Local Distance Functions: A Taxonomy, New Algorithms, and an Evaluation

Deva Ramanan
UC Irvine
dramanan@ics.uci.edu

Simon Baker
Microsoft Research
sbaker@microsoft.com

Abstract

We present a taxonomy for local distance functions where most existing algorithms can be regarded as approximations of the geodesic distance defined by a metric tensor. We categorize existing algorithms by **how**, **where** *and* **when** *they estimate the metric tensor. We also extend the taxonomy along each axis.* **How***: We introduce hybrid algorithms that use a combination of dimensionality reduction and metric learning to ameliorate over-fitting.* **Where***: We present an exact polynomial time algorithm to integrate the metric tensor along the lines between the test and training points under the assumption that the metric tensor is piecewise constant.* **When***: We propose an interpolation algorithm where the metric tensor is sampled at a number of references points during the offline phase, which are then interpolated during online classification. We also present a comprehensive evaluation of all the algorithms on tasks in face recognition, object recognition, and digit recognition.*

1. Introduction

The K-nearest neighbor (K-NN) algorithm is a simple but effective tool for classification. It is well suited for multi-class problems with large amounts of training data, which are relatively common in vision. Despite its simplicity, K-NN and its variants are competitive with the state-of-the-art on various vision benchmarks [29, 3, 14].

The key ingredient in K-NN is the choice of the distance function or metric. The distance function captures the type of invariances used for measuring similarity between pairs of examples. These functions are often hand-constructed, but recent approaches have tried to learn them from training data [9, 25, 15, 14, 2]. Until recently, such methods focused primarily on learning a single global metric.

The optimal distance measure for 1-NN is the probability that the pair of examples belong to different classes [19]. The resulting function can be quite complex, and in part motivates the analysis of distance functions that vary across the space of examples. For those motivated by psychophysical studies, there also exists evidence that humans define categorical boundaries in terms of local relationships between exemplars [21]. The last few years have seen an increased interest in such local distance functions for nearest neighbor classification [29, 6, 7, 20], though similar ideas were explored earlier in the context of local discriminant analysis [13, 4] and locally weighted learning [1, 22]. Recent approaches have also leveraged local metrics for gaussian process regression [26] and multiple kernel learning [10].

Our first contribution is to present a taxonomy for local distance functions. In particular, we show how most existing algorithms can be regarded as approximations of the geodesic distance defined by a metric tensor. We categorize existing algorithms in terms of **how**, **where**, and **when** they estimate the metric tensor. See Table 1 for a summary. In terms of **how**, existing algorithms obtain a metric either[1] by dimensionality reduction such as principle components analysis (PCA) or linear discriminant analysis (LDA) [8], or by explicit metric-learning [2, 9, 15]. In terms of **where**, existing algorithms sample the metric tensor: (a) for the whole space ("global"), (b) for each class ("per-class") [15], (c) at the training points ("per-exemplar") [6, 7], or (d) at the test point [13, 1]. In terms of **when**, existing algorithms either estimate the metric tensor: (a) offline during training or (b) online during classification.

Our second contribution is to extend the taxonomy along each dimension. In terms of **how**, we introduce hybrid algorithms that use a combination of dimensionality reduction and metric learning to ameliorate over-fitting. In terms of **where**, we consider algorithms to integrate the metric tensor along the line between the test point and the training point that it is being matched to. We present an exact polynomial time algorithm to perform this integration under the assumption that the metric tensor is piecewise constant. In terms of **when**, we consider a combined online-offline algorithm. In the offline training phase, a representation of the metric tensor is estimated by sampling it at a number of reference points In the online phase, the metric tensor is estimated by interpolating the samples at the reference points.

Our third contribution is to present a comprehensive evaluation of the algorithms in our framework, both prior and new. We show results on a diverse set of problems,

[1]Another approach is to define a distance function analytically based on high level reasoning. An example is the tangent distance [24], where the metric captures user-defined invariances in the data. We restrict our analysis to situations where the metric is not user-defined in this way.

Table 1. A summary of our taxonomy for local distance functions. Algorithms can be classified by **how**, **when**, and **where** they estimate the metric tensor. We also propose a number of extensions to the taxonomy (marked in **bold**.)

How	Dim. Reduction/LDA
	Metric Learning
	Hybrid (Sec. 2.2.3)

		Where				
		Whole Space	Each Class	Training Points	Test Point	Line Integral
When	Offline	Global	Per-Class	Small DB Only	N/A	N/A
	Online	As Above	As Above	Per-Exemplar	Lazy	Too Slow
	Interp.	As Above	As Above	**Sec. 2.4.3**	**Sec. 2.4.3**	**Sec. 2.3.5**

including object recognition using face recognition using MultiPIE [12], Caltech 101 [5], and digit recognition using MNIST [18]. To spur further progress in this area and allow other researchers to compare their results with ours, we will make the raw feature vectors, class membership data, and training-test partitions of the data available on a website with the goal of defining a standard benchmark.

2. Taxonomy and Algorithms

2.1. Background

We assume that we are working with a classification problem defined in an K dimensional feature space R^K. Each test sample is mapped into this space by computing a number of features to give a point $\mathbf{x} \in R^K$. Similarly, each training sample is mapped into this space to give a point point $\mathbf{x}_i \in R^K$. Where appropriate, in this paper we denote test points by $\mathbf{x}$ and training points by $\mathbf{x}_i$. We do not consider problems where all that can be computed is the pairwise distance between a test samples and a training sample $\mathrm{Dist}(\mathbf{x}, \mathbf{x}_i)$. A number of local distance function and metric learning algorithms have been proposed for this more general setting [6, 7]. Note, however, that such general metrics often measure the distance between two images under some correspondence of local features and that approximate correspondence can be captured by a pyramid vector representation [11]. Many, if not most, distance functions either explicitly or implicitly embed data in a finite dimensional vector space. Our work applies to this common case.

A Mahanobolis metric is a $K \times K$ symmetric, positive definite matrix M. (In many cases the positive definite condition is relaxed to just non-negativity in which case, strictly speaking M is a psuedometric.) A metric defines the distance between a pair of test and training points $\mathbf{x}, \mathbf{x}_i \in R^K$ via:

$$\mathrm{Dist}_M(\mathbf{x}, \mathbf{x}_i) \;=\; (\mathbf{x}_i - \mathbf{x})^{\mathrm{T}} M (\mathbf{x}_i - \mathbf{x}). \qquad (1)$$

The metric tensor is a generalization of a single constant metric to a possibly different metric $MT(\mathbf{x})$ at each point in the space. Roughly speaking, $MT(\mathbf{x})$ can be thought of as defining a local distance function $\mathrm{Dist}_{MT(\mathbf{x})}$ at each $\mathbf{x}$.

Given a piecewise differentiable curve $\mathbf{c}(\lambda)$, where $\lambda \in [0, 1]$, the length of the curve is given by:

$$\mathrm{Length}(\mathbf{c}(\lambda)) \;=\; \int_0^1 \left[\frac{\mathrm{d}\mathbf{c}}{\mathrm{d}\lambda}\right]^{\mathrm{T}} MT(\mathbf{c}(\lambda)) \frac{\mathrm{d}\mathbf{c}}{\mathrm{d}\lambda}\, \mathrm{d}\lambda. \qquad (2)$$

The geodesic distance between a pair of test and training points is the distance of the shortest curve between them:

$$\mathrm{Dist}_{\mathrm{Geo}}(\mathbf{x}, \mathbf{x}_i) \;=\; \min_{\mathbf{c}(0)=\mathbf{x}, \mathbf{c}(1)=\mathbf{x}_i} \mathrm{Length}(\mathbf{c}(\lambda)). \qquad (3)$$

If the metric tensor is (close to) zero within each class, and has a large non-zero component across the boundary of each class, then perfect classification can be obtained in the absence of noise using the geodesic distance and just a single training sample in each connected component of each class. Estimating such an ideal metric tensor is impossible. But at heart, we argue that all local distance function algorithms are based on the hope that better classification can be obtained by using such a spatially varying metric tensor and an approximation to the geodesic distance.

2.2. How

2.2.1 Dimensionality Reduction Algorithms

Perhaps the simplest metric learning algorithms are the various linear dimensionality reduction algorithms. For example, Principle Components Analysis (PCA) [8] projects the data into a low-dimensional subspace and then measures distance in the projected space. If P is the (orthonormal) $d \times K$ dimensional PCA projection matrix, the corresponding distance between a pair of test and training points:

$$||P(\mathbf{x}_i - \mathbf{x})||^2 \;=\; (\mathbf{x}_i - \mathbf{x})^T M_{\mathrm{PCA}} (\mathbf{x}_i - \mathbf{x}) \qquad (4)$$

where $M_{\mathrm{PCA}} = P^{\mathrm{T}} P$ is the corresponding metric. Rather than an explicit projection, one can use a Mahanolobis distance [8] (the inverse of the sample covariance matrix.)

A more powerful approach is to learn a subspace that preserves the variance between class labels using Linear Discriminant Analysis (LDA) [8]. LDA has a large number of variants. In this paper we consider a weighted form of **Regularized LDA** [8] described below. We assume we are given a set of triples $\{x_i, y_i, w_i\}$ consisting of data points, class labels, and weights. We can interpret integer weights as specifying the number of times an example appears in an equivalent unweighted dataset. We search for the $d \times K$ dimensional LDA basis V that maximizes:

$$\arg\max_V \mathrm{tr}(V^T \Sigma_B V) \quad \text{s.t.} \quad V^T(\Sigma_W + \lambda I)V = I \qquad (5)$$

$$\Sigma_B = \frac{1}{C}\sum_j \bar{\mu}_j \bar{\mu}_j^T, \quad \Sigma_W = \frac{\sum_i w_i \bar{x}_i \bar{x}_i^T}{\sum_i w_i}, \quad \bar{x}_i = x_i - \mu_{y_i}$$

$$\bar{\mu}_j = \mu_j - \frac{1}{C}\sum_j \mu_j, \quad \mu_j = \frac{\sum_{\{i:y_i=j\}} w_i x_i}{\sum_{\{i:y_i=j\}} w_i}$$

where C is the number of classes. This can be solved with a generalized eigenvalue problem. In general, Σ_W will of rank K or higher, while the rank of Σ_B is upper bounded by C. If $C < K$ we obtain use a bagged estimate of Σ_B obtained by averaging together covariance matrices estimated from sub-sampled data [28]. The regularization parameter λ enforces an isotropic Gaussian prior on Σ_W that ensures that the eigenvalue problem is well conditioned when Σ_W is low rank. We found this explicit regularization was essential for good performance even when Σ_W is full rank. We used a fixed $\lambda = 1$ for all our experiments.

As defined in Equation (5), V is not guaranteed to be orthonormal. We explicitly apply Gram-Schmidt orthonormalization to compute V_G and define the LDA metric: $M_{\text{LDA}} = V_G^{\text{T}} V_G$ where M_{LDA} is a $K \times K$ matrix of rank d. An alternative approach [13] is to use the scatter matrices directly to define a full-rank metric $M = S_W^{-1} S_B S_W^{-1}$. Empirically, we found the low-rank metric M_{LDA} to perform slightly better, presumably due to increased regularization.

2.2.2 Metric Learning Algorithms

Recently, many approaches have investigated the problem of learning a metric M that minimizes an approximate K-NN error on a a training set of data and class label pairs $\{x_i, y_i\}$ [9, 25, 15, 14, 2]. Relevant Component Analysis (RCA) [2] finds the basis V that maximizes the mutual information between x_i and its projection Vx_i while requiring that that distances between projected points from the same class remain small. Neighborhood Component Analysis (NCA) [9] minimizes a probabilistic approximation of the leave-one-out cross validation error. Recent approaches minimize a hinge-loss approximation of the misclassification error [15, 14, 25].

The state-of-the-art at the time of writing seems to be the Large Margin Nearest Neighbor (LMNN) algorithm [15]. LMNN defines a loss function L that penalizes large distances between "target" points that are from the same class while enforcing the constraint that, for each point i, "imposter" points k from different classes are 1 unit further away than a target neighbors j. We define a generalization that uses a weighted error similar to (5):

$$\begin{aligned} L(M) = & \sum_{\{ij\}\in Targ} w_i \text{Dist}_M(\mathbf{x}_i, \mathbf{x}_j) + \qquad (6) \\ & C \sum_{\{ijk\}\in Imp} w_i h(\text{Dist}_M(\mathbf{x}_i, \mathbf{x}_k) - \text{Dist}_M(\mathbf{x}_i, \mathbf{x}_j)) \end{aligned}$$

The constraint is enforced softly with a hinge loss function h, making the overall objective function convex. It can be solved with a semi-definite program (SDP). We used publically available LMNN code [15] in our experiments.

2.2.3 New Algorithm: Hybrid Approach

Some local distance function algorithms invoke multiple metrics at various stages. For example, many local algorithms require an initial global metric to produce a *short list* of candidate training points which are then processed by a more expensive local metric. More generally in our taxonomy below, many local algorithms use an initial global metric to determining **where** in the space to estimate the a local approximation to the metric tensor. An example are interpolation algorithms which estimate the local metric at a test point with a weighted combination of reference metrics - here, a global metric produces the weight (see Sec 2.4.3).

The metric learning algorithm need not be the same in the two stages. In this paper we consider a hybrid approach where different metric learning algorithms are used in the two steps. Specifically, we use LMNN to decide where to estimate the metric tensor and populate a short list, and Regularized LDA to estimate the local metric. The local estimation of a metric is more dependent on regularization. Empirically we found LMNN to be more powerful when supplied with ample training data and LDA to be far easier to regularize when estimating a local metric.

2.3. Where

2.3.1 Over the Whole Space: Global

The baseline approach is to assume that the metric tensor is constant throughout the space and compute a single **Global** metric for the whole space using all of the training data $MT(\mathbf{x}) = M_{\text{Global}}$. If the metric tensor is a constant, then the shortest path geodesic distance in Equation (3) is the distance along the direct path $\mathbf{c}(\lambda) = \mathbf{x} + \lambda(\mathbf{x}_i - \mathbf{x})$ between the points which simplifies to:

$$\text{Dist}_{\text{Global}}(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x}_i - \mathbf{x})^{\text{T}} M_{\text{Global}} (\mathbf{x}_i - \mathbf{x}). \qquad (7)$$

2.3.2 For Each Class: Per-Class

Another alternative is to approximate the metric tensor with a constant metric for each class, commonly referred to as a **Per-Class** metric [15]:

$$MT(\mathbf{x}_i) = M^j_{\text{Per-Class}} \quad \text{where} \quad j = \text{Class(x)}_\text{i}. \qquad (8)$$

In particular, there is a different metric M^j_{PerClass} for each class j. One simple approach to learn M^j_{PerClass} is to optimize the weighted score from (5) or (6) where $w_i = 1$ for $\{i : y_i = j\}$ and 0 otherwise. When computing the distance from a test point $\mathbf{x}$ to a training point $\mathbf{x}_i$ we assume the metric tensor is constant along all paths from $\mathbf{x}$ to $\mathbf{x}_i$:

$$\text{Dist}_{\text{PC}}(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x}_i - \mathbf{x})^{\text{T}} M^{\text{Class}(\mathbf{x}_i)}_{\text{Per-Class}} (\mathbf{x}_i - \mathbf{x}). \qquad (9)$$

In general metrics learned for each class may not be comparable, since they are trained independently. We make LDA metrics comparable by normalizing each to have a unit trace. In all our experiments, we learn a rank $d = 50$ class-specific LDA metric. To normalize the LMNN metrics, we follow the approach introduced in [7] and [15] and define a generalization of (6) that learns metrics M_c for all classes together:

$$L(\{M_c\}) = \sum_{\{ij\} \in Targ} \mathrm{Dist}_{M_{y_i}}(\mathbf{x}_i, \mathbf{x}_j) + \tag{10}$$
$$C \sum_{\{ijk\} \in Imp} h(\mathrm{Dist}_{M_{y_i}}(\mathbf{x}_i, \mathbf{x}_k) - \mathrm{Dist}_{M_{y_i}}(\mathbf{x}_i, \mathbf{x}_j)).$$

2.3.3 At the Training Points: Per-Exemplar

Another approximation is to assume that the metric tensor is constant within the neighborhood of each training sample $\mathbf{x}_i$ and use a **Per-Exemplar** metric [6, 7]:

$$MT(\mathbf{x}) \;=\; M^{\mathbf{x}_i}_{\mathrm{Per-Exemplar}} \tag{11}$$

for points $\mathbf{x}$ close to $\mathbf{x}_i$. One simple approach to learn $M^{\mathbf{x}_i}_{\mathrm{Per-Exemplar}}$ is to optimize (5) or (6) with $w_i = 1$ and 0 otherwise. Again, these metrics may not be comparable. As before, we trace-normalize the LDA metrics. In theory, one can define an extension of (10) that simultaneously learns all LMNN exemplar metrics. This is infeasible due to the number on training exemplars, and so we independently learn LMNN exemplar metrics. In practice, this still performs well because the initial pruning step (creating a shortlist) tends to select close-by exemplars whose metrics are learned with similar data. Note that [7] is able to jointly learn metrics by learning diagonal metrics on a subset of the training data deemed "focal" exemplars.

When computing the distance from a test point $\mathbf{x}$ to a training point $\mathbf{x}_i$ we assume the metric tensor is constant along all paths from $\mathbf{x}$ to $\mathbf{x}_i$ and so use the metric of $\mathbf{x}_i$:

$$\mathrm{Dist}_{\mathrm{PE}}(\mathbf{x}, \mathbf{x}_i) \;=\; (\mathbf{x}_i - \mathbf{x})^{\mathrm{T}} M^{\mathbf{x}_i}_{\mathrm{Per-Exemplar}}(\mathbf{x}_i - \mathbf{x}). \tag{12}$$

2.3.4 At the Test Point: Lazy

We can also estimate the metric tensor at the test point:

$$MT(\mathbf{x}) \;=\; M^{\mathbf{x}}_{\mathrm{Lazy}}. \tag{13}$$

This approach requires learning the metric at run time which is why we refer to it as **Lazy**. The training set for the Lazy algorithm consists of K training points $\mathbf{x}_i$ close to the test point $\mathbf{x}$. In our experiments, we use a Global metric to define which points are the closest and set $K = 50$. We set w_i for these training points to be 1 and 0 otherwise, and optimize (5) or (6) to learn $M^{\mathbf{x}}_{\mathrm{Lazy}}$. We fix the rank d of the LDA metric to the number of distinct classes present in the K training neighbors. When computing the distance from a test point $\mathbf{x}$ to a training point $\mathbf{x}_i$ we assume the metric tensor is constant along all paths from $\mathbf{x}$ to $\mathbf{x}_i$ and use the metric estimated at the test point $\mathbf{x}$:

$$\mathrm{Dist}_{\mathrm{Lazy}}(\mathbf{x}, \mathbf{x}_i) \;=\; (\mathbf{x}_i - \mathbf{x})^{\mathrm{T}} M^{\mathbf{x}}_{\mathrm{Lazy}}(\mathbf{x}_i - \mathbf{x}). \tag{14}$$

Examples of the **Lazy** algorithm are the local discriminant adaptive algorithm of [13], the locally weighted learning algorithm of [1], and the locally adpative metric algorithm of [4]. These algorithms both use a global metric to select the K nearest neighbors of a test point $\mathbf{x}$ which are then classified locally.

2.3.5 New Algorithm: Integration Along a Line

Computing the geodesic distance in Equations (2) and (3) in high dimensions in intractable. In 2D or 3D the space can be discretized into a regular grid of vertices (pixels/voxels) and then either an (approximate) distance transform or a shortest path algorithm used to estimate the geodesic distance. In higher dimensions, however, discretizing the space yields a combinatorial explosion in the number of vertices to be considered. To avoid this problem, all of the above algorithms assume that the metric tensor is locally (or globally) constant and so can approximate the geodesic distance in with the simple metric distance in Equation (1).

One way to obtain a better approximation of the geodesic is to assume that the space it not curved locally and approximate the minimum in Equation (3) with the integral along the line between the test point $\mathbf{x}$ and the training point $\mathbf{x}_i$:

$$\mathbf{c}(\lambda) \;=\; \mathbf{x} + \lambda(\mathbf{x}_i - \mathbf{x}). \tag{15}$$

Substituting Equation (15) into Equation (2) and simplifying yeilds the **Line Integral** distance:

$$\mathrm{Dist}_{\mathrm{Line}}(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x}_i - \mathbf{x})^{\mathrm{T}} \left[\int_0^1 MT(\mathbf{c}(\lambda))\, \mathrm{d}\lambda \right] (\mathbf{x}_i - \mathbf{x}). \tag{16}$$

Note that the line distance is an upper bound on the geodesic distance:

$$\mathrm{Dist}_{\mathrm{Geo}}(\mathbf{x}, \mathbf{x}_i) \;\leq\; \mathrm{Dist}_{\mathrm{Line}}(\mathbf{x}, \mathbf{x}_i). \tag{17}$$

In Appendix A we present a polynomial time algorithm to estimate the integral in Equation (16) exactly under the assumption that the metric tensor is piecewise constant.

2.4. When

2.4.1 Offline

Offline algorithms compute the metric at train time, before any test point are given. Therefore, in Table 1 only the **Global**, **Per-Class** and **Per-Exemplar** algorithms could be used offline. The **Lazy** and **Line Integral** approaches require the test point and so cannot be applied. The Global approach is the most common [9, 25, 15, 14, 2], and also the least computationally demanding. Given a dataset with M training points in C classes in a K dimensional space, just a single metric must be estimated and the storage requirement is only $O(K^2)$. The Per-Class approach [15] must estimate C metrics, and so requires storing $O(CK^2)$. The Per-Exemplar approach has also been used in the past [6, 7]. Since the number of training points M is typically much larger than the number of classes, the storage cost $O(MK^2)$ can make the application of this approach in an offline fashion intractable for very large datasets including ours. One can however, define an "online" Per-Exemplar that first computes a short-list of exemplars for a test point using a global metric. One can then learn metrics of each short-listed exemplar on-the-fly, use them to re-rank the exemplars, and discard them without any need for storage. We include this algorithm in our experiments.

2.4.2 Online

Once the test point has been provided, all of the algorithms in Table 1 could be applied. Recomputing the metrics used by **Global**, **Per-Class** for each new test point would be redundant and so in practice it makes little sense to apply these algorithms. Recall that **Per-Exemplar**, even though essentially an off-line algorithm, can only be practically implemented on-line due to storage limitations. On the other hand, the **Lazy** algorithm [13, 1] can only be applied during classification of a specific test point. The **Line Integral** algorithm could also be applied in an online fashion, however due to the computational cost, the interpolation approach described next is more practical.

2.4.3 New Algorithms: Interpolation

We expect the metric tensor $MT(\mathbf{x})$ to vary smoothly over the space $\mathbf{x}$. This suggests that we can interpolate the tensor from a sparse set of samples. Assume that a subset of the training examples have been selected. We refer to these points $\mathbf{x}_{\mathrm{RP}_i}$ for $i = 1, \dots, R$ as reference points. In our experiments we chose the reference points at random, however, a more principled approach to distribute the reference points uniformly could have been used.

Offline, local metrics $M_{RP}^{\mathbf{x}_{\mathrm{RP}_i}}$ are computed for the reference points by optimizing (5) or (6) with $w_{RP_i} = 1$ and 0 otherwise. One could learn these metrics jointly but we found independently learning this models sufficed. Online, the local metrics $M_{RP}^{\mathbf{x}_{\mathrm{RP}_i}}$ are interpolated to obtain the final metric. One approach is a nearest neighbor (NN) interpolation $MT(\mathbf{x}) = M_{RP}^{\mathbf{x}_{\mathrm{RP}_i}}$ where for all $j = 1, \dots, R$:

$$\mathrm{Dist}_{\mathrm{Global}}(\mathbf{x}, \mathbf{x}_{\mathrm{RP}_i}) \leq \mathrm{Dist}_{\mathrm{Global}}(\mathbf{x}, \mathbf{x}_{\mathrm{RP}_j}). \tag{18}$$

Here we use a Global metric to define closeness to the references points. Another approach would be to use that reference point's own metric, as is done in Appendix A. We saw similar performance in either case with a slight speed up with Global. Another approach would be to use a smooth weighted average of the metrics:

$$MT(\mathbf{x}) = \sum_{i=1}^{R} w_i(\mathbf{x}) M_{RP}^{\mathbf{x}_{\mathrm{RP}_i}} \tag{19}$$

$$w_i(\mathbf{x}) = \frac{e^{-\frac{1}{\sigma}\mathrm{Dist}_{\mathrm{Global}}(\mathbf{x}, \mathbf{x}_{\mathrm{RP}_i})}}{\sum_{j=1}^{R} e^{-\frac{1}{\sigma}\mathrm{Dist}_{\mathrm{Global}}(\mathbf{x}, \mathbf{x}_{\mathrm{RP}_j})}}. \tag{20}$$

In our experiments, we varied σ between .01 and .001. Given a fixed $\mathrm{Dist}_{\mathrm{Global}}$, we can compute $w_{ij} = w_j(\mathbf{x_i})$. Interpreting these values as the probability that training point $\mathbf{x_i}$ selects metric $M_{RP}^{\mathbf{x}_{\mathrm{RP}_j}}$, we can learn each $M_{RP}^{\mathbf{x}_{\mathrm{RP}_j}}$ so as to minimize expected loss by optimizing (5) and (6) with weights w_{ij}. One could define a weighted version of (10) that simultaneously learns all reference metrics. Since this involves a sum over the cross product of all imposter triples with all reference points, we chose not to do this.

Empirically we found that the smooth weighted average metric in Equation (19) significantly outperformed the nearest neighbor algorithm. The computational cost of computing the smooth weighted average metric in Equation (19) is significantly more, however. Below, we describe a different way of training the metrics at the reference points that eliminates this difference.

We hypothesized that the smooth interpolation outperformed NN interpolation because the smoothing acted as an additional regularizer. To achieve the same regularization for NN interpolation, we applied a cross-validation step to construct a regularized version of $M_{RP}^{\mathbf{x}_{\mathrm{RP}_i}}$

$$M_{CV}^{\mathbf{x}_{\mathrm{RP}_j}} = \frac{1}{Z_j} \sum_{i \neq j} w_i(\mathbf{x}) M_{RP}^{\mathbf{x}_{\mathrm{RP}_i}} \tag{21}$$

where Z_j is a normalization factor included to ensure the sum of weights $w_i(\mathbf{x})$ for $j \neq i$ is 1. We found NN interpolation with the above cross-validated metrics performed similarly to the smooth interpolation, but was significantly faster. As such, our interpolation results reported in our experiments are generated with NN interpolation with $M_{CV}^{\mathbf{x}_{\mathrm{RP}_j}}$.

So far we have described the interpolation of the reference point metrics for the test point $\mathbf{x}$. The same interpolation could be performed anywhere in the feature space. For example, it could be performed at the training points $\mathbf{x}_i$. This leads to a space-efficient version of Per-Exemplar. It can also be used along the lines between the test point $\mathbf{x}$ and the training points $\mathbf{x}_i$ an in the Line Integral algorithm described in Section 2.3.5 and Appendix A.

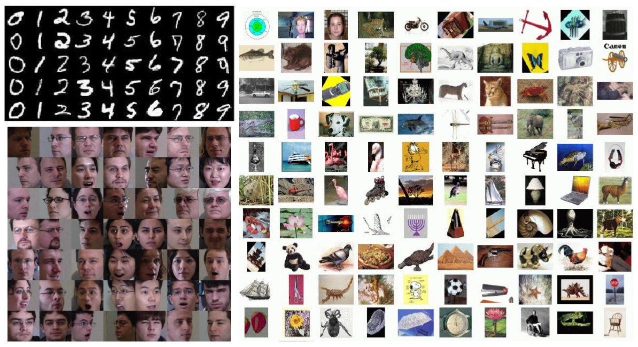
Figure 1. Benchmark databases. Bottom Left: MultiPIE [12]. Right: Caltech 101 [5]. Top Left: MNIST [18].

3. Experiments

3.1. Databases and Evaluation Procedure

We compared the various algorithms on a diverse set of problems (see Figure 1), including object recognition using face recognition using MultiPIE [12], Caltech 101 [5], and digit recognition using MNIST [18].

MultiPIE [12] is a larger version of the CMU PIE database [23] that includes over 700,000 images of over 330 subjects collected over multiple sessions over around 6 months. We extract the face regions automatically in the 5 frontal-most cameras (-45 degrees to +45 degrees) using the Viola-Jones face detector [27]. The face regions were then resampled and normalized to 80×80 grayscale patches with zero mean and unit variance. The patches were then projected into their first 254 principle components (95% of the empirical variance.) Different sessions were used as training and testing data. We randomly generated 10 batches of 1000 examples as our test sets.

Caltech 101 [5] is a widely used benchmark for image classification consisting of over 9,000 images from 101 object categories. We base our results on the widely-used single-cue baseline of spatial pyramid kernels [17]. We use the publically available feature computation code from the author's website, which generates a sparse feature vector of visual word histograms for each image. We project the vectors such that 95% of the variance is captured, and then use them in our local distance function framework. We follow the established protocol of leave-one-out cross validation with 30 training examples per class.

MNIST [18] is a well-studied dataset of 70,000 examples of digit images. We followed the pre-processing steps outlined by [15]. The original 28×28 dimensional images were deskewed and projected into their leading 164 principle components which were enough to capture 95% of the data variance. The normal MNIST train/test split [18] has two problems. (1) The training data is very dense with error rates around 1-2% making it hard to see any difference in performance. (2) There is only one partition making it impossible to estimate confidence intervals. To rectify these problems, we sub-sampled both the train and test data, in 10 batches of 1000 examples each.

To speed up run-time search, it is common to find a *short-list* of neighbors of a test point using an efficient distance function, before scoring a more computationally demanding local function. Many previous approaches using a per-exemplar distance [7, 6] do this so as to keep run-times reasonable. We follow a similar approach for all our algorithms. We score the local distance function over a set of 50 neighbors computed with the global metric.

To eliminate any dependence on a specific indexing structure, both our recognition rate and timing results were obtained using a brute-force linear scan. Preliminary results using a KD-tree to generate the short-list showed that the relative performance of the algorithms was unaffected. Also, for many algorithms the computation time is dominated by the local computation anyway. The development of efficient indexing structures specially tuned for local distance functions is left as an area for future work.

3.2. Results

We include the full results of our study in a set of webpages in the supplemental material. We include: (1) recognition rates, (2) error rates, (3) percentage reduction in the error rate, (4) computation times, and (5) standard deviations of all of these measures. We include results for (1) LDA, (2) LMNN metric learning, and (3) the hybrid algorithm. A subset of the results are included in Table 2. Due to respace restrictions, we only include recognition rate and computation time results for all variants of the hybrid algorithm because overall it performs the best.

3.3. Discussion of the Results

How: Comparing the results in Table 2 with those in the supplemental material, overall we found that the hybrid algorithm outperforms LDA and LMNN metric learning. Metric-learning approaches such as LMNN tend to outperform simpler dimensionality reduction schemes such as LDA when given enough data. Overfitting becomes more of an issue when learning local metrics.

When: Overall, we found the best approach to be to learn local metrics online at the test point, as in [13, 1] We hypothesize two possible explanations. (1) Overfitting is less of an issue when the metric is being estimated at the test point. (2) Just a single metric is being used, avoiding any need to learn multiple per-exemplar metrics in a way such that they are comparable or normalized appropriately [7, 15].

Where: In applications where the computational cost of learning the metric online at the test point is too great, there are two reasonable alternatives. (1) Use the interpolation algorithm to estimate the metric at the test point. (2) Use a per-class metric. Both of these algorithms perform slightly

MultiPIE Hybrid Recognition Rate

	Global	Class	Train	Test	Line
Offline	59.2 ± 1.9	65.6 ± 1.4	N/A	N/A	N/A
Online	59.2 ± 1.9	65.6 ± 1.4	68.9 ± 1.2	**70.2 ± 1.4**	N/A
Interp.	59.2 ± 1.9	65.6 ± 1.4	65.4 ± 1.5	65.4 ± 1.5	65.4 ± 1.5

Time (seconds)

Global	Class	Train	Test	Line
3.5 ± 0.0	12.6 ± 0.1	N/A	N/A	N/A
-	-	1219 ± 12	**63.3 ± 0.4**	N/A
-	-	14.3 ± 0.2	22.0 ± 0.1	184.5 ± 1.6

Caltech Hybrid Recognition Rate

	Global	Class	Train	Test	Line
Offline	51.1 ± 4.7	55.2 ± 4.9	N/A	N/A	N/A
Online	51.1 ± 4.7	55.2 ± 4.9	58.3 ± 3.2	**60.1 ± 4.2**	N/A
Interp.	51.1 ± 4.7	55.2 ± 4.9	53.0 ± 4.0	52.0 ± 3.9	52.6 ± 4.3

Time (seconds)

Global	Class	Train	Test	Line
0.1 ± 0.0	0.3 ± 0.0	N/A	N/A	N/A
-	-	223.7 ± 29.0	**11.3 ± 1.3**	N/A
-	-	0.7 ± 0.0	1.6 ± 0.1	40.3 ± 6.9

Sampled MNIST Hybrid Recognition Rate

	Global	Class	Train	Test	Line
Offline	92.4 ± 1.2	**93.8 ± 0.9**	N/A	N/A	N/A
Online	92.4 ± 1.2	93.8 ± 0.9	92.9 ± 0.8	93.7 ± 0.8	N/A
Interp.	92.4 ± 1.2	93.8 ± 0.9	92.6 ± 0.7	92.5 ± 0.7	92.6 ± 0.8

Time (seconds)

Global	Class	Train	Test	Line
0.1 ± 0.0	**0.5 ± 0.0**	N/A	N/A	N/A
-	-	311.1 ± 3.1	15.7 ± 0.5	N/A
-	-	3.1 ± 0.2	5.3 ± 0.2	97.8 ± 1.6

Table 2. Recognition rate and computation time results for all variants of the hybrid algorithm (Section 2.2.3.) We do not include timing results for online/interpolated global/class metrics because an offline implementation is more efficient. The results for LDA and LMNN metric learning, as well as error rates and percentage reduction in error rates, are included in the supplemental material. Overall, online estimation of the metric tensor at the test point in our hybrid framework performed the best.

worse than online estimation at the test point, but are far more efficient. For MNIST, a per-class metric slightly performed the best, consistent with the results from [15].

We found that there was little to be gained by integrating the metric tensor along the line between the training and test points. While this is a negative result and therefore primarily of theoretical interest, it is still important to show that little can be gained by such an approach.

Regularization: Overall, we found the generalization ability of the learned metrics to be a recurring key issue. Local metrics are learned using a local subset of the training data, and so overfitting is a fundamental difficulty. As a result, regularized LDA was often competitive with LMNN. We obtained good results when using LMNN to train a global model but found LDA was generally better at estimating the local metrics (e.g. in the hybrid algorithm.)

Previous reported results: Our MultiPIE results are comparable to those reported in [12], but are obtained automatically without manually marked fiducial locations. Our score of 60.1 on Caltech is comparable to the score of 62.2 we obtained by running the author's code from [17], which itself is slightly below the reported value of 64.6. We hypothesize the chi-squared kernel from [17] performs a form of local weighting. Our MNIST results are not directly comparable to previous work because we use subsampled data, but our class-specific learned-metric baseline produces the state-of-the-art MNIST results on the full train/test split [15]. We verified that the author's code reproduced the reported results, and ran the same code on random train/test splits. Overall, though our benchmark results do not always advance the state-of-the-art, our evaluation clearly reveals the benefit of local approaches over global baselines.

4. Conclusion

We have presented a taxonomy of local distance functions in terms of **how**, **where**, and **when** they estimate the metric tensor and approximate the geodesic distance. We have extended this taxonomy along all three axis. We performed a thorough evaluation of the full combination of algorithms on 3 large scale, diverse datasets. Overall, the main conclusions are that the hybrid, online estimation of the metric at the test point performs the best. Such a method is straightforward to implement, reasonably efficient, and provided a consistent improvement over global methods across all our datasets. We will make all our feature vectors and results freely available on a website to further research in this field.

References

[1] C. Atkeson, A. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:11–73, 1997.

[2] A. Bar-Hillel, T. Hertz, N. Shental, and D. Weinshall. Learning and mahalanobis metric from equivalence constraints. *Journal of Machine Learning Research*, 6:937–965, 2005.

[3] O. Boiman, E. Shechtman, and M. Irani. In defense of nearest-neighbor based image classification. In *CVPR*, 2008.

[4] C. Domeniconi, J. Peng, and D. Gunopulos. Locally adaptive metric nearest-neighbor classification. *IEEE PAMI*, 24(9):1281–1285, 2002.

[5] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. In *IEEE CVPR, Workshop on Generative-Model Based Vision*, 2004.

[6] A. Frome, Y. Singer, and J. Malik. Image retrieval and classification using local distance functions. In *NIPS*, 2007.

[7] A. Frome, Y. Singer, F. Sha, and J. Malik. Learning globally-consistent local distance functions for shape-based image retrieval and classification. In *ICCV*, 2007.

[8] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press Professional, Inc., 1990.

[9] J. Goldberger, S. Roweis, and R. Salakhutdinov. Neighborhood components analysis. In *NIPS*, 2005.

[10] M. Gonen and E. Alpaydin. Localized multiple kernel learning. In *ICML*, 2008.

[11] K. Grauman and T. Darrell. Efficient learning with sets of features. *Journal of Machine Learning Research*, 8:725–760, 2007.

[12] R. Gross, I. Matthews, J. Cohn, T. Kanade, and S. Baker. Multi-PIE. In *Proceedings of the IEEE International Conference on Automatic Face and Gesture Recognition*, 2008.

[13] T. Hastie and R. Tibshirani. Discriminant adaptive nearest neighbor classification. *PAMI*, 18(6):607–616, 1996.

[14] M. Kumar, P. Torr, and A. Zisserman. An invariant large margin nearest neighbour classifier. In *ICCV*, 2007.

[15] K.Weinbeger and L. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 2009. To appear.

[16] F. Labelle and J. Shewchuk. Anisotropic voronoi diagrams and guaranteed-quality anisotropic mesh generation. In *Proceedings of the 19th Annual Symposium on Computational Geometry*, 2003.

[17] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006.

[18] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[19] S. Mahamud and M. Hebert. The optimal distance measure for object detection. In *CVPR*, 2003.

[20] T. Malisiewicz and A. Efros. Recognition by associatino via learning per-exemplar distances. In *CVPR*, 2008.

[21] E. Rosch and C. Mervis. Family resembleances: Studies in the internal structure of categories. *Cognitive Psychology*, 7:573–605, 1975.

[22] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter-sensitive hashing. In *ICCV*, pages 750–757, 2003.

[23] T. Sim, S. Baker, and M. Bsat. The cmu pose, illumination, and expression (pie) database. In *Proceedings of the IEEE International Conference on Automatic Face and Gesture Recognition*, May 2002.

[24] P. Simard, Y. LeCun, and J. Denker. Efficient pattern recognition using a new transformation distance. In *NIPS*, 1993.

[25] L. Torresani and K. Lee. Large margin component analysis. In *NIPS*, 2006.

[26] R. Urtasun and T. Darrell. Local probabilistic regression for activity-independent human pose inference. In *CVPR*, 2008.

[27] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001.

[28] X. Wang and X. Tang. Random sampling LDA for face recognition. In *Computer Vision and Pattern Recognition*, volume 2, 2004.

[29] H. Zhang, A. Berg, M. Maire, and J. Malik. Svm-knn: Discriminative nearest neighbor classification for visual category recognition. In *CVPR*, 2006.

A. Polynomial Time Line Integral Algorithm

We now present an exact polynomial-time algorithm to compute the integral in Equation (16) along the line in Equation (15) under the assumption that the metric tensor is piecewise constant, and using the interpolation algorithm in Section 2.4.3. This should be compared with the exponential cost in the dimension of an (approximate) distance transform that splits the space into a regular grid of voxels.

We assume that the metric tensor has been sampled at the reference points to estimate $M_{RP}^{\mathbf{x}_{\mathrm{RP}_i}}$. We also assume that the metric tensor is piecewise constant and its value at any given point $\mathbf{x}$ is the value at the reference point which is closest, as measured by that reference points metric:

$$MT(\mathbf{x}) \;=\; M_{RP}^{\mathbf{x}_{\mathrm{RP}_i}} \tag{22}$$

where for all $j = 1, \ldots, R$:

$$(\mathbf{x}-\mathbf{x}_{\mathrm{RP}_i})^{\mathrm{T}} M_{RP}^{\mathbf{x}_{\mathrm{RP}_i}} (\mathbf{x}-\mathbf{x}_{\mathrm{RP}_i}) \;\le\; (\mathbf{x}-\mathbf{x}_{\mathrm{RP}_j})^{\mathrm{T}} M_{RP}^{\mathbf{x}_{\mathrm{RP}_j}} (\mathbf{x}-\mathbf{x}_{\mathrm{RP}_j}). \tag{23}$$

For any pair of reference points i and j we know that the metric tensor is constant in the region defined by Equation (23); i.e. the space can be divided into a set of $R(R-1)/2$ regions where the metric tensor is constant. This division into regions is similar to the "anisotropic voronoi diagram" of [16]. The voronoi diagram will actually have less regions in the case where there is another reference point $\mathbf{x}_{\mathrm{RP}_k}$ that is closer than either $\mathbf{x}_{\mathrm{RP}_i}$ or $\mathbf{x}_{\mathrm{RP}_j}$. This oversegmentation of the space does not affect the accuracy of the estimate of the the integral in Equation (16).

To compute the integral in Equation (16) we need to break the domain $\lambda \in [0,1]$ into the segments for which it is constant. The boundaries of the regions in Equation (23) are defined by the equalities:

$$(\mathbf{x}-\mathbf{x}_{\mathrm{RP}_i})^{\mathrm{T}} M_{RP}^{\mathbf{x}_{\mathrm{RP}_i}} (\mathbf{x}-\mathbf{x}_{\mathrm{RP}_i}) \;=\; (\mathbf{x}-\mathbf{x}_{\mathrm{RP}_j})^{\mathrm{T}} M_{RP}^{\mathbf{x}_{\mathrm{RP}_j}} (\mathbf{x}-\mathbf{x}_{\mathrm{RP}_j}) \tag{24}$$

Equation (24) is a quadratic and in general the solution is a quadratic hyper-surface. The intersection of this boundary hypersurface and the line between the test and training points in Equation (16) can be computed by substituting Equation (16) into Equation (24). The result is a quadratic in the single unknown λ which can be solved to give either $0, 1$ or 2 solutions in the range $[0,1]$.

Once all the solutions that lie in $\lambda \in [0,1]$ have been computed, they can be sorted. This breaks the line in Equation (16) into a number of segements where the metric tensor is constant on each one. The appropriate value of the metric tensor is then computed for each such segment by taking the midpoint of the segment and computing the closest reference point using Equation (23). The lengths of all the segments can then be computed and added up. In total, there can be at most $2R(R-1)/2$ intersection points. Computing each one takes time $\mathrm{O}(K^2)$. There are at most $1 + R(R-1)$ segments of the line between the test and training points. The search for the closest reference point for each one takes $\mathrm{O}(RK^2)$ because there are R reference points and the cost of computing the distance to it is $\mathrm{O}(K^2)$. The total computation cost is therefore $\mathrm{O}(R^3K^2)$.