent standard Gaussian distributions. They serve as noise. For each class, data are evenly drawn from each of the two normal subclasses. Results are shown in the second column of Table I.

Mixture and Multiplicative Noise. The data are distributed as in multi-Gaussians, with additional noisy predictors. For the mixture case, the noisy variables are distributed according to a mixture of Gaussians: the sum of a standard normal and a Gaussian with mean and standard deviation randomly chosen in the intervals $[-7, 7]$ and $[1, 5]$, respectively. For the multiplicative noise, the noisy variables are the product of two Gaussians, with the same settings for means and standard deviations as for the mixture case. These data were generated to compare RBF-SVM and LAMANNA in presence of noisy features with more complex distributions.

2) Results: Table I shows that all methods have similar performances for the multi-Gaussians problem, with C4.5 being the worst performer. When the noisy predictors are added to the problem (noisy Gaussians), we observe different levels of deterioration in performance among the eight methods. LAMANNA shows the most robust behavior in presence of noise. K-NN is instead the worst performer. We also observe that C4.5 has similar error rates in both cases; we noticed, in fact, that for the majority of the 10 independent trials we run it uses only the first two input features to build the decision tree. In Fig. 4, we plot the performances of LAMANNA and RBF-SVM as a function of an increasing number of noisy features (for the same multi-Gaussians problem with Gaussian, mixture, and multiplicative noisy features). The standard deviations for RBF-SVM (in order of increasing number of noisy features) are as follows.

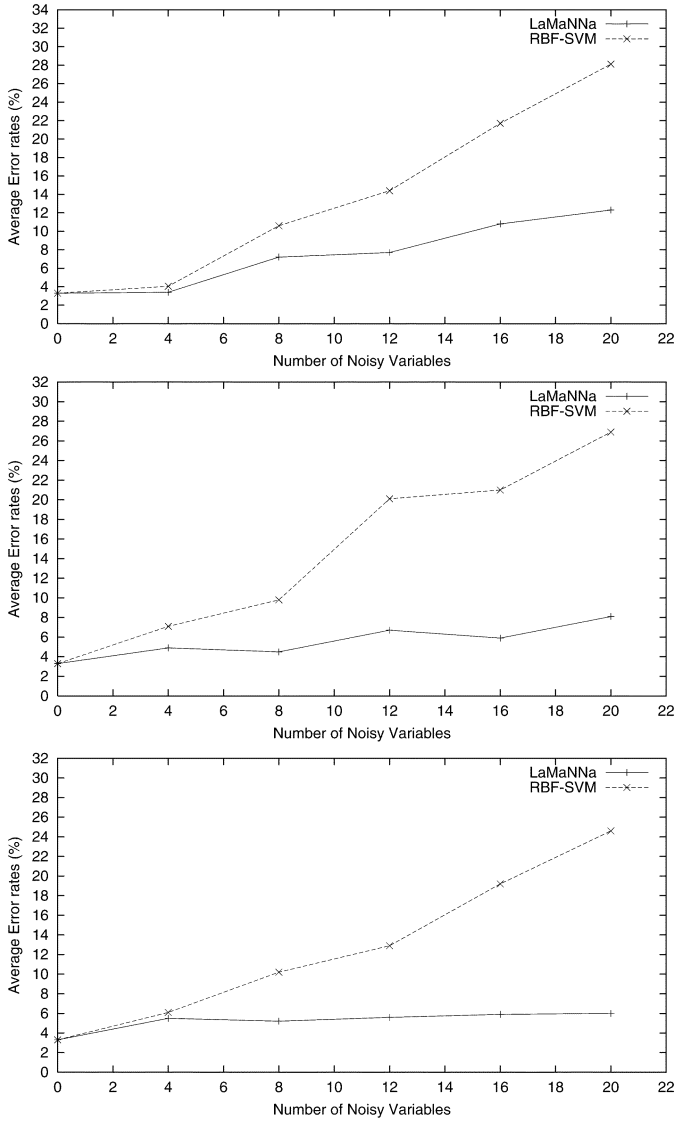

Fig. 4. Average error rates of LAMANNA and RBF-SVM as a function of the number of noisy predictors. The noisy predictors have: standard Gaussian distributions (top); mixture of Gaussians distributions (middle); multiplicative Gaussian distributions (bottom).

Gaussian noise: 0.01, 0.03, 0.03, 0.03 and 0.03. Mixture noise: 0.02, 0.03, 0.04, 0.04, and 0.02. Multiplicative noise: 0.02, 0.03, 0.02, 0.03, and 0.03. The standard deviations for LAMANNA are as follows. Guassian noise: 0.18, 0.2, 0.3, 0.3, and 0.3. Mixture noise: 0.2, 0.2, 0.3, 0.2, and 0.3. Multiplicative noise: 0.2, 0.2, 0.2, 0.2, and 0.2. In all three cases, the LAMANNA technique shows a considerable improvement over RBF-SVM as the amount of noise increases. The rather flat performance curve of LAMANNA demonstrates that our technique successfully filters out the noisy features.

We have also computed the normalized eigenvalues of the principal components for the three 22-dimensional (20 noisy features + two actual features) data sets we have considered. The eigenvalues are shown in Fig. 5. The plot shows that, in the presence of Gaussian noise, the eigenvalues drop drastically, and most information is captured by the first two principal components. In the other two cases, however, such a separation is not

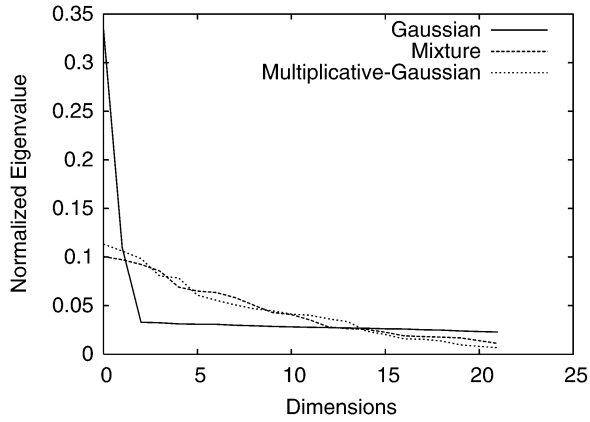

Fig. 5. Normalized eigenvalues of the principal components for the three 22-dimensional data sets augmented with different kind of noisy features.

evident. The eigenvalues decrease gradually. The elimination of the noisy features in these cases is not a trivial task.

B. Experiments on Real Data

While simulated data are informative for comparison studies, it is highly likely that artificially constructed examples will not correspond to situations that are likely to occur in practice. Thus, in this section, we examine the performance of the competing classification methods using real world data. One of the advantages of real data is that they are generated without any knowledge of the classification procedures that it will be used to test.

In our experiments we used eight different real data sets. They are all taken from the UCI Machine Learning Repository. For the Iris, Sonar, Liver, and Vote data we perform leave-one-out cross validation to measure performance, since the number of available data is limited for these data sets. For the Breast, OQ-letter and Pima data we randomly generated five independent training sets of size 200. For each of these, an additional independent test sample consisting of 200 observations was generated. For the noisy-Pima data we performed 10 two-fold cross validation, with 60% of the data for training and 40% for testing. Table II shows the cross-validated error rates for the eight methods under consideration on the eight real data.

1) Problems:

1) **Iris data**. This data set consists of $n = 4$ measurements made on each of $l = 100$ iris plants of $J = 2$ species. The two species are iris versicolor and iris virginica. The problem is to classify each test point to its correct species based on the four measurements. The results on this data set are shown in the first column of Table II.
2) **Sonar data**. This data set consists of $n = 60$ frequency measurements made on each of $l = 208$ data of $J = 2$ classes ("mines" and "rocks"). The problem is to classify each test point in the 60-dimensional feature space to its correct class. The results on this data set are shown in the second column of Table II.
3) **Liver data**. This example has $n = 6$ attribute values regarding blood test results and drinking habits. There are $J = 2$ classes, establishing the liver disorder condition,

TABLE II
AVERAGE CLASSIFICATION ERROR RATES FOR REAL DATA

	Iris	Sonar	Liver	Vote	Breast	OQ	Pima	Noisy-Pima
LaMaNNa	4.0	11.0	28.1	**2.6**	3.0	3.5	**19.3**	**24.7**
RBF-SVM	4.0	12.0	**26.1**	3.0	3.1	3.4	21.3	25.1
ADAMENN	**3.0**	9.1	30.7	3.0	3.2	**3.1**	20.4	31.5
Machete	5.0	21.2	27.5	3.4	3.5	7.4	20.4	28.2
Scythe	4.0	16.3	27.5	3.4	2.7	5.0	20.0	29.4
DANN	6.0	**7.7**	30.1	3.0	**2.2**	4.0	22.2	28.1
K-NN	6.0	12.5	32.5	7.8	2.7	5.4	24.2	30.9
C4.5	8.0	23.1	38.3	3.4	4.1	9.2	23.8	32.3

and $l = 345$ samples. The results on this data set are shown in the third column of Table II.

4) **Vote data**. This data set includes votes for each of the U.S. House of Representatives Congressmen on the 16 key votes identified by the Congressional Quarterly Almanac. The data set consists of $l = 232$ instances after removing missing values, and $J = 2$ classes (democrat and republican). The instances are represented by $n = 16$ boolean valued features. The average leave-one-out cross validation error rates are shown in the fourth column of Table II.
5) **Wisconsin breast cancer data**. The data set consists of $n = 9$ medical input features that are used to make a binary decision on the medical condition: determining whether the cancer is malignant or benign. The data set contains 683 examples after removing missing values. Average error rates for this problem are shown in the fifth column of Table II. The standard deviations are: 0.2, 0.2, 0.2, 0.2, 0.2, 0.9, 0.9, and 0.9, respectively.
6) **OQ data**. This data set consists of $n = 16$ numerical attributes and $J = 2$ classes. The objective is to identify black-and-white rectangular pixel displays as one of the two capital letters "O" and "Q" in the English alphabet. There are $l = 1536$ instances in this data set. The character images were based on 20 different fonts, and each letter within these 20 fonts was randomly distorted to produce a file of 20 000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15. The average error rates over five independent runs are shown in the sixth column of Table II. The standard deviations are: 0.2, 0.2, 0.2, 0.3, 0.2, 1.1, 1.5, and 2.1, respectively.
7) **Pima Indians Diabete data**. This data set consists of $n = 8$ numerical medical attributes and $J = 2$ classes (tested positive or negative for diabetes). There are $l = 768$ instances. Average error rates over five independent runs are shown in the seventh column of Table II. The standard deviations are: 0.4, 0.4, 0.4, 0.4, 0.4, 2.4, 2.1, and 0.7, respectively.
8) **Noisy Pima Indians Diabete data**. This data set consists of $n = 60$ numerical attributes and $J = 2$ classes. It was obtained by adding 52 independent mixture of Gaussians to the original eight dimensions of the Pima data. These additional 52 dimensions serve as noise. Mean and standard deviations of the Gaussians are set as in the mixture noise for simulated data. Average error rates over 10 independent runs are shown in the last column of Table II. The standard deviations are: 0.4, 0.4, 0.5, 2.1, 2.7, 2.4, 3.1, and 1.6, respectively.

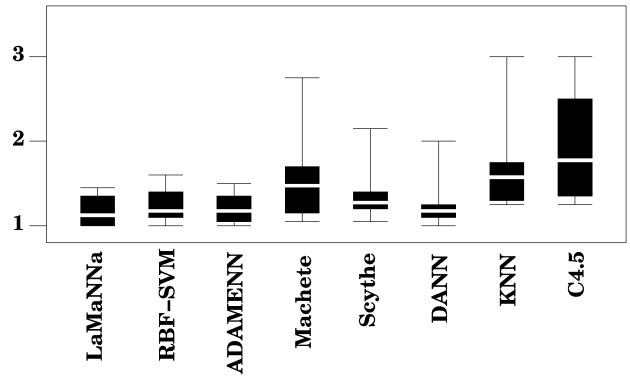

Fig. 6. Performance distributions for real data.

2) Results: Table II shows that LAMANNA achieves the best performance in 3/7 of the real data sets; in one case it shows the second best performance, and in the remaining four its error rate is still quite close to the best one.

It seems natural to quantify this notion of robustness; that is, how well a particular method m performs on average across the problems taken into consideration. Following [12], we capture robustness by computing the ratio b_m of the error rate e_m of method m and the smallest error rate over all methods being compared in a particular example:

$$b_m = \frac{e_m}{\min\limits_{1 \le k \le 8} e_k}.$$

Thus, the best method m^* for that example has $b_{m^*} = 1$, and all other methods have larger values $b_m \ge 1$, for $m \ne m^*$. The larger the value of b_m, the worse the performance of the mth method is in relation to the best one for that example, among the methods being compared. The distribution of the b_m values for each method m over all the examples, therefore, seems to be a good indicator concerning its robustness. For example, if a particular method has an error rate close to the best in every problem, its b_m values should be densely distributed around the value 1. Any method whose b value distribution deviates from this ideal distribution reflect its lack of robustness.

Fig. 6 plots the distribution of b_m for each method over the eight real data sets. The dark area represents the lower and upper quartiles of the distribution that are separated by the median. The outer vertical lines show the entire range of values for the distribution. One of the outer vertical lines for the LAMANNA method is not visible because it coincides with the limit of the lower quartile. The spread of the error distribution for LAMANNA is narrow and close to one. The spread for ADAMENN has a similar behavior, with the outer bar reaching a slightly higher value. The results clearly demonstrate that LAMANNA (and ADAMENN) obtained the most robust performance over the data sets.

The poor performance of the Machete and C4.5 methods might be due to the greedy strategy they employ. Such recursive peeling strategy removes at each step a subset of data points

permanently from further consideration. As a result, changes in an early split, due to any variability in parameter estimates, can have a significant impact on later splits, thereby producing different terminal regions. This makes predictions highly sensitive to the sampling fluctuations associated with the random nature of the process that produces the traning data, thus, leading to high-variance predictions. The Scythe algorithm, by relaxing the winner-take-all splitting strategy of the Machete algorithm, mitigates the greedy nature of the approach, and thereby achieves better performance.

In [14], the authors show that the metric employed by the DANN algorithm approximates the weighted Chi-squared distance, given that class densities are Gaussian and have the same covariance matrix. As a consequence, we may expect a degradation in performance when the data do not follow Gaussian distributions and are corrupted by noise, which is likely the case in real scenarios like the ones tested here.

We observe that LAMANNA avoids expensive cross validation by using a principled technique for setting the procedural parameters. Our approach to efficient and automatic settings of parameters leverages the sparse solution provided by SVMs. As a result, our algorithm has only one adjudtable tuning parameter, the number K of neighbors in the final nearest neighbor rule. This parameter is common to all nearest neighbor techniques. On the other hand, the competing techniques have multiple parameters whose values must be determined through cross validation: ADAMENN has *six* parameters; Machete/Scythe each has *four* parameters; DANN has *two* parameters. Furthermore, LAMANNA speeds up the online computation since it applies the nearest neighbor rule only once, whereas ADAMENN, for example, applies it to each point within a region centered on the query. In Section VIII, we provide a complexity analysis of online computation for each of the locally adaptive methods used in our experiments. The analysis shows that our algorithm has the lowest online computational complexity. We also observe that the construction of SVMs for LAMANNA is carried out offline only once. Also, there exist algorithmic and computational results that show SVM learning can be made practical for large-scale problems [17].

The LAMANNA technique offers improvement in accuracy over the RBF-SVM algorithm alone, for both the (noisy) simulated and real data sets. The reason for such performance gain has to do with the increase in margin in the transformed space by our local weighting scheme, as shown in Section VI.

Assigning large weights, locally in the neighborhoods of the support vectors, to input features close to the gradient direction corresponds to increasing the spatial resolution along those directions, thereby improving the separability of classes.

VIII. Complexity Analysis

Here we provide the complexity analysis of online computation for each of the locally adaptive competing methods. In this analysis, we assume that the procedural parameters for each method has been determined.

For a test point, DANN must first compute $l/5$ nearest neighbors that are used to estimate the DANN metric, which requires $O(l \log l)$ operations. For n dimensions, the metric requires $O((l/5)n^2)$ operations. Therefore, the overall complexity of DANN is $O(l \log l + ln^2)$.

Given a test point, ADAMENN first computes $l/5$ nearest neighbors to estimate feature relevance, which requires $O(l \log l)$ operations. Each of the $l/5$ nearest neighbors is then used to estimate conditional predictions along each dimensions, which requires $O((l/5)l \log l + n(l/5))$ operations. Thus, the computational complexity of ADAMENN is bounded by $O(nl^2 \log l + nl)$.

Machete and Scythe, on the other hand, involve a "peeling" process. Each peel removes $(1 - \alpha)l$ points from further consideration, where $0 < \alpha < 1$. The total number of peels is $\log_{1/\alpha}(l/K)$. At each peel t, feature relevance along each dimension is estimated, which requires $O(n\alpha^t l \log \alpha^t l)$. This is performed for $\log_{1/\alpha}(l/K)$ times. Thus, the complexity of both Machete and Scythe is $O\left((\log_{1/\alpha}(l/K))\left(n\left(\alpha^{\log_{1/\alpha}(l/K)}l\right)\log\left(\alpha^{\log_{1/\alpha}(l/K)}l\right)\right)\right)$, which is bounded by $O\left((l/K)\left(n(\alpha^{(l/K)}l)\log(\alpha^{(l/K)}l)\right)\right)$. Since the peeling process demands "patience," $\alpha \approx 1$. Therefore, Machete and Scythe have a time bound $O(n(l^2/K)\log l)$.

Given a test point, our algorithm, LAMANNA, computes a gradient vector, which requires $O(l_s n)$ operations, where l_s is the number of support vectors computed by SVMs (in general $l_s \ll l$). Then, it performs $O(l \log l)$ operations for nearest neighbor computation. Thus, the total number of operations is bounded by $O(l \log l + l_s n)$

The previous analysis shows that LAMANNA has the lowest online computational complexity. In addition, our algorithm is prone to further computational online speed-up if required by a specific application. In fact, one can compute offline a map of the gradients and, thus, of the weight vectors in different regions of the input space. When a query is given, the closest gradient vector can be identified and, thus, the corresponding local weight vector will be readily available.

IX. Related Work

In [3], Amari and Wu improve SVM classifiers by modifying kernel functions. A primary kernel is first used to obtain support vectors. The kernel is then modified in a data dependent way by using the information of the support vectors: the factor that drives the transformation has larger values at positions close to support vectors. The modified kernel enlarges the spatial resolution around the boundary so that the separability of classes is increased.

The resulting transformation depends on the distance of data points from the support vectors, and it is therefore a local transformation, but is independent of the boundary's orientation in input space. Likewise, our transformation metric depends on the distance of the query point from the support vectors; this dependence is driven by the A factor in the exponential weighting scheme (5), which is defined in (8). Moreover, since we weigh features, our metric is directional, and depends on the orientation of local boundaries in input space. This dependence is driven by the measure of feature relevance (3), which has the effect of increasing the spatial resolution along discriminant directions around the boundary.

X. Conclusion and Future Work

We have described a large margin nearest neighbor classification method and demonstrated its efficacy through experimental results. The proposed technique offers accuracy improvements over the SVM alone, supported by the theoretical result of margin increase. In addition, our technique has the potential of scaling up to large data sets. It speeds up, in fact, the online classification process by computing offline the information relevant to define local weights. It also avoids expensive cross validation by using a principled technique for setting procedural parameters.

A considerable amount of work has been done for efficient discovery of nearest neighbors. Unfortunately, all methods suffer from the curse of dimensionality, and therefore they become less effective in high dimensions. Our scheme could also be used to improve the effectiveness of nearest neighbor search in high dimensions by virtue of the local dimensionality reduction resulting from the feature weights. We intend to further explore this direction of research in our future work.

References

[1] D. Aha, "Lazy learning," *Artif. Intell. Rev.*, vol. 11, pp. 1–5, 1997.

[2] S. Akaho, "SVM maximizing margin in the input space," in *Proc. 6th Kernel Machines Workshop Learning Kernels NIPS*, 2002, pp. 1–19.

[3] S. Amari and S. Wu, "Improving support vector machine classifiers by modifying kernel functions," *Neural Netw.*, vol. 12, pp. 783–789, 1999.

[4] C. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artif. Intell. Rev.*, vol. 11, pp. 11–73, 1997.

[5] R. E. Bellman, *Adaptive Control Processes*. Princeton, NJ: Princeton Univ. Press, 1961.

[6] M. Brown, W. Grundy, D. Lin, N. Cristianini, C. Sugnet, T. Furey, M. Ares, and D. Haussler, "Knowledge-based analysis of microarray gene expressions data by using support vector machines," *Proc. Nat. Acad. Sci.*, vol. 97, no. 1, pp. 262–267, 2000.

[7] W. S. Cleveland and S. J. Devlin, "Locally weighted regression: An approach to regression analysis by local fitting," *J. Amer. Statist. Assoc.*, vol. 83, pp. 596–610, 1988.

[8] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, pp. 21–27, 1967.

[9] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge, U.K.: Cambridge Univ. Press, 2000.

[10] C. Domeniconi, J. Peng, and D. Gunopulos, "Locally adaptive metric nearest neighbor classification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 9, pp. 1281–1285, Sep. 2002.

[11] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.

[12] J. H. Friedman, "Flexible Metric Nearest Neighbor Classification," Dept. Statist., Stanford Univ., Stanford, CA, 1994.

[13] G. H. Hardy, J. E. Littlewood, and G. Polya, *Inequalities*. Cambridge, U.K.: Cambridge Univ. Press, 1973.

[14] T. Hastie and R. Tibshirani, "Discriminant adaptive nearest neighbor classification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 18, no. 6, pp. 607–615, Jun. 1996.

[15] T. K. Ho, "Nearest neighbors in random subspaces," *Lecture Notes in Computer Science: Advances in Pattern Recognition*, pp. 640–648, 1998.

[16] T. Joachims, "Text categorization with support vector machines," in *Proc. Eur. Conf. Machine Learning*, 1998, pp. 137–142.

[17] ——, "Making large-scale SVM learning practical," in *Advances in Kernel Methods—Support Vector Learning*, B. Schölkopf, C. Burger, and A. Smola, Eds. Cambridge, MA: MIT Press, 1999.

[18] D. G. Lowe, "Similarity metric learning for a variable-kernel classifier," *Neural Computat.*, vol. 7, no. 1, pp. 72–85, 1995.

[19] G. J. McLachlan, *Discriminant Analysis and Statistical Pattern Recognition*. New York: Wiley, 1992.

[20] E. Osuna, R. Freund, and F. Girosi, "Training support vector machines: An application to face detection," in *Proc. Computer Vision and Pattern Recognition*, 1997, pp. 130–136.

[21] J. Peng, D. Heisterkamp, and H. K. Dai, "LDA/SVM driven nearest neighbor classification," *IEEE Trans. Neural Netw.*, vol. 14, no. 4, pp. 940–942, Jul. 2003.

[22] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, MA: Morgan-Kaufmann, 1993.

[23] R. Rifkin and A. Klautau, "In defense of one-vs-all classification," *J. Mach. Lear. Res.*, vol. 5, pp. 101–141, 2004.

[24] S. Salzberg, "A nearest hyperrectangle learning method," in *Mach. Learn.*, 1991, vol. 6, pp. 251–276.

[25] J. Shawe-Taylor, P. L. Bartlett, R. C. Williamson, and M. Anthony, "Structural risk minimization over data-dependent hierarchies," *IEEE Trans. Inf. Theory*, vol. 44, no. 5, pp. 1926–1940, May 1998.

[26] C. J. Stone, "Nonparametric regression and its applications (with discussion)," *Ann. Statist.*, vol. 5, pp. 595–595, 1977.

[27] V. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer-Verlag, 1995.

[28] ——, *Statistical Learning Theory*. New York: Wiley, 1998.

Carlotta Domeniconi received the Laurea degree in computer science from the University of Milano, Milan, Italy, in 1992, the M.S. degree in information and communication technologies from the International Institute for Advanced Scientific Studies, Salerno, Italy, in 1997, and the Ph.D. degree in computer science from the University of California, Riverside, in 2002.

She is currently an Assistant Professor in the Information and Software Engineering Department, George Mason University, Fairfax, VA. Her research interests include machine learning, pattern recognition, data mining, and feature relevance estimation, with applications in text mining and bioinformatics.

Dr. Domeniconi is a recipient of a 2004 Ralph E. Powe Junior Faculty Enhancement Award.

Dimitrios Gunopulos received the B.S. degree in computer engineering and informatics from the University of Patras, Patras, Greece, in 1990 and the M.A. and Ph.D. degrees from Princeton University, Princeton, NJ, in 1992 and 1995, respectively.

He is currently an Associate Professor in the Department of Computer Science and Engineering, University of California, Riverside. His research is in the areas of data mining, databases, and algorithms. His current interests include efficient indexing techniques for moving points, approximating range queries, similarity searching in sequence databases, local adaptive classification techniques, information retrieval in peer-to-peer systems, and analysis of sensor data. He has held positions at the IBM Almaden Research Center (1996–1998) and at the Max-Planck-Institut for Informatics (1995–1996).

Dr. Gunopulos is currently an Associate Editor for the IEEE Transactions on Knowledge and Data Engineering and on the Editorial Advisory Board of the Elsevier *Information Systems Journal*.

Jing Peng (M'03) received the M.A. degree in computer science from Brandeis University, Waltham, MA, the B.S. degree in computer science from Beijing Institute of Aeronautics and Astronautics, Beijing, China, and the Ph.D. degree in computer science from Northeastern University, Boston, MA, in 1994.

Since 2001, he has been an Assistant Professor of electrical engineering and computer science at Tulane University, New Orleans, LA. He was on the Faculty of Computer Science, Oklahoma State University, Stillwater, OK, from 1999 to 2001. Prior to that, he was a Postdoctoral Fellow in the Center for Research in Intelligent Systems, University of California, Riverside. His research interests include machine learning, classification, image databases, and learning and vision applications. He has served on the program committees of several international conferences.