Efficient Local Flexible Nearest Neighbor Classification

Carlotta Domeniconi, Dimitrios Gunopulos
Dept. of Computer Science, University of California, Riverside, CA 92521
{*carlotta,dg*}*@cs.ucr.edu*

Abstract

The nearest neighbor technique is a simple and appealing method to address 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 local adaptive metric becomes crucial in order to keep class conditional probabilities close to uniform, and therefore to minimize the bias of estimates. We propose a technique that computes a locally flexible metric by means of Support Vector Machines (SVMs). The maximum margin boundary found by the SVM is used to determine the most discriminant direction over the query's neighborhood. Such direction provides a local weighting scheme for input features. We present experimental evidence, together with a formal justification, of classification performance improvement over the SVM algorithm alone and over a variety of adaptive learning schemes, by using both simulated and real data sets. Moreover, the proposed method has the important advantage of superior efficiency over the most competitive technique used in our experiments.

1 Introduction

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

The K nearest neighbor classification method [6, 13, 16, 17, 20, 21] is a simple and appealing approach to this problem: it finds the K nearest neighbors 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 [7, 10] 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.

The nearest neighbor rule becomes less appealing with finite training samples, however. This is due to the curse of dimensionality [4]. 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. Figure 1 illustrates a case in point, where class boundaries are parallel to the coordinate axes. For query **a**, dimension X is more relevant, because a slight move along the X axis may change the class label, while for query **b**, dimension Y is more relevant. For query **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 [11, 12, 9] have been proposed to try to minimize bias in high dimensions by using locally adaptive mechanisms. The "lazy learning" approach used by these methods, while appealing in many ways, requires a considerable amount of on-line computation, which makes it difficult for such techniques to scale up to large data sets. The feature weighting scheme they introduce, in fact, is query based and is applied on-line when the test point is presented to the "lazy learner".

In this paper we propose a locally adaptive metric classification method which, although still founded on a query based weighting mechanism, computes off-line the information relevant to define local weights. Preliminary results of this approach appear in [8]; here we further motivate our technique, and provide a theoretical justification that supports the experimental performance results.

Our technique uses support vector machines (SVMs) as a guidance for the process of defining a local flexible metric. SVMs have been successfully used as a classification tool in a variety of areas [14, 5, 18], 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 SVM's learning algorithm, and therefore SVMs are a natural choice for seeking local discriminant directions between classes.

The solution provided by SVMs allows to determine locations in input space where class conditional probabilities are likely to be not constant, and guides the extraction of local information in such areas. This process produces highly stretched neighborhoods along boundary directions when the query is close to the boundary.

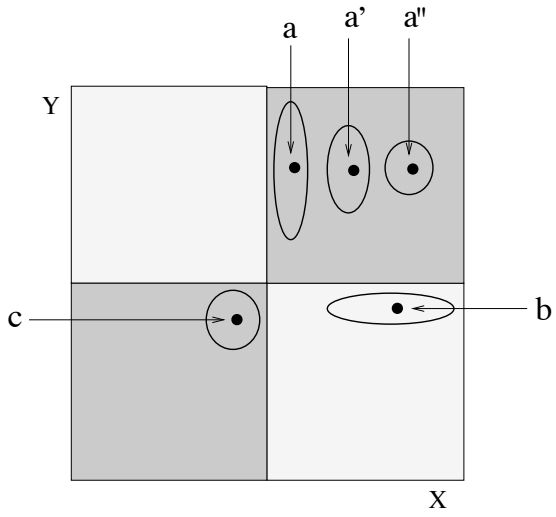

Figure 1: Feature relevance varies with query locations.

As a result, the class conditional probabilities tend to be constant in the modified neighborhoods, whereby better classification performance can be achieved. The amount of elongation-constriction decays as the query moves further from the boundary vicinity. This phenomenon is exemplified in Figure 1 by queries a, $a^{'}$ and $a^{''}$. In this paper we present experimental evidence of the accuracy achieved by means of this local weighting scheme.

The sparse solution given by SVMs also provides principled guidelines to efficiently set the input parameters of our technique. This is a major advantage over the ADAMENN technique [9], which has a competitive behavior but requires six tunable input parameters.

Furthermore, the technique proposed here speeds up the classification process since it computes off-line the information relevant to define local weights, and applies the nearest neighbor rule only once, whereas ADAMENN applies it at each point within a region centered at the query. Indeed, the major strength of our technique is that it is capable of providing local feature weightings using a global decision scheme, specifically the SVM boundary. This mechanism allows an off-line computation of the relevant information to define weights, leaving only local refinements to an on-line stage. This results in a method which is much more efficient than current local adaptive techniques for nearest neighbor classification [11, 12, 9], which act iteratively on the computation of neighborhoods.

We present theoretical and experimental results that show that using SVMs to locally weight features results in improved performance over both adaptive nearest neighbor techniques and the SVM classification method itself. Our approach is related to [2]. In [2], Amari and Wu improve support vector machine classifiers by

modifying kernel functions. 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 is local; moreover, since we weight features, our metric is also directional, and depends on the orientation of local boundaries in input space.

2 Adaptive Metric Nearest Neighbor Classification Techniques

K nearest neighbor methods are based on the assumption of smoothness of the target functions, which translates to locally constant class posterior probabilities for a classification problem. This assumption, 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 to produce a modified local neighborhood in which the posterior probabilities are approximately constant.

The techniques proposed in the literature [11, 12, 9] 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 a given query point $\mathbf{q}$, 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 in general be obtained.

3 Learning with SVMs

We are given l observations. Each observation consists of a pair: a vector $\mathbf{x}_i \in \Re^n$, $i = 1, \ldots, l$, and the associated class label $y_i \in \{-1, 1\}$. In the simple case of two linearly separable classes, a support vector machine 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 classifier that satifies the maximum margin property, i.e. the classifier whose decision boundary has the maximum minimum distance from the closest training point.

If the two classes are non-separable, the SVM looks for the hyperplane that maximizes the margin and that, at the same time, minimizes a quantity proportional to the number of misclassification errors. The trade-off between margin and 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 $f(\mathbf{x}) = \sum_i \alpha_i y_i \mathbf{x}_i^T \cdot \mathbf{x} - b$, and the coefficients α_i are the solution of a convex quadratic problem, defined over the hypercube $[0, C]^l$. 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 non-zero α_i will influence the solution. These points are the support vectors, i.e. the points that lie closest to the separating hyperplane. 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 non-linear 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 $f(\mathbf{x}) = \sum_i \alpha_i y_i \phi^T(\mathbf{x}_i) \cdot \phi(\mathbf{x}) - b$.

If one were given a function $K(\mathbf{x}, \mathbf{y}) = \phi^T(\mathbf{x}) \cdot \phi(\mathbf{y})$, 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) that for each continuous positive definite function $K(\mathbf{x}, \mathbf{y})$ there exists a mapping ϕ such that $K(\mathbf{x}, \mathbf{y}) = \phi^T(\mathbf{x}) \cdot \phi(\mathbf{y})$, $\forall \mathbf{x}, \mathbf{y} \in \Re^n$. By making use of such function K (*kernel function*), the equation for $f(\mathbf{x})$ can be rewritten as

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

4 Feature Weighting

The maximum margin boundary found by the SVM is used here to determine local discriminant directions over query's neighborhoods. 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. Formally, the definition of our weighting scheme proceeds as follows.

SVMs classify patterns according to the $sign(f(\mathbf{x}))$. Clearly, in the general case of a non-linear feature mapping ϕ, the SVM classifier gives a non-linear boundary $f(\mathbf{x}) = 0$ in input space. The gradient vector $\mathbf{n_d} = \nabla_{\mathbf{d}} f$, computed at any point $\mathbf{d}$ of the level curve $f(\mathbf{x}) = 0$, gives the perpendicular direction to the decision boundary in input space at $\mathbf{d}$. As such, the vector $\mathbf{n_d}$ identifies the orientation in input space on which the projected training data are well separated, locally over $\mathbf{d}$'s neighborhood. Therefore, the orientation given by $\mathbf{n_d}$, and any orientation close to it, is highly informative for the classification task at hand, and we can use such information 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 non uniform, and therefore estimation of local feature relevance becomes crucial. Let $\mathbf{d}$ be the closest point to $\mathbf{q}$ on the boundary $f(\mathbf{x}) = 0$: $\mathbf{d} = \arg\min_{\mathbf{p}} ||\mathbf{q} - \mathbf{p}||$, subject to the constraint $f(\mathbf{p}) = 0$. Then we know that the gradient $\mathbf{n_d}$ identifies a direction along which data points between classes are well separated.

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 a nearest neighbor rule at $\mathbf{q}$, 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 constricted (large weight) along $\mathbf{n_d}$ and along directions close to it, thus excluding from $\mathbf{q}$'s neighborhood points along $\mathbf{n_d}$ that are far from $\mathbf{q}$. The farther we move from the $\mathbf{n_d}$ direction, the less discriminant the correspondent orientation becomes. This means that class labels are likely not to change along those orientations, and distances should be elongated (small weight), thus including in $\mathbf{q}$'s neighborhood points which are likely to be similar to $\mathbf{q}$ in terms of the class conditional probabilities.

This principle is in analogy with a local linear discriminant analysis approach. In fact, the orientation of the gradient vector identifies the direction, locally at the query point, on which the projected training data are well separated. This property guides the process of generating modified neighborhoods with homogeneous class conditional probabilities.

Formally, we can measure how close a direction $\mathbf{t}$ is to $\mathbf{n_d}$ by considering the dot product $\mathbf{n_d}^T \cdot \mathbf{t}$. In particular, by denoting with $\mathbf{u}_j$ the 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

$$R_j(\mathbf{q}) \equiv |\mathbf{u}_j^T \cdot \mathbf{n_d}| = |n_{\mathbf{d},j}| \tag{2}$$

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

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

$$w_j(\mathbf{q}) = (R_j(\mathbf{q}))^t / \sum_{i=1}^{n} (R_i(\mathbf{q}))^t \tag{3}$$

where $t = 1, 2$, giving rise to linear and quadratic weightings, respectively. We propose the following exponential weighting scheme

$$w_j(\mathbf{q}) = exp(AR_j(\mathbf{q})) / \sum_{i=1}^{n} exp(AR_i(\mathbf{q})) \tag{4}$$

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 (2), and in general gives rise to better performance improvement. In fact, the exponential weighting scheme conveys stability to the method by preventing neighborhoods to extend infinitely in any direction. This is achieved by avoiding zero weights, which is instead allowed by the linear and quadratic weightings.

Thus, (4) can be used as weights associated with features for weighted distance computation

$$D(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} w_i (x_i - y_i)^2}. \tag{5}$$

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, 3].

One may be tempted to use the weights $w_j(\mathbf{q})$ directly in the SVM classification, by applying the weighted distance measure (5) 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, the weights $w_j(\mathbf{q})$ are based on the local (to $\mathbf{q}$) orientation of the decision boundary, and therefore they are meaningful for local distance computation of $\mathbf{q}$ from its neighbors. The weights $w_j(\mathbf{q})$, in fact, convey information on how distances should be constricted or elongated locally at $\mathbf{q}$: we desire to achieve constricted distances along directions close to the gradient direction, and elongated distances along directions far from 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.

5 Local Flexible Metric Classification based on SVMs

To estimate the orientation of local boundaries, we move from the query point along the input axes 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 an input axis i, i.e. when a point $\mathbf{p}_i$ is reached that satisfies the condition $sign(f(\mathbf{q})) \times sign(f(\mathbf{p}_i)) = -1$. 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 $\mathbf{n_d}$ with the gradient vector $\mathbf{n}_{\mathbf{d}_i} = \nabla_{\mathbf{d}_i} f$, computed at $\mathbf{d}_i$.

We desire that the parameter A in the exponential weighting scheme (4) 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 a query point $\mathbf{q}$ is to the boundary by computing its distance from the closest non bounded support vector: $B_{\mathbf{q}} = \min_{\mathbf{s}_i} ||\mathbf{q} - \mathbf{s}_i||$, where the minimum is taken over the non bounded ($0 < \alpha_i < C$) support vectors $\mathbf{s}_i$. Following the same principle, in [2] 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 $A = D - B_{\mathbf{q}}$, where D is a constant input parameter of the algorithm. In our experiments we set D equal to the approximated average distance between the training points $\mathbf{x}_k$ and the boundary:

$$D = \frac{1}{l} \sum_{\mathbf{x}_k} \{\min_{\mathbf{s}_i} ||\mathbf{x}_k - \mathbf{s}_i||\}. \tag{6}$$

If A becomes negative it is set to zero.

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 higher the effect of the R_j's values will be on distances computation.

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

1. Compute the approximated closest point $\mathbf{d}_i$ to $\mathbf{q}$ on the boundary;
2. Compute the gradient vector $\mathbf{n}_{\mathbf{d}_i} = \nabla_{\mathbf{d}_i} f$;
3. Set feature relevance values $R_j(\mathbf{q}) = |n_{\mathbf{d}_i,j}|$ 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 (6);
6. Set $\mathbf{w}$ according to (4);
7. Use the resulting $\mathbf{w}$ for K-nearest neighbor classification at the query point $\mathbf{q}$.

Figure 2: The LFM-SVM algorithm

We observe that this principled guideline for setting the parameters of our technique takes advantage of the sparseness representation of the solution provided by the SVM. 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, the computation of D's value is carried out once and off-line.

The resulting local flexible metric technique based on SVMs (LFM-SVM) is summarized in Figure 2. 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.

6 Weighting Features Increases the Margin

In this section we formally show that our weighting scheme increases the margin of the solution provided by the SVM. Our discussion holds for Gaussian kernels. This property explains the performance improvements achieved by our method over the SVM alone, as shown in our experiments. The same argument holds for polynomial kernels with an odd exponent also. The flow of the reasoning for a polynomial kernel is similar to the Gaussian one, and we omit it.

Consider the Gaussian radial basis function kernel (which we use in our experiments): $K(\mathbf{s}_i, \mathbf{x}) = e^{-\gamma ||\mathbf{s}_i - \mathbf{x}||^2}$. The expression of the decision boundary in equation (1) becomes: $f(\mathbf{x}) = \sum_{\mathbf{s}_i \in SV} \alpha_i y_i e^{-\gamma ||\mathbf{s}_i - \mathbf{x}||^2} - b = 0$. Consider now the j component of the gradient vector $\mathbf{n}_{\mathbf{d}} = \nabla_{\mathbf{d}} f = (\frac{\partial}{\partial x_1} f_{\mathbf{d}}, \ldots, \frac{\partial}{\partial x_n} f_{\mathbf{d}})$ computed with

respect to $\mathbf{x}$ at point $\mathbf{d}$:

$$n_{\mathbf{d},j} = \frac{\partial}{\partial x_j} f_{\mathbf{d}} = 2\gamma \sum_{\mathbf{s}_i \in SV} \alpha_i y_i (s_{ij} - d_j) e^{-\gamma ||\mathbf{s}_i - \mathbf{d}||^2} \tag{7}$$

where $\mathbf{d}$ is the closest point to the query $\mathbf{q}$ on the boundary. Our local measure of relevance (2) for feature j is then given by

$$R_j(\mathbf{q}) = |n_{\mathbf{d},j}| = |2\gamma \sum_{\mathbf{s}_i \in SV} \alpha_i y_i (s_{ij} - d_j) e^{-\gamma ||\mathbf{s}_i - \mathbf{d}||^2}|. \tag{8}$$

We partition SV in SV^- and SV^+, i.e. $SV = SV^- \cup SV^+$, where SV^- is the set of support vectors with label $y = -1$, and SV^+ is the set of support vectors with label $y = +1$. We can rewrite equation (8) as follows:

$$R_j(\mathbf{q}) = |2\gamma(\sum_{\mathbf{s}_i \in SV^+} \alpha_i (s_{ij} - d_j) e^{-\gamma ||\mathbf{s}_i - \mathbf{d}||^2} - \sum_{\mathbf{s}_i \in SV^-} \alpha_i (s_{ij} - d_j) e^{-\gamma ||\mathbf{s}_i - \mathbf{d}||^2})|. \tag{9}$$

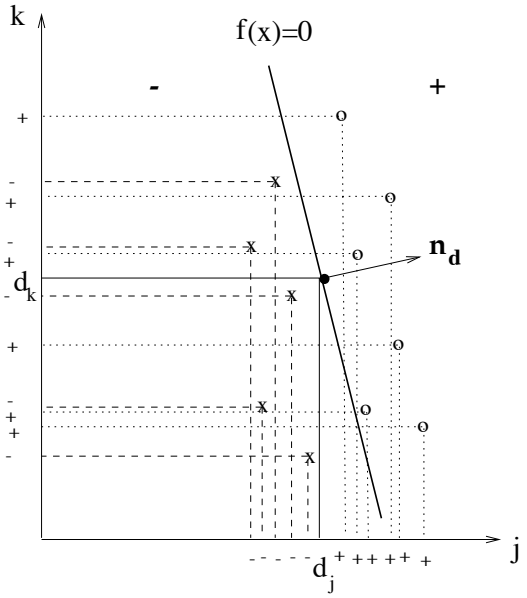

Figure 3: Illustration of a case in which conditions (10) are satisfied. Negative support vectors are represented as "x" and positive support vectors are denoted with "o". Along feature j, d_j separates the positive support vectors from the negative ones. Along feature k support vectors with opposite sign shuffle. A large weight is assigned to feature j and a small weight to feature k.

We want to identify the conditions that make $R_j(\mathbf{q})$ large. We observe that γ, α_i, and the exponentials in (9) are all positive terms. A large value for $R_j(\mathbf{q})$ is

obtained when the terms $(s_{ij} - d_j)$ are all positive for $\mathbf{s}_i \in SV^+$ and all negative for $\mathbf{s}_i \in SV^-$, or vice versa. These conditions are satisfied when

$$d_j < s_{ij} \quad \forall \mathbf{s}_i \in SV^+ \quad \textit{and} \quad d_j > s_{ij} \quad \forall \mathbf{s}_i \in SV^- \tag{10}$$

or

$$d_j > s_{ij} \quad \forall \mathbf{s}_i \in SV^+ \quad \textit{and} \quad d_j < s_{ij} \quad \forall \mathbf{s}_i \in SV^-. \tag{11}$$

Figure 3 illustrates a case in which conditions (10) are satisfied. Along the orientation identified by feature j, d_j separates the positive support vectors from the negative ones. As a consequence, the value of $R_j(\mathbf{q})$ is large, and a large weight is assigned to feature j. On the other hand, along the orientation identified by feature k, the negative support vectors mix with the positive support vectors. Therefore, the terms $(s_{ik} - d_k)$ becomes positive and negative for different $\mathbf{s}_i$ within either SV^+ or SV^-, and cancel each other in equation (9). As a result, the value of $R_k(\mathbf{q})$ is small, and a small weight is assigned to feature k.

By assigning a large weight to feature j in Figure 3, and in general to input features close to the gradient direction, locally in neighborhood of support vectors, we increase the distance between the support vectors in SV^- and SV^+. This corresponds to improve the separability of classes along those orientations, and therefore the margin. As a consequence, better classification results can be achieved as also demonstrated in our experiments.

7 Experimental Results

In the following we compare several classification methods using both simulated and real data. We compare the following classification approaches: **LFM-SVM** algorithm described in Figure 2. SVM^{light} [15] with radial basis kernels is used to build the SVM classifier. **RBF-SVM** classifier with radial basis kernels. We used SVM^{light} [15], 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 LFM-SVM. **ADAMENN**-adaptive metric nearest neighbor technique [9]. 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** [11]. 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** [11]. 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 [12]. 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 [19].

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 for each method were determined empirically through cross-validation.

7.1 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 1.

7.1.1 The 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 1 shows the results for this problem. The standard deviations are: 0.17, 0.01, 0.01, 0.01, 0.01 0.01, 0.01 and 1.50, respectively.
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 1. The standard deviations are: 0.18, 0.01, 0.02, 0.01, 0.01, 0.01, 0.01 and 1.60, respectively.

Table 1: Average classification error rates for simulated data.

	MultiGauss	NoisyGauss
LFM-SVM	**3.3**	**3.4**
RBF-SVM	**3.3**	5.3
ADAMENN	3.4	4.1
Machete	3.4	4.3
Scythe	3.4	4.8
DANN	3.7	4.7
K-NN	**3.3**	7.0
C4.5	5.0	5.1

7.1.2 Results

Table 1 shows that all methods have similar performances for the MultiGaussians problem, with C4.5 being the worst performer. When the noisy predictors are added to the problem (NoisyGaussians), we observe different levels of deterioration

in performance among the eight methods. LFM-SVM 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 Figure 4 we plot the performances of LFM-SVM and RBF-SVM as a function of an increasing number of noisy features (for the same MultiGaussians problem). The standard deviations for RBF-SVM (in order of increasing number of noisy features) are: 0.01, 0.01, 0.03, 0.03, 0.03 and 0.03. The standard deviations for LFM-SVM are: 0.17, 0.18, 0.2, 0.3, 0.3 and 0.3. The LFM-SVM technique shows a considerable improvement over RBF-SVM as the amount of noise increases.

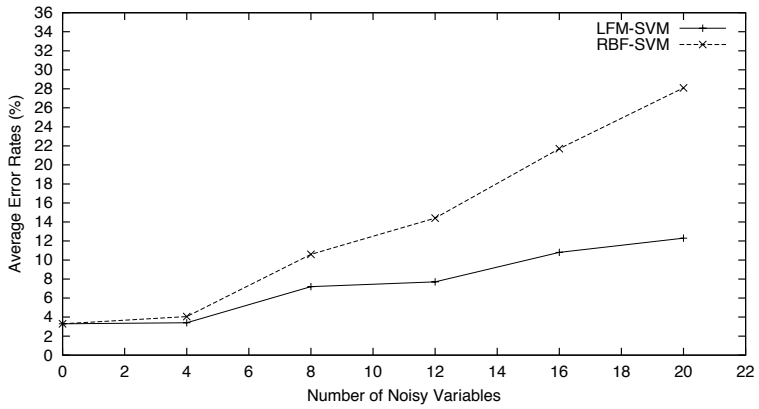

Figure 4: Average Error Rates of LFM-SVM and RBF-SVM as a function of an increasing number of noisy predictors.

7.2 Experiments on Real Data

We used six different real data sets. They are all taken from UCI Machine Learning Repository at http://www.cs.uci.edu/~mlearn/MLRepository.html. For the Iris, Sonar 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. Table 2 shows the cross-validated error rates for the eight methods under consideration on the six real data.

7.2.1 The 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 2. 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 2. 3. **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 CQA. 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 third column of Table 2. 4. **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 fourth column of Table 2. The standard deviations are: 0.2, 0.2, 0.2, 0.2, 0.2, 0.9, 0.9 and 0.9, respectively. 5. **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 fifth column of Table 2. The standard deviations are: 0.2, 0.2, 0.2, 0.3, 0.2, 1.1, 1.5 and 2.1, respectively. 6. **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 sixth column of Table 2. The standard deviations are: 0.4, 0.4, 0.4, 0.4, 0.4, 2.4, 2.1 and 0.7, respectively.

7.2.2 Results

Table 2: Average classification error rates for real data.

	Iris	Sonar	Vote	Breast	OQ	Pima
LFM-SVM	4.0	11.0	**2.6**	3.0	3.5	**19.3**
RBF-SVM	4.0	12.0	3.0	3.1	3.4	21.3
ADAMENN	**3.0**	9.1	3.0	3.2	**3.1**	20.4
Machete	5.0	21.2	3.4	3.5	7.4	20.4
Scythe	4.0	16.3	3.4	2.7	5.0	20.0
DANN	6.0	**7.7**	3.0	**2.2**	4.0	22.2
K-NN	6.0	12.5	7.8	2.7	5.4	24.2
C4.5	8.0	23.1	3.4	4.1	9.2	23.8

Table 2 shows that LFM-SVM achieves the best performance in 2/6 of the real data sets; 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 Friedman [11], 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: $b_m = e_m / \min_{1 \leq k \leq 8} e_k$. Thus, the best method m^* for that example has $b_{m^*} = 1$, and all other methods have larger values $b_m \geq 1$, for $m \neq m^*$. 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.

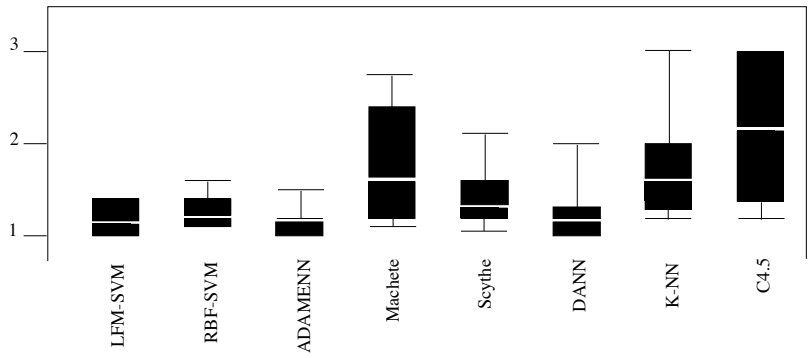

Figure 5: Performance distributions for real data.

Figure 5 plots the distribution of b_m for each method over the seven 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. The outer vertical lines for the LFM-SVM method are not visible because they coincide with the limits of the lower and upper quartiles. The spread of the error distribution for LFM-SVM 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 LFM-SVM (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 [12], 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 the sparse solution given by SVMs provides LFM-SVM with principled guidelines to efficiently set the input parameters. This is an important advantage over ADAMENN, which has six tunable input parameters. Furthermore, LFM-SVM speeds up the classification process since it applies the nearest neighbor rule only once, whereas ADAMENN applies it at each point within a region centered at the query. We also observe that the construction of the SVM for LFM-SVM is carried out off-line only once, and there exist algorithmic and computational results which make SVM training practical also for large-scale problems [15].

The LFM-SVM offers performance improvements over the RBF-SVM algorithm alone, for both the (noisy) simulated and real data sets. The reason for such performance gain may rely on the effect of our local weighting scheme on the separability of classes, and therefore on the margin, as shown in section 6. Assigning large weights to input features close to the gradient direction, locally in neighborhoods of support vectors, corresponds to increase the spatial resolution along those orientations, and therefore to improve the separability of classes. As a consequence, better classification results can be achieved as demonstrated in our experiments.

8 Conclusions

We have described a locally adaptive metric classification method, formally motivated the approach, and demonstrated its efficacy through experimental results. The proposed technique offers performance improvements over the SVM alone, and has the potential of scaling up to large data sets. It speeds up, in fact, the classification process by computing off-line the information relevant to define local weights. It also applies the nearest neighbor rule only once, whereas ADAMENN applies it at each point within a region centered at the query.

Acknowledgments

This research has been supported by the National Science Foundation under grants NSF CAREER Award 9984729 and NSF IIS-9907477, by the US Department of Defense, and a research award from AT&T.

References

[1] D. Aha, "Lazy Learning", *Artificial Intelligence Review*. 11:1-5. 1997.

[2] S. Amari and S. Wu, "Improving support vector machine classifiers by modifying kernel functions", *Neural Networks*, 12, pp. 783-789, 1999.

[3] C. Atkeson, A.W. Moore, and S. Schaal, "Locally Weighted Learning", *Artificial Intelligence Review*. 11:11-73. 1997.

[4] R.E. Bellman, *Adaptive Control Processes*. Princeton Univ. Press, 1961.

[5] 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 using support vector machines", Tech. Report, University of California in Santa Cruz, 1999.

[6] W.S. Cleveland and S.J. Devlin, "Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting", *J. Amer. Statist. Assoc.* **83**, 596-610, 1988

[7] T.M. Cover and P.E. Hart, "Nearest Neighbor Pattern Classification", *IEEE Trans. on Information Theory*, pp. 21-27, 1967.

[8] C. Domeniconi and D. Gunopulos, "Adaptive Nearest Neighbor Classification using Support Vector Machines", *Advances in Neural Information Processing Systems 14*, 2001.

[9] C. Domeniconi, J. Peng, and D. Gunopulos, "An Adaptive Metric Machine for Pattern Classification", *Advances in Neural Information Processing Systems 13*, 2000.

[10] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*. John Wiley & Sons, Inc., 1973.

[11] J.H. Friedman "Flexible Metric Nearest Neighbor Classification", Tech. Report, Dept. of Statistics, Stanford University, 1994.

[12] T. Hastie and R. Tibshirani, "Discriminant Adaptive Nearest Neighbor Classification", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 18, No. 6, pp. 607-615, 1996.

[13] T.K. Ho, "Nearest Neighbors in Random Subspaces", *Lecture Notes in Computer Science: Advances in Pattern Recognition*, pp. 640-648, 1998.

[14] T. Joachims, "Text categorization with support vector machines", *Proc. of European Conference on Machine Learning*, 1998.

[15] T. Joachims, "Making large-scale SVM learning practical" *Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf and C. Burger and A. Smola (ed.), MIT-Press, 1999. http://www-ai.cs.uni-dortmund.de/thorsten/svm_light.html

[16] D.G. Lowe, "Similarity Metric Learning for a Variable-Kernel Classifier", *Neural Computation* **7**(1):72-85, 1995.

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

[18] E. Osuna, R. Freund, and F. Girosi, "Training support vector machines: An application to face detection", *Proc. of Computer Vision and Pattern Recognition*, 1997.

[19] J.R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan-Kaufmann Publishers, Inc., 1993.

[20] S. Salzberg, A Nearest Hyperrectangle Learning Method. *Machine Learning* **6**:251-276, 1991.

[21] C.J. Stone, Nonparametric regression and its applications (with discussion). *Ann. Statist.* **5**, 595, 1977.

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

[23] V. Vapnik, *Statistical Learning Theory*. Wiley, 1998.