BoostMap: An Embedding Method for Efficient Nearest Neighbor Retrieval

Vassilis Athitsos, *Member, IEEE,* Jonathan Alon, *Member, IEEE,* Stan Sclaroff, *Member, IEEE,* and George Kollios, *Member, IEEE*

***Abstract*— This paper describes BoostMap, a method for efficient nearest neighbor retrieval under computationally expensive distance measures. Database and query objects are embedded into a vector space, in which distances can be measured efficiently. Each embedding is treated as a classifier that predicts for any three objects X, A, B whether X is closer to A or to B. It is shown that a linear combination of such embedding-based classifiers naturally corresponds to an embedding and a distance measure. Based on this property, the BoostMap method reduces the problem of embedding construction to the classical boosting problem of combining many weak classifiers into an optimized strong classifier. The classification accuracy of the resulting strong classifier is a direct measure of the amount of nearest neighbor structure preserved by the embedding. An important property of BoostMap is that the embedding optimization criterion is equally valid in both metric and non-metric spaces. Performance is evaluated in databases of hand images, handwritten digits, and time series. In all cases, BoostMap significantly improves retrieval efficiency with small losses in accuracy compared to brute-force search. Moreover, BoostMap significantly outperforms existing nearest neighbor retrieval methods, such as Lipschitz embeddings, FastMap, and VP-trees.**

***Index Terms*— Indexing methods, embedding methods, similarity matching, multimedia databases, nearest neighbor retrieval, nearest neighbor classification, non-Euclidean spaces.**

I. Introduction

Nearest neighbor retrieval is the task of identifying the database objects that are the most similar to a given query object. The most straightforward algorithm for nearest neighbor retrieval is brute-force search: we simply measure all distances between the query and the database objects. Clearly, as database size increases, brute-force search can become computationally demanding, or even impractical. This problem is exacerbated in domains with computationally expensive distance measures. Such measures occur frequently in pattern recognition. Examples include the Kullback-Leibler distance for matching probability distributions [1], dynamic time warping for matching time series [2], [3], or the edit distance [4] for matching strings and biological sequences. We introduce an embedding method, called BoostMap, for efficient nearest neighbor retrieval in such domains. BoostMap maps database and query objects into a real vector space, where distances can be computed orders-of-magnitude faster than in the original space. These fast embedding-based distances can be used to speed up nearest neighbor retrieval.

Our method makes two key contributions to the current state of the art. The first contribution is defining a new quantitative criterion of embedding quality, that directly measures how well the embedding preserves the nearest neighbor structure of the original space. The key idea is that any embedding F naturally defines a binary classifier $\tilde{F}$ that predicts, for any three objects X, A, B whether X is closer to A or to B, by simply checking if $F(X)$ is closer to $F(A)$ or to $F(B)$. If F never makes any mistakes, then F perfectly preserves nearest neighbor structure. We show that the error rate of F on a specific set of triples (X, A, B) is a direct measure of the amount of nearest neighbor structure preserved by F. This is in contrast to the global measures of stress and distortion typically used for evaluating embedding quality [5], which take into account all pairwise distances between objects, and thus mainly depend on pairs of objects that are not nearest neighbors of each other.

The second contribution is an algorithm for constructing and optimizing embeddings according to the proposed measure of embedding quality. We show that any linear combination of embedding-based binary classifiers $\tilde{F}$ naturally corresponds to an embedding and a distance measure. Consequently, the problem of constructing a multidimensional embedding is reduced to the classical boosting problem of combining many weak classifiers into an optimized strong classifier. The BoostMap method is based on this reduction and performs embedding optimization using AdaBoost [6]. An important property of BoostMap is that the embedding optimization criterion is equally valid in both metric and non-metric spaces.

The experiments evaluate the usefulness of BoostMap for efficient nearest neighbor retrieval and classification in relatively large databases with non-metric distance measures like the chamfer distance [7], shape context matching [8] and dynamic time warping [2], [3]. Using BoostMap leads to significant improvements in retrieval efficiency, with small losses in accuracy compared to brute-force search. Furthermore, BoostMap significantly outperforms existing methods for efficient nearest neighbor retrieval and classification in non-Euclidean spaces, such as Lipschitz embeddings [5], FastMap [9], and VP-trees [10].

II. Related Work

Various methods have been employed for speeding up nearest neighbor retrieval. The reader can refer to [5], [11]–[13] for comprehensive reviews of existing nearest neighbor methods. A large amount of work focuses on efficient nearest neighbor retrieval in multidimensional vector spaces [14]–[22]. Particular mention should be made to Locality Sensitive

Vassilis Athitsos, Jonathan Alon, Stan Sclaroff and George Kollios are with the Computer Science Department, Boston University, Boston MA, USA.

Hashing (LSH) [23], an approximate nearest neighbor method that has been shown theoretically to scale well with the number of dimensions and has produced good results in practice [24]–[26]. However, those methods can only be applied in vector spaces. The focus of this paper is on nearest neighbor retrieval in non-vector spaces induced by computationally expensive distance measures.

A more general class of spaces is the class of metric spaces, i.e., spaces with a metric distance measure. Examples of metric distance measures are the edit distance for strings [4], the Hausdorff distance for edge images [27], or bipartite matching for sets [28]. A number of nearest neighbor methods have been designed that are applicable to arbitrary metric spaces; the reader is referred to [12] for a comprehensive survey of such methods. VP-trees [10] hierarchically partition the database into a tree structure by partitioning, at each node, the set of objects based on whether they are closer than a threshold to a specific object, called a *pivot* object. A similar structure, called metric trees, has been proposed independently in [29]. MVP-trees [30] are an extension of VP-trees, where multiple pivot points are used at each node. M-trees [31] are a variant of metric trees explicitly designed for dynamic databases. Slim-trees [32] improve on M-trees by minimizing the overlap between nodes. An approximate variant of M-trees is proposed in [33], and achieves additional speed-ups by sacrificing the guarantee of always retrieving the true nearest neighbors. A general problem with the above-mentioned tree-based indexing methods is that their performance tends to approach brute-force search as the intrinsic dimensionality of the space increases. The reason is that, as dimensionality increases, distances to pivot objects are less likely to warrant pruning of large portions of the database.

Two alternative indexing methods for general metric spaces are AESA [34] and LAESA [35]. Those methods compute the exact distance between the query and a small set of database objects and then use the triangle inequality to establish lower bounds on the distance between the query and the database objects. However, by relying on the triangle inequality, those methods cannot handle non-metric distance measures such as the ones used in our experiments.

In domains with a computationally expensive distance measure, significant speed-ups can be obtained by embedding objects into another space with a more efficient distance measure. Several methods have been proposed for embedding arbitrary spaces into a Euclidean or pseudo-Euclidean space [9], [36]–[41]. Some of these methods, in particular Multidimensional Scaling (MDS) [41], Bourgain embeddings [5], [36], [42], Locally Linear Embedding (LLE) [38] and Isomap [39], need to evaluate exact distances between the query and most or all database objects, and thus are not designed for efficient nearest neighbor retrieval. Methods that can be used for efficient retrieval include Lipschitz embeddings [5], FastMap [9], MetricMap [40], and SparseMap [37].

BoostMap, the method described in this paper, was introduced in [43] and is an embedding method for efficient nearest neighbor retrieval. A key difference between BoostMap and existing embedding methods is that BoostMap optimizes a direct measure of the amount of nearest neighbor structure preserved by the embedding. Another distinguishing feature of BoostMap is that it addresses the problem of embedding optimization from a machine learning perspective, in contrast to the geometric perspective taken by existing methods [5], [9], [37], [40]. As a result, the embedding optimization criterion in BoostMap does not rely on any geometric properties, and is equally valid in Euclidean, metric, and non-metric spaces. In contrast, FastMap [9] and MetricMap [40] are based on Euclidean properties, and the design of Bourgain embeddings [36], [37] and SparseMap [37] (which is an approximation of Bourgain embeddings) is based on metric properties.

Non-metric distance measures are frequently used in pattern recognition. Examples of non-metric distance measures are the chamfer distance [7], shape context matching [8], dynamic time warping [3], or the Kullback-Leibler (KL) distance [1]. Methods that are designed for general metric spaces can still be applied when the distance measure is non-metric. However, methods that are exact for metric spaces become inexact in non-metric spaces, and no theoretical guarantees of performance can be made. BoostMap can guarantee correct retrieval results in metric spaces, as do Lipschitz embeddings and SparseMap [5], [37].

In several domains where BoostMap is applicable, methods have been proposed for speeding up similarity queries under the specific distance measures used in those domains. Various techniques have been proposed for time series databases using non-metric distance functions [3], [44], [45]. These techniques use the filter-and-refine approach [5], and use efficient distance approximations in the filter step.

One of the distance measures tested in the experiments is shape context matching [8]. Shape context matching is based on the shape context feature, which describes the distribution of points around a given location. Several methods have been proposed for speeding up similarity matching and classification using shape context. In [46], efficient retrieval is attained by pruning based on comparisons of a small subset of shape context features, and also using vector quantization. In [25] the Earth Mover's Distance between shape context features is efficiently approximated using an embedding, and then Locality Sensitive Hashing is applied. In [47] a discriminative classifier is learned based on correspondences of shape context features between the test object and a small number of training objects.

It is natural that a method designed for a specific distance measure, like the above-mentioned methods for time series matching and shape context matching, can sometimes lead to better performance than a general method applicable to arbitrary distance measures. At the same time, our method, which is general, does outperform some methods designed for specific distances [47], [48] in our experiments, and thus may be a viable alternative in applications where such methods are being used to improve efficiency. Furthermore, a method applicable to arbitrary measures has the advantage of being readily applicable in arbitrary settings and novel applications.

For nearest neighbor *classification* applications, there are also methods that explicitly speed up classification, and are not concerned with retrieving the true nearest neighbors. Condensing methods [49]–[51] speed up classification by trying to

identify training objects whose removal from the database does not hurt classification accuracy. By removing those objects from the database, the query needs to be compared to fewer objects in order to be classified. Approaches that speed up classification without reducing the database size are described in [52], [53]. In those approaches, tree-based index structures are constructed separately for each class. When it becomes clear, during search, that a specific class cannot achieve a majority of k-nearest neighbor votes, then that class is dropped from consideration. This way, the winning class can be identified without having to retrieve the actual k-nearest neighbors.

III. Background

Let $\mathbb{X}$ be a space of objects, and D be a distance measure in $\mathbb{X}$. If D is computationally expensive, a way to speed up retrieval is to embed objects into another space with a more efficient distance measure. Typically we construct an embedding $F : \mathbb{X} \rightarrow \mathbb{R}^d$, where distances in $\mathbb{R}^d$ are measured using a weighted Minkowski (L_p) metric, like the Euclidean (L_2) distance or the Manhattan (L_1) distance. We use Δ to denote the distance measure used in $\mathbb{R}^d$. We use the notation $\Delta_F(X_1, X_2)$ as shorthand for $\Delta(F(X_1), F(X_2))$.

Evaluating L_p distances in $\mathbb{R}^d$ takes time linear to the length d of the vectors. There are many spaces where we need to use distance measures that take time superlinear to the length of the objects. Such distance measures are common in spaces where objects are represented as sets or sequences of features or tokens, and where measuring the distance between two objects involves establishing optimal correspondences between the features/tokens of the two objects. Some examples of such spaces are:

- the space of binary edge images with the chamfer distance [7]. Each edge image in this space is represented as a set of edge pixels. Computing the chamfer distance involves computing the distance from each edge pixel in one image to its nearest edge pixel in the other image, and takes $O(d \log d)$ time for images with d edge pixels.
- the space of strings with the edit distance [4]. This distance is computed using dynamic programming, and takes time $O(d_1 d_2)$, where d_1 and d_2 are the lengths of the two strings. A related distance measure is dynamic time warping [2], [3], for comparing time series. Variants of the edit distance are also used for matching proteins and DNA sequences.

A. Some Simple Embeddings

Given any space $\mathbb{X}$ with a distance measure D, we can extend D to define the distance between elements of $\mathbb{X}$ and subsets of $\mathbb{X}$. Let $X \in \mathbb{X}$ and $\mathbb{P} \subset \mathbb{X}$. Then,

$$D(X, \mathbb{P}) = \min_{P \in \mathbb{P}} D(X, P) \ . \tag{1}$$

Given a subset $\mathbb{P} \subset \mathbb{X}$, a simple 1D embedding $F^{\mathbb{P}} : \mathbb{X} \rightarrow \mathbb{R}$ can be defined as follows:

$$F^{\mathbb{P}}(X) = D(X, \mathbb{P}) \ . \tag{2}$$

The set $\mathbb{P}$ that is used to define $F^{\mathbb{P}}$ is called a *reference set*. In many cases $\mathbb{P}$ can consist of a single object P, which is typically called a *reference object* or a *vantage object* [5]. In that case, we denote the embedding as F^P:

$$F^P(X) = D(X, P) \ . \tag{3}$$

We call F^P a *reference object embedding*.

Another family of simple, 1D embeddings is proposed in [9]. The idea there is to choose two objects $X_1, X_2 \in \mathbb{X}$, called pivot objects, and then, given an arbitrary $X \in \mathbb{X}$, to define the embedding F^{X_1, X_2} of X to be the *projection* of X onto the "line" $\overline{X_1 X_2}$:

$$F^{X_1, X_2}(X) = \frac{D(X, X_1)^2 + D(X_1, X_2)^2 - D(X, X_2)^2}{2D(X_1, X_2)} \ . \tag{4}$$

The reader can find in [9] an intuitive geometric interpretation of this equation, based on the Pythagorean theorem. We call F^{X_1, X_2} a *line projection embedding*.

A multidimensional embedding can be constructed by concatenating such 1D embeddings: if $F_1, \ldots, F_d$ are 1D embeddings, we can define a d-dimensional embedding F as $F(X) = (F_1(X), \ldots, F_d(X))$. In existing work, 1D embeddings defined using reference sets (Eq. 2) are used to form *Lipschitz* embeddings [5], and line projection embeddings (Eq. 4) are used to construct FastMap embeddings [9].

B. Embedding Application: Filter-and-refine Retrieval

Let F be an embedding from a space $\mathbb{X}$ with distance measure D to $\mathbb{R}^d$ with distance measure Δ, and let $\mathbb{U} \subset \mathbb{X}$ be a database of objects. We can use F to speed up k-nearest neighbor retrieval by applying the filter-and-refine framework [5], in which retrieval is done as follows:

- Offline preprocessing step: compute and store vector $F(U)$ for every database object $U \in \mathbb{U}$.
- Embedding step: given a query object Q, compute $F(Q)$.
- Filter step: rank all database objects in order of the distance of their embeddings from $F(Q)$.
- Refine step: for some integer p that is a parameter of the algorithm, rerank the p highest-ranked database objects by evaluating their exact distances D to Q.
- Output: return the k highest-ranked database objects.

The filter step provides a preliminary ranking of database objects by comparing d-dimensional vectors using the distance measure Δ. The refine step applies D only to the top p candidates. Assuming that Δ is significantly more efficient than D, filter-and-refine retrieval is much more efficient than brute-force retrieval.

IV. Overview of The BoostMap Method

An embedding F is *proximity-preserving* when it perfectly preserves proximity relations between triples of objects, i.e., when it holds for all $X, A, B \in X$ that

$$D(X, A) \leq D(X, B) \Leftrightarrow \Delta_F(X, A) \leq \Delta_F(X, B) \ . \tag{5}$$

If Eq. 5 does not hold for some triple (X, A, B), we say that F *fails* on that triple. In BoostMap, the goal is to

construct an embedding that is as close to being proximity-preserving as possible. For the purpose of speeding up nearest neighbor retrieval it is sufficient to limit our attention to triples (X, A, B) of a specific type, as discussed in Section V-C.

Deciding for a triple (X, A, B) whether X is closer to A or to B is a binary classification problem (we ignore the typically rare case where X is equally far from A and B). Any embedding F defines a binary classifier $\tilde{F}$ that decides whether X is closer to A or to B by simply checking if $F(X)$ is closer to $F(A)$ or to $F(B)$. Our goal is to construct an embedding F whose associated classifier $\tilde{F}$ is as accurate as possible. In Section V we show that the task of optimizing a multidimensional embedding that is a concatenation of 1D embeddings is equivalent to the task of designing a good linear combination of classifiers. The latter task is a natural fit for boosting methods proposed in the machine learning literature. In the BoostMap method we optimize embedding quality using AdaBoost [6], as described in Section VI.

V. Associating Embeddings with Classifiers

A. Using Embeddings to Define Classifiers

As in previous sections, $\mathbb{X}$ is a space of objects and D is a distance measure defined on $\mathbb{X}$. If (X, A, B) is a triple of objects in $\mathbb{X}$, one of the following three cases must be true:

- X is closer to A than to B.
- X is equally far from A and B.
- X is closer to B than to A.

In order to denote, for each triple (X, A, B), which of those three possibilities is true, we define the *proximity order* P of triple (X, A, B) as follows:

$$P(X,A,B) = \begin{cases} 1 & \text{if } D(X,A) < D(X,B) \ . \\ 0 & \text{if } D(X,A) = D(X,B) \ . \\ -1 & \text{if } D(X,A) > D(X,B) \ . \end{cases} \quad (6)$$

In spaces where distances can take any value within some range of real numbers, it is typically unusual for an object to have the exact same distance to two database objects. Consequently, we consider the task of estimating $P(X, A, B)$ to be a *binary* classification task.

Let F be an embedding that maps (X, D) to $(\mathbb{R}^d, \Delta)$. We can guess whether X is closer to A or to B by checking if $F(X)$ is closer to $F(A)$ or to $F(B)$. More formally, for every embedding F we define a classifier $\tilde{F}$ as follows:

$$\tilde{F}(X,A,B) = \Delta_F(X,B) - \Delta_F(X,A) \ . \quad (7)$$

If we define $\text{sign}(x)$ to be 1 for $x > 0$, 0 for $x = 0$, and -1 for $x < 0$, then $\text{sign}(\tilde{F}(X, A, B))$ is an estimate of $P(X, A, B)$.

B. Using Classifiers to Define Embeddings

At this point we have established that every embedding $F : (\mathbb{X}, D) \rightarrow (\mathbb{R}^d, \Delta)$ corresponds to a binary classifier $\tilde{F}$ of triples of objects. It is shown in [54] that the converse does not hold: there exist classifiers H of triples such that $H \neq \tilde{F}$ for all embeddings F. At the same time, there is always an embedding F such that $H = \tilde{F}$, if H is of the following form:

$$H(X,A,B) = \sum_{j=1}^{J} \alpha_j \tilde{F}_j(X,A,B) \ , \quad (8)$$

where J is any positive integer and each F_j is an embedding mapping $\mathbb{X}$ and D to some real vector space $\mathbb{R}^{d_j}$ and some distance measure Δ^j.

Proposition 1: If classifier H is of the form of Eq. 8, then we can define an embedding F and distance measure Δ such that $F : (\mathbb{X}, D) \rightarrow (\mathbb{R}^d, \Delta)$ and $H = \tilde{F}$, for some integer d.

Proof: Given that $H(X, A, B) = \sum_{j=1}^{J}(\alpha_j \tilde{F}_j(X, A, B))$, we define F and Δ as follows:

$$F(X) = (F_1(X), \ldots, F_J(X)) \ .$$

$$\Delta(F(X_1), F(X_2)) = \sum_{j=1}^{J}(\alpha_j \Delta^j(F_j(X_1), F_j(X_2))) \ .$$

Embedding F maps $\mathbb{X}$ into a d-dimensional vector space, where $d = \sum_{j=1}^{J} d_j$, and Δ is the sum of individual distances Δ^j that correspond to embeddings F_j.

Given these definitions, the proof that $H = \tilde{F}$ can be obtained in a few simple steps, by starting from the definition of $\tilde{F}$ in Equation 7:

$$\begin{aligned} \tilde{F}(X,A,B) &= \Delta_F(X,B) - \Delta_F(X,A) \\ &= \sum_{j=1}^{J}(\alpha_j \Delta^j_{F_j}(X,B)) - \sum_{j=1}^{J}(\alpha_j \Delta^j_{F_j}(X,A)) \\ &= \sum_{j=1}^{J}(\alpha_j (\Delta^j_{F_j}(X,B) - \Delta^j_{F_j}(X,A))) \\ &= \sum_{j=1}^{J}(\alpha_j \tilde{F}_j(X,A,B)) = H(X,A,B) \ . \end{aligned}$$

□

We have shown that if classifier H is a weighted linear combination of classifiers corresponding to embeddings, then H itself is equivalent to a specific embedding F and a specific distance measure Δ. By the word "equivalent" we mean that, for any (X, A, B), H misclassifies (X, A, B) if and only if F fails on that triple. This equivalence allows us to map the problem of embedding optimization to the problem of optimizing a weighted linear combination of binary classifiers, which is exactly the problem that boosting methods are designed to solve.

C. Classification Error As a Measure of Embedding Quality

Suppose that we have a database $\mathbb{U} \subseteq \mathbb{X}$, and in our application we are only interested in retrieving up to $k_{\max}$ nearest neighbors for each query object $X \in \mathbb{X}$. An example of such an application is k-nearest neighbor classification, where for every test object we want to retrieve k database objects, so $k_{\max} = k$ in that case. We denote the set of the $k_{\max}$ nearest neighbors of X in $\mathbb{U}$ as $NN(X, \mathbb{U}, k_{\max})$. In order to achieve perfect retrieval accuracy of up to $k_{\max}$ nearest neighbors using an embedding F, it suffices that classifier $\tilde{F}$ be perfect on a set of triples $\mathbb{T}_{k_{\max}}$ defined as follows:

$$\mathbb{T}_{k_{\max}} = \{(X,A,B) | X \in \mathbb{X}, A \in NN(X, \mathbb{U}, k_{\max}), B \in \mathbb{U}\} \ . \quad (9)$$

In the above equation, B can be any database object.

In the ideal case where $\tilde{F}$ makes no mistakes on triples in $\mathbb{T}_{k_{\max}}$, the following holds: $\forall X, A \in \mathbb{X}, \forall k \in \{1, \ldots, k_{\max}\}$, A is the k-nearest neighbor of X in $\mathbb{U}$ iff $F(A)$ is the k-nearest neighbor of $F(X)$ in $F(\mathbb{U})$. Therefore, using F we can retrieve the correct $k_{\max}$ nearest neighbors for any query object, without needing to measure any exact distances D. If $\tilde{F}$ misclassifies triples that are not in $\mathbb{T}_{k_{\max}}$, retrieval accuracy is not affected. In the typical case where $\tilde{F}$ is not perfectly accurate on $\mathbb{T}_{k_{\max}}$, the error rate of $\tilde{F}$ on $\mathbb{T}_{k_{\max}}$ provides us with a quantitative measure of how well F preserves the $k_{\max}$-nearest neighbor structure of $\mathbb{X}$. Higher error rates indicate that the proximity rankings obtained using F are less reliable. Therefore, if we wish to optimize classifier $\tilde{F}$, it is important to optimize it for accuracy on the set $\mathbb{T}_{k_{\max}}$, as opposed to, for example, accuracy on the set of all possible triples (which was done in [43]).

It is interesting to compare the measure of embedding quality we have proposed, i.e., the classification error on the set $\mathbb{T}_{k_{\max}}$, with the measures of stress and distortion that are often used to evaluate embedding quality [5]. The measure proposed here is fundamentally a local measure. Assuming that $k_{\max} \ll |\mathbb{U}|$, the vast majority of triples of objects (X, A, B) are such that neither A nor B is one of the $k_{\max}$-nearest neighbors of X, and therefore we are not concerned about classifying such triples correctly. In contrast, stress and distortion are global measures that are affected by every pair of objects, although the vast majority of pairs of objects (X, A) are such that X and A are not $k_{\max}$-nearest neighbors of each other. Arguably a method that minimizes stress or distortion spends most of its effort on pairs of objects that have no bearing on how well the embedding preserves nearest neighbor structure.

VI. BoostMap: Optimizing Embedding Construction Using Boosting

As stated in Section V-C, our goal is to construct an embedding $F_{\text{out}} : (\mathbb{X}, D) \rightarrow (\mathbb{R}^d, \Delta)$ in a way that minimizes the classification error of classifier $\tilde{F}_{\text{out}}$ on a specific set of triples. The building blocks we will use for embedding construction are simple, 1D embeddings defined using database objects, according to Eqs. 3 and 4. By applying Eq. 7 to each such 1D embedding we obtain a large pool of binary classifiers. As long as such embeddings preserve at least a small amount of the structure of the original space, we expect the corresponding binary classifiers to be more accurate than a random guess. In other words, we expect the classifiers associated with 1D embeddings to behave as *weak classifiers* [6].

Based on the above considerations, and using the correspondence we have established between embeddings and classifiers, we reduce the problem of embedding optimization to the problem of optimizing a weighted linear combination of binary weak classifiers. Naturally, this is exactly the problem that boosting methods [6], [55] have been designed to solve. In our embedding construction algorithm we have chosen to use the AdaBoost method [6].

The AdaBoost algorithm is shown in Algorithm 1. The inputs to AdaBoost are a set of objects o_i, together with their corresponding class labels y_i, which are equal either to -1 or to 1. In our problem, each o_i corresponds to a triple of objects of $\mathbb{X}$. The goal in AdaBoost is to construct a strong classifier that achieves much higher accuracy than the individual weak classifiers.

Algorithm 1: The AdaBoost algorithm. This description is largely copied from [6].

input : $(o_1, y_1), \ldots, (o_\beta, y_\beta)$; $o_i \in \mathcal{G}, y_i \in \{-1, 1\}$.
output: Strong classifier $H : \mathcal{G} \rightarrow \mathbb{R}$.
Initialize $w_{i,1} = \frac{1}{\beta}$, for $i = 1, \ldots, \beta$.
for training round $j = 1, \ldots, J$: **do**

1. Train weak learner using training weights $w_{i,j}$, and obtain weak classifier $h_j : \mathcal{G} \rightarrow \mathbb{R}$, and a corresponding weight $\alpha_j \in \mathbb{R}$.

2. Set training weights $w_{i,j+1}$ for the next round as follows:

$$w_{i,j+1} = \frac{w_{i,j} \exp(-\alpha_j y_i h_j(o_i))}{z_j} \quad . \tag{10}$$

where z_j is a normalization factor (chosen so that $\sum_{i=1}^{\beta} w_{i,j+1} = 1$).

end
Output the final classifier:

$$H(x) = \sum_{j=1}^{J} \alpha_j h_j(x). \tag{11}$$

The BoostMap algorithm is an adaptation of AdaBoost to the problem of embedding construction. In order to apply AdaBoost to our problem we need to perform some preprocessing before invoking AdaBoost, we need to specify how to implement the first step of the AdaBoost main loop, and finally we need to convert the output classifier of AdaBoost into an embedding. We now describe in detail how to perform each of these steps.

A. Inputs and Preprocessing

The inputs to the BoostMap algorithm are the following:

- A database $\mathbb{U}$ of objects in some space $\mathbb{X}$ with distance measure D.
- A positive integer $k_{\max}$ specifying the maximum number of nearest neighbors we will be interested in retrieving using the resulting embedding.
- A set $\mathbb{C} \subset \mathbb{U}$ of candidate reference and pivot objects. Elements of $\mathbb{C}$ will be used to define 1D embeddings.
- A set $\mathbb{L} \subset \mathbb{U}$ of training objects. Elements of $\mathbb{L}$ will be used to form training triples, i.e., the o_i's used by AdaBoost.
- Matrices of distances: from each $X_1 \in \mathbb{C}$ to each $X_2 \in \mathbb{C}$, from each $X_1 \in \mathbb{C}$ to each $X_2 \in \mathbb{L}$, and from each $X_1 \in \mathbb{L}$ to each $X_2 \in \mathbb{L}$.

In addition, we need to specify parameters that control the runtime of the training algorithm:

- The size β of the set $\mathcal{G}$ of training triples.
- The number γ of weak classifiers to consider at each training round.
- The number δ of classifiers selected after a quick scan at each training round j. These selected classifiers are then evaluated more thoroughly in order to choose h_j and α_j.
- A parameter $Z_{\max}$ that will be used for deciding when to stop the training algorithm.

The role of these parameters will be fully explained in the description of the training algorithm.

The goal of the training algorithm is to construct an embedding F_{out} in a way that minimizes the classification error of the corresponding classifier $\tilde{F}_{\text{out}}$ on $\mathbb{T}_{k_{\max}}$, the set of triples defined in Eq. 9. During the course of the algorithm we need to keep in memory a matrix of distances from every object in $\mathbb{C}$ to every object included in any of the training triples. To reduce the memory requirements we choose training triples not from the entire database, but from a smaller set $\mathbb{L} \subset \mathbb{U}$.

Given $\mathbb{L}$, we define $k' \equiv \lceil \frac{k_{\max}|\mathbb{L}|}{|\mathbb{U}|} \rceil$. Then, we choose β training triples $o_i = (X_i, A_i, B_i)$ randomly, subject to the constraints that: 1). A_i is a k'-nearest neighbor of X_i in $\mathbb{L} - \{X_i\}$, and 2). A_i and B_i are *not* equally far from X_i. We set class label y_i of o_i -1 or 1, according to the proximity order $P(X_i, A_i, B_i)$, as defined in Eq. 6. The formula we use for setting k' makes the training triple selection process approximate sampling from $\mathbb{T}_{k_{\max}}$, under the constraint that each X_i, A_i and B_i must be an element of $\mathbb{L}$. If A_i is one of the k'-nearest neighbors of X_i in $\mathbb{L}$, A_i is likely to be one of the the $k_{\max}$-nearest neighbors of X_i in $\mathbb{U}$ (unless $|\mathbb{L}| < \frac{|\mathbb{U}|}{k_{\max}}$, in which case A_i is just likely to be one of the $\frac{|\mathbb{U}|}{|\mathbb{L}|}$ nearest neighbors of X_i).

Now we proceed to specify how to implement the training algorithm, i.e., how to implement Step 1 of the main loop of AdaBoost, as shown in Algorithm 1. We should note that, in Algorithm 1, Step 2 of the main loop is fully specified.

B. Choosing the Next Weak Classifier and Weight

At training round j, given training weights $w_{i,j}$, the weak learner is called to provide us with a weak classifier h_j and a weight α_j. In BoostMap the weak learner simply evaluates a large number of weak classifiers, and finds the best classifier and best weight for that classifier. Each weak classifier is a classifier $\tilde{F}_i$ where F_i is a 1D embedding. In [56]–[58] we have described alternative families of weak classifiers that can be used within the context of this algorithm.

As described in [6], the function $Z_j(h, \alpha)$ gives a measure of how useful it would be to choose $h_j = h$ and $\alpha_j = \alpha$ at training round j:

$$Z_j(h, \alpha) = \sum_{i=1}^{\beta} (w_{i,j} \exp(-\alpha y_i h(X_i, A_i, B_i))) \ . \qquad (12)$$

The full details of the significance of Z_j can be found in [6]. Here it suffices to say that if $Z_j(\tilde{F}, \alpha) < 1$ then choosing $h_j = h$ and $\alpha_j = \alpha$ is overall beneficial, and is expected to reduce the training error. Overall, lower values of $Z_j(\tilde{F}, \alpha)$ are preferable to higher values.

Finding the optimal α for a given classifier h and computing the Z_j value attained using that optimal α are very common operations in our algorithm, so we define specific notation:

$$A_{\min}(h, j, l) = \text{argmin}_{\alpha \in [l, \infty)} Z_j(h, \alpha) \ . \qquad (13)$$

$$Z_{\min}(h, j, l) = \min_{\alpha \in [l, \infty)} Z_j(h, \alpha) \ . \qquad (14)$$

In the above equations j specifies the training round, and l specifies the smallest value we allow for α. Function $A_{\min}(h, j, l)$ returns the α that minimizes $Z_j(h, \alpha)$, subject to the constraint that $\alpha \geq l$. We compute $A_{\min}$ following the optimization method described in [6]. Argument l will be used to ensure that no classifier has a negative weight. In Section VI-C we will use classifier weights to define a weighted L_1 distance measure Δ in $\mathbb{R}^d$, and non-negative weights ensure that Δ is a metric.

The number of classifiers to evaluate is specified by parameter γ of the algorithm. As an implementation choice, half of these γ classifiers are reference-object embeddings and half are line-projection embeddings. A very simple way of implementing Step 1 from Algorithm 1 is to: 1). define γ weak classifiers by picking randomly $\gamma/2$ reference objects and $\gamma/2$ pairs of pivot objects from $\mathbb{C}$, and 2). set h_j and α_j to be the classifier h (among those γ weak classifiers) and weight α that minimize $Z_j(h, \alpha)$.

The implementation that we actually use in our experiments differs from the above description in two ways. The first difference is that, before selecting a new weak classifier, we check whether removing or modifying the weight of an already selected weak classifier would improve the strong classifier. Removals and weight modifications that improve the strong classifier are given preference because they do not increase the complexity of the strong classifier.

The second difference is that, instead of evaluating every weak classifier h using function $Z_{\min}$, we first evaluate all weak classifiers using an alternative measure, the training error $\Lambda_j(h)$:

$$\Lambda_j(h) = \sum_{i=1}^{\beta} w_{i,j} \frac{y_i - \text{sign}(h(X_i, A_i, B_i))}{2} \ . \qquad (15)$$

Using function Λ_j we select the best δ classifiers (given parameter δ), and then among those δ classifiers we choose the best one using function $Z_{\min}$. Function $Z_{\min}$ takes an order of magnitude more time to compute than the training error, because computing $Z_{\min}$ involves searching for the optimal weight $A_{\min}$, whereas computing the training error does not involve such a search. The step of selecting a smaller set of classifiers based on training error can be skipped if the running time of the training algorithm is not a concern.

Our implementation of Step 1 of the AdaBoost algorithm is shown in Algorithm 2. In that algorithm, we denote with H_j the classifier assembled by AdaBoost after j training rounds, so that $H_j = \sum_{i=1}^{j} \alpha_i h_i$. It is possible that some weak classifier occurs multiple times in H_j, i.e., that there exist $i, g < j$ such that $h_i = h_g$. Without loss of generality we assume that we also have an alternative representation of H_{j-1} as a weighted linear combination of unique weak classifiers. We denote that representation as $H_{j-1} = \sum_{i=1}^{K_{j-1}} \alpha'_{i,j-1} h'_{i,j-1}$.

Note that, as specified in Step 26 of Algorithm 2, the algorithm terminates when we select a new weak classifier h_j for which $Z_j(h_j, \alpha_j) \geq Z_{\max}$, meaning that we have failed to find a weak classifier that would be more than marginally beneficial to add to the strong classifier.

Algorithm 2: The steps of the BoostMap training algorithm. Steps 3-23 implement Step 1 of the AdaBoost algorithm, as shown in Algorithm 1.

1 Initialize training weights $w_{i,1} \leftarrow \frac{1}{\beta}$, for $i = 1, \ldots, \beta$,
2 Initialize $H_0 \leftarrow 0$, $j \leftarrow 1$.
3 $z \leftarrow \min_{c=1,\ldots,K_{j-1}} Z_j(h'_{c,j-1}, -\alpha'_{c,j-1})$.
4 **if** $z < 1$ **then**
  `/* Remove an already selected weak classifier, by adding its negation. */`
5 $g \leftarrow \text{argmin}_{c=1,\ldots,K_{j-1}} Z_j(h'_{c,j-1}, -\alpha'_{c,j-1})$.
6 $h_j \leftarrow h'_{g,j-1}$.
7 $\alpha_j = -\alpha'_{g,j-1}$.
8 **Goto** Step 27.
9 **end**
10 $z \leftarrow \min_{c=1,\ldots,K_{j-1}} Z_{\min}(h'_{c,j-1}, j, -\alpha'_{c,j-1})$.
11 **if** $z < Z_{\max}$ **then**
  `/* Modify the weight of an already selected weak classifier. The third argument of` $Z_{\min}$ `ensures that the new weight` $\alpha'_{g,j}$ `of` h'_g `is non-negative. */`
12 $g \leftarrow \text{argmin}_{c=1,\ldots,K_{j-1}} Z_{\min}(h'_{c,j-1}, j, -\alpha'_{c,j-1})$.
13 $h_j \leftarrow h'_{g,j-1}$.
14 $\alpha_j \leftarrow A_{\min}(h'_g, j, -\alpha'_{g,j-1})$.
15 **Goto** Step 27.
16 **end**
17 $\mathbb{F}_{j1} \leftarrow \{F^{X_1}, \ldots, F^{X_{\gamma/2}}\}$, where $X_1, \ldots, X_{\gamma/2}$ are random elements of the set $\mathbb{C}$ of candidate objects.
18 $\mathbb{F}_{j2} \leftarrow \{F^{X_{i,1}, X_{i,2}} \mid i = 1, \ldots, \gamma/2\}$, where $X_{1,1}, X_{1,2}, ..., X_{\gamma/2,1}, X_{\gamma/2,2}$ are random objects of $\mathbb{C}$.
19 $\mathbb{F}_j \leftarrow \mathbb{F}_{j1} \cup \mathbb{F}_{j2}$.
20 $\tilde{\mathbb{F}}_J \leftarrow \{\tilde{F} \mid F \in \mathbb{F}_j\}$.
21 $\mathbb{H}_j \leftarrow$ set of the δ classifiers h in $\tilde{\mathbb{F}}_J$ with the smallest $\Lambda_j(h)$.
22 $h_j \leftarrow \text{argmin}_{h \in \mathbb{H}_j} Z_{\min}(h, j, 0)$.
23 $\alpha_j \leftarrow A_{\min}(h_j, j, 0)$.
24 **if** $Z_j(h_j, \alpha_j) \geq Z_{\max}$ **then**
25 **return** H_{j-1}
26 **end**
27 $z_j \leftarrow Z_j(h_j, \alpha_j)$.
28 Set weights $w_{i,j+1}$ for training round $j+1$ using Eq. 10.
29 $j \leftarrow j + 1$.
30 **Goto** Step 3.

C. Defining an Embedding and a Distance Measure

The output of AdaBoost is a strong classifier H. Without loss of generality, we can write H as $H = \sum_{c=1}^{d} \alpha'_c \tilde{F}_c$, where each $\tilde{F}_c$ is associated with a unique 1D embedding F_c. Classifier H has been trained to estimate, for triples of objects (X, A, B), if X is closer to A or to B. However, our final goal is to construct not a classifier, but an embedding. To achieve that we use Proposition 1, to convert H into an embedding $F_{\text{out}} : \mathbb{X} \rightarrow \mathbb{R}^d$ and a distance measure Δ:

$$F_{\text{out}}(x) = (F_1(x), ..., F_d(x)) \ . \tag{16}$$

$$\Delta((u_1, ..., u_d), (v_1, ..., v_d)) = \sum_{c=1}^{d} (\alpha'_c |u_c - v_c|) \ . \tag{17}$$

Δ is a weighted Manhattan (L_1) distance measure. Δ is a metric, because the training algorithm ensured that all α_c's are non-negative, and thus we can apply to the resulting embedding any additional indexing, clustering and visualization tools that are available for L_1 metric spaces.

VII. Properties of BoostMap Embeddings

In this section we take a closer look at some properties of the proposed algorithm for constructing embeddings, and of the resulting embeddings.

A. Contractiveness

An embedding $F : (\mathbb{X}, D) \rightarrow (\mathbb{R}^d, \Delta)$ is contractive if for any $X_1, X_2 \in \mathbb{X}$ it holds that $\Delta_F(X_1, X_2) \leq D(X_1, X_2)$. As explained in [5], when an embedding is contractive, then filter-and-refine retrieval can guarantee retrieval of the true nearest neighbors.

The output embedding $F_{\text{out}} : (\mathbb{X}, D) \rightarrow (\mathbb{R}^d, \Delta)$, constructed as described in Eqs. 16-17, can be made contractive by dividing $\Delta(F_{\text{out}}(X_1), F_{\text{out}}(X_2))$ with a normalization term, provided that D is metric. First, we address the case where F_{out} contains no line projection embeddings:

Proposition 2: Let $X_1, X_2 \in \mathbb{X}$. Suppose that, in Eq. 17, $\alpha'_i > 0$ for all i, and suppose that D is metric. If all dimensions of F_{out} are reference-object embeddings, then it holds that:

$$\frac{1}{\sum_{i=1}^{d} \alpha'_i} \Delta(F_{\text{out}}(X_1), F_{\text{out}}(X_2)) \leq D(X_1, X_2) \ . \tag{18}$$

Proof: If each dimension of F_{out} is a reference-object embedding, then F_{out} can be represented as $F_{\text{out}} = (F^{P_1}, \ldots, F^{P_d})$, where d is the dimensionality of F_{out} and P_i are reference objects. We will denote $F_{\text{out}}(X_1)$ as $(x_{1,1}, \ldots, x_{1,d})$ and $F_{\text{out}}(X_2)$ as $(x_{2,1}, \ldots, x_{2,d})$. First, based on the triangle inequality, we can easily see that:

$$|x_{1,i} - x_{2,i}| = |D(X_1, P_i) - D(X_2, P_i)| \leq D(X_1, X_2) \ . \tag{19}$$

Using this observation, we can complete the proof:

$$\begin{aligned}
\frac{1}{\sum_{i=1}^{d} \alpha'_i} \Delta_F(X_1, X_2) &= \frac{1}{\sum_{i=1}^{d} \alpha'_i} \sum_{i=1}^{d} (\alpha'_i |x_{1,i} - x_{2,i}|) \\
&\leq \frac{1}{\sum_{i=1}^{d} \alpha'_i} \sum_{i=1}^{d} (\alpha'_i D(X_1, X_2)) \\
&= \frac{1}{\sum_{i=1}^{d} \alpha'_i} D(X_1, X_2) \sum_{i=1}^{d} \alpha'_i \\
&= D(X_1, X_2) \ .
\end{aligned}$$

□

If F_i, the i-th dimension of F_{out}, is a line-projection embedding, then it is shown in [5] that $|F_i(X_1) - F_i(X_2)| \leq 3D(X_1, X_2)$. Therefore, if we divide $\Delta(X_1, X_2)$ by $3\sum_{i=1}^{d} \alpha'_i$, then F_{out} is contractive even in the case where some of its dimensions are line-projection embeddings.

We should point out that the distance measures D used in the experiments are non-metric, and thus the resulting embeddings are not contractive. No existing domain-independent embedding method [9], [37], [40] is contractive in non-metric spaces. Consequently, none of these methods, including BoostMap, can guarantee perfect retrieval accuracy in non-metric spaces.

B. Complexity

Before we start the training algorithm, we need to compute three distance matrices: distances between objects in $\mathbb{C}$, distances between objects in $\mathbb{L}$, and distances from objects in $\mathbb{C}$ to objects in $\mathbb{L}$. If (as in our experiments) $|\mathbb{C}| = |\mathbb{L}|$, then the number of distances that we need to precompute is quadratic to $|\mathbb{C}|$. During training, at each training round we evaluate γ weak classifiers by measuring their performance on β training triples, which takes $O(\beta\gamma)$ time. In contrast, FastMap [9], SparseMap [37], and MetricMap [40] do not require training at all. However, we should emphasize that the cost of training is a *one-time preprocessing cost*. In many applications this cost is acceptable, as long as it results in better trade-offs between retrieval accuracy and efficiency.

Computing the d-dimensional embedding F_{out} of an object takes $O(d)$ time and requires measuring between d and $2d$ exact distances D: one distance for each reference object embedding in F_{out}, and two distances for each line projection embedding in F_{out}. For comparison, the number of distance evaluations required by other methods to embed an object is: $2d$ for FastMap, d for SparseMap, and $d+1$ for MetricMap. It follows that, for BoostMap and these other methods, computing the embeddings of all database objects takes $O(d|\mathbb{U}|)$ time, inserting a new object to the database requires $O(d)$ time to compute its embedding, and computing the d-dimensional embedding of a query object also takes $O(d)$ time.

Comparing, during the filter step, the embedding of the query to the embeddings of n database objects takes time $O(dn)$. As d increases, this becomes more expensive. However, in our experiments, the filter step always takes negligible time; retrieval time is dominated by the few exact distance computations we need to perform at the embedding step and the refine step.

VIII. Experiments

In this section we experimentally evaluate BoostMap by comparing it to several alternative existing methods for efficient nearest neighbor retrieval. Experiments are performed in four different domains: hand shape classification using a database of hand images, offline handwritten digit recognition using the MNIST database [59], online handwritten digit recognition using the isolated digits benchmark (category 1a) of the UNIPEN Train-R01/V07 database [60], and similarity-based retrieval of time series using a benchmark time series dataset [44].

A. Datasets

Here we provide details about each of the four datasets we use in the experiments. More detailed descriptions of each dataset can be found at the experiments section of [54].

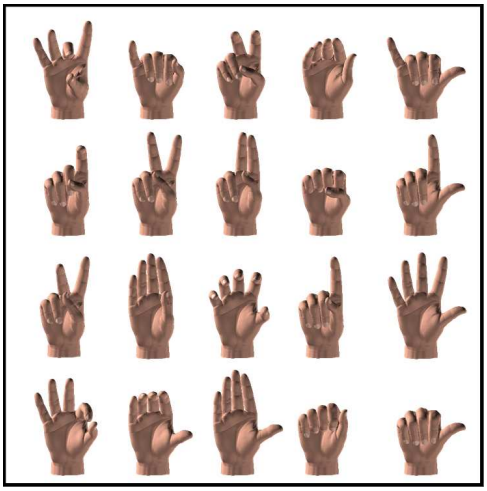


Fig. 1. The 20 handshapes used in the ASL handshape dataset.

1) ASL Handshape Dataset: The ASL handshape dataset consists of a database of $80,640$ synthetic images of hands, generated using the Poser 5 software [61], and a query set of 710 real images of hands. All images display the hand in one of 20 different 3D handshape configurations (Fig. 1), which are all commonly used in American Sign Language (ASL). For each of the 20 handshapes we synthetically generate a total of $4,032$ database images that correspond to different 3D orientations of the hand.

The query images are obtained from video sequences of a native ASL signer either performing individual handshapes in isolation or signing in ASL. The hand locations were extracted from those sequences using the method described in [62]. Accurate localization of the hand in such sequences remains a very challenging task, and hand localization fails in more than 50% of the frames. In these experiments we only use frames where the hand is localized correctly. The query images are obtained from the original frames by extracting the subwindow corresponding to the hand region. Database and query images are normalized so that the minimum enclosing circle of the hand region has radius 120.

The distance measure used to compare images is the chamfer distance [7], [63], which operates on edge images. The synthetic images generated by Poser can be rendered directly as edge images. For the query images we simply apply the Canny edge detector [64]. On a AMD Athlon 2GHz processor, we can compute on average 715 chamfer distances per second. Nearest-neighbor classification via brute-force search takes about 112 seconds per query, and yields an error rate of 67%.

2) Offline Handwritten Digit Dataset (MNIST): The MNIST dataset of handwritten digits [59] contains $60,000$ database images, and $10,000$ query images. Each image is a 28x28 image displaying an isolated digit between 0 and 9 (Fig. 2). The distance measure used is shape context matching [8], which achieves a nearest-neighbor classification error of 0.54%. As can be seen on the MNIST web site (`http://yann.lecun.com/exdb/mnist/`), shape context matching outperforms in accuracy a large number of other methods on the MNIST dataset. Using our own

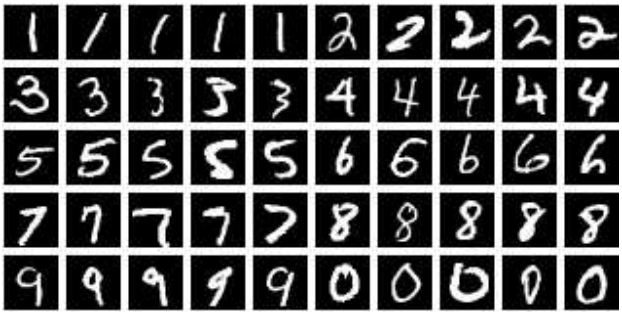


Fig. 2. Example images from the MNIST dataset of handwritten digits.

optimized C++ implementation, and running on an AMD Opteron 2.2GHz processor, we can compute on average 15 shape context distances per second. Using brute-force search for nearest neighbor classification averages approximately 60 minutes per query.

3) Online Handwritten Digit Dataset (UNIPEN): We use the isolated digits benchmark (category 1a) of the UNIPEN Train-R01/V07 database [60], which consists of $15,953$ digit examples. The digits have been randomly divided into $10,630$ database objects and $5,323$ query objects. Each digit is preprocessed exactly as described in [48]. Each extracted point is represented by three features: 2D normalized location $(\tilde{x}_i, \tilde{y}_i)$ and the tangent angle θ_i of the line segment between $(\tilde{x}_i, \tilde{y}_i)$ and $(\tilde{x}_{i-1}, \tilde{y}_{i-1})$. The distance measure D used for classification is dynamic time warping [2]. On an AMD Athlon 2.0GHz processor, we can compute on average 890 DTW distances per second. Therefore, nearest neighbor classification using brute-force search takes about 12 seconds per query, and yields an error rate of 1.90%.

4) Time-series dataset: We use the time series benchmark dataset described in [44]. To generate that dataset, various real datasets were used as seeds, and multiple copies of every real sequence were constructed by incorporating small variations in the original patterns as well as additions of random compression and decompression in time. As described in [54], we randomly split the data into $1,000$ queries and $31,818$ database objects. Distances in this dataset are measured using constrained dynamic time warping, with a warping length $\delta = 10\%$ of the total length of the shortest sequence [44]. On average, on an AMD Opteron 2.2GHz processor, we can compute 60 distances per second. Consequently, brute-force retrieval of the nearest neighbors of a query takes on average 530 seconds.

B. Evaluation Methodology and Parameter Choices

In our experiments, retrieval time is dominated by the number of exact distance computations that we perform. Other operations, such as the filter step of filter-and-refine retrieval, take negligible time (less than 0.1 seconds/query for all computations that are not part of measuring an exact distance). Consequently, we mainly report processing time using the number of exact distances we need to measure per query.

In evaluating k-nearest neighbor retrieval accuracy, we consider the retrieval result for a query to be correct if and only if *all* k-nearest neighbors of the query have been correctly identified. For example,if we measure accuracy on 50-nearest neighbor retrieval for a particular method and set of parameters, 95% retrieval accuracy means that for 95% of the queries we successfully identify all 50 nearest neighbors.

For filter-and-refine retrieval we must specify two parameters: d, which is the dimensionality of the embedding, and p, which specifies the number of exact distances to measure during the refine step. In all experiments, we use the d and p values that maximize efficiency given a specific setting for retrieval accuracy. As an example, if we want to measure the efficiency of FastMap for 50-nearest neighbor retrieval with 95% accuracy, we first find, for each d, the smallest value of p (denoted as p_d) that is needed to obtain the desired accuracy when a d-dimensional FastMap embedding is used during the filter step. If values d and p_d are specified, then the number of exact distance computations per query is also specified. This way, for each d, we compute the number of exact distance computations per query needed in order to attain the desired 95% accuracy. After computing that number for each d, we simply select the d that minimizes the number of exact distance computations, and we report results for that d (and its associated p_d).

With respect to the additional free parameters that are needed by the BoostMap algorithm (see Sec. VI-A), here we provide the default values, used in all experiments unless noted otherwise:

- $k_{\max} = 50$.
- $|\mathbb{C}| = |\mathbb{L}| = 5000$, except for the UNIPEN dataset, which is the smallest among our four datasets. For UNIPEN experiments, $|\mathbb{C}| = |\mathbb{L}| = 3500$, due to the relatively small size of the database.
- $\beta = 300,000$.
- $\gamma = 2000$.
- $\delta = 200$.
- $Z_{\max} = .9999$.

C. Methods Used for Comparison Purposes

We compare BoostMap to several alternative methods for nearest neighbor retrieval:

- **FastMap [9]**. We construct FastMap embeddings by running the FastMap algorithm on a subset of the database, containing 5000 objects (3500 objects for the UNIPEN dataset). The subset used for each dataset is the set $\mathbb{C}$ used for BoostMap.
- **Random reference objects (RRO)**. We construct a multi-dimensional Lipschitz embedding as a concatenation of multiple 1D embeddings, where each 1D embedding is obtained by choosing a random reference object P from the database.
- **Random line projections (RLP)**. We construct a multi-dimensional embedding as a concatenation of multiple 1D embeddings, each of which is defined by choosing two random database objects X_1, X_2 as pivot objects.
- **VP-trees [10]**. VP-trees rely on the triangle inequality to achieve efficient retrieval while always finding the true nearest neighbors. Since the distance measures in our

experiments are non-metric, using a method similar to [65] we modify the search algorithm so that it guarantees correct retrieval results if the triangle inequality is satisfied up to a constant ζ. Larger values of ζ lead to more accurate results and slower retrieval time. We should note that in some experiments we use values of ζ that are smaller than 1, in order to compare VP-trees with other methods at ultra-efficient settings. VP-trees were not designed to be used with such small ζ values (which lead to unreasonably aggressive pruning), and this is reflected in the corresponding results, where VP-trees are much less accurate than the other methods when $\zeta < 1$.

- **Brute-Force Search**. For nearest neighbor classification only, we provide results obtained using brute-force search on random subsets of the database, vs. the number of objects in those subsets.

D. Evaluation on Nearest Neighbor Retrieval

To measure retrieval accuracy, we first find the true k-nearest neighbors of each query in each dataset, using brute-force search. Then, we compare the results obtained by each method with the correct results obtained from brute-force search. In Figs. 3, 4, 5, and 6 we compare BoostMap to alternative methods on the task of k-nearest neighbor retrieval, on all four datasets. BoostMap clearly outperforms all other methods in three of the four datasets, namely the ASL handshape, MNIST and UNIPEN datasets. The performance difference between BoostMap and the other methods varies depending on the setting, i.e., the desired accuracy and the number of nearest neighbors to retrieve. In many settings BoostMap achieves retrieval times that are from 50% to over 300% faster than the times attained by the best alternative method. The time series dataset is the only dataset where BoostMap is not the best-performing method. In that dataset, using random reference objects provides results that are roughly as good as those of BoostMap for 90% and 95% retrieval accuracy, and results that are better than those of BoostMap for 99% retrieval accuracy.

The results on the time series dataset illustrate one limitation of the training algorithm: since we use AdaBoost as the underlying training method, the classifier that is constructed is not a globally optimal classifier. AdaBoost is essentially a greedy optimization method that finds locally optimal solutions. It is possible in some cases to obtain a better classifier using random choices. In [57] we describe improvements that allow the BoostMap method to outperform random reference objects on the time series dataset.

We have also performed experiments to evaluate the sensitivity of performance to the settings of the various parameters (namely $\beta, |\mathbb{C}|, k_{\max}, \gamma, \delta$) used in the training algorithm. The results are shown on Figs. 7 and 8. These experiments were performed on the ASL handshape dataset and the UNIPEN dataset. In each experiment we varied only one parameter, while the other parameters were set as specified in Section VIII-B, with the following exceptions:

- In the experiments where β does not vary, we set $\beta = 100,000$, to speed up the training algorithm.
- In all cases, γ is not allowed to be smaller than $2|\mathbb{C}|$, and δ is not allowed to be smaller than γ. Consequently, γ and δ are set to 2000 and 200 respectively when possible, and otherwise they are set to the highest legal value, given the settings of the other parameters.

As one would expect, decreasing the number β of training triples, while speeding up the offline training algorithm, leads to worse online performance. Similarly, online performance deteriorates by decreasing the size of $\mathbb{C}$, the set of candidate reference objects and pivot subjects. As a reminder, in all experiments, the size of $\mathbb{L}$ (the set of objects from which we form training triples) is set equal to $|\mathbb{C}|$. Decreasing γ leads to worse performance on the handshape dataset, but has very small impact on the UNIPEN dataset, at least within the ranges tested (50 to 2000). A similar result is obtained for parameter δ: decreasing the value of δ leads to somewhat worse performance on the handshape dataset, but makes very little difference on the UNIPEN dataset.

Fig. 8 displays the effects of varying parameter $k_{\max}$, which is used in choosing training triples. Parameter $k_{\max}$ represents the maximum number of nearest neighbors that we are interested in retrieving. As we see in Fig. 8, actual performance does not vary significantly as we shift the value of $k_{\max}$ within a fairly large range: between 50 and 600 for the handshape dataset and between 3 and 600 for the UNIPEN dataset. We also display results obtained by setting $k_{\max} = |\mathbb{U}|$, where $|\mathbb{U}|$ is the size of the database. With that setting, training triples are chosen entirely randomly from the set of all possible triples formed by objects of $\mathbb{L}$, as was done in the first implementation of the BoostMap algorithm [43]. The results show that, in general, using $k_{\max} \ll |\mathbb{U}|$ leads to better performance. The only exception is the result for $k_{\max} = 16$ for the handshape dataset, which produces performance worse than than using $k_{\max} = |\mathbb{U}|$.

E. Evaluation on Nearest Neighbor Classification

Here we evaluate the performance of the proposed methods on the task of efficient nearest neighbor classification. Evaluation is performed on all datasets except for the time series dataset, which does not contain class label information. In each dataset we compare BoostMap with the following methods: RRO, RLP, FastMap, and VP-trees. We should note that, in order for VP-trees to attain the efficiency of other methods (i.e., fewer than 1000 exact distance computations per query), we had to set parameter ζ to values smaller than 1, as discussed in Section VIII-C. Consequently, the results obtained using VP-trees in these experiments were much worse than those of the other methods.

1) Classification Experiments on the ASL Handshape Dataset: Classification on the ASL handshape dataset is challenging. Unlike typical handshape recognition settings, which assume a fixed 3D orientation for each handshape, in this dataset the orientation is arbitrary. Our goal is to identify for each query image which of the 20 handshapes it displays. Given the vast difference in appearance between different 3D orientations of the same shape, it is not surprising that k-nearest neighbor classification using brute-force search has a very high error rate of 67%. That rate is achieved using $k = 1$.

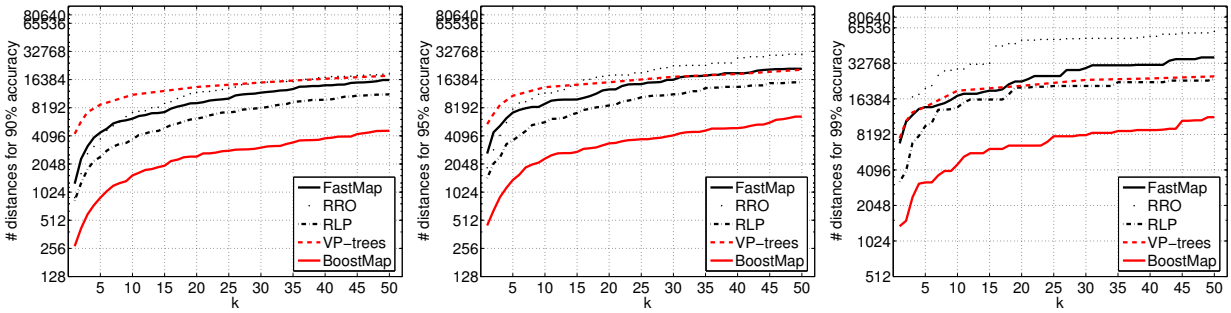


Fig. 3. Comparing performance on the ASL handshape dataset, using the chamfer distance as the exact distance measure. Each graph shows the number of exact distance computations needed by each method to achieve correct retrieval of all k nearest neighbors (k ranging from 1 to 50) for 90%, 95%, and 99% of the 710 query objects.

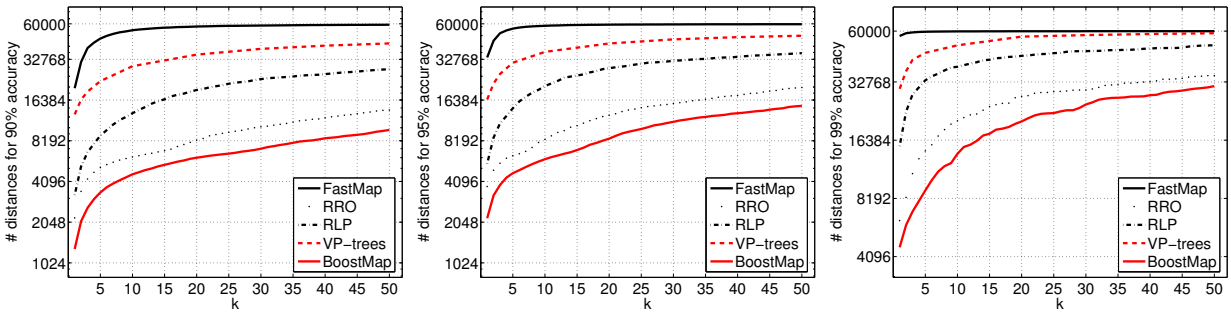


Fig. 4. Comparing performance on the MNIST dataset, using shape context matching as the exact distance measure. Each graph shows the number of exact distance computations needed by each method to achieve correct retrieval of all k nearest neighbors (k ranging from 1 to 50) for 90%, 95%, and 99% of the 10,000 query objects.

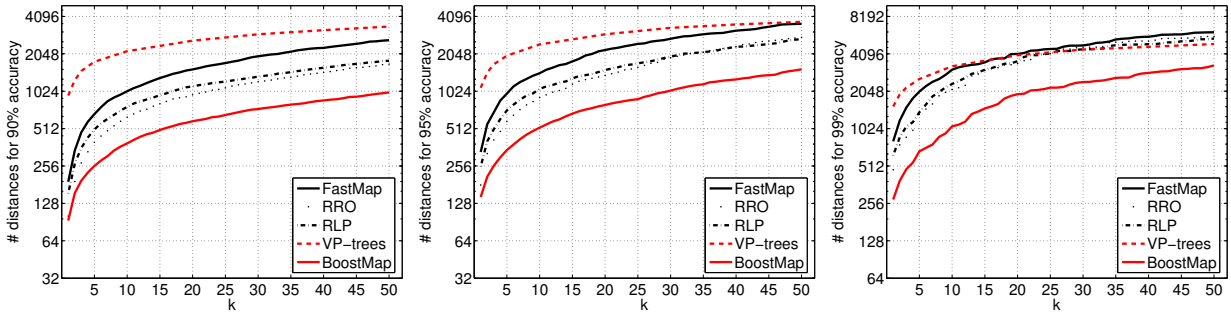


Fig. 5. Comparing performance on the UNIPEN dataset, using DTW as the exact distance measure. Each graph shows the number of exact distance computations needed by each method to achieve correct retrieval of all k nearest neighbors (k ranging from 1 to 50) for 90%, 95%, and 99% of the 5,323 query objects.

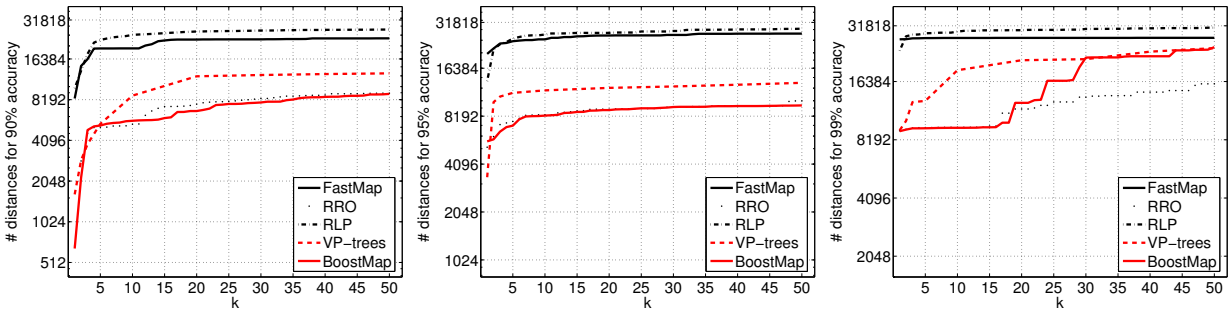


Fig. 6. Comparing performance on the time series database, using constrained DTW as the exact distance measure. Each graph shows the number of exact distance computations needed by each method to achieve correct retrieval of all k nearest neighbors (k ranging from 1 to 50) for 90%, 95%, and 99% of the 1,000 query objects.

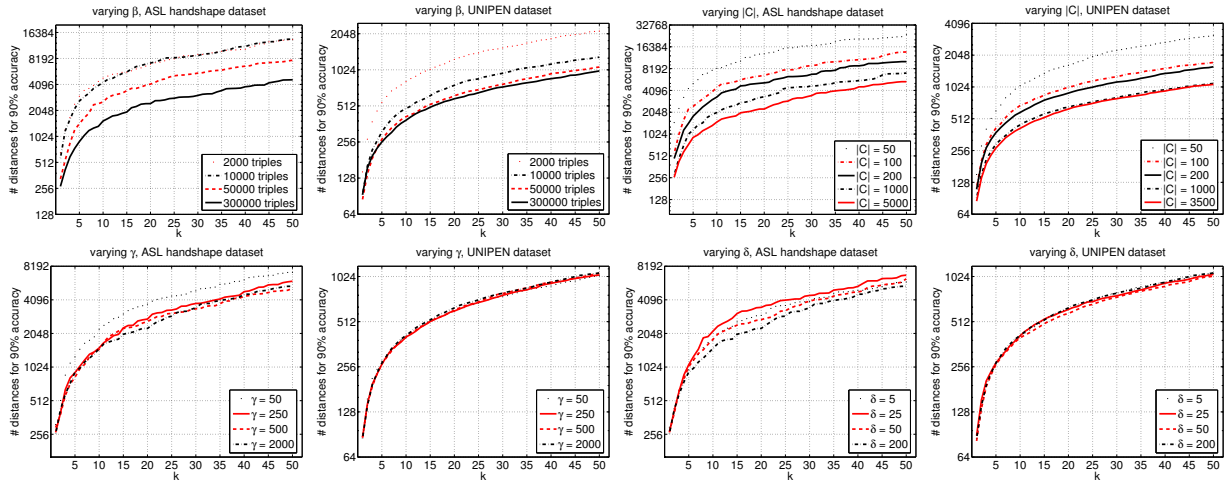


Fig. 7. Comparing performance of BoostMap embeddings on the ASL handshape dataset and the UNIPEN dataset, when varying parameters of the training algorithm. Each graph shows, for different parameter settings, the number of exact distance computations needed to achieve correct retrieval of all k nearest neighbors (k ranging from 1 to 50) for 90% of the query objects. The parameters that vary are: β (the number of training triples), $|\mathbb{C}|$ (the number of candidate reference and pivot objects), γ (the number of weak classifiers considered at each training round), and δ (the number of weak classifiers for which Equations 13 and 14 are evaluated at each training round). In each graph, the parameters that do not vary are set as described in the text.

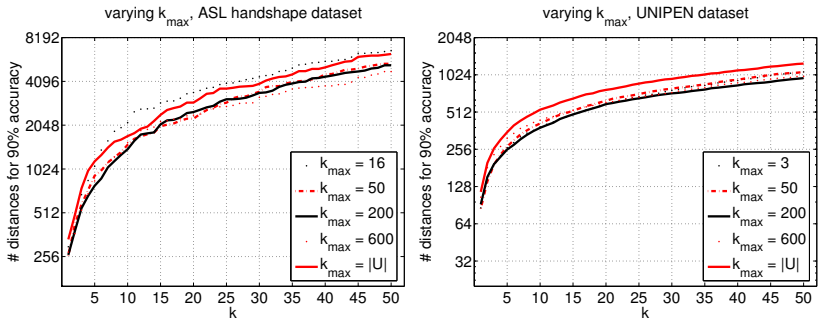


Fig. 8. Comparing performance on the ASL handshape dataset and the UNIPEN dataset, when varying parameter $k_{\max}$ of the training algorithm. Each graph shows, for five different values of $k_{\max}$, the number of exact distance computations needed to achieve correct retrieval of all k nearest neighbors (k ranging from 1 to 50) for 90% of the query objects. With $|\mathbb{U}|$ we denote the size of the database.

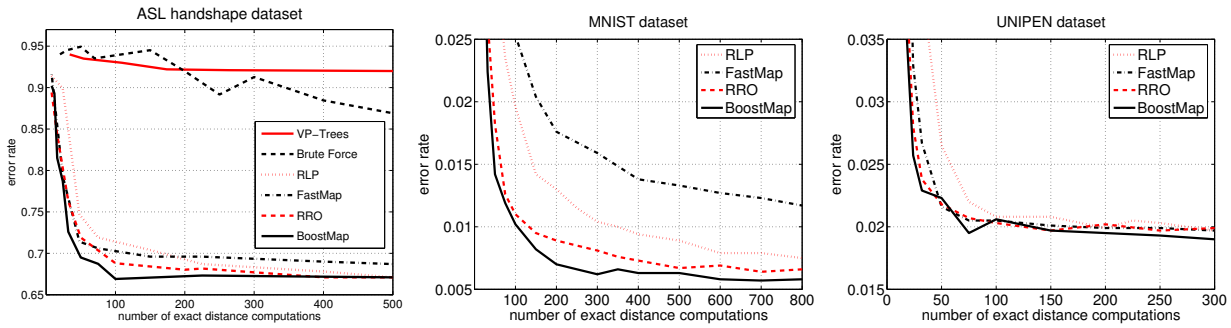


Fig. 9. Comparing classification accuracy vs. efficiency trade-offs achieved by the BoostMap, RRO, RLP, and FastMap methods on the ASL handshape dataset (left), the MNIST dataset (center), and the UNIPEN dataset (right). For the handshape dataset we also show results using VP-trees and using brute-force search on random subsets of the database, of different size. Results for VP-trees and brute-force search are discussed in the text for the other two datasets.

Fig. 9 displays the error rate attained using several different methods. Overall, BoostMap produces better results than the other methods. At the cost of 100 exact distance computations, BoostMap attains an error rate of 67%, which essentially equals the error rate of brute-force search. Therefore, using BoostMap we obtain a speed up factor of 800 over brute-force search, with no losses in classification accuracy. In terms of actual running time, using BoostMap we can classify about 3.5 queries per second, whereas it takes 112 seconds on average to classify a query using brute-force search.

For a cost of 100 exact distance computations, RRO achieves an error rate of 69%, whereas RLP and FastMap achieve error rates of 70%. RRO and RLP achieve an error rate of 67% at 400 distances and 500 distances respectively, whereas for FastMap the error rate is 69% at 500 distances. We also note that VP-trees and brute-force search perform significantly worse than the other methods when allowing no more than 500 distant computations per query on average.

Overall, it is fair to say that the accuracy we obtain on the ASL handshape dataset is not at the level where it

Method	Distances per query object	Speed-up factor	Seconds per query object	Error rate
brute force	60,000	1	3,696	0.54%
VP-trees [10]	21,152	2.84	1303	0.63%
CNN [51]	1,060	56.6	65.3	2.40%
VP-trees	800	75	49.3	24.8%
brute force	800	75	49.3	2.15%
BoostMap	800	75	49.3	0.58%
brute force	50	1200	3.1	14.6%
Zhang [47]	50	1200	3.1	2.55%
BoostMap	50	1200	3.1	1.50%
Cascade [66]	50	1200	3.1	0.83%

TABLE I

SPEEDS AND ERROR RATES ACHIEVED BY DIFFERENT METHODS ON THE MNIST DATASET. RESULTS USING BRUTE-FORCE SEARCH WERE OBTAINED BY CHOOSING A RANDOM SUBSET OF THE DATABASE, INCLUDING AS MANY OBJECTS AS THE NUMBER SHOWN IN THE "DISTANCES PER QUERY OBJECT" COLUMN.

Method	Distances per query object	Speed-up factor	Seconds per query object	Error rate
brute force	10,630	1.0	11.94	1.90%
VP-trees [10]	1,899	5.6	2.13	1.90%
brute force	150	70.9	0.17	10.7%
CSDTW	150	70.9	0.17	2.90%
RRO	150	70.9	0.17	1.97%
RLP	150	70.9	0.17	2.08%
BoostMap	150	70.9	0.17	1.97%
brute force	32	332	0.036	38.4%
RRO	32	332	0.036	2.38%
RLP	32	332	0.036	3.92%
BoostMap	32	332	0.036	2.29%

TABLE II

SPEEDS AND ERROR RATES ACHIEVED BY DIFFERENT METHODS ON THE UNIPEN DATASET. TO MAKE IT EASIER TO COMPARE DIFFERENT METHODS, FOR SOME METHODS WE SHOW MULTIPLE RESULTS, WHICH CORRESPOND TO DIFFERENT NUMBERS OF EXACT DISTANCE EVALUATIONS PER QUERY.

can be useful for actual applications. We should emphasize that this low accuracy is not caused by BoostMap or the other embedding methods, it is inherent in the choice of the underlying distance measure, i.e., the chamfer distance, which produces a high error rate even when using brute-force search. Reliable handshape classification of hand images displaying arbitrary 3D orientations is still an open problem.

2) Classification Experiments on the MNIST Dataset: As a reminder, exact k-nearest neighbor classification using shape context matching achieves an error rate of 0.54%, with classification time per object equal to about 60 minutes. That rate is achieved using $k = 8$. Fig. 9 displays the error rate attained using filter-and-refine retrieval with the BoostMap, RRO, RLP, and FastMap methods. BoostMap achieves an error rate of 0.58% at a cost of 800 exact distance computations. At the same cost of 800 exact distance computations, the RRO, RLP and FastMap methods obtain error rates of 0.66%, 0.75%, and 1.17% respectively.

In [47] a discriminative classifier is trained using shape context features, and achieves an error rate of 2.55% on the MNIST dataset. Overall, the cost of classifying a test object using the method in [47] is the cost of evaluating 50 exact distances. At the same cost of 50 exact distances per query, BoostMap achieves a classification error of 1.50%. We should point out that in [66] we describe a method for further improving the performance of BoostMap (0.83% error rate at the cost of 50 distances per query), by combining multiple BoostMap embeddings in a cascade structure.

Two additional methods that can be used for speeding up nearest neighbor classification are the well-known condensed nearest neighbor (CNN) method [51] and VP-trees [10]. Both methods achieve significantly worse tradeoffs between accuracy and efficiency compared to our method. CNN requires 1060 exact distances, and yields an error rate of 2.40%. With VP-trees the error rate is 0.63%, but an average of $21,152$ exact distances need to be measured per query. At 800 exact distances per query, the error rate is a very high 24.8%.

Table I summarizes the results of all the different methods.

3) Classification Experiments on the UNIPEN Dataset: Fig. 9 displays the error rate attained using filter-and-refine retrieval with the BoostMap, RRO, RLP, and FastMap methods on the UNIPEN dataset. Exact k-nearest neighbor classification using brute-force search achieves an error rate of 1.90% on this dataset. That rate is achieved using $k = 1$. BoostMap achieves an error rate of 1.95% at a cost of 75 exact distance computations, and an error rate of 1.90 at a cost of 300 distance computations. At a cost of 300 exact distance computations, the RRO, RLE, and FastMap methods obtains error rates of 1.99%, 1.97%, and 1.97 respectively. As in the other datasets, VP-trees do not work very well, yielding an error rate of 17%. Overall, BoostMap achieves a 35-fold speed-up over brute-force search, while achieving the same error rate, thus reducing classification time per query from 12 seconds to 0.34 seconds.

We should note that the CSDTW method [48], which has been explicitly designed for classifying time series and in particular for online handwritten character recognition, achieves an error rate of 2.90% on the UNIPEN dataset at a cost equivalent to 150 exact computations of DTW distances. For the same cost, BoostMap attains a significantly lower error rate of 1.97%. Even at a cost of 32 distance computations per query, BoostMap achieves an error rate of 2.26%, which is still lower than that of CSDTW. The advantage of CSDTW over our method is that it requires significantly less memory; in our method, we store in memory the embeddings of all database objects, and this requires about 1.4MB for a 32-dimensional embedding.

Table II provides a summary of classification results obtained using different methods, including VP-trees, different embedding methods, and CSDTW.

IX. DISCUSSION

The foundation of the BoostMap method has been the correspondence that we established in Sec. V between embeddings and classifiers: the association of every embedding with a corresponding classifier, and the proof that any linear combination of such embedding-based classifiers naturally corresponds to an embedding and a distance measure. By

treating embeddings as classifiers and embedding construction as a problem of learning how to estimate the proximity order of triples of objects, we obtain an algorithm that directly maximizes the amount of nearest neighbor structure preserved by the embedding. We emphasize that the optimization criterion that we use does not rely on any geometric assumptions and is equally principled for Euclidean, metric, and non-metric spaces. Furthermore, the local nature of our optimization criterion is in contrast with the global nature of measures such as stress and distortion, which focus on preservation of all pairwise distances and are not direct indicators of how well nearest neighbor structure is preserved.

In the filter-and-refine retrieval framework, embedding-based similarity rankings are used to select a small number of candidate nearest neighbors. Since our embedding construction method directly maximizes the accuracy of these similarity rankings, fewer candidates need to be evaluated during the refine step. As evidenced in our experiments, BoostMap leads to significantly better trade-offs between retrieval accuracy and efficiency compared to alternative methods, many times attaining $50\% - 300\%$ faster retrieval time than the best alternative method. Furthermore, in all datasets BoostMap led to significant computational savings over brute-force search, savings that in many settings were between one and two orders of magnitude.

Our training algorithm relies on sampling sets of candidate objects and training objects from the database. This scheme works as long as the sampled objects are representative of the distribution of database objects, but may fail in cases where the number of samples is only a small fraction of the database size, and where the database has a large amount of local structure. Handling such cases is an interesting topic for future investigation. Another topic for investigation is whether embedding quality can improve by using different variants of boosting, such as LogitBoost [55], or FloatBoost [67], in place of AdaBoost, during embedding construction.

X. Conclusion

The main topic of this paper has been embedding-based nearest neighbor retrieval and classification in spaces with computationally expensive distance measures. We have established a correspondence between embeddings and classifiers that allows us to reduce embedding construction to the problem of boosting many weak classifiers into a strong classifier. The proposed embedding construction algorithm is domain-independent, and directly maximizes the amount of nearest neighbor structure preserved by the embedding. Furthermore, embedding optimization does not rely on any Euclidean or metric properties. The resulting embeddings outperform alternative methods on several datasets, and lead to speed-ups of orders of magnitude over brute-force search.

Acknowledgments

This work was supported by NSF grants IIS-0308213, IIS-0329009, and CNS-0202067, and by ONR grant N00014-03-1-0108.

References

[1] T. M. Cover and J. A. Thomas, *Elements of information theory*. New York, NY, USA: Wiley-Interscience, 1991.

[2] J. B. Kruskall and M. Liberman, "The symmetric time warping algorithm: From continuous to discrete," in *Time Warps*. Addison-Wesley, 1983.

[3] E. Keogh, "Exact indexing of dynamic time warping," in *International Conference on Very Large Data Bases*, 2002, pp. 406–417.

[4] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics*, vol. 10, no. 8, pp. 707–710, 1966.

[5] G. Hjaltason and H. Samet, "Properties of embedding methods for similarity searching in metric spaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 5, pp. 530–549, 2003.

[6] R. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine Learning*, vol. 37, no. 3, pp. 297–336, 1999.

[7] H. Barrow, J. Tenenbaum, R. Bolles, and H. Wolf, "Parametric correspondence and chamfer matching: Two new techniques for image matching," in *International Joint Conference on Artificial Intelligence*, 1977, pp. 659–663.

[8] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 4, pp. 509–522, 2002.

[9] C. Faloutsos and K. I. Lin, "FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets," in *ACM International Conference on Management of Data (SIGMOD)*, 1995, pp. 163–174.

[10] P. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *ACM-SIAM Symposium on Discrete Algorithms*, 1993, pp. 311–321.

[11] C. Böhm, S. Berchtold, and D. A. Keim, "Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases," *ACM Computing Surveys*, vol. 33, no. 3, pp. 322–373, 2001.

[12] G. R. Hjaltason and H. Samet, "Index-driven similarity search in metric spaces," *ACM Transactions on Database Systems*, vol. 28, no. 4, pp. 517–580, 2003.

[13] D. A. White and R. Jain, "Similarity indexing: Algorithms and performance," in *Storage and Retrieval for Image and Video Databases (SPIE)*, 1996, pp. 62–73.

[14] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *International Conference on Very Large Data Bases*, 1998, pp. 194–205.

[15] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima, "The A-tree: An index structure for high-dimensional spaces using relative approximation," in *International Conference on Very Large Data Bases*, 2000, pp. 516–526.

[16] K. Chakrabarti and S. Mehrotra, "Local dimensionality reduction: A new approach to indexing high dimensional spaces," in *International Conference on Very Large Data Bases*, 2000, pp. 89–100.

[17] C. Li, E. Chang, H. Garcia-Molina, and G. Wiederhold, "Clustering for approximate similarity search in high-dimensional spaces," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 4, pp. 792–808, 2002.

[18] Ö. Egecioglu and H. Ferhatosmanoglu, "Dimensionality reduction and similarity distance computation by inner product approximations," in *International Conference on Information and Knowledge Management*, 2000, pp. 219–226.

[19] K. V. R. Kanth, D. Agrawal, and A. Singh, "Dimensionality reduction for similarity searching in dynamic databases," in *ACM International Conference on Management of Data (SIGMOD)*, 1998, pp. 166–176.

[20] R. Weber and K. Böhm, "Trading quality for time with nearest-neighbor search," in *International Conference on Extending Database Technology: Advances in Database Technology*, 2000, pp. 21–35.

[21] N. Koudas, B. C. Ooi, H. T. Shen, and A. K. H. Tung, "LDC: Enabling search by partial distance in a hyper-dimensional space," in *IEEE International Conference on Data Engineearing*, 2004, pp. 6–17.

[22] E. Tuncel, H. Ferhatosmanoglu, and K. Rose, "VQ-index: An index structure for similarity searching in multimedia databases," in *Proc. of ACM Multimedia*, 2002, pp. 543–552.

[23] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *International Conference on Very Large Databases*, 1999, pp. 518–529.

[24] A. Frome, D. Huber, R. Kolluri, T. Bulow, and J. Malik, "Recognizing objects in range data using regional point descriptors," in *European Conference on Computer Vision*, vol. 3, 2004, pp. 224–237.

[25] K. Grauman and T. J. Darrell, "Fast contour matching using approximate earth mover's distance," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2004, pp. I: 220–227.

[26] G. Shakhnarovich, P. Viola, and T. Darrell, "Fast pose estimation with parameter-sensitive hashing," in *IEEE International Conference on Computer Vision*, 2003, pp. 750–757.

[27] D. Huttenlocher, D. Klanderman, and A. Rucklige, "Comparing images using the Hausdorff distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 9, pp. 850–863, 1993.

[28] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–87, 1955.

[29] J. Uhlman, "Satisfying general proximity/similarity queries with metric trees," *Information Processing Letters*, vol. 40, no. 4, pp. 175–179, 1991.

[30] T. Bozkaya and Z. Özsoyoglu, "Indexing large metric spaces for similarity search queries," *ACM Transactions on Database Systems (TODS)*, vol. 24, no. 3, pp. 361–404, 1999.

[31] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," in *International Conference on Very Large Data Bases*, 1997, pp. 426–435.

[32] C. Traina, Jr., A. Traina, B. Seeger, and C. Faloutsos, "Slim-trees: High performance metric trees minimizing overlap between nodes," in *7th International Conference on Extending Database Technology (EDBT)*, 2000, pp. 51–65.

[33] P. Zezula, P. Savino, G. Amato, and F. Rabitti, "Approximate similarity retrieval with M-trees," *The VLDB Journal*, vol. 4, pp. 275–293, 1998.

[34] E. Vidal, "New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (AESA)," *Pattern Recognition Letters*, vol. 15, no. 1, pp. 1–7, 1994.

[35] L. Micó and E. Vidal, "A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements," *Pattern Recognition Letters*, vol. 15, no. 1, pp. 9–17, 1994.

[36] J. Bourgain, "On Lipschitz embeddings of finite metric spaces in Hilbert space," *Israel Journal of Mathematics*, vol. 52, pp. 46–52, 1985.

[37] G. Hristescu and M. Farach-Colton, "Cluster-preserving embedding of proteins," CS Department, Rutgers University, Tech. Rep. 99-50, 1999.

[38] S. Roweis and L. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, pp. 2323–2326, 2000.

[39] J. Tenenbaum, V. d. Silva, and J. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, pp. 2319–2323, 2000.

[40] X. Wang, J. T. L. Wang, K. I. Lin, D. Shasha, B. A. Shapiro, and K. Zhang, "An index structure for data mining and clustering," *Knowledge and Information Systems*, vol. 2, no. 2, pp. 161–184, 2000.

[41] F. Young and R. Hamer, *Multidimensional Scaling: History, Theory and Applications*. Hillsdale, New Jersey: Lawrence Erlbaum Associates, 1987.

[42] N. Linial, E. London, and Y. Rabinovich, "The geometry of graphs and some of its algorithmic applications," in *IEEE Symposium on Foundations of Computer Science*, 1994, pp. 577–591.

[43] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios, "BoostMap: A method for efficient approximate similarity rankings," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2004, pp. 268–275.

[44] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh, "Indexing multi-dimensional time-series with support for multiple distance measures," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003, pp. 216–225.

[45] B.-K. Yi, H. V. Jagadish, and C. Faloutsos, "Efficient retrieval of similar time sequences under time warping," in *IEEE International Conference on Data Engineering*, 1998, pp. 201–208.

[46] G. Mori, S. Belongie, and J. Malik, "Shape contexts enable efficient retrieval of similar shapes," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 2001, pp. 723–730.

[47] H. Zhang and J. Malik, "Learning a discriminative classifier using shape context distances," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 2003, pp. 242–247.

[48] C. Bahlmann and H. Burkhardt, "The writer independent online handwriting recognition system frog on hand and cluster generative statistical dynamic time warping," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 3, pp. 299–310, 2004.

[49] V. S. Devi and M. N. Murty, "An incremental prototype set building technique," *Pattern Recognition*, vol. 35, no. 2, pp. 505–513, 2002.

[50] G. W. Gates, "The reduced nearest neighbor rule," *IEEE Transactions on Information Theory*, vol. 18, no. 3, pp. 431–433, 1972.

[51] P. E. Hart, "The condensed nearest neighbor rule," *IEEE Transactions on Information Theory*, vol. 14, no. 3, pp. 515–516, 1968.

[52] T. Liu, K. Yang, and A. W. Moore, "The ioc algorithm: efficient many-class non-parametric classification for high-dimensional data." in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004, pp. 629–634.

[53] T. Liu, A. W. Moore, and A. G. Gray, "Efficient exact k-nn and nonparametric classification in high dimensions." in *Neural Information Processing Systems*, 2003.

[54] V. Athitsos, "Learning embeddings for indexing, retrieval, and classification, with applications to object and shape recognition in image databases," Ph.D. dissertation, Boston University, 2006.

[55] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting," *Annals of Statistics*, vol. 28, no. 2, pp. 337–374, 2000.

[56] J. Alon, V. Athitsos, and S. Sclaroff, "Online and offline character recognition using alignment to prototypes," in *International Conference on Document Analysis and Recognition*, 2005, pp. 839–843.

[57] V. Athitsos, M. Hadjieleftheriou, G. Kollios, and S. Sclaroff, "Query-sensitive embeddings," in *ACM International Conference on Management of Data (SIGMOD)*, 2005, pp. 706–717.

[58] V. Athitsos and S. Sclaroff, "Boosting nearest neighbor classifiers for multiclass recognition," in *IEEE Workshop on Learning in Computer Vision and Pattern Recognition*, 2005.

[59] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[60] I. Guyon, L. Schomaker, and R. Plamondon, "Unipen project of on-line data exchange and recognizer benchmarks," in *12th International Conference on Pattern Recognition*, 1994, pp. 29–33.

[61] *Poser 5 Reference Manual*, Curious Labs, Santa Cruz, CA, August 2002.

[62] Q. Yuan, S. Sclaroff, and V. Athitsos, "Automatic 2D hand tracking in video sequences." in *IEEE Workshop on Applications of Computer Vision*, 2005, pp. 250–256.

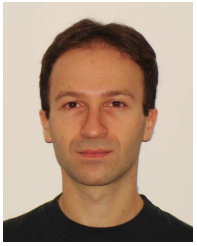
[63] V. Athitsos and S. Sclaroff, "Estimating hand pose from a cluttered image," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2003, pp. 432–439.

[64] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, 1986.

[65] S. C. Sahinalp, M. Tasan, J. Macker, and Z. M. Özsoyoglu, "Distance based indexing for string proximity search," in *IEEE International Conference on Data Engineering*, 2003, pp. 125–136.

[66] V. Athitsos, J. Alon, and S. Sclaroff, "Efficient nearest neighbor classification using a cascade of approximate similarity measures," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2005, pp. 486–493.

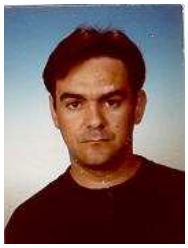
[67] S. Z. Li and Z. Q. Zhang, "Floatboost learning and statistical face detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1112–1123, 2004.



Vassilis Athitsos received the BS degree in mathematics from the University of Chicago in 1995, the MS degree in computer science from the University of Chicago in 1997, and the PhD degree in computer science from Boston University in 2006. In 2005-2006 he worked as a researcher at Siemens Corporate Research, developing methods for database-guided medical image analysis. Since October 2006 he is a postdoctoral research associate at the Computer Science department at Boston University. His research interests include computer vision, machine learning, and data mining. His recent work has focused on efficient similarity-based retrieval, human motion analysis and recognition, shape modeling and detection, and medical image analysis.

Jonathan Alon received the BSc degree in physics and computer science from Tel Aviv University in 1994, and both the MA degree and PhD degree in computer science from Boston University in 2001 and 2006. In September 2006 he joined the image processing group at NegevTech as an algorithms engineer, developing methods for visual inspection for the semiconductor industry. His research interests include computer vision and pattern recognition. His published work is in the areas of gesture recognition, efficient database retrieval, and character recognition.

Stan Sclaroff is an Associate Professor of Computer Science at Boston University, where he founded the Image and Video Computing research group. He received the PhD degree from MIT in 1995. In 1996, he received an ONR Young Investigator Award and an NSF Faculty Early Career Development Award. Dr. Sclaroff has coauthored numerous scholarly publications in the areas of tracking, video-based analysis of human motion and gesture, surveillance, deformable shape matching and recognition, as well as image/video database indexing, retrieval and data mining methods. He has served on the technical program committees of over 60 computer vision conferences and workshops. Stan Sclaroff has served as an Associate Editor for IEEE Transactions on Pattern Analysis, 2000-2004, and 2006-present. He is a Senior Member of the IEEE.



George Kollios received his Diploma in Electrical and Computer Engineering in 1995 from the National Technical University of Athens, Greece; and the M.Sc. and Ph.D. degrees in Computer Science from Polytechnic University, New York in 1998 and 2000 respectively. He is currently an Associate Professor in the Computer Science Department at Boston University in Boston, Massachusetts. His research interests include spatio-temporal indexing, data mining, multimedia indexing, and sensor and stream data management. He is the recipient of an NSF CAREER Award and his research is supported by NSF and other agencies. He also received a Best Paper Award in IEEE ICDE 2004 for his paper with title “Approximate Aggregation Techniques for Sensor Databases”. Prof. Kollios was the General Chair for the SSTD 2007, a Co-Organizer of the IDM NSF workshop for 2004, and he served as the local arrangements chair for SSDBM 2003 and IEEE ICDE 2004. He is an Associate Editor for the IEEE Transactions on Knowledge and Data Engineering. He has served in many technical program committees for top database and data mining conferences including VLDB, ACM SIGKDD, IEEE ICDE, ICML, and IEEE ICDM and has been a reviewer for top journals including IEEE TKDE, ACM TODS, VLDB Journal, and Information Systems. He is a member of ACM and IEEE Computer Society.