This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2015.2507130, IEEE Transactions on Knowledge and Data Engineering

Semi-Supervised Nonlinear Distance Metric Learning via Forests of Max-Margin Cluster Hierarchies

David M. Johnson, Caiming Xiong, and Jason J. Corso *Senior Member, IEEE*

Abstract—Metric learning is a key problem for many data mining and machine learning applications, and has long been dominated by Mahalanobis methods. Recent advances in nonlinear metric learning have demonstrated the potential power of non-Mahalanobis distance functions, particularly tree-based functions. We propose a novel nonlinear metric learning method that uses an iterative, hierarchical variant of semi-supervised max-margin clustering to construct a forest of cluster hierarchies, where each individual hierarchy can be interpreted as a weak metric over the data. By introducing randomness during hierarchy training and combining the output of many of the resulting semi-random weak hierarchy metrics, we can obtain a powerful and robust nonlinear metric model. This method has two primary contributions: first, it is semi-supervised, incorporating information from both constrained and unconstrained points. Second, we take a relaxed approach to constraint satisfaction, allowing the method to satisfy different subsets of the constraints at different levels of the hierarchy rather than attempting to simultaneously satisfy all of them. This leads to a more robust learning algorithm. We compare our method to a number of state-of-the-art benchmarks on k-nearest neighbor classification, large-scale image retrieval and semi-supervised clustering problems, and find that our algorithm yields results comparable or superior to the state-of-the-art.

Index Terms—clustering, classification, and association rules; data mining; image/video retrieval; machine learning; similarity measures

1 Introduction

Many elemental data mining problems—nearest neighbor classification, retrieval, clustering—are at their core dependent on the availability of an effective measure of pairwise distance. Ad hoc selection of a metric, whether by relying on a standard such as Euclidean distance or attempting to select a domain-appropriate kernel, is unreliable and inflexible. It is thus attractive to approach metric selection as a learning problem, and attempt to train strong problem-specific distance measures using data and semantic information.

A wide range of methods have been proposed to address this learning problem, but the field has traditionally been dominated by algorithms that assume a linear model of distance, particularly Mahalanobis metrics [1]. Linear methods have primarily benefited from two advantages. First, they are generally easier to optimize, allowing for faster learning and in many cases a globally optimal solution to the proposed problem [2], [3], [4], [5]. Second, they allow the original data to be easily projected into the new metric space, meaning the metric can be used in conjunction with other methods that operate only on an explicit feature representation (most notably approximate nearest neighbor methods—needed if the metric is to be applied efficiently to large-scale problems).

However, for many types of data a linear approach is not appropriate. Images, videos, documents and histogram representations of all kinds are ill-suited to linear models. Even an ideal Mahalanobis metric will be unable to capture the true semantic structure of these types of data, particularly over larger distances where local linearity breaks down. Kernelized versions of popular Mahalanobis methods [2], [6] have been proposed to handle such data, but these approaches have been limited by high complexity costs. For this reason, researchers have begun to seek alternate metric models that are inherently capable of handling nonlinear data.

These nonlinear metrics are necessarily a broad class of models, encompassing a range of learning modalities and metric structures. One early example of nonlinear metrics (for facial recognition, in this case) by Chopra et al. [7] was based on deep learning strategies. The method was effective, but required long training times and extensive tuning of hyperparameters. Other methods sought to resolve the problem by taking advantage of local linearity in the data, and learning multiple localized linear metrics [8], [9], [10], [11]. These techniques have generally proven superior to single-metric methods, but have also tended to be expensive.

- *D.M. Johnson is with the Department of Computer Science and Engineering, SUNY at Buffalo, Buffalo, NY, 14260, and the Department of Electrical Engineering and Computer Science, UM Ann Arbor, Ann Arbor, MI, 48109. E-mail: davidjoh@buffalo.edu*
- *C. Xiong is a senior researcher at Metamind in Palo Alto, CA, 94301. E-mail: caimingxiong@ucla.edu*
- *J. J. Corso is with the Department of Electrical Engineering and Computer Science, UM Ann Arbor, Ann Arbor, MI, 48109. E-mail: jjcorso@umich.edu*

Most recently, several works have explored metrics that take advantage of tree structures to produce flexible nonlinear transformations of the data. Kedem et al. [12] proposed a method that trained a set of gradient-boosted regression trees and added the regression outputs for each region directly to the data, producing an explicit nonlinear transformation that shifted similar points together and dissimilar points apart. However, this method relies heavily on the (linear) large-margin nearest neighbor process used to initialize it, and often closely mirrors its performance.

Finally, our previous work on this topic [13] formulated the pairwise-constrained metric learning problem as a pair-classification problem, and solved it by direct application of random forests, yielding an implicit nonlinear transformation of the data. However, while this metric could be trained efficiently, it suffered from poor scalability at inference time due to the lack of an explicit feature representation, which made common metric tasks such as nearest neighbor search expensive on larger datasets.

In order to overcome the limitations of these methods, we propose a novel tree-based nonlinear metric with several advantages over existing algorithms. Our metric first constructs a model of the data by computing a forest of semi-random cluster hierarchies, where each tree is generated by iteratively applying a partially-randomized binary semi-supervised max-margin clustering objective. As a result, each tree directly encodes a particular model of the data's full semantic structure, and the structure of the tree itself can thus be interpreted as a weak metric. By merging the output from a forest of these weak metrics, we can produce a final metric model that is powerful, flexible, and resistant to overtraining (due to the independent and semi-random nature of the hierarchy construction).

This methodology provides two significant contributions: first, unlike previous tree-based nonlinear metrics, it is semi-supervised, and can incorporate information from both constrained and unconstrained points into the learning algorithm. This is an important advantage in many problem settings, particularly when scaling to larger datasets where only a tiny proportion of the full pairwise constraint set can realistically be collected or used in training.

Second, the iterative, hierarchical nature of our training process allows us to relax the constraint satisfaction problem. Rather than attempting to satisfy every available constraint simultaneously, at each hierarchy node we can optimize an appropriate constraint subset to focus on, leaving others to be addressed lower in the tree (or in other hierarchies in the forest). By selecting constraints in this way, we can avoid situations where we are attempting to satisfy incoherent constraints [14], and thereby better model hierarchical data structures.

Additionally, we propose a scalable and highly accurate algorithm for obtaining approximate nearest neighbors within our learned metric's space. This renders the metric tractable for large-scale retrieval or nearest-neighbor classification problems, and overcomes a major limitation our previous tree-based metric.

2 Semi-supervised max-margin hierarchy forests

In this section we describe in detail our Hierarchy Forest Distance (HFD) model, as well as our procedures for training and inference. First, however, we must clarify that, despite its name, HFD does not fit the strict definition of a distance metric. While our proposed measure *is* symmetric and nonnegative, it does not satisfy the triangle inequality. While this is a compromise from a theoretical perspective, in practical terms it is a necessity. In order to design a similarity measure capable of modeling complex nonlinear spaces, the triangle inequality *must* be violated.

2.1 Hierarchy forests

The structure of the HFD model draws some basic elements from random forests [15], in that it is composed of T trees trained independently in a semi-random fashion, with individual nodes in the trees defined by a splitting function that divides the local space into two or more segments. Each hierarchy tree represents a distance function $\mathcal{H}(\mathbf{a}, \mathbf{b})$, where $\{\mathbf{a}, \mathbf{b}\} \in \mathbb{R}^d$ are points in the original input space. The overall distance function is then

$$D(\mathbf{a}, \mathbf{b}) = \frac{1}{T} \sum_{t=1}^{T} \mathcal{H}_t(\mathbf{a}, \mathbf{b}) \ . \tag{1}$$

However, there are significant differences between the two methods. HFD is conceptually distinct from random forests (and the Random Forest Distance (RFD) metric [13]) in that the individual components of the forest represent cluster hierarchies rather than decision trees (we discuss this distinction and its implications in Section 2.3). HFD also differs from the most common form of random forest in that it does not do bootstrap sampling on its training points, and its splitting functions are linear combinations rather than single-feature thresholds.

We will now describe our Hierarchy Forest Distance model in detail. For a high-level overview, see Algorithms 1 (learning) and 2 (inference).

2.2 Hierarchy forest distance

The full hierarchy forest distance is effectively the mean of a number of weak distance functions $\mathcal{H}_t$, each corresponding to one hierarchy in the forest. These distance functions, in turn, are representations of the structure of the individual hierarchies—the further apart two instances fall within a hierarchy, the greater the distance between them (see Fig. 1). Specifically, we formulate each metric as a modified form of a hierarchy distance

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2015.2507130, IEEE Transactions on Knowledge and Data Engineering

function we previously proposed for use in hierarchy comparison [16]:

$$\mathcal{H}_t(\mathbf{a},\mathbf{b}) = \begin{cases} 0 & \text{if } \mathcal{H}_{tl(\mathbf{a},\mathbf{b})} \text{is a leaf node} \\ p_t(\mathbf{a},\mathbf{b}) \cdot \frac{|\mathcal{H}_{tl(\mathbf{a},\mathbf{b})}|}{N} & \text{otherwise,} \end{cases} \tag{2}$$

where $\mathcal{H}_t$ represents a particular hierarchy, a and b are input points, $\mathcal{H}_{tl}$ denotes the l^{th} node in $\mathcal{H}_t$ and $\mathcal{H}_{tl(\mathbf{a},\mathbf{b})}$ is the lowest node in $\mathcal{H}_t$ that contains both $\mathbf{a}$ and $\mathbf{b}$. The $|\cdot|$ notation represents the number of training points (out of the training set of size N) contained in a node's subtree. Pairs that share a leaf node are given a distance of 0 because they are maximally similar under $\mathcal{H}_t$, and the size of the leaf nodes is defined by hyperparameter rather than by data. Each non-leaf node $\mathcal{H}_{tl}$ is assigned (via max-margin clustering) a projection function $\mathcal{P}_{tl}$ and associated binary linear discriminant S_{tl} that divides the data in that node between the two child nodes. $p_t(a,b)$ is a certainty term determined by the distance of the projected points a and b from the decision hyperplane at $\mathcal{H}_{tl(\mathbf{a},\mathbf{b})}$:

$$p_t(\mathbf{a},\mathbf{b}) = \frac{1}{1+\exp(\alpha \cdot \mathcal{P}_{tl(\mathbf{a},\mathbf{b})}(\mathbf{x}_a))} - \frac{1}{1+\exp(\alpha \cdot \mathcal{P}_{tl(\mathbf{a},\mathbf{b})}(\mathbf{x}_b))}, \tag{3}$$

where α is a hyperparameter that controls the sensitivity of p and $\mathcal{P}_{tl(\mathbf{a},\mathbf{b})}$ is the projection function at $\mathcal{H}_{tl(\mathbf{a},\mathbf{b})}$ (see (5)). Thus, p ranges from 0 to 1, approaching 0 when the projections of both a and b are near the decision boundary, and 1 when both are far away. The full distance formulation for a hierarchy is also confined to this range, with a distance approaching 1 corresponding to points that are widely separated at the root node, and 0 to points that share a leaf node.

2.3 HFD learning and inference

The fact that the trees used in HFD represent cluster hierarchies rather than decision trees has significant implications for HFD training, imposing stricter requirements on the learned splitting functions. While the goal of decision tree learning is ultimately to yield a set of pure single-class leaf nodes, a cluster hierarchy instead seeks to accurately group data elements at *every* level of the tree. Thus, if the hierarchy learning algorithm divides the data poorly at or near the root node, there is no way for it to recover from this error later on. This is partially mitigated by learning a forest in place of a single tree, but even in this case the *majority* of hierarchies in the forest must correctly model the high-level semantic relationship between any two data elements.

For this reason, HFD requires a robust approach to the hierarchy splitting problem that reliably generates semantically meaningful splits. Additionally, in order to allow for efficient metric inference, our splitting algorithm must generate explicit and efficiently evaluable splitting functions at each node.

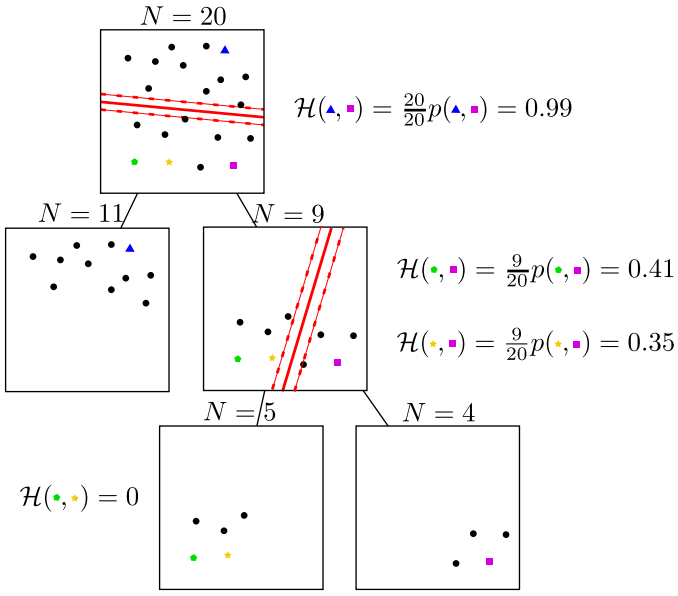


Fig. 1. A small example of a single hierarchy metric, illustrating how metric distance between two points is determined by the tree depth at which the points are separated and the projected distance of the two points from their final max-margin splitting hyperplane.

Given these constraints, we approach the hierarchy learning problem as a series of increasingly fine-grained flat semi-supervised clustering problems, and we solve these flat clustering problems via max-margin clustering (MMC) [17], [18], [19], [20].

Max-margin clustering has a number of advantages that make it ideal for our problem. In addition to their widespread use in support vector machines for classification, max-margin and large-margin techniques have proven highly effective in the metric learning domain [4], [21], [22], and many, including MMC, can be solved in linear time [20], [23], [24]. Most importantly, MMC returns a simple and explicit splitting function which can be computed efficiently and applied to points outside the initial clustering.

We employ a novel relaxed form of semi-supervised MMC, which uses pairwise must-link (ML) and cannot-link (CL) constraints to improve semantic clustering performance. Constraints of this type indicate either semantic similarity (ML) or dissimilarity (CL) between pairs of points, and do not require the availability of class labels.

We describe our semi-supervised MMC technique in Section 3.

2.3.1 HFD learning

We train each tree in the HFD model independently, with each tree using the same data and constraint sets. Training is hence easily parallelized. Assume an unlabeled training dataset $\mathbf{X}_0$ and pairwise constraint set $\mathcal{L}_0$. Denote a must-link constraint set $\mathcal{L}_0^{\mathcal{M}}$ and cannot-link constraint set $\mathcal{L}_0^{\mathcal{C}}$, such that $\mathcal{L}_0 = \mathcal{L}_0^{\mathcal{M}} \cup \mathcal{L}_0^{\mathcal{C}}$.

Algorithm 1 HFD learning

$t \leftarrow 0$
for $t < T$ **do**
  BUILDTREE(t, 0, $\mathbf{X}^0$, $\mathcal{L}_0$)
  $t \leftarrow t + 1$
end for

function BUILDTREE(t, l, $\mathbf{X}_{tl}$, $\mathcal{L}_{tl}$)
  $\mathbf{w}_{tl} \leftarrow$ LEARNSPLIT($\mathbf{X}_{tl}$, $\mathcal{L}_{tl}$)
  Create child nodes c_L and c_R
  Divide $\mathbf{X}_{tl}$ among c_L and c_R using (6)
  if $|\mathbf{X}_{tc_L}| > minsize$ **then**
    Use $\mathbf{X}_{tc_L}$ to determine $\mathcal{L}_{tc_L}$
    BUILDTREE(t, c_L, $\mathbf{X}_{tc_L}$, $\mathcal{L}_{tc_L}$)
  end if
  if $|\mathbf{X}_{tc_R}| > minsize$ **then**
    Use $\mathbf{X}_{tc_R}$ to determine $\mathcal{L}_{tc_R}$
    BUILDTREE(t, c_R, $\mathbf{X}_{tc_R}$, $\mathcal{L}_{tc_R}$)
  end if
end function

function LEARNSPLIT($\mathbf{X}_{tl}$, $\mathcal{L}_{tl}$)
  Select split features $\mathcal{K}_{tl}$ and build $\mathbf{X}_{tl}^{\mathcal{K}}$ via (4)
  if $\mathcal{L}^{\mathcal{C}} \neq \emptyset$ **then**
    Use CCCP to solve (8) for $\mathbf{w}_{tl}$
  else
    Use block-coord. descent to solve (12) for $\mathbf{w}_{tl}$
  end if
  return $\mathbf{w}_{tl}$
end function

Training of individual trees proceeds in a top-down manner. At each node $\mathcal{H}_{tl}$ we begin by selecting and storing a local feature subset $\mathcal{K}_{tl}$ by uniformly sampling $d_k < d$ features from the full feature set. We then use $\mathcal{K}_{tl}$ to construct our modified local data:

$$\mathbf{X}_{tl}^{\mathcal{K}} = \left\{ \begin{bmatrix} \mathbf{x}_j^{\mathcal{K}_{tl}} & 1 \end{bmatrix} \;\middle|\; \mathbf{x}_j \in \mathbf{X}_{tl} \in \right\} \quad . \tag{4}$$

Where the unit feature appended to $\mathbf{x}_j^{\mathcal{K}_{tl}}$ allows MMC to learn a bias term.

For each node $\mathcal{H}_{tl}$, our split function learning algorithm can operate in either a semi-supervised or unsupervised mode, so before we begin learning we must check for constraint availability. We require at least 1 cannot-link constraint in order to carry out semi-supervised MMC, so we check whether $\mathcal{L}_{tl}^{\mathcal{C}} = \emptyset$, and then apply either semi-supervised or unsupervised MMC (see Section 3) to $\mathbf{X}_{tl}^{\mathcal{K}}$ and $\mathcal{L}_{tl}$. The output of our split learning algorithm is the weight vector $\mathbf{w}_{tl}$, which, along with $\mathcal{K}_{tl}$, forms the splitting function S_{tl}:

$$\mathcal{P}_{tl}(\mathbf{x}) = \mathbf{w}_{tl}^T \begin{bmatrix} \mathbf{x}_j^{\mathcal{K}_{tl}} & 1 \end{bmatrix} \tag{5}$$

$$S_{tl}(\mathbf{x}) = \begin{cases} \text{send } \mathbf{x} \text{ left} & \mathcal{P}_{tl}(\mathbf{x}) \leq 0 \\ \text{send } \mathbf{x} \text{ right} & \mathcal{P}_{tl}(\mathbf{x}) > 0 \end{cases} \quad . \tag{6}$$

We then apply S_{tl} to divide $\mathbf{X}_{tl}$ among $\mathcal{H}_{tl}$'s children. After this, we must also propagate the constraints down the tree. We do this by iterating through $\mathcal{L}_{tl}$ and checking the point membership of each child node $\mathcal{H}_{tj}$—if $\mathbf{X}_{tj}$ contains both points covered by a constraint, then we add that constraint to $\mathcal{L}_{tj}$.

As a result, constraints in $\mathcal{L}_{tl}$ whose constrained points are separated by $\mathcal{H}_{tl}$'s splitting function effectively disappear in the next level of the hierarchy. This results in a steady narrowing of the constraint-satisfaction problem as we reach further down the tree, in accordance with the progressively smaller regions of the data space we are processing. We continue this process until we reach a stopping point (in our experiments, a minimum node size threshold), falling back on unsupervised MMC as we exhaust the relevant cannot-link constraints.

2.3.2 HFD Inference

Algorithm 2 HFD inference

function INFERDISTANCE($\mathbf{a}, \mathbf{b}$)
  $t \leftarrow 0$
  $D \leftarrow 0$
  for $t < T$ **do**
    $D \leftarrow D +$ TREEDISTANCE($t, 0, \mathbf{a}, \mathbf{b}$)
    $t \leftarrow t + 1$
  end for
  return $\frac{D}{T}$
end function

function TREEDISTANCE($t, l, \mathbf{a}, \mathbf{b}$)
  Retrieve split features $\mathcal{K}_{tl}$ and build $\mathbf{a}^{\mathcal{K}_{tl}}$ and $\mathbf{b}^{\mathcal{K}_{tl}}$
  Apply (6) to $\mathbf{a}^{\mathcal{K}_{tl}}$ and $\mathbf{b}^{\mathcal{K}_{tl}}$ to get $S_{tl}(\mathbf{a})$ and $S_{tl}(\mathbf{b})$
  if $S_{tl}(\mathbf{a}) = S_{tl}(\mathbf{b})$ **then**
    return TREEDISTANCE($t, S_{tl}(\mathbf{a}), \mathbf{a}, \mathbf{b}$)
  else
    return output of (2) for $t, l, \mathbf{a}$ and $\mathbf{b}$
  end if
end function

Metric inference on learned HFD structures is straightforward. We feed two points $\mathbf{x}_1$ and $\mathbf{x}_2$ to the metric and track their progress down each tree $\mathcal{H}_t$. At each node $\mathcal{H}_{tl}$, we compute $S_{tl}(\mathbf{x}_1)$ and $S_{tl}(\mathbf{x}_2)$. If $S_{tl}(\mathbf{x}_1) = S_{tl}(\mathbf{x}_2)$, we continue the process in the indicated child node. If not, then we have found $\mathcal{H}_{tl(\mathbf{x}_1,\mathbf{x}_2)}$, so we compute and return $\mathcal{H}_t(\mathbf{x}_1, \mathbf{x}_2)$ as described in (2). The results from each tree are then combined as per (1).

3 LEARNING SPLITTING FUNCTIONS

In order to learn strong, optimized $\mathcal{P}_{tl}$ functions at each hierarchy node, our method relies on the Max-Margin Clustering (MMC) framework. In most nodes, our method uses semi-supervised MMC (SSMMC) to incorporate pairwise constraint information into the split function learning process. Below, we describe a novel relaxed SSMMC formulation (based on the state-of-the-art method described in [24]) designed to function in our hierarchical problem setting.

3.1 Relaxed semi-supervised max-margin clustering

Semi-supervised MMC incorporates a set of must-link ($\mathcal{L}^{\mathcal{M}}$) and cannot-link ($\mathcal{L}^{\mathcal{C}}$) pairwise semantic constraints into the clustering problem. SSMMC thus seeks to simultaneously maximize the cluster assignment margin of each point (as in unsupervised MMC) and an additional set of margin terms reflecting the satisfaction of each pairwise constraint.

Existing SSMMC formulations, however, are poorly suited to hierarchy learning. Because cannot-link constraints disappear from the hierarchy learning problem once they are satisfied (see Section 2.3.1), the number of relevant cannot-link constraints will generally decrease much more quickly than the number of must-link constraints. This will lead to highly imbalanced constraint sets in lower levels of the hierarchy.

Under the original SSMMC formulation, imbalanced cases such as these may well yield trivial one-class solutions wherein the ML constraints are well satisfied, but the few CL constraints are highly *un*satisfied. To address this problem, we simply separate the ML and CL constraints into two distinct optimization terms, each with equal weight.

Second, and more significantly, we must modify SSMMC to handle the iterative nature of the hierarchy splitting problem. Consider a case with 4 semantic classes: apple, orange, bicycle and motorcycle. In a binary hierarchical setting, the most reasonable way to approach this problem is to first separate apples and oranges from bicycles and motorcycles, then divide the data into pure leaf nodes lower in the tree.

Standard SSMMC, however, will instead attempt to simultaneously satisfy the cannot-link constraints between *all* of these classes, which is impossible. As a result, the optimization algorithm may seek a compromise solution that weakly violates all or most of the constraints, rather than one that strongly satisfies a subset of the constraints and ignores the others (e.g. that separates apples and oranges from bicycles and motorcycles).

We handle this complication by relaxing the clustering algorithm to focus on only a *subset* of the CL constraint set, and integrate the selection of that subset into the optimization problem. Thus, our variant of semi-supervised MMC simultaneously optimizes the discriminant $\mathbf{w}$ to satisfy a subset of the CL constraint set $\mathcal{L}^{\mathcal{C}'} \subset \mathcal{L}^{\mathcal{C}}$, *and* seeks the $\mathcal{L}^{\mathcal{C}'}$ that can best be satisfied by a binary linear discriminant.

For an empirical comparison of baseline SSMMC and our relaxation, see Section 6.9.

3.1.1 Relaxed SSMMC formulation

First, we define some notation. $\mathcal{L}^{\mathcal{M}}$ and $\mathcal{L}^{\mathcal{C}}$ are the sets of ML and CL constraints, respectively, and $L^{\mathcal{M}}$ and $L^{\mathcal{C}}$ are the sizes of those sets. We set the size of the selected CL constraint subset $\mathcal{L}^{\mathcal{C}'}$ via the hyperparameter $L^{\mathcal{C}'}$. η_j denotes a slack variable for pairwise constraint j, $\mathcal{U}$ is the set of unconstrained points (i.e. points not referenced by any pairwise constraint), U is the size of that set and ξ_i are slack variables for the unconstrained points. j_1 and j_2 represent the two points constrained by pairwise constraint j.

For convenience, define the following function representing the joint projection value of two different points onto two particular cluster labels $y_1, y_2 \in \{-1, 1\}$:

$$\phi(\mathbf{x}_1, \mathbf{x}_2, y_1, y_2) = y_1 \mathbf{w}^T \mathbf{x}_1 + y_2 \mathbf{w}^T \mathbf{x}_2 \ . \tag{7}$$

We can now define our relaxed SSMMC formulation thus:

$$\begin{aligned}
&\min_{\mathbf{w},\eta,\xi,\mathcal{L}^{\mathcal{C}'}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{L^{\mathcal{M}}} \sum_{j \in \mathcal{L}^{\mathcal{M}}} \eta_j + \frac{1}{L^{\mathcal{C}'}} \sum_{j \in \mathcal{L}^{\mathcal{C}'}} \eta_j + \frac{C}{U} \sum_{i=1}^{U} \xi_i \\
&\text{s.t.} \\
&\quad \forall j \in \mathcal{L}^{\mathcal{M}}, \forall s_{j1}, s_{j2} \in \{-1, 1\}, s_{j1} \neq s_{j2} : \\
&\qquad \max_{z_{j1} = z_{j2}} \phi(\mathbf{x}_{j1}, \mathbf{x}_{j2}, z_{j1}, z_{j2}) - \phi(\mathbf{x}_{j1}, \mathbf{x}_{j2}, s_{j1}, s_{j2}) \\
&\qquad \geq 1 - \eta_j, \quad \eta_j \geq 0 \\
&\quad \exists\ \mathcal{L}^{\mathcal{C}'} \subset \mathcal{L}^{\mathcal{C}} \text{ of size } L^{\mathcal{C}'} \text{ s.t. :} \\
&\qquad \forall j \in \mathcal{L}^{\mathcal{C}'}, \forall s_{j1}, s_{j2} \in \{-1, 1\}, s_{j1} = s_{j2} : \\
&\qquad\quad \max_{z_{j1} \neq z_{j2}} \phi(\mathbf{x}_{j1}, \mathbf{x}_{j2}, z_{j1}, z_{j2}) - \phi(\mathbf{x}_{j1}, \mathbf{x}_{j2}, s_{j1}, s_{j2}) \\
&\qquad\quad \geq 1 - \eta_j, \quad \eta_j \geq 0 \\
&\quad \forall i \in \mathcal{U} : \\
&\qquad \max_{y_i^s \in \{-1,1\}} 2 y_i^s \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i, \\
&\qquad \xi_i \geq 0 \ .
\end{aligned} \tag{8}$$

As in [24], the must-link and cannot-link constraints each impose a soft margin on the difference in score between the highest-scoring joint projection that satisfies the constraint and the highest scoring joint projection that does *not* satisfy the constraint. Each unconstrained point is subject to a separate soft-margin constraint enforcing strong association with a single cluster. The parameters λ and C are optimization weighting factors that determine the relative importance of the max-margin objective and the unsupervised slack variable objective, respectively.

3.1.2 Semi-supervised MMC optimization

We optimize our SSMMC formulation via a Constrained Concave Convex Procedure (CCCP) [25]. In most respects, this is identical to the process used in [24].

The CCCP described in [24] consists of two steps per iteration. In step one, the best cluster assignments (based on the current $\mathbf{w}$) for each point and constraint pair are locked in, reducing the non-convex global optimization problem to a convex local one. In step two, this convex problem is solved via subgradient descent, yielding a new $\mathbf{w}$ which is used to initialize the next iteration. This process continues until the weight vector converges.

In our modified version of this process, step one contains an additional component: the selection of the

current $\mathcal{L}^{\mathcal{C}'}$. We define $\mathcal{L}^{\mathcal{C}'}$ as the set of cannot-link constraints of size $L^{\mathcal{C}'} \leq L^{\mathcal{C}}$ that minimizes $\frac{1}{\mathcal{L}^{\mathcal{C}'}} \sum_{j \in \mathcal{L}^{\mathcal{C}'}} \eta_j$ (i.e. the sum of the cannot-link constraint margin violations). For a given $\mathbf{w}$, this can be trivially optimized by simply selecting the $L^{\mathcal{C}'} \leq L^{\mathcal{C}}$ cannot-link constraints with the largest margins. The following subgradient optimization then considers only this constraint subset. As with the cluster assignments, we update this selection at each iteration, allowing the algorithm to converge jointly (albiet locally) on an optimal discriminant function and optimal set of cannot-link constraints to be satisfied at the current level of the hierarchy.

Because CCCP finds only a local maximum, it is sensitive to parameter initialization. We address this problem in the same way as [24], using an LDA-like heuristic to initialize $\mathbf{w}^{(0)}$ based on the available pairwise constraints:

$$\mathbf{S}^{\mathcal{M}} = \frac{1}{L^{\mathcal{M}}} \sum_{j \in \mathcal{L}^{\mathcal{M}}} (\mathbf{x}_{j1} - \mathbf{x}_{j2})(\mathbf{x}_{j1} - \mathbf{x}_{j2})^T \tag{9}$$

$$\mathbf{S}^{\mathcal{C}} = \frac{1}{L^{\mathcal{C}}} \sum_{j \in \mathcal{L}^{\mathcal{C}}} (\mathbf{x}_{j1} - \mathbf{x}_{j2})(\mathbf{x}_{j1} - \mathbf{x}_{j2})^T \tag{10}$$

$$\mathbf{S}^{\mathcal{C}}\mathbf{w}^{(0)} = \lambda_e \mathbf{S}^{\mathcal{M}}\mathbf{w}^{(0)} \ , \tag{11}$$

with (11) being a straightforward general eigenproblem. This process yields a $\mathbf{w}^{(0)}$ that maximizes the ratio of must-link-pair similarity to cannot-link-pair similarity. We then initialize our chosen set of cannot-link constraints using this heuristically determined $\mathbf{w}^{(0)}$.

3.2 Unsupervised MMC

Because HFD progressively eliminates constraints as the hierarchy grows deeper (see Section 2.3.1), at lower levels of the hierarchy we may encounter nodes where there are no cannot-link constraints available, and hence SSMMC cannot proceed [24]. In these cases, we compute $\mathcal{P}_{tl}$ via the unsupervised Membership Requirement MMC (MRMMC) formulation proposed in [20]:

$$\min_{\mathbf{w},\xi,\beta} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{C}{N}\sum_{i=1}^{N}\xi_i + \frac{1}{2h}\left(\beta_{-1} + \beta_1\right)$$

s.t.

$$\forall i : \max_{y_i^s \in \{-1,1\}} 2y_i^s \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i,$$

$$\xi_i \geq 0,$$

$$j \in \{-1, 1\} : \exists \text{ a set } \mathcal{L}_j \text{ containing } h \text{ indices } i \text{ s.t.}$$

$$\max_{\mathcal{L}_j}\left(\min_{i \in \mathcal{L}_j} 2j\mathbf{w}^T\mathbf{x}_i\right) \geq 1 - \beta_j \ , \tag{12}$$

This formulation uses (for all points) the same unsupervised margin constraint seen in (8). However, in order to prevent degenerate solutions in which all elements are assigned to a single cluster, MRMMC enforces a membership requirement constraint, applying an additional soft margin constraint that drives each cluster to have at least h points strongly assigned to it (in our algorithm we set $h = \max(1, \frac{U}{20})$). We optimize this problem via block-coordinate descent as described in [20].

4 Fast approximate nearest neighbors in hierarchy metric space

Algorithm 3 Fast approximate HFD nearest neighbors

```
t ← 0
O ← ∅
for t < T do
    O ← O ∪ TreeNeighbors(t, 0, a)
    t ← t + 1
end for
for {x ∈ O} do
    InferDistance(x, a)
end for
return the k points in O with the smallest distance

function TreeNeighbors(t, l, a)
    if l is a leaf node then
        O_tl ← k_O points sampled from l
        if |X_tl| < k_O then
            Sample from parent node(s) as needed
        end if
        return O_tl
    else
        Retrieve split features K_tl and build a^{K_tl}
        Apply (6) to a^{K_tl} to get S_tl(a)
        return TreeNeighbors(t, S_tl(a), a)
    end if
end function
```

One problem with this approach is the potentially high (though still trivially parallelizable) cost of computing each pairwise distance, as compared to a Euclidean or even Mahalanobis distance. This is worsened, for many applications, by the unavailability of traditional fast approximate nearest-neighbor methods (e.g. kd-trees [26], hierarchical k-means [27] or hashing [28]), which require an explicit representation of the data in the metric space in order to function.

We address the latter problem by introducing our own fast approximate nearest-neighbor process (described in Algorithm 3), which takes advantage of the tree-based structure of the metric to greatly reduce the number of pairwise distance computations needed to compute a set of nearest-neighbors for a query point x.

We begin by tracing the path taken by x through each tree in the forest, and thus identifying each leaf node containing x. We then seek $k_{\mathcal{O}}$ candidate neighbors from each tree, beginning by sampling other training points from the identified leaf nodes, then, if necessary, moving up the tree parent-node-by-parent-node until $k_{\mathcal{O}}$ candidates have been found. The candidate sets from each tree are then combined to yield a final candidate neighbor set $\mathcal{O}$, such that $|\mathcal{O}| \leq T \cdot k_{\mathcal{O}}$. We then compute

the full hierarchy distance $D(x, y)$ for all $y \in \mathcal{O}$, sort the resulting distances, and return the k closest points.

This approximation method functions by assuming that, intuitively, a point's nearest-neighbors within the full forest metric space are very likely to *also* be nearest-neighbors under at least *one* of the individual tree metrics. We evaluate this method empirically on several small-to-mid-size datasets, and the results strongly support the validity of this approximation (Section 6.2).

5 Complexity Analysis

5.1 HFD training

First let q be the number of pairwise constraints, n be the number of training points. The overall complexity of standard semi-supervised max-margin clustering is then $\mathbf{O}(d^3 + d(q+n))$ [24], which in our method becomes $\mathbf{O}(d_k^3 + d_k(q+n))$. The d_k^3 component stems from the initialization step of the optimization process (see Section 3.1.2), while $d_k(q+n)$ represents the CCCP procedure. In our case, we must also compute $L^{C'}$ in each CCCP iteration, adding an additional $d_k + q \log q$ factor (the cost of scoring and sorting the cannot-link constraint margins). Our total cost for computing relaxed SSMMC is thus $\mathbf{O}(d_k^3 + d_k(q+n) + q \log q)$

In our hierarchical setting, we are using SSMMC in a divide-and-conquer fashion, which steadily reduces n and q as the algorithm proceeds. If we assume that we divide the data roughly in half with each SSMMC operation, the second and third complexity components become $d_k(q+n)\log(q+n) + q\log^2 q$ for the whole tree. The first component depends on the total number of nodes, which is heavily dependent on the minimum node size parameter. In the worst case, however, the number of nodes is $\mathbf{O}(n)$, so the first complexity component becomes nd_k^3, yielding a total training complexity of $\mathbf{O}(nd_k^3 + d_k(q+n)\log(q+n) + q\log^2 q)$ for each tree. Given the trivially parallel nature of forest training, this can also be considered the cost for training a full HFD model, provided T processors are available.

This does mean that training time is strongly influenced by the d_k parameter. Fortunately, in cases where both d and n are large, HFD can achieve strong performance even with $d_k \ll d$ (see Section 6.8).

5.2 HFD inference

Computing a single HFD metric distance requires one traversal of each tree in the forest, for a complexity cost of $\mathbf{O}(Td_k \log n)$. Many of the most common applications of a metric require computing nearest-neighbors between the training set and a test set of size m. This requires mn distance evaluations, so a brute force nearest-neighbor search under HFD costs $\mathbf{O}(mnTd_k \log n)$, or $\mathbf{O}(nTd_k \log n)$ for a single test point.

Our approximate nearest-neighbor algorithm significantly reduces this cost. Computing candidate neighbors for a single point costs only $\mathbf{O}(T \log n)$. There will be at most $Tk_{\mathcal{O}}$ candidates for each set, so the cost for computing distances to each candidate is $\mathbf{O}(T^2 k_{\mathcal{O}} d_k \log n)$, or $\mathbf{O}(T^2 d_k \log n)$ if we ignore the parameter. It should be noted, however, that in practice there is significant overlap between the candidate sets returned by different trees, and this overlap increases with T. The actual cost of this step is thus generally much lower.

Thus, the worst-case complexity of our approximate nearest-neighbor method, when applied to an entire dataset, is $\mathbf{O}(mT^2 d_k \log n)$, or $\mathbf{O}(mTd_k \log n)$ if fully parallelized, and in practice the performance is generally much better than the worst case.

5.3 Memory complexity

In order to represent a trained cluster hierarchy in memory, we must store the learned weight vector for each node in each tree. The total number of nodes per tree, again, is $\mathbf{O}(n)$ in the worst case. Storing the weight vectors for a tree thus has $\mathbf{O}(nd_k)$ memory complexity.

However, in order to enable our fast approximate nearest neighbors algorithm, we must also store the set of training points present in each node. The allocation of training points to subtrees follows the divide-and-conquer pattern, so the number of stored points in each tree is $\mathbf{O}(n \log n)$. The total memory complexity of HFD is thus $\mathbf{O}(Tnd_k + Tn \log n)$.

6 Experiments

Below we present several experiments quantifying HFD's performance. First, we validate the accuracy and efficiency of our approximate nearest-neighbor retrieval method. We then carry out benchmark comparisons against other state-of-the-art metric learning techniques in the k-nearest neighbor classification, large-scale image retrieval and semi-supervised clustering domains.

6.1 Datasets

Dataset	#Samples	#Features	#Classes
sonar	208	60	2
ionosphere	351	34	2
balance	625	4	3
segmentation	2310	18	7
USPS	11,000	256	10
MAGIC	19,020	10	2
CIFAR	60,000	300	10/20/100
diabetes	768	8	2
breast	683	10	2
german credit	1000	24	2
haberman	306	3	2
DBWorld	64	3721	2
arcene	100	10000	2

TABLE 1
Dataset statistics

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2015.2507130, IEEE Transactions on Knowledge and Data Engineering

We use a range of datasets, from small- to large-scale, to evaluate our method. For small to mid-range data, we use a number of well-known UCI sets [29]: sonar, ionosphere, balance, segmentation and MAGIC, as well as the USPS handwritten digits dataset [30].

For our larger scale experiments, we relied on the CIFAR tiny image datasets [31]. CIFAR-10 consists of 50,000 training and 10,000 test images, spread among 10 classes. CIFAR-100 also contains 50,000 training and 10,000 testing images, but has 2 different label sets—a coarse set with 20 classes, and a fine set with 100 classes. All CIFAR instances are 36x36 color images, which we have reduced to 300 features via PCA.

In all our experiments, the data is normalized to 0 mean and unit variance.

6.2 Approximate nearest-neighbor retrieval

Because we use it for retrieval in all of our other experiments, we first evaluate the accuracy cost and efficiency benefits of our approximate nearest-neighbor method. We evaluate accuracy by training an HFD model with 100 trees. We then return 50 approximate nearest-neighbors for each point in the dataset and compute mean average precision (mAP) relative to the ground truth 50 nearest-neighbors (obtained via brute force search). Average precision scores are computed at 10, 20, 30, 40 and 50 nearest-neighbors. We do retrieval at $k_{\mathcal{O}} = 1$, 3, 5, 10, 20 and 30, and report both the mAP results and the time taken (as a proportion of the brute force time) at each value on several datasets.

$k_{\mathcal{O}}$	Sonar	Seg.	USPS
1	0.812	0.715	0.6559
3	0.965	0.904	0.8700
5	0.987	0.956	0.9274
10	0.997	0.987	0.9710
20	0.998	0.994	0.9897
30	0.998	0.995	0.9945

TABLE 2
Approximate nearest-neighbor retrieval mAP scores

$k_{\mathcal{O}}$	Sonar	Seg.	USPS
1	0.499	0.042	0.014
3	0.547	0.074	0.034
5	0.729	0.097	0.049
10	0.860	0.147	0.076
20	1.002	0.221	0.112
30	0.997	0.270	0.136

TABLE 3
Approximate nearest-neighbor retrieval times (as a proportion of brute force search time)

The results clearly show that our approximation method yields significant reductions in retrieval time on larger datasets with minimal loss of accuracy. Note that all other results we report for HFD are generated using this approximation method. We use $k_{\mathcal{O}} = 5$ for the CIFAR datasets, and $k_{\mathcal{O}} = 10$ for all other data.

6.3 Comparison methods and parameters

In the following experiments, we compare our HFD model against a number of state-of-the-art metric learning techniques: DCA [32], LMNN [4], [22], GB-LMNN [12], ITML [2][1], Boostmetric [3] and RFD [13]. With the exception of RFD and GB-LMNN (both of which incorporate tree structures into their metrics), all are Mahalanobis methods that learn purely linear transforms of the original data.

We did not extensively tune any of the hyperparameters for HFD, instead using a common set of values for all datasets. We set $T = 500$ (for HFD, RFD and GB-LMNN), $L^{\mathcal{C}'} = 0.25L^{\mathcal{C}}$, $\lambda = 0.01$, $C = 1$, $\epsilon_1 = \epsilon_2 = 0.01$ and $\alpha = 0.5$. We use $d_k = \frac{d}{3}$ on all datasets except balance, where we use $d_k = d$ (this was the only dataset that appeared to be highly sensitive to this or any other parameter in HFD). As a stop criterion for tree training, we set a minimum node size of 5 for all but the CIFAR dataset, where we use 30 (as a means of saving some computation time).

6.4 Nearest neighbor classification

We next test our method using k-nearest neighbor classification (we use $k = 5$ for all datasets). Each dataset is evaluated using 5-fold cross-validation. For the weakly-supervised methods, in each fold we use 1,000 constraints per class (333 must-link, 667 cannot-link, drawn randomly from the training data) for the sonar, ionosphere, balance and segmentation datasets. For USPS and magic, we use 30,000 constraints in each fold. We repeated each experiment 10 times with different random constraint sets, and reported the average result in Fig. 4.

We found that HFD achieved the best score on 4 out of the 6 UCI datasets tested, and was competitive on the remaining two (sonar and USPS). On the CIFAR data, only the DCA method (which performs poorly on most of the UCI sets) is competitive with HFD, and on the 100 class problem HFD has a clear advantage.

6.5 Retrieval

To evaluate our method's performance (as well as, implicitly, the effectiveness of our approximate nearest-neighbor algorithm) on large-scale tasks, we computed semantic retrieval precision on labeled CIFAR tiny image datasets. For the weakly-supervised methods, we sample 600,000 constraints from the training data, again with a 1:2 must-link to cannot-link ratio, sampled evenly from all classes. Though this does represent a large amount

1. For all ITML experiments, we cross-validated across 9 different γ values and reported the best results.

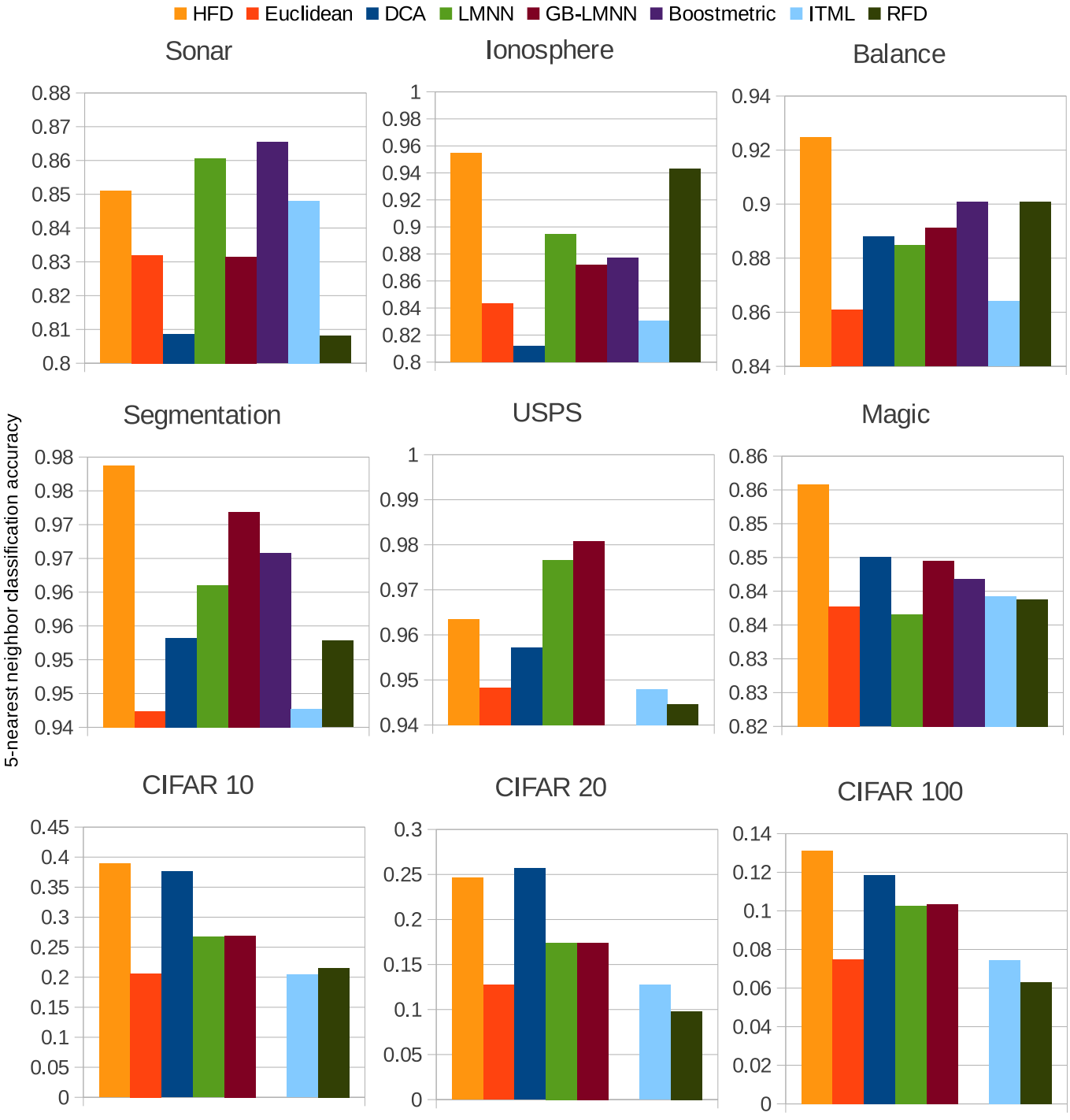


Fig. 2. 5-nearest neighbor classification accuracy under HFD and benchmark metrics. (View in color)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2015.2507130, IEEE Transactions on Knowledge and Data Engineering

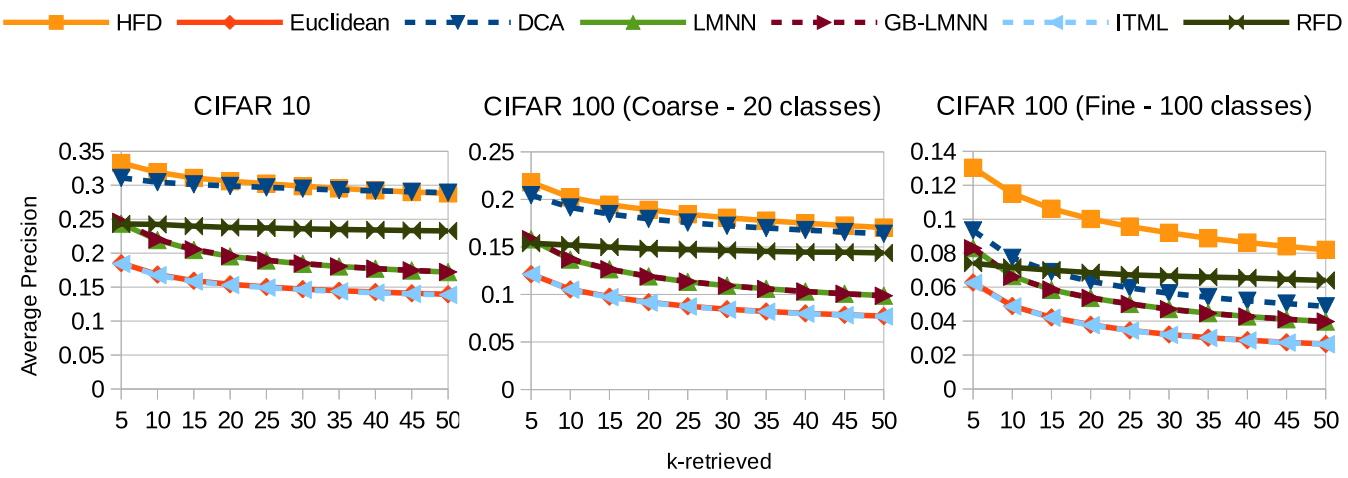


Fig. 3. Large-scale semantic image retrieval results for our method and benchmarks. Only DCA is competitive with our method on the 10 and 20 class datasets, and HFD significantly outperforms all other algorithms on the 100 class problem. (View in color)

of training data, we note that it contains less than 0.1% of the full constraint set for this data. We do not report Boostmetric results on these sets because we were unable to obtain them.

Our results can be found in Fig. 3, which shows retrieval accuracy at 5 through 50 images retrieved on each dataset. HFD is clearly the best-performing method across all 3 problems. While DCA is competitive with HFD on the 10-class and 20-class sets, this performance drops off significantly on the more difficult 100-class problem.

The particularly strong performance of HFD on the 100-class problem may be due to the relaxed SSMMC formulation, which allows our method to effectively divide the very difficult 100-class discrimination problem into a sequence of many broader, easier problems, and thus make more effective use of its cannot-link constraints than the other metrics.

TABLE 4
Semi-supervised clustering results (V-Measure)

	Sonar			Balance		
	60	**120**	**180**	**45**	**90**	**180**
Euclidean	0.0493	0.0493	0.0493	0.2193	0.2193	0.2193
DCA	0.0959	0.1098	0.1386	0.0490	0.2430	0.3817
ITML	0.0650	0.0555	0.0644	0.2221	0.1915	0.2155
RFD	0.0932	0.1724	0.2699	0.1398	0.1980	0.3004
HFD	**0.1267**	**0.2296**	**0.3518**	**0.3059**	**0.5128**	**0.6149**

	Segmentation			USPS		
	70	**175**	**350**	**3k**	**5k**	**10k**
Euclidean	0.6393	0.6393	0.6393	0.6493	0.6493	0.6493
DCA	0.0510	0.2537	0.6876	0.5413	0.4359	0.4473
ITML	0.5682	0.6365	0.5931	0.6447	0.6445	0.6420
RFD	**0.7887**	**0.8157**	**0.8367**	**0.8248**	**0.8402**	0.8745
HFD	**0.7788**	**0.8090**	**0.8367**	0.7258	0.7397	**0.9087**

6.6 Semi-supervised clustering

In order to analyze the metrics holistically, in a way that takes into account not just ordered rankings of distances but the relative values of the distances themselves, we began by performing semi-supervised clustering experiments. We sampled varying numbers of constraints from each of the datasets presented and used these constraints to train the metrics. Note that only weakly- or semi-supervised metrics could be evaluated in this way, so only DCA, ITML, RFD and HFD were used in this experiment.

In order to evaluate the quality of the clusters produced, we used an external cluster evaluation metric called V-Measure [33]. V-measure computes two separate entropy-based scores, representing homogeneity (the extent to which each cluster contains elements from only a single class) and completeness (the extent to which all elements from a given class are assigned to the same cluster). The final score is a geometric mean of the two component scores.

After training, the learned metrics were applied to the dataset and used to retrieve the 50 nearest-neighbors and corresponding distances for each point. RFD and HFD return distances on a 0-1 scale, so we converted those to similarities by simply subtracting from 1. For the other methods, the distances were converted to similarities by applying a Gaussian kernel (we used $\sigma = 0.1$, 1, 10, 100 or 1000—whichever yielded the best results for that metric and dataset).

We then used the neighbor and similarity data to construct a number of sparse similarity matrices from varying numbers of nearest-neighbors (ranging from 5 to 50) and computed a spectral clustering [34] solution for each. We recorded the best result for each metric-dataset pair. Again, we repeated this testing process 10 times for each pair. The average results can be seen in

Table 4—the numbers below the dataset names indicate the number of constraints used in that test.

The tree based methods, RFD and HFD, demonstrated a consistent and significant advantage on this data. Between the two tree-based methods, HFD yielded better results on the sonar and balance data, while both were competitive on the segmentation and USPS datasets.

It is notable that the difference between the euclidean performance and that of the tree-based metrics is much more pronounced in the clustering domain. This would suggest that the actual distance values (as opposed to the distance rankings) returned by the tree-based metrics contain much stronger semantic information than those returned by the linear methods.

6.7 Multimetric methods

Many existing nonlinear metric learning methods take a multimetric approach, learning a number of different metrics (often one for each instance) spread throughout the data space. There are a number of successful (though generally not scalable) techniques in this class of methods, and we compare several such metrics against HFD. We were unable to obtain working code for these methods, and thus present the comparison data from [13].

These experiments were conducted on the k-nearest neighbor classification task, with $k = 11$, using 3-fold cross-validation. In each case 1% of all must-link and 1% of all cannot-link pairwise constraints were used. We repeat each experiment 10 times, using different constraint sets and cross-validations folds, and report the average results. The results can be seen in Table 6.

TABLE 5
Nearest neighbor classification accuracy against multimetric methods

	Balance	Diabetes	Breast	German	Haberman
HFD	**0.892**	**0.759**	**0.969**	**0.748**	**0.726**
ISD [11] (L1)	0.886	0.713	**0.969**	0.723	0.723
ISD (L2)	0.884	0.731	**0.97**	0.726	**0.727**
FSM [8]	0.866	0.658	0.898	0.725	0.724
FSSM [9]	0.857	0.678	0.888	0.725	0.724

In each case HFD matches or outperforms the multimetric methods. This may be because, while these methods are collectively nonlinear, they do assume the existence of a single linear metric *at each instance*. Given local linearity, this is not an unreasonable assumption in most cases, but may be problematic for points located in more sparsely populated regions of the data space. By contrast, HFD is fully nonlinear, and makes no assumptions about local linearity in the data.

6.8 High-dimensional data

While high-dimensional data (and particularly data where $d \gg n$) presents a challenge to any machine learning application, it is particularly troublesome for traditional Mahalanobis metrics. Solving for the Mahalanobis matrix $\mathbf{M}$ requires optimizing d^2 independent variables. When d is large, this quickly becomes both prohibitively costly and analytically dubious. By contrast, HFD needs only a subset of the available features in each node, and computes only a linear combination over this subset. As a result, high-dimensional data presents no special challenge to the method.

To illustrate this, we assess HFD's performance on the DBWorld emails [29] and arcene [35] datasets. The DBWorld data consists of 64 emails divided into two classes, represented by binary vectors indicating presence or absence of rooted words from the corpus (a total of 3721 features). As an example text dataset, DBWorld is comparatively quite small, both in number of samples and dimensionality, but it is nonetheless an extreme challenge for traditional metric learning techniques. The arcene dataset (we use only the training data, because ground truth labels were not published for the testing set) consists of 100 mass spectrometer readings of blood samples taken from either cancerous or healthy patients. Each reading has 10,000 features, including a number of artificial ''probe" features with no information content.

For both datasets, we performed a 5-nearest neighbor classification experiment, using 5-fold cross-validation. For DBWorld, we used 20 must-link and 20 cannot-link constraints in each fold, while for arcene we used 100. We report the average results across 10 runs of this experiment below.

We were unable to obtain results on either dataset for any of our comparison Mahalanobis metrics (the code either failed outright or produced no results even after days of runtime). Baseline Euclidean 5-NN accuracy was 0.546 for DBWorld and 0.720 for arcene. RFD did return results for this data, but did not perform well, yielding an accuracy of only 0.577 on DBWorld and 0.616 on arcene.

TABLE 6
HFD 5-nearest neighbor classification accuracy on high-dimensional datasets

Dataset	d_k						
	5	**10**	**25**	**50**	**100**	**200**	**500**
DBWorld	0.723	0.772	0.792	0.809	0.828	0.861	0.857
arcene	0.755	0.775	0.789	0.786	0.793	0.801	0.765

In both datasets, even with each node using only a tiny subset of the features (and with only a small subset of the pairwise constraints), HFD is learning an useful and effective model of the data, clearly outperforming the baseline. The learned model progressively increases in quality up to the $d_k = 500$ level, at which point the optimization space at each node is likely too large to obtain any additional information from the very limited training data. The noticeable decrease in performance at $d_k = 500$ on the arcene data may be attributable to the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2015.2507130, IEEE Transactions on Knowledge and Data Engineering

probe features—as the dimensionality of the optimization problem at each node increases relative to n, the odds of some of these noise features being erroneously assigned a high weight increases.

6.9 Baseline versus relaxed SSMMC

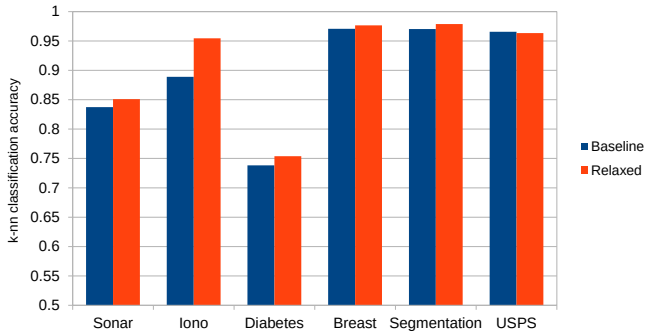


Fig. 4. 5-nearest neighbor classification accuracy under HFD, using baseline or relaxed SSMMC for $\mathcal{P}_{tl}$ function learning.

In order to empirically evaluate the effects of relaxing SSMMC to function in the hierarchical domain, we trained HFD models on several datasets using baseline SSMMC [24] to learn the $\mathcal{P}_{tl}$ functions. We then evaluated their performance at the k-nearest neighbor classification task, as described in Section 6.4, and compared the performance to that obtained by our relaxed formulation. The results are shown in Fig. 4.

The experiment show that our relaxation yields a significant improvement in performance for some, though not all, data. Relaxed SSMMC has a significant advantage on the sonar, ionosphere and diabetes sets, and a minor advantage on the breast and segmentation sets. On the USPS set, the two techniques yield essentially identical results.

These discrepancies can be interpreted in several ways. One possibility is that a sufficiently large training set enables the baseline algorithm to overcome the obstacles we identified in Section 3. With enough constraints and unconstrained points, the algorithm may not encounter badly unbalanced constraint sets until it reaches very low levels of the tree, at which point the learned hierarchy may already be a reasonably strong metric. Large numbers of constraints may also reduce the problem posed by the need for compromise among many irreconcilable cannot-link constraints—with enough constraints, the optimization may be able to achieve a similar result to the relaxed method by seeking the "least-bad" compromise.

Another possible explanation may be the level of linearity in the data. USPS and segmentation are both relatively linear sets, compared to sonar and ionosphere. With a sufficiently linear constraint satisfaction problem, the difficulties posed by local minima may be greatly reduced, allowing either algorithm to locate strong solutions.

Regardless, the results demonstrate that our relaxation approach can provide significant improvements in performance, and even in the worst case is comparable to the baseline.

7 Conclusion

In this paper, we have presented a novel semi-supervised nonlinear distance metric learning procedure based on forests of cluster hierarchies constructed via an iterative max-margin clustering procedure. We further propose a novel relaxed constraint formulation for max-margin clustering which improves the performance of the method in hierarchical problem settings. Our results show that this algorithm is competitive with the state-of-the-art on small- and medium-scale datasets, and superior for large-scale problems. We also present a novel inmetric approximate nearest-neighbor retrieval algorithm for our method that greatly decreases retrieval times for large data with little reduction in accuracy.

In the future, we hope to expand this metric to less-well-explored learning settings, such as those with more complex semantic relationship structures (e.g., hierarchies or "soft" class membership). By extending our method to incorporate relative similarity constraints, we could learn semi-supervised metrics even where binary pairwise constraints are no longer meaningful.

Acknowledgments

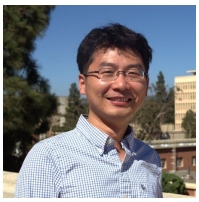
We are grateful for the support in part provided through the following grants: NSF CAREER IIS-0845282, ARO YIP W911NF-11-1-0090, DARPA Minds Eye W911NF-10-2-0062, DARPA CSSG D11AP00245, and NPS N00244-11-1-0022. Findings are those of the authors and do not reflect the views of the funding agencies.

References

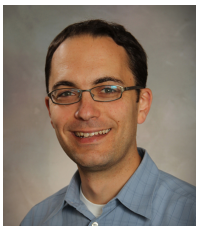
[1] A. Bellet, A. Habrard, and M. Sebban, "A survey on metric learning for feature vectors and structured data," *arXiv preprint arXiv:1306.6709*, 2013.
[2] J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon, "Information-theoretic metric learning," in *ICML*, 2007.
[3] C. Shen, J. Kim, L. Wang, and A. van den Hengel, "Positive semidefinite metric learning with boosting," in *NIPS*, 2009.
[4] J. Blitzer, K. Q. Weinberger, and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," in *NIPS*, 2005.
[5] Y. Ying and P. Li, "Distance metric learning with eigenvalue optimization," *JMLR*, vol. 13, 2012.
[6] R. Chatpatanasiri, T. Korsrilabutr, P. Tangchanachaianan, and B. Kijsirikul, "A new kernelization framework for mahalanobis distance learning algorithms," *Neurocomputing*, vol. 73, no. 10, 2010.
[7] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *CVPR*, 2005.
[8] A. Frome, Y. Singer, and J. Malik, "Image retrieval and classification using local distance functions," in *NIPS*, 2006.
[9] A. Frome, Y. Singer, F. Sha, and J. Malik, "Learning globally-consistent local distance functions for shape-based image retrieval and classification," in *ICCV*.

[10] K. Q. Weinberger and L. K. Saul, "Fast solvers and efficient implementations for distance metric learning," in *ICML*, 2008.
[11] D.-C. Zhan, M. Li, Y.-F. Li, and Z.-H. Zhou, "Learning instance specific distances using metric propagation," in *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 1225–1232.
[12] D. Kedem, S. Tyree, K. Weinberger, F. Sha, and G. Lanckriet, "Non-linear metric learning," in *NIPS*, 2012.
[13] C. Xiong, D. M. Johnson, R. Xu, and J. J. Corso, "Random forests for metric learning with implicit pairwise position dependence," in *SIGKDD*, 2012.
[14] K. L. Wagstaff, S. Basu, and I. Davidson, "When is constrained clustering beneficial, and why?" *Ionosphere*, vol. 58, no. 60.1, 2006.
[15] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, 2001.
[16] D. M. Johnson, C. Xiong, J. Gao, and J. J. Corso, "Comprehensive cross-hierarchy cluster agreement evaluation," in *AAAI Late-Breaking Papers*, 2013.
[17] L. Xu, J. Neufeld, B. Larson, and D. Schuurmans, "Maximum margin clustering," in *NIPS*, 2004.
[18] B. Zhao, F. Wang, and C. Zhang, "Efficient multiclass maximum margin clustering," in *ICML*, 2008.
[19] K. Zhang, I. W. Tsang, and J. T. Kwok, "Maximum margin clustering made practical," *TNN*, vol. 20, no. 4, 2009.
[20] M. Hoai and F. De la Torre, "Maximum margin temporal clustering," in *AISTATS*, 2012.
[21] C. Xiong, D. M. Johnson, and J. J. Corso, "Efficient max-margin metric learning," in *ECDM*, 2012.
[22] C. Shen, J. Kim, and L. Wang, "Scalable large-margin mahalanobis distance metric learning," *TNN*, vol. 21, no. 9, 2010.
[23] Y. Hu, J. Wang, N. Yu, and X.-S. Hua, "Maximum margin clustering with pairwise constraints," in *ICDM*, 2008.
[24] H. Zeng and Y.-m. Cheung, "Semi-supervised maximum margin clustering with pairwise constraints," *TKDE*, vol. 24, no. 5, 2012.
[25] A. L. Yuille and A. Rangarajan, "The concave-convex procedure," *Neural Computation*, vol. 15, no. 4, 2003.
[26] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, Sep. 1975. [Online]. Available: http://doi.acm.org/10.1145/361002.361007
[27] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration." in *VISAPP (1)*, 2009.
[28] A. Gionis, P. Indyk, R. Motwani *et al.*, "Similarity search in high dimensions via hashing," in *VLDB*, 1999.
[29] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml
[30] J. J. Hull, "A database for handwritten text recognition research," *PAMI*, vol. 16, no. 5, 1994.
[31] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Master's thesis, Department of Computer Science, University of Toronto*, 2009.
[32] S. C. Hoi, W. Liu, M. R. Lyu, and W.-Y. Ma, "Learning distance metrics with contextual constraints for image retrieval," in *CVPR*, 2006.
[33] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure." in *EMNLP-CoNLL*, 2007.
[34] J. Shi and J. Malik, "Normalized cuts and image segmentation," *PAMI*, vol. 22, no. 8, 2000.
[35] I. Guyon, S. Gunn, A. Ben-Hur, and G. Dror, "Result analysis of the nips 2003 feature selection challenge," in *Advances in Neural Information Processing Systems*, 2004, pp. 545–552.

David M. Johnson is a PhD student in the Computer Science and Engineering Department of SUNY Buffalo, and currently working as a visiting scholar at the Electrical Engineering and Computer Science Department of the University of Michigan. He recieved his BS Degree in Neuroscience from Brandeis University in 2009. His research interests include active and semi-supervised learning and computer vision.



Caiming Xiong is a senior researcher at Metamind. Previously he worked as a Postdoctoral Researcher in the Department of Statistics at the University of California, Los Angeles. He received his Ph.D. in computer science and engineering from SUNY Buffalo in 2014, and a B.S. and M.S. from Huazhong University of Science and Technology in 2005 and 2007, respectively. His research interests include interactive learning and clustering, computer vision and human-robot interaction.



Jason J. Corso is an associate professor of Electrical Engineering and Computer Science at the University of Michigan. He received his PhD and MSE degrees at The Johns Hopkins University in 2005 and 2002, respectively, and the BS Degree with honors from Loyola College In Maryland in 2000, all in Computer Science. He spent two years as a post-doctoral fellow at the University of California, Los Angeles. From 2007-14 he was a member of the Computer Science and Engineering faculty at SUNY Buffalo. He is the recipient of a Google Faculty Research Award 2015, the Army Research Office Young Investigator Award 2010, NSF CAREER award 2009, SUNY Buffalo Young Investigator Award 2011, a member of the 2009 DARPA Computer Science Study Group, and a recipient of the Link Foundation Fellowship in Advanced Simulation and Training 2003. Corso has authored more than one-hundred peer-reviewed papers on topics of his research interest including computer vision, robot perception, data science, and medical imaging. He is a member of the AAAI, ACM, AMS and the IEEE.