Fast Similarity Search for Learned Metrics

Prateek Jain Brian Kulis Kristen Grauman
Department of Computer Sciences
University of Texas at Austin
{pjain, kulis, grauman}@cs.utexas.edu

UTCS Technical Report TR-07-48

September 14, 2007

Abstract

We propose a method to efficiently index into a large database of examples according to a learned metric. Given a collection of examples, we learn a Mahalanobis distance using an information-theoretic metric learning technique that adapts prior knowledge about pairwise distances to incorporate similarity and dissimilarity constraints. To enable sub-linear time similarity search under the learned metric, we show how to encode a learned Mahalanobis parameterization into randomized locality-sensitive hash functions. We further formulate an indirect solution that enables metric learning and hashing for sparse input vector spaces whose high dimensionality make it infeasible to learn an explicit weighting over the feature dimensions. We demonstrate the approach applied to systems and image datasets, and show that our learned metrics improve accuracy relative to commonly-used metric baselines, while our hashing construction permits efficient indexing with a learned distance and very large databases.

1 Introduction

The success of any distance-based indexing, clustering, or classification scheme depends critically on the quality of the chosen distance metric, and the extent to which it accurately reflects the true underlying relationships between the examples in a particular data domain. An optimal distance metric should report small distances for examples that are similar in the parameter space of interest (or that share a class label), and conversely should report large distances for examples that are unrelated. General-purpose measures, such as ℓ_p norms, are simple to compute and are amenable to existing fast search methods, but they are not necessarily well-suited for all learning problems with a given data representation. This is especially of concern when one might want the learner to tend to different aspects of similarity for different problems with the same representation. For example, given a collection of face images, in one scenario we might want a distance function that considers images to be close if they contain the same person, whereas in another case we would prefer that proximity reflect the similarity of two facial expressions.

Recent advances in metric learning make it possible to learn distance or kernel functions that are more effective for a given problem, provided some partially labeled data or constraints are available [29, 28, 13, 3, 7, 11]. By taking advantage of the prior information, these techniques offer improved accuracy when indexing or classifying examples. Analyzing large volumes of data is of great interest in computational biology, vision, and natural language processing, making fast similarity search with learned metrics an important challenge.

However, previous approaches to metric learning have limited applicability to very large data sets, since the specialized learned distance functions preclude the direct use of known efficient search techniques. Data structures for efficient exact search are known to be ineffective for high-dimensional spaces and can (depending on the data distribution) degenerate to brute force search [10, 27]; approximate search methods can guarantee sub-linear time performance but are defined for general-purpose metrics, such as the Hamming distance [18], ℓ_p norms [6], or an inner product [4]. As such, in order to find the most similar examples for a given input, metric learning approaches are currently (in the worst case) forced to exhaustively scan all previously seen examples to evaluate their similarity to the input under the learned metric; likewise, in order to cluster a collection of data, all pairwise distances are required. This is a limiting factor that thus far prevents the use of metric learning with very large data sets.

In this work we introduce a method for fast approximate similarity search with learned metrics. We show how to construct randomized hash functions that integrate knowledge attained from partially labeled data or paired constraints, so that examples may be efficiently indexed or grouped according to the learned metric without resorting to a naive exhaustive scan of all items. We present a straightforward solution for the case of relatively low-dimensional input vector spaces, and further derive a solution to accommodate very high-dimensional but sparse data for which explicit input space computations for both metric learning and hashing are infeasible. The former contribution makes fast indexing accessible for numerous existing metric learning methods [29, 28, 13, 3, 7], while the latter is of particular interest for commonly used representations in text processing and vision. We demonstrate our approach by learning metrics and hash functions for systems and image datasets, and analyze its performance with classification tasks.

2 Related Work

Recent work has yielded various approaches to the metric learning problem, including several techniques to learn a combination of existing kernels given partially labeled data [19, 5], as well as a number of formulations for learning the parameterization of a Mahalanobis metric given some number of class labels or paired constraints [29, 28, 23, 13, 3, 7]. Embedding functions can be useful both to capture (as closely as possible) a desired set of provided distances between points, as well as to provide an efficient approximation for a known but computationally expensive distance function of interest [22, 1, 16]. However, in contrast to learned metrics, such geometric embeddings are meant to mirror a fixed distance function and do not adapt to reflect supervised constraints.

Xing et al. learn a Mahalanobis metric for k-means clustering by using semidefinite programming to minimize the sum of squared distances between similarly labeled examples, while requiring a certain lower bound on the distances between examples with different labels [29]. In related techniques, Globerson and Roweis [13] constrain within-class distances to be zero and maximize between-class distances [13], while Weinberger et al. formulate the problem in a large-margin k-nearest-neighbors setting [28]. In addition to using labeled data, research has shown how metric learning can proceed with weaker supervisory information, such as equivalence constraints [3], or relative constraints [23, 11]. For example, equivalence constraints are exploited in the Relevant Component Analysis method of Bar-Hillel et al. [3]; the Support Vector Machine-based approach of Schultz and Joachims [23] incorporates relative constraints over triples of examples, and is extended by Frome et al. to learn example-specific local distance functions [11]. Davis et al. develop an information-theoretic approach that accommodates any linear constraints on pairs of examples, and provide an efficient optimization solution that forgoes expensive eigenvalue decomposition [7].

Multi-dimensional scaling [8], Locally-Linear Embeddings [22], and IsoMap [26] provide ways to capture known distances in a low-dimensional space, and provably low-distortion geometric embeddings have also been explored (e.g., [2]). The BoostMap approach of Athitsos et al. learns efficient Euclidean-space embeddings that preserve proximity as dictated by useful expensive distance measures [1].

In order to efficiently index multi-dimensional data, data structures based on spatial partitioning and recursive hyperplane decomposition have been developed, including $k-d$-trees [10] and metric trees [27]. Some such data structures support the use of arbitrary metrics. However, while their expected query time performance may be logarithmic in the database size, selecting useful partitions can be expensive and requires good heuristics; worse, particular data distributions can result in a brute force search, and in high-dimensional spaces all exact search methods are known to provide little improvement over a naive linear scan [18].

As such, researchers have considered the problem of *approximate* similarity search, where a user is afforded explicit tradeoffs between the guaranteed accuracy versus speed of a search. Several randomized approximate search algorithms have been developed that allow even high-dimensional data to be searched in time sub-linear in the size of the database [18, 4, 6]. Indyk and Motwani [18] and Charikar [4] propose locality-sensitive hashing (LSH) techniques to index examples in Hamming space in sub-linear time, and Datar et al. extend LSH for ℓ_p norms in [6]. Data-dependent variants of LSH have also been suggested. Georgescu et al. select space partitions in a data-driven manner, in an effort to use more meaningful hash functions for a given data distribution [12],

In contrast to previous work, we address the problem of sub-linear time approximate similarity search for a learned metric. While randomized algorithms like LSH have been employed heavily to mitigate the time complexity of identifying similar examples [24], their use has been restricted to generic measures for which the appropriate hash functions are already defined, i.e., direct application to learned metrics was not possible. We instead devise a method that allows knowledge attained from partially labeled data or paired constraints to be incorporated into the hash functions, without any additional loss in accuracy relative to the learned metric beyond the quantified loss induced by the approximate search technique.

The goals of Shakhnarovich et al. [25] are most similar to ours, since they also address both the need to compare examples according to their "hidden" parameters as well as the need to search for similar examples in large databases very quickly. However, while they have shown the advantage of applying hash functions to feature dimensions that most reveal parameter-space similarity, any examples indexed must be sorted according to the input space (non-learned) distance. Our approach offers a seamless integration of a learned metric, in that examples are both hashed and sorted according to learned constraints. In addition, the proposed method stands to benefit several existing methods for metric learning.

3 Approach

The main idea of our approach is to learn a parameterization of a Mahalanobis metric based on whatever labels or paired constraints are provided for some training examples, and then to encode the learned information into randomized hash functions. These functions will guarantee that the more similar inputs are under the learned metric, the more likely they are to collide in a hash table. After constructing hash tables containing all of the initial training (database) examples, examples similar to a new instance are found in sub-linear time in the size of the database by evaluating the learned metric between the new example and any examples with which it shares a hash bucket. In the following we will first give a brief background of the metric learning approach we employ, and then we describe how to incorporate that metric into the hash functions, either for low-dimensional (Section 3.2.1) or sparse high-dimensional data (Section 3.2.2).

3.1 Information-Theoretic Metric Learning

As in [29, 28, 13, 7, 23, 3], we are given n points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, with all $\boldsymbol{x}_i \in \Re^d$, and wish to compute a positive-definite (p.d.) $d \times d$ matrix A to parameterize the squared Mahalanobis distance:

$$d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i - \boldsymbol{x}_j)^T A (\boldsymbol{x}_i - \boldsymbol{x}_j), \tag{1}$$

for all $i, j = 1, \ldots, n$. Note that a generalized inner product (kernel) measures the pairwise similarity associated with that distance: $s_A(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T A \boldsymbol{x}_j$. The Mahalanobis distance is often used with A as the inverse of the sample covariance when data is assumed to be Gaussian, or with A as the identity matrix if the squared Euclidean distance is suitable. Given a set of inter-point distance constraints, however, we can directly learn a matrix A to yield a measure that is more accurate for a given classification or clustering problem. We adopt the information-theoretic metric learning method of Davis et al. [7], since it is efficient, flexible in terms of the constraint specification, and fares well empirically for classification and clustering tasks.

Given an initial $d \times d$ p.d. matrix A_0 specifying prior knowledge about inter-point distances, the learning task is posed as an optimization problem that minimizes the LogDet divergence between matrix A_0 and A, subject to a set of constraints specifying pairs of examples that are similar or dissimilar. The LogDet divergence is a Bregman divergence and is defined over the cone of positive definite matrices:

$$D_{\ell \mathrm{d}}(A, A_0) = \mathrm{tr}(AA_0^{-1}) - \log\det(AA_0^{-1}) - d.$$

Given pairs of similar points $\mathcal{S}$ and dissimilar points $\mathcal{D}$, we will require that $d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq u$ for a small value of u for all $(i,j) \in \mathcal{S}$, and likewise that $d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq l$ for a sufficiently large value of l for all $(i,j) \in \mathcal{D}$. In semi-supervised multi-class settings, the constraints are taken directly from the provided labels: points in the same class must be similar, points in different classes are constrained to be dissimilar.

To compute A, the LogDet divergence is minimized while enforcing the desired constraints:

$$\begin{aligned} \min_{A \succeq 0} \quad & D_{\ell \mathrm{d}}(A, A_0) \\ \text{s. t.} \quad & d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq u \quad (i,j) \in \mathcal{S}, \\ & d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq \ell \quad (i,j) \in \mathcal{D}. \end{aligned} \tag{2}$$

In order to guarantee the existence of a feasible A, slack variables may be introduced into the above. The optimal solution is obtained via repeated Bregman projections that project the current solution onto a single constraint. The update to A_{t+1} from A_t is given by:

$$A_{t+1} = A_t + \beta_t A_t (\boldsymbol{x}_{i_t} - \boldsymbol{x}_{j_t})(\boldsymbol{x}_{i_t} - \boldsymbol{x}_{j_t})^T A_t, \tag{3}$$

where $\boldsymbol{x}_{i_t}$ and $\boldsymbol{x}_{j_t}$ are the constrained data points for iteration t, and β_t is a projection parameter computed by the algorithm. When the dimensionality of the data is very high, we cannot explicitly work with A, and so the update in (3) cannot be performed. However, we may still implicitly update the Mahalanobis matrix A via updates in kernel space for an equivalent kernel learning problem in which $K = X^T A X$ for $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]$. If K_0 is an input kernel matrix of the data, the appropriate update is:

$$K_{t+1} = K_t + \beta_t K_t (\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^T K_t, \tag{4}$$

where the vectors $\boldsymbol{e}_{i_t}$ and $\boldsymbol{e}_{j_t}$ refer to the i_t-th and j_t-th standard basis vectors, respectively, and the projection parameter β_t is the same as in (3). See [7] for details on the algorithm.

In Section 3.2.1 we will use the A that results at convergence to modify random hash functions, in the event that A_t can be manipulated directly. In Section 3.2.2 we derive an implicit formulation that enables information-theoretic learning with high-dimensional sparse inputs for which A_t cannot be explicitly represented.

3.2 Locality-Sensitive Hash Functions for Learned Metrics

A locality-sensitive hashing scheme is a distribution on a family $\mathcal{F}$ of hash functions operating on a collection of objects, such that for two objects $\boldsymbol{x}, \boldsymbol{y}$,

$$\Pr_{h \in \mathcal{F}} [h(\boldsymbol{x}) = h(\boldsymbol{y})] = sim(\boldsymbol{x}, \boldsymbol{y}), \tag{5}$$

where $sim(\boldsymbol{x}, \boldsymbol{y})$ is some similarity function defined on the collection of objects [4, 18]. When $h(\boldsymbol{x}) = h(\boldsymbol{y})$, $\boldsymbol{x}$ and $\boldsymbol{y}$ collide in the hash table. Because the probability that two inputs collide is equal to the similarity between them, highly similar objects are indexed together in the hash table with high probability. Existing LSH functions can accommodate the Hamming distance [18], ℓ_p norms [6], and inner products [4].

In the following we introduce hash functions that can accommodate a learned Mahalanobis distance, where we want to retrieve examples $\boldsymbol{x}_i$ for an input $\boldsymbol{x}_q$ for which the value $d_A(\boldsymbol{x}_i, \boldsymbol{x}_q)$ resulting from (1) is small, or, in terms of the kernel form, for which the value of $s_A(\boldsymbol{x}_i, \boldsymbol{x}_q) = \boldsymbol{x}_q^T A \boldsymbol{x}_i$ is high.

3.2.1 Explicit Formulation

Given the matrix A for a metric learned as above[1], such that $A = G^T G$, we generate the following randomized hash functions $h_{\boldsymbol{r},A}$ which accept an input point and return a binary hash key bit:

$$h_{\boldsymbol{r},A}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{r}^T G \boldsymbol{x} \geq 0 \\ 0, & \text{otherwise} \end{cases}, \tag{6}$$

where the vector $\boldsymbol{r}$ is chosen at random from a d-dimensional Gaussian distribution with zero mean and unit variance. This construction leverages earlier results showing that (1) the probability of two unit vectors having a dot products with random vector $\boldsymbol{r}$ that are opposite in sign is proportional to the angle between them [14], and (2) the sign of $\boldsymbol{r}^T \boldsymbol{x}_i$ is therefore a locality-sensitive function for the inner product of any two inputs $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ [4].

Thus by parameterizing the hash functions instead by G (which is computable since A is p.d.), we obtain the following relationship:

$$\Pr\left[h_{\boldsymbol{r},A}(\boldsymbol{x}_i) = h_{\boldsymbol{r},A}(\boldsymbol{x}_j)\right] = 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{\boldsymbol{x}_i^T A \boldsymbol{x}_j}{\sqrt{|G\boldsymbol{x}_i||G\boldsymbol{x}_j|}}\right), \tag{7}$$

which sustains the requirement of (5) for a learned Mahalanobis metric, whether A is computed using the method of [7] or otherwise [29, 28, 13, 23, 3]. Essentially we have shifted the random hyperplane $\boldsymbol{r}$ according to A, and by factoring it by G we allow the random hash function itself to "carry" the information about the learned metric. Note that (6) assumes that the input dimension d is low enough that A can be explicitly handled in memory, allowing the updates in (3). In Section 3.3 we describe how indexing proceeds from these hash keys.

3.2.2 Implicit Formulation

We are also interested in the case where the dimensionality d may be very high, but the examples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ are sparse and therefore representable. For example, bag-of-words text representations for large corpora [21] or multi-dimensional multi-resolution histogram representations used in vision [15, 16] are prime situations where one must work with a very high-dimensional but sparse feature space. Even though the examples are each sparse, *the matrix A can be dense*, with values for each dimension. Therefore, in this setting A cannot be explicitly represented, so the update in (3) and hash functions in (6) are infeasible to compute. Thus, in the following we show how to achieve efficient hashing without manipulating A directly.

We denote high-dimensional sparse inputs by $\phi(\boldsymbol{x})$ to mark their distinction from the dense inputs $\boldsymbol{x}$ handled in Section 3.2.1. In this case, we assume that implicitly $A_0 = I$. As in the explicit formulation above, the goal is to wrap G into the hash function, but now we must do so without working directly with G. In the following, we will show that an appropriate hash function $h_{\boldsymbol{r},A}$ for inputs $\phi(\boldsymbol{x})$ can be defined as:

$$h_{\boldsymbol{r},A}(\phi(\boldsymbol{x})) = \begin{cases} 1, & \text{if } \boldsymbol{r}^T \phi(\boldsymbol{x}) + \sum_{i=1}^{c} \gamma_i^r \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}) \geq 0 \\ 0, & \text{otherwise} \end{cases}, \tag{8}$$

[1] In all further notation, a variable without an iteration t subscript denotes its value after convergence, thus here we are referring to A once (3) converges.

where c is the total number of constrained data points in $\mathcal{S} \bigcup \mathcal{D}$, $\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x})$ is the original kernel function value between constrained example $\boldsymbol{x}_i$ and the input $\boldsymbol{x}$, and γ_i^r are coefficients computed once (offline) during metric learning, and will be defined below. Note that while G is dense and therefore not manageable, $\boldsymbol{r}$ is by definition as sparse as all $\phi(\boldsymbol{x})$ inputs, and we need to generate random values for the nonzero entries only. Thus $\boldsymbol{r}^T\phi(\boldsymbol{x})$ is practical to compute.

Next we present a construction to express G in terms of the constrained data points, and a method to compute (8) efficiently. Our construction relies on two technical lemmas which we prove in this section. Recall the update rule for A from (3): $A_{t+1} = A_t + \beta_t A_t \boldsymbol{v}_t \boldsymbol{v}_t^T A_t$, where β_t is the projection parameter for iteration t, and $\boldsymbol{v}_t$ equals $\phi(\boldsymbol{x}_{i_t}) - \phi(\boldsymbol{x}_{j_t})$ if points i and j correspond to the distance constraint under consideration at iteration t. Since A_t is positive semi-definite, we can factorize it as $A_t = G_t^T G_t$, which allows us to rewrite the update as:

$$A_{t+1} = G_t^T(I + \beta_t G_t \boldsymbol{v}_t \boldsymbol{v}_t^T G_t^T)G_t.$$

As a result, if we factorize $I + \beta_t G_t \boldsymbol{v}_t \boldsymbol{v}_t^T G_t^T$, we can derive an explicit update for G_{t+1}:

$$G_{t+1} = (I + \beta_t G_t \boldsymbol{v}_t \boldsymbol{v}_t^T G_t^T)^{1/2} G_t = (I + \alpha_t G_t \boldsymbol{v}_t \boldsymbol{v}_t^T G_t^T)G_t, \tag{9}$$

where the second equality follows from Lemma 1 using $\boldsymbol{y} = G_t\boldsymbol{v}_t$, and α_t is defined accordingly.

Lemma 1. *Let $B = I + \beta \boldsymbol{y}\boldsymbol{y}^T$ be positive semi-definite. Then $B^{1/2} = I + \alpha \boldsymbol{y}\boldsymbol{y}^T$, with $\alpha = (\pm\sqrt{1 + \boldsymbol{y}^T\boldsymbol{y}\beta} - 1)/\boldsymbol{y}^T\boldsymbol{y}$.*

Proof. Consider $(I + \alpha\boldsymbol{y}\boldsymbol{y}^T)^2$. Expanding yields $I + 2\alpha\boldsymbol{y}\boldsymbol{y}^T + \alpha^2(\boldsymbol{y}^T\boldsymbol{y})\boldsymbol{y}\boldsymbol{y}^T = I + (2\alpha + \alpha^2\boldsymbol{y}^T\boldsymbol{y})\boldsymbol{y}\boldsymbol{y}^T$. For the lemma to hold, we require that $(I + \alpha\boldsymbol{y}\boldsymbol{y}^T)^2 = I + \beta\boldsymbol{y}\boldsymbol{y}^T$, and this holds when $2\alpha + \alpha^2\boldsymbol{y}^T\boldsymbol{y} = \beta$. Solving this quadratic equation for α, we obtain the desired result. Furthermore, α is real-valued: the eigenvalues of B are 1 and $1 + \beta\boldsymbol{y}^T\boldsymbol{y}$, which is greater than or equal to 0 since B is positive semi-definite. Thus $\sqrt{1 + \boldsymbol{y}^T\boldsymbol{y}\beta}$ is real, and so α is real. $\square$

Let $\Phi = [\phi(\boldsymbol{x}_1)\phi(\boldsymbol{x}_2)...\phi(\boldsymbol{x}_c)]$ be the $d \times c$ matrix of all c constrained data points, and let $K_0 = \Phi^T\Phi$ be the kernel matrix for all constrained input pairs. We now prove the following lemma:

Lemma 2. *For all t, if $G_0 = I$ and $S_0 = 0$, then*

$$\begin{aligned} G_{t+1} &= I + \Phi S_{t+1}\Phi^T \\ S_{t+1} &= S_t + \alpha_t(I + S_tK_0)(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^T(I + K_0S_t^T)(I + K_0S_t). \end{aligned}$$

Proof. We can prove this by induction. In the base case, $S_0 = 0$, implying $G_0 = I$ and $G_0^TG_0 = A_0 = I$. In the inductive case, expand Eqn. 9 as $G_{t+1} = (I + \alpha_t G_t \boldsymbol{v}_t \boldsymbol{v}_t^T G_t^T)G_t$ by plugging in the inductive hypothesis $G_t = I + \Phi S_t \Phi^T$. After algebraic simplification, G_{t+1} can be expressed in the form $I + \Phi S_{t+1}\Phi^T$ using the update for S given above. $\square$

According to Lemma 2, G_t can be expressed as $G_t = I + \Phi S_t \Phi^T$, where S_t is a $c \times c$ matrix of coefficients that determines how much weight each pair of constrained inputs has in its contribution to G. Initially, S_0 is set to the all-zeros $c \times c$ matrix, and from there every S_{t+1} is iteratively updated in $O(c^2)$ time via

$$S_{t+1} = S_t + \alpha_t(I + S_tK_0)(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})(\boldsymbol{e}_{i_t} - \boldsymbol{e}_{j_t})^T(I + K_0S_t^T)(I + K_0S_t).$$

Note that this update ensures that G_t remains in the form $G_t = I + \Phi S_t \Phi^T$ once it is updated according to (9).[2] Using this result, at convergence of the metric learning algorithm we can compute $G\phi(\boldsymbol{x})$ in terms of the constrained input pairs $(\phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j))$ as follows:

$$G\phi(\boldsymbol{x}) = \phi(\boldsymbol{x}) + \Phi S \Phi^T \phi(\boldsymbol{x}) = \phi(\boldsymbol{x}) + \sum_{i=1}^{c}\sum_{j=1}^{c} S_{ij}\phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_j)^T\phi(\boldsymbol{x}).$$

[2]We also stress that, since the dimensionality of the data is high, we compute updates implicitly using (4) when performing metric learning; however, since the projection parameters are the same for (3) and (4), the updates for S still can be computed.

Therefore, we have

$$\boldsymbol{r}^T G\phi(\boldsymbol{x}) = \boldsymbol{r}^T\phi(x) + \sum_{i=1}^{c}\sum_{j=1}^{c} S_{ij}\boldsymbol{r}^T\phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_j)^T\phi(\boldsymbol{x}) = \boldsymbol{r}^T\phi(\boldsymbol{x}) + \sum_{i=1}^{c}\gamma_i^r\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}),$$

where $\gamma_j^r = \sum_i S_{ij}\boldsymbol{r}^T\phi(\boldsymbol{x}_j)$, and is a notation substitution for the first equality. This substitution reflects that the values of each γ_j^r rely only on known constrained inputs, and thus can be efficiently computed as soon as the metric learning algorithm has converged, that is, prior to hashing anything into the database.

Finally, having determined this expression for $\boldsymbol{r}^T G\phi(\boldsymbol{x})$, we arrive at our hash function definition in (8). Note the analogy between the use of $\boldsymbol{r}^T G\boldsymbol{x}$ and $\boldsymbol{r}^T G\phi(\boldsymbol{x})$ in (6) and (8), respectively.

3.3 Searching Hashed Examples

Having defined locality-sensitive hash functions for learned metrics, we can apply existing methods [18, 4] to perform sub-linear time approximate similarity search. Given n data points in a Hamming space and an input $\boldsymbol{x}_q$, approximate near-neighbor techniques guarantee retrieval of example(s) within the radius $(1+\epsilon)D$ from $\boldsymbol{x}_q$ in $O(n^{1/1+\epsilon})$ time, where the true nearest neighbor is at a distance of D from $\boldsymbol{x}_q$.

To generate a b-bit hash key for every example, we select b random vectors $[\boldsymbol{r}_1, \dots, \boldsymbol{r}_b]$ to form b hash functions. The hash key for an input $\boldsymbol{x}$ is then the concatenation of the outputs of (6) (or similarly, the outputs of (8) for a sparse and high-dimensional input $\phi(\boldsymbol{x})$). The problem of indexing into the database with $\boldsymbol{x}_q$ is reduced to hashing with these same b functions and retrieving items corresponding to database bit vectors having minimal Hamming distances to the query bit vector. For this step, we employ the technique for approximate search in Hamming space developed by Charikar [4]. Given the list of database hash keys, $M = O(n^{1/(1+\epsilon)})$ random permutations of the bits are formed, and each list of permuted hash keys is sorted lexicographically to form M sorted orders. A query hash key is indexed into each sorted order with a binary search, and the $2M$ nearest examples found this way are the approximate nearest neighbors. See [4] for details.

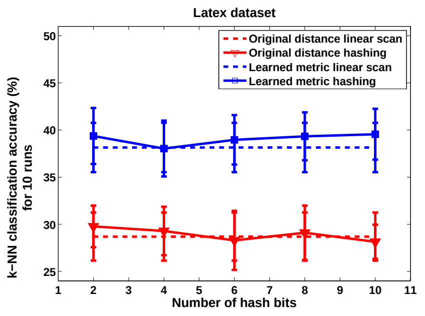
Having identified these nearest bit vectors, we then compute the actual learned kernel values between them and the original input. The hashed neighbors are ranked according to these scores, and this ranked list is used for k-nn classification, clustering, etc., depending on the application at hand. The tradeoff in the selection of b is as follows: larger values will increase the accuracy of how well the keys themselves reflect the metric of interest, but will also increase computation time and can lead to too few collisions in the hash tables. On the other hand, if b is lower, hashing will be faster, but the key will only coarsely reflect our metric, and too many collisions may result.

3.4 Computational Complexity Analysis

We now briefly discuss the computational complexity of each step of our algorithm. The first step, offline metric learning, costs $O(d^2)$ per projection in the low-dimensional case (using (3)) and $O(c^2)$ per projection in the high-dimensional case (using (4)). Thus, each iteration of cycling through all m constraints costs $O(md^2)$ and $O(mc^2)$ for the low-dimensional and high-dimensional cases, respectively. In the high-dimensional case, we must also maintain and update S, which costs $O(c^2)$ per projection.

Once the metric learning step is complete, we must generate hash keys for each item in the database. In the case of dense inputs, computing $h_{\boldsymbol{r},A}(\boldsymbol{x})$ costs $O(d)$ time for each $\boldsymbol{r}$ after we precompute $\boldsymbol{r}^T G$ once. In the high-dimensional case, we can compute $\boldsymbol{r}^T\phi(\boldsymbol{x})$ efficiently by only considering the non-zero elements of $\phi(\boldsymbol{x})$. The γ_i^r may also be computed in this way. Further, we store the non-zeros of $\sum_i \gamma_i^r\phi(\boldsymbol{x}_i)$ and, as a result, computing $h_{\boldsymbol{r},A}(\boldsymbol{x})$ requires $O(z)$ time, where z is the number of non-zeros in $\phi(\boldsymbol{x})$.

Given a new data point, after computing its hash key as above, we must compute the learned kernel values to the $2M$ approximate nearest neighbors. In the low-dimensional case, we compute $A\boldsymbol{x}$ once in $O(d^2)$ time, and then each learned kernel function value can be computed in $O(d)$ time, for a total of $O(Md + d^2)$ time.



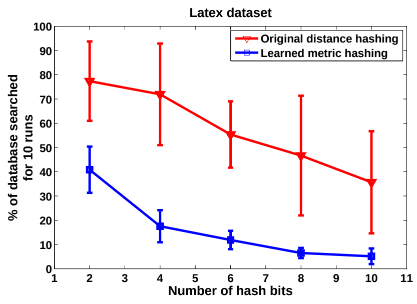


Figure 1: Comparison of the accuracy (left) and time requirements (right) when hashing with the original and learned metrics for the Latex dataset. Left plot shows k-nn classification accuracy; right plot shows the search time in terms of the percentage of database items searched per query, both as a function of the number of hash bits b. Results are from ten runs with random query / database partitions, with ϵ = 1.5. Our learned hash functions reduce the search to about 5% of the database, with virtually no loss in accuracy over the exhaustive linear scan.

In the high-dimensional case, we can compute the M learned kernel values in $O(M\bar{z} + \bar{z}^2)$ total time using a similar construction as with computing $h_{\boldsymbol{r},A}(\boldsymbol{x})$, where $\bar{z}$ is the maximum number of non-zeros for any of the c constrained points. An alternate method leads to $O(Mc + c^2)$ total time for the computation.

4 Results

We first evaluate our method for learned metric hashing on a nearest neighbor (nn) classification problem using data from the CLARIFY system of Ha et al. [17]. CLARIFY assists a programmer in diagnosing errors by identifying previously seen abnormal termination reports with similar program features, and pointing the programmer to other users who have had similar problems. We experiment with a database of n=3825 such examples collected from the Latex typesetting program. The features are $d = 20$-dimensional, and so our explicit formulation for learning hash functions is most appropriate. Paired similarity constraints are generated with information-theoretic metric learning using 20 labeled examples from each class. For 10 random partitions of the data, we extract 30 examples for each of its nine classes, and treat the remainder as database examples. We measure the $k = 4$-nearest-neighbor classification accuracy and search times over all 270 queries per run, under four settings: the "original" Euclidean distance metric and a linear scan, the original distance with LSH, the learned metric with a linear scan, and the learned metric with LSH. For both hashing cases we fixed $\epsilon = 1.5$.

Figure 1 shows the resulting accuracy and complexity gains. By incorporating the paired constraints, the learned metric shows clear accuracy gains over the unconstrained Euclidean distance, yielding about 10% higher correct classification rates. The k-nn rates for the both associated hashed results are on average as good as the linear scan results, and in this case, have little dependence on the number of hash functions used. As b increases, however, the hash keys become more specific and allow larger amounts of the database to be ignored for any given query (righthand plot). When searching only 5% of the database, our learned hash functions suffer no loss in accuracy yet enable an average $13x$ speedup (maximum speedup $34x$) relative to an exhaustive scan with the learned metric (including the overhead cost of computing the hash keys). Interestingly, for the same values of ϵ and b, the number of examples searched with the learned hash functions is noticeably lower than that of the generic hash functions, and has a tighter distribution. While the indexing guarantees remain the same, we infer that just as the learned metric adjusts the feature space so that in-class examples are more closely clustered, the learned hash functions better map them to distinct keys.

In a second set of experiments, we evaluate the implicit formulation of our approach using a large collection of images from 101 categories, called the Caltech101 [9]. This dataset is a commonly used benchmark for object recognition, and poses an interesting challenge for metric learning, since there is not only a large number of categories, but within each category examples can be fairly diverse in appearance. To compare

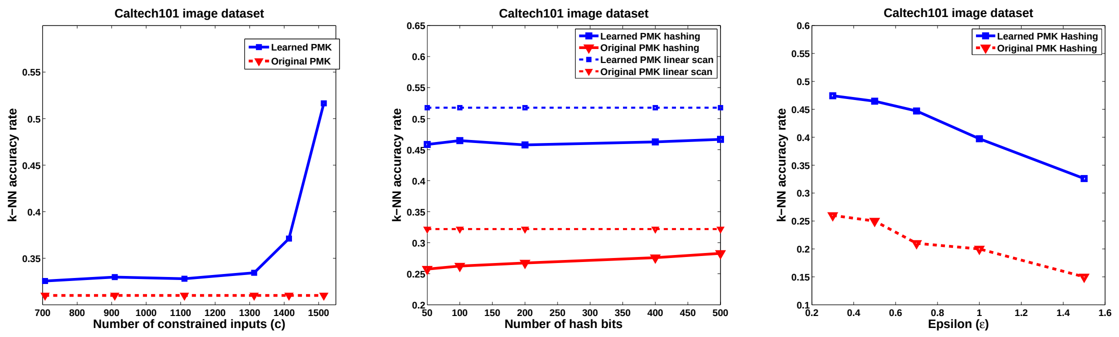


Figure 2: Comparison of the k-nn classification accuracy when hashing with the original and learned PMK for Caltech101 image data. Left plot shows the accuracy improvements over the original PMK [15] as we learn a PMK with an increasing number of paired constraints. Middle plot shows classification accuracy as a function of the number of hash bits, according to a linear scan or hashing with either kernel. Right plot shows the accuracy-search time tradeoff when using the original or learned hashing functions.

these images we consider learning a kernel based on the pyramid match kernel (PMK) of Grauman et al.[15]. The PMK uses multi-dimensional, multi-resolution histogram pyramids to estimate the correspondence between two sets of local image features. To hash with the baseline PMK, the authors embed the pyramids in such a way that standard inner product LSH functions are applicable [16]. The pyramids and this embedding space are very sparse but extremely high-dimensional, thus necessitating the implicit form of our learned metric hashing technique.

As above, we posed a k-nn classification task, and evaluated both the baseline metric (PMK in this case) and learned metric for their accuracy when used in either a linear scan mode or with the approximate hashing search. We represented images with sets of local SIFT [20] features as is done in [15]. We used 15 training examples per class for the database, 2454 images as queries (at least 15 per class), and measured the classification success in terms of the mean recognition rate per class, as is standard practice for this dataset. In order to examine the quality of a learned PMK as a function of the number of paired constraints, we selected varying numbers of images to constrain for kernel learning. The left plot of Figure 2 shows the linear scan k-nn accuracy that results for increasing values of c, and illustrates the clear gain over the (non-learned) kernel baseline (k=1). With 15 constraints per category, our learned kernel outperforms the baseline 52% to 32%.[3] The very best reported performance on this dataset is due to Frome et al.[11], who learn a local feature metric per-training example that yields 60% accuracy. Our single learned kernel function yields 52% accuracy, and at test time requires comparisons to be computed with significantly fewer examples, even if we were to search training examples with a linear scan.

In the middle plot of Figure 2, we compare the linear scan accuracies against those obtained with approximate search, for varying numbers of hash bits. For both the original and learned kernels, the large search time speedup does come at the cost of several points in accuracy. Our learned hash functions still achieve 47% accuracy, and require about $10x$ less computation time than the linear scan when accounting for the hash key computation. Our research code is unoptimized, making this a rather conservative illustration of the speedups our approach offers. The rightmost plot shows the accuracy of our learned PMK hashing compared to the baseline as a function of ϵ. For higher values of ϵ we are guaranteed to search fewer examples, but as expected must then sacrifice some accuracy. Overall, experiments with both datasets validate our approach for sub-linear time hashing with learned metrics and kernels.

[3]Note that the accuracy reported by Grauman and Darrell is 50% using the baseline kernel, but within an SVM classifier; here we find that with k-nn the PMK yields only 32% accuracy. Our learned PMK exceeds the SVM accuracy with a simple k-nn classifier.

5 Conclusions

We have introduced a method to learn randomized hash functions that reflect constraints on partially labeled data points, in order to permit efficient approximate similarity search for learned metrics. Our experiments with systems and image datasets demonstrate both the accuracy advantages to be had with information-theoretic metric learning, as well as the significant computational efficiency gains that the proposed algorithm enables. The proposed technique makes it practical to scale powerful metric learning methods to classification and clustering problems with large databases. In the future we would like to explore the active selection of paired constraints for metric learning, and its impact on our similarity search framework. We are also interested in considering how local learning paradigms could benefit from this faster search capability, and in future experiments we plan to study additional datasets from alternate domains.

References

[1] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios. BoostMap: A Method for Efficient Approximate Similarity Rankings. In *CVPR*, 2004.

[2] M. Badoiu, E. Demaine., M. Hajiaghayi, and P. Indyk. Low-Dimensional Embedding with Extra Information. In *Proceedings of the 20th Symposium on Computational Geometry*, 2004.

[3] A. Bar-Hillel, T. Hertz, N. Shental, and D. Weinshall. Learning a Mahalanobis Metric from Equivalence Constraints. *Journal of Machine Learning Research*, 6:937–965, June 2005.

[4] M. Charikar. Similarity Estimation Techniques from Rounding Algorithms. In *ACM Symp. on Theory of Computing*, 2002.

[5] K. Crammer, J. Keshet, and Y. Singer. Kernel design using boosting. In *NIPS*, 2002.

[6] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. In *Proceedings of the 20th Annual Symposium on Computational Geometry*, 2004.

[7] J. Davis, B. Kulis, P. Jain, S. Sra, and I. Dhillon. Information-Theoretic Metric Learning. In *ICML*, 2007.

[8] R. Duda, P. Hart, and D. Stork. *Pattern Classification*, chapter 10. John Wiley and Sons, Inc., New York, 2 edition, 2001.

[9] L. Fei-Fei, R. Fergus, and P. Perona. Learning Generative Visual Models from Few Training Examples: an Incremental Bayesian Approach Tested on 101 Object Cateories. In *Workshop on Generative Model Based Vision*, Washington, D.C., June 2004.

[10] J. Freidman, J. Bentley, and A. Finkel. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Transactions on Mathematical Software*, 3(3):209–226, September 1977.

[11] A. Frome, Y. Singer, and J. Malik. Image retrieval and classification using local distance functions. In B. Scholkopf, J. Platt, and T. Hofmann, editors, *Advances in Neural Information Processing Systems 19*, Cambridge, MA, 2007. MIT Press.

[12] B. Georgescu, I. Shimshoni, and P. Meer. Mean Shift Based Clustering in High Dimensions: A Texture Classification Example. In *CVPR*, 2003.

[13] A. Globerson and S. Roweis. Metric Learning by Collapsing Classes. In *NIPS*, 2005.

[14] M. Goemans and D. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *JACM*, 42(6):1115–1145, 1995.

[15] K. Grauman and T. Darrell. The Pyramid Match Kernel: Discriminative Classification with Sets of Image Features. In *ICCV*, 2005.

[16] K. Grauman and T. Darrell. Pyramid Match Hashing: Sub-Linear Time Indexing Over Partial Correspondences. In *CVPR*, 2007.

[17] J. Ha, C. Rossbach, J. Davis, I. Roy, H. Ramadan, D. Porter, D. Chen, and E. Witchel. Improved error reporting for software that uses black-box components. In *Programming Language Design and Implementation*, 2007.

[18] P. Indyk and R. Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *30th Symposium on Theory of Computing*, 1998.

[19] G. Lanckriet, N. Cristinanini, P. Bartlett, L. Ghaoui, and M. Jordan. Learning the kernel matrix with semidefinite programming. In *Journal of Machine Learning Research*, volume 5, pages 27–72, 2004.

[20] D. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *IJCV*, 60(2), 2004.

[21] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification, 1998.

[22] S. Roweis and L. Saul. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science*, 290(5500):2323–2326, 2000.

[23] M. Schultz and T. Joachims. Learning a Distance Metric from Relative Comparisons. In *NIPS*, 2003.

[24] G. Shakhnarovich, T. Darrell, and P. Indyk, editors. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. 2006.

[25] G. Shakhnarovich, P. Viola, and T. Darrell. Fast Pose Estimation with Parameter-Sensitive Hashing. In *ICCV*, 2003.

[26] J. Tenenbaum, V. de Silva, and J. Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, December 2000.

[27] J. Uhlmann. Satisfying General Proximity / Similarity Queries with Metric Trees. *Information Processing Letters*, 40:175–179, 1991.

[28] K. Weinberger, J. Blitzer, and L. Saul. Distance Metric Learning for Large Margin Nearest Neighbor Classification. In *NIPS*, 2006.

[29] E. Xing, A. Ng, M. Jordan, and S. Russell. Distance Metric Learning, with Application to Clustering with Side-Information. In *NIPS*, 2002.