Boston University Computer Science Tech. Report No. 2005-009, April 06, 2005.
To appear in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2005.

Efficient Nearest Neighbor Classification Using a Cascade of Approximate Similarity Measures

Vassilis Athitsos, Jonathan Alon, and Stan Sclaroff
Computer Science Department
Boston University
111 Cummington Street
Boston, MA 02215
email: {athitsos, jalon, sclaroff}@cs.bu.edu

Abstract

This paper proposes a method for efficient nearest neighbor classification in non-Euclidean spaces with computationally expensive similarity/distance measures. Efficient approximations of such measures are obtained using the BoostMap algorithm, which produces embeddings into a real vector space. A modification to the BoostMap algorithm is proposed, which uses an optimization cost that is more appropriate when our goal is classification accuracy as opposed to nearest neighbor retrieval accuracy. Using the modified algorithm, multiple approximate nearest neighbor classifiers are obtained, that provide a wide range of trade-offs between accuracy and efficiency. The approximations are automatically combined to form a cascade classifier, which applies the slower and more accurate approximations only to the hardest cases. The proposed method is experimentally evaluated in the domain of handwritten digit recognition using shape context matching. Results on the MNIST database indicate that a speed-up of two to three orders of magnitude is gained over brute force search, with minimal losses in classification accuracy.

1. Introduction

Nearest neighbor classifiers are appealing because of their simplicity and their ability to model a wide range of complex, non-parametric distributions. However, finding the nearest neighbors of an object in a large database can take too long, especially in domains that employ computationally expensive distance measures.

This problem is illustrated in [3], where a computationally expensive image similarity measure, called shape context matching, is used for optical character recognition. A three-nearest-neighbor classifier using shape context matching and 20,000 training objects yields an error rate of only 0.63% on the MNIST database of handwritten digits. Unfortunately, recognizing a single test image takes over 20 minutes on a state-of-the-art PC, because it involves performing shape context matching between the test image and all 20,000 training objects.

The complexity of nearest neighbor classification using shape context matching is an instance of a more general problem, i.e., the problem of finding nearest neighbors in non-Euclidean spaces with computationally expensive distance measures. In this paper, we propose a general method that can be applied to arbitrary non-Euclidean similarity measures, including shape context matching, Dynamic Time Warping [5], or the chamfer distance [2].

The key idea underlying the proposed method is that nearest neighbor classification is not the same problem as nearest neighbor retrieval. In nearest neighbor retrieval we want to identify the nearest neighbors of test objects in the database. In nearest neighbor classification, the goal is to produce the right class label for every test object. Our method improves efficiency by sacrificing nearest neighbor *retrieval* accuracy in such a way that *classification* accuracy is minimally affected.

The first contribution of this paper is a modified version of the BoostMap embedding algorithm [1]. The modification consists of using an embedding optimization cost that is more appropriate for classification accuracy, as opposed to nearest neighbor retrieval accuracy. The second contribution is a method for building a cascade of approximate nearest neighbor classifiers. Using BoostMap embeddings and the filter-and-refine framework [12], we construct a sequence of approximations of the original nearest neighbor classifier. The first approximation in that sequence is relatively fast, but also has a relatively high error rate. Each successive approximation in the sequence is slower and more accurate than the previous one. These approximations are combined in a cascade structure, whereby easy cases are classified by earlier classifiers, and harder cases are passed on to the slower but more accurate classifiers.

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

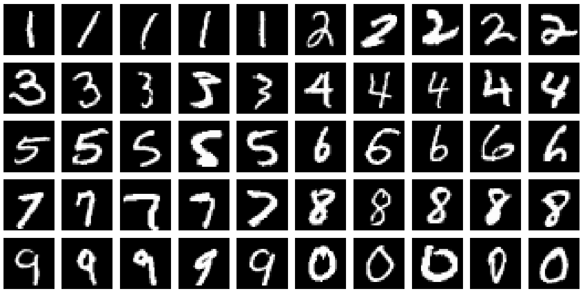

Figure 1: Some examples from the MNIST database of images of handwritten digits.

In our experiments with handwritten digit recognition using the MNIST database [15], we obtain speed-ups of two to three orders of magnitude compared to exact shape context matching, with minimal loss in classification accuracy (about 0.1% or less). Our method also yields significant speed-ups compared to using BoostMap, a state-of-the-art method for approximating a similarity measure.

2. Related Work

Typically, the problem of efficient nearest neighbor classification is addressed by using indexing methods, that can find the nearest neighbor or neighbors of the test object (also called query object) without having to exhaustively compare the test object with every single training object. Several hashing and tree structures [4, 11, 23, 24] have been proposed for indexing. However, the performance of such methods degrades in high dimensions. Locality sensitive hashing (LSH) [14] is a method for *approximate* nearest neighbor retrieval that scales better with the number of dimensions, but LSH is only applicable for specific metrics, and cannot be applied to arbitrary distance measures.

In domains where the distance measure is computationally expensive, embedding methods can been used for speeding up nearest neighbor retrieval. Such methods include Lipschitz embeddings [12], FastMap [7], MetricMap [22], SparseMap [13], and BoostMap [1]. Embedding methods define functions that map every object into a vector of real numbers. Given a query object, instead of computing its exact distance to each training object, we simply map the query to a vector and compare that vector with the vectors corresponding to the training objects. Typically vectors are compared using a Euclidean (L_2) or L_p distance metric, that can be several orders of magnitude faster to evaluate compared to the exact distance measure in the original space.

The goal of existing embedding methods is efficient and accurate nearest neighbor retrieval. The main difference of our method is that it aims at efficient and accurate *classification*. Our method builds on top of existing embedding methods, and shows how to improve classification efficiency by constructing a cascade of approximate nearest neighbor classifiers.

The efficiency of nearest neighbor classifiers can also be improved using condensing methods [6, 8, 10]. Those methods try to identify and remove training objects whose removal does not negatively affect classification accuracy. In the experiments we compare our method to the condensing method described in [10].

Shape context matching [3] is a computationally expensive non-Euclidean similarity measure. It is based on the shape context feature, a rich local feature that describes the distribution of points around a given location. Shape context has been successfully applied in various recognition domains, like trademark recognition [3], hand shape detection and classification [19] and body pose estimation [9, 18].

Methods that improve the efficiency of shape context matching can be useful in a wide range of domains. In [17], efficient retrieval is attained by pruning based on comparisons of a small subset of shape context features, and also using vector quantization on the space of those features. In [9] shape context features are matched based on the Earth Mover's Distance (EMD). The EMD is efficiently approximated using an embedding, and then Locality Sensitive Hashing is applied on top of the embedding. In [25] a discriminative classifier is learned based on correspondences of shape context features between the test object and a small number of prototypes per class. In experiments on the MNIST database, [25] reports an error rate of 2.55% by comparing the test object to only 50 prototypes. In comparison, the original brute force method in [3] is much slower (20,000 distance computations per test object), but achieves a significantly lower error rate(0.63%).

Our algorithm constructs a cascade of nearest neighbor classifiers. Cascades of classifiers have been very popular in recent years [16, 19, 21]. However, typically cascades are applied to a binary "detection" problem, where the goal is to determine whether a given image window contains an instance of some class. Our method produces a cascade of classifiers for a multi-class problem, using approximations of the original nearest neighbor classifier as building blocks.

3. Tuning BoostMap for Classification Accuracy

BoostMap [1] is a method for constructing embeddings that are optimized for preserving the similarity structure of the original space. In this section we introduce a modified version of the algorithm, that is more appropriate when our goal is nearest neighbor classification accuracy. We will first give a very basic description of BoostMap (the reader is referred to [1] for the details). Then we will motivate and describe our modification to the algorithm.

3.1. Embeddings and BoostMap

We use X to denote a set of objects, and $D_X(x_1, x_2)$ to denote a distance measure between objects $x_1, x_2 \in X$. In our example application, X is a set of images of handwritten digits (Fig. 1), and D_X is shape context matching as defined in [3]. However, any X and D_X can be plugged in the formulations described in this paper.

An embedding $F : X \rightarrow \mathbb{R}^d$ is a function that maps any object $x \in X$ into a d-dimensional vector $F(x) \in \mathbb{R}^d$. Distances in $\mathbb{R}^d$ are measured using some L_p metric, most commonly the Euclidean (L_2) or Manhattan (L_1) metric. It is assumed that measuring a single L_p distance between two vectors is significantly faster than measuring a single distance D_X between two objects of X. This assumption is obeyed in our example application: on our PC, over a million L_1 distances between high-dimensional vectors in $\mathbb{R}^{100}$ can be measured in one second, whereas only 15 shape context distances can be evaluated per second.

In BoostMap [1], the building blocks used to construct an embedding F are simple, one-dimensional (1D) embeddings. Any object $r \in X$ can be used to define a one-dimensional embedding $F^r : X \rightarrow \mathbb{R}$, as follows:

$$F^r(x) = D_X(x, r) . \quad (1)$$

In plain terms, F^r maps each object of X to a single real number. Furthermore, if x_1 is very similar to x_2 under D_X, then in many domains we can expect (and in metric spaces we can guarantee) that the distances $D_X(x_1, r)$ and $D_X(x_2, r)$ will be similar, meaning that F^r will map x_1 and x_2 to nearby points on the real line [1].

Each 1D embedding F^r acts as a classifier for the following binary classification problem: given three objects $q, a, b \in X$, is q closer to a or to b? F^r provides an answer by simply checking if $F^r(q)$ is closer to $F^r(a)$ or to $F^r(b)$. Since F^r is a simple, 1D embedding, it is expected to act as a weak classifier [1, 20], i.e., it will probably have a high error rate, but still it is expected to be more accurate than a random guess, which would have an error rate of 50%.

The BoostMap algorithm uses a training set S of triples (q, a, b), picked randomly from the available training objects. The algorithm constructs, using AdaBoost [20], an embedding $F : X \rightarrow \mathbb{R}^d$ optimized for classification accuracy on triples of objects. In BoostMap, distances in $\mathbb{R}^d$ are measured using a weighted L_1 metric.

3.2. The Modified BoostMap Algorithm

The original BoostMap algorithm aims at preserving similarity rankings, and minimizes the fraction of triples (q, a, b) where q is closer to a than to b, but $F(q)$ is closer to $F(b)$ than to $F(a)$. If our end goal is classification accuracy, then for some triples of objects the minimization criterion of BoostMap is either irrelevant or problematic. For example, suppose that q is an image of the digit “2”, and a and b are images, respectively, of digits “0” and “1”. In that case, for the purposes of classification, it is irrelevant whether the embedding maps q closer to a or to b.

A more interesting example is the following: suppose q and b are both images of the digit “2”, and a is an image of the digit “0”. Furthermore, suppose that, a is the nearest neighbor of q among training objects according to shape context matching. In that case, we would actually prefer an embedding that maps q closer to b than to a, although the original BoostMap algorithm would penalize for that.

To address these problems, we propose the following guidelines for selecting training triples and measuring the optimization cost:

- Useful training triples are triples (q, a, b) such that a is one of the nearest neighbors of q among objects *of the same class* as q, and b is one of the nearest neighbors of q among objects of some other class.
- The optimization cost that should be minimized is the number of training triples (q, a, b) such that the output embedding F maps q closer to b than to a, regardless of whether q is actually closer to b than to a in terms of distance measure D_X. Since a is of the same class as q, we want q to be mapped closer to a than to b.

In our example application, we apply these guidelines as follows: we choose a subset X' of 5,000 training objects, and for every object $q \in X'$, we form 20 triples (q, a, b) (for a total of 100,000 training triples), such that a is the nearest neighbor of q among all objects in X' of the same class as q, and b is one of the five nearest neighbors of q among objects in X' that belong to some class different than that of q. In other words, in order to choose b given q, we perform the following three steps:

1. Choose (randomly) a class C different than that of q.
2. Find the five nearest neighbors of q among objects of class C in X'.
3. Set b randomly to one of those five objects.

Instead of finding the five nearest neighbors, we experimented with different values, ranging from two to 20, without observing much difference in the results.

Overall, our implementation of the BoostMap algorithm is exactly as described in [1], except for the way we choose training triples, the optimization cost that we use, and the fact that we use only 1D embeddings of the type defined in Eq. 1.

4. Cascading Approximate Classifiers

Suppose that we have used some embedding method, like BoostMap, and we have obtained an embedding F. In this section we will use the filter-and-refine retrieval framework [12] to define, based on F, multiple approximations of the nearest neighbor classifier corresponding to the distance measure D_X. These approximations cover a wide range of tradeoffs between accuracy and efficiency. We will also show how to automatically combine such approximations into a cascade classifier, that can approximate the accuracy of the slowest approximation, while being much more efficient than the slowest approximation.

4.1. Filter-and-refine Classification

We can use embedding F for nearest neighbor classification in a filter-and-refine framework [12]. In particular, given parameters p and k, and given a query object q, we perform the following steps:

- Embedding step: compute the embedding $F(q)$ of the query. If F is a d-dimensional embedding obtained with BoostMap, then $F(q) = (F^{r_1}(q), \dots, F^{r_d}(q))$, where each F^{r_i} is defined using some object $r_i \in X$, according to Eq. 1. Computing $F(q)$ consists of computing d distances D_X between q and objects r_i.

- Filter step: compute the L_1 distances between $F(q)$ and the embeddings of all training objects. Rank all database objects in order of decreasing similarity to the query, based on these distances. Mark the p highest-ranked objects. Typically this step is very fast, because it only involves measuring L_1 distances.

- Refine step: rerank the p objects marked by the filter step, by evaluating the D_X distances between q and each of those p objects.

Given the final ranking produced by the filter and refine steps, we can then classify q based on majority voting among its k nearest neighbors. Note that we can have $p = 0$. In that case no refine step is performed, we simply use the ranking produced by the filter step.

The performance of filter-and-refine classification is measured in terms of accuracy and efficiency. In many domains, the running time is dominated by the number of exact distances measured at the embedding step and the refine step. In our example application, the filter step, i.e., comparing the query to the entire database using the L_1 metric, takes less time than evaluating the shape context matching distance D_X between a single pair of objects. Overall, increasing embedding dimensionality and the parameter p for the filter step tends to improve accuracy, but at the cost of computing more D_X distances per query object.

4.2. Constructing a Cascade

Let F be the embedding obtained using BoostMap (or some other embedding method), and let d' be the dimensionality of F. For any $d \in \{1, \dots, d'\}$, we define embedding F_d to be the embedding consisting of the first d dimensions of F. Given positive integers $d \leq d'$ and p we define filter-and-refine process $F_{d,p}$ to be the the filter-and-refine process that uses F_d as the embedding, and p as the parameter for the filter step. Naturally, given a query object q, as d and p increase, the approximate similarity ranking obtained using process $F_{d,p}$ will get closer to the correct ranking. For example, if p is equal to the number of training objects, then process $F_{d,p}$ becomes equivalent to brute force search: at the refine step we simply compare the query object with every training object. On the other hand, small d and p allow the filter-and-refine process $F_{d,p}$ to give results very fast.

With appropriate choices of d_i and p_i we can construct a sequence $\mathbb{P} = (P_1, ..., P_s)$ of s filter-and-refine processes $P_i = F_{d_i,p_i}$, such that each successive process P_i is less efficient and more accurate than the preceding process P_{i-1}. Table 1, in the experiments, provides an example of such a sequence $\mathbb{P}$. We want to use these processes to construct a cascade classifier, that can approach the accuracy of the slowest and most accurate process, while being significantly more efficient than the slowest process.

In order to construct such a cascade, we need to answer the following question: how can we identify, for each process P_i, the test objects for which P_i provides sufficient information for classification? In other words, given the similarity ranking that P_i produces for test object q, how can we tell whether we can reliably classify q using that ranking, or whether we need to pass on q to P_{i+1}, i.e., the next filter-and-refine process?

We will answer this question by defining a quantity $K(q, P_i)$ which will be a measure of confidence in the classification result for object q obtained using process P_i. We define $K(q, P_i)$ as follows: $K(q, P_i)$ is the highest integer k such that all the k nearest neighbors of q retrieved using process P_i belong to a single class. For example, suppose that, according to P_i, the 50 nearest neighbors of test object q belong to class "1", and the 51st neighbor belongs to class "2". Then, $K(q, P_i) = 50$.

We use quantity $K(q, P_i)$ to define a criterion for when P_i should be used to classify q, as follows: Given a threshold t_i, if $K(q, P_i) \geq t_i$ then we assign to q the class label of the t_i nearest neighbors found using P_i. Otherwise, we pass on q to P_{i+1}, i.e., the next filter-and-refine process in the sequence $\mathbb{P}$.

The intuition behind this criterion is that if, for some test object, process P_i reports that that test object is surrounded by a large number of training objects of a single class, then we can confidently assign that class to the test object, without needing to spend any additional computation. Note that we are not concerned about actually finding the true nearest neighbor of the test object: we stop as soon as we have sufficient information about the class label of the test object.

```
input  : $\mathbb{P}$: sequence of filter-and-refine processes
         $P_1, \cdots, P_s$.
         $s$: number of filter-and-refine processes in sequence $\mathbb{P}$.
         $X_{\text{train}}$: set of training objects.
         $X_{\text{validation}}$: set of validation objects.
         $B$: set of validation objects that are misclassified by process $P_s$.
         $e$: a parameter specifying how many objects
         should be misclassified, at most, at each step
         in the cascade.
output : $t_1, \cdots, t_{s-1}$: The thresholds $t_i$ to be used with
         each process $P_i$.
$X_{\text{validation}} = X_{\text{validation}} - B$
for $i = 1 : (s-1)$ do
    for $q \in X_{\text{validation}}$ do
        $K_q = K(q, P_i)$
        $N_q$ = nearest neighbor of $q$ in $X_{\text{train}}$ according to $P_i$
        $Y_q$ = class label of $q$
        $C_q$ = class label of $N_q$
    end
    $t_i = \text{size}(X_{\text{train}}) + 1$
    for $t = 1 : \text{size}(X_{\text{train}})$ do
        $X_1 = \{q \in X_{\text{validation}} | K_q \geq t\}$
        $X_2 = \{q \in X_1 | Y_q \neq C_q\}$
        if $\text{size}(X_2) \leq e$ then
            $t_i = t$
            break;
        end
    end
    $X_{\text{validation}} = \{q \in X_{\text{validation}} | K_q < t_i\}$
end
```

Algorithm 1: The algorithm for choosing thresholds for a cascade of filter-and-refine processes.

Naturally, in order to achieve good results, given a sequence $\mathbb{P}$ we need to choose good values for thresholds t_i. We choose those values using a validation set, sampled from the set of training objects. Let $e \geq 0$ be an integer parameter that specifies how many objects from the validation set we are allowed to misclassify by each P_i. Then, for the first process P_1 we can simply set t_1 to the smallest threshold t satisfying the following property: if we use process P_1 to classify (using 1-nearest neighbor accuracy) all validation objects q satisfying the criterion $K(q, P_1) \geq t$, we misclassify at most e validation objects. For example, if $e = 2$, we can try all thresholds until we find the smallest threshold t such that, if we find all validation objects with $K(q, P_1) \geq t$ and we label those objects with the label of the t nearest neighbors retrieved using P_1, we misclassify no more than two objects.

Process	Dimensions (d)	p	Threshold
P_1	10	0	50
P_2	20	0	56
P_3	40	0	51
P_4	60	0	51
P_5	80	0	50
P_6	100	0	41
P_7	100	20	41
P_8	100	40	41
P_9	100	60	17
P_{10}	100	80	20
P_{11}	100	100	20
P_{12}	100	150	24
P_{13}	100	200	11
P_{14}	100	250	4
P_{15}	100	300	5
P_{16}	100	700	NA

Table 1: The sequence $\mathbb{P} = P_1, \ldots, P_{16}$ of filter-and-refine processes that was passed as input to Algorithm 1. We used this sequence both with BoostMap embeddings and with BoostMap-C embeddings. The dimensions column specifies the dimensionality of the embedding, and p is the parameter specifying the number of distances to measure in the refine step. We also show the threshold chosen by the cascade learning algorithm, using embeddings from BoostMap-C and setting $e = 0$. Naturally, no threshold is needed for the final step in the cascade.

After we have determined the right threshold for process P_1, we proceed to select an appropriate threshold for P_2, using the same parameter e, and using only the validation objects q that P_1 does not classify (i.e., for which $K(q, P_1) < t_1$). Proceeding this way recursively we can choose all thresholds t_i. Naturally, the last process P_s in the sequence does not need a threshold, because P_s is the last step in the cascade, and therefore it needs to classify all objects that are passed on to it.

A slight problem with the above procedure for determining thresholds is that there may be some validation objects q that even the final process P_s will misclassify, and for which $K(q, P_i)$ is very high for all processes P_i. In our experiments, such objects were identified in practice. These objects are essentially outliers that look very similar to a large number of objects from another class. These objects are likely to influence the threshold choice t_i for every P_i, so that t_i is large enough to avoid misclassifying those objects, even though they will end up being misclassified anyway at the final step P_s. We use a simple method to avoid this problem: before choosing thresholds, we identify all validation objects that P_s misclassifies. We then remove those objects from the validation set, and proceed with threshold selection.

The exact algorithm for picking thresholds for each step in a cascade is described in Algorithm 1.

5. Experiments

We applied our method to define a cascade of approximate similarity measures, using shape context matching as the original distance measure D_X. We evaluated our method on the publicly available MNIST database [15] of handwritten digits. MNIST consists of 60,000 images that are used as training objects, and 10,000 images that are used as test objects. Fig. 1 shows some of those images. The exact shape context matching error rates, obtained by comparing the test object to all training objects, as described in [3], were 0.63% for 20,000 training objects and 0.54% for 60,000 training objects, with classification time per object equal to about 22 minutes and 66 minutes respectively.

We used the original BoostMap algorithm [1] and BoostMap-C, i.e., the modified algorithm proposed in this paper, to learn a 100-dimensional embedding F. Filter-and-refine classification with $p = 700$ gave error rates of 0.74% for BoostMap, and 0.72% for BoostMap-C, for 20,000 training objects. For 60,000 training objects, both methods had an error rate of 0.58%. These error rates were obtained at the cost of computing 800 exact distances per test object, i.e., spending about 53 seconds to classify each test object. In Fig. 2 we plot error rate vs. number of exact distance evaluations per test object, using 60,000 training objects. We see that BoostMap-C attains its peak accuracy at around 300 exact distance computations per object, but it takes BoostMap about 800 exact distance computations per object to reach the same accuracy.

We applied Algorithm 1 to construct a cascade of classifiers, using different values of e, ranging from 0 to 4. The training and validation sets passed to the algorithm were disjoint sets of 20,000 and 10,000 objects respectively, randomly picked from the original MNIST training set of 60,000 objects. The sequence $\mathbb{P}$ of filter-and-refine processes that was passed as an input to Algorithm 1 is shown in Table 1. We constructed the sequence by hand, i.e., we manually picked d and p for each process. The thresholds were learned automatically by the algorithm. We did not experiment with any other sequence of processes, so we have no reason to believe that the sequence we manually constructed is particularly good or bad with respect to other possible sequences. Our guideline in constructing the sequence was simply to provide an adequate number of steps, ranging from really fast and inaccurate to really slow and accurate, with the constraint that each cascade step could reuse the work done at the previous steps.

For 20,000 objects, and passing $e = 0$ to Algorithm 1, using BoostMap we obtained an error rate of 0.75%, at an average cost of measuring about 149.0 distances per test object, which translates to average classification time of 9.9 seconds per test object. Using the modified algorithm BoostMap-C, the resulting cascade yielded an error rate of 0.74%, at an average cost of measuring 92.5 distances per test object, which translates to average classification time of 6.2 seconds per test object. Overall, using a cascade speeds up average recognition time significantly, compared to using BoostMap or BoostMap-C without a cascade, and by over two orders of magnitude over brute force search, at the cost of about 0.1% increase in the error rate. We also see that the BoostMap-C cascade gives faster average classification time for essentially the same accuracy obtained with the BoostMap cascade.

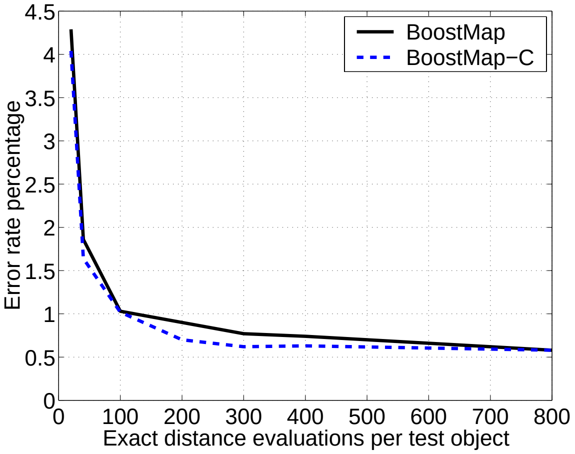

Figure 2: Error rates attained using BoostMap and BoostMap-C, without a cascade, vs. number of exact D_X evaluations per test object. We note that, for each number of D_X evaluations per object, BoostMap-C gives either as good or somewhat better results, compared to BoostMap. 60,000 training objects were used.

Finally, we evaluated the same cascades using as a training set all 60,000 training objects of the MNIST database. For each cascade, the thresholds were set to the same values that were employed in the experiments with 20,000 training objects. The results, for parameter e ranging from 0 to 4, are shown in Fig. 3. For $e = 0$ and using BoostMap we got an error rate of 0.66% at an average cost of 123 exact distance computations per test object. For $e = 0$ and using BoostMap-C we got an error rate of 0.61% at an average cost of only 77.3 distance computations per test object. This is a speed-up of almost three orders of magnitude over brute-force search, which achieves an error rate of 0.54%. As seen in Fig. 3, cascades using BoostMap-C achieve better tradeoffs of accuracy versus efficiency compared to cascades using BoostMap.

Note that increasing the training size from 20,000 to 60,000 objects improved both the accuracy and the efficiency of the cascade classifier. This result may seem surprising at first glance, and is in stark contrast to traditional nearest-neighbor methods, where recognition time increases as training set size increases. By taking a closer look at the results, we found that, as training size increased, and

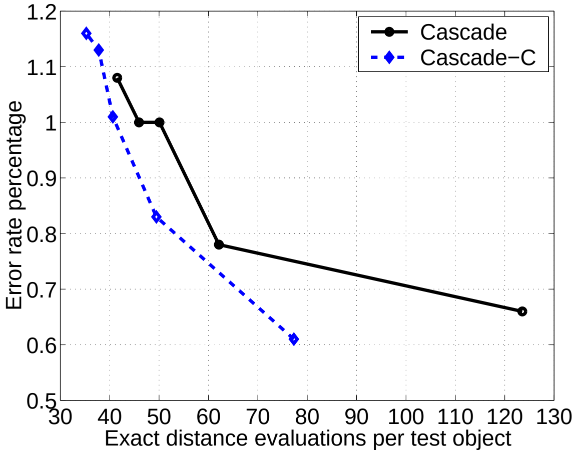

Figure 3: Error rates attained by cascade classifiers vs. average number of exact D_X evaluations per test object. Cascade and Cascade-C correspond to the five cascades that were learned, using embeddings constructed with BoostMap and BoostMap-C respectively. For each of these two embedding methods, we obtained five cascades by running Algorithm 1 with parameter e set respectively to $e = 0, 1, 2, 3, 4$. 60,000 training objects were used.

the processes P_i and thresholds t_i remained fixed, the quantity $K(q, P_i)$ tended to increase, meaning that more objects were classified at earlier steps in the cascade.

In [25] a discriminative classifier is trained using shape context features, and achieves an error rate of 2.55% on the MNIST dataset while measuring only distances D_X between the test object and 50 prototypes. That method is not a nearest-neighbor method, so after learning the classifier only the 50 prototypes are needed, and the rest of the training set is discarded. Overall, the cost of classifying a test object using the method in [25] is the cost of evaluating 50 D_X distances. Using BoostMap-C and the full training set of 60,000 objects, with parameter $e = 1$ we obtain a cascade that yields an error rate of 0.83%, while measuring on average 49.5 evaluations of D_X per test object. Also, as Fig. 3 shows, several cascades obtained using BoostMap and BoostMap-C achieve error rates under 1.2% at an average cost ranging from 35 to 50 D_X evaluations per test object.

We also compared our method to two well known methods for speeding up nearest neighbor classification: the condensed nearest neighbor (CNN) method [10] and vp-trees [24]. For the vp-trees the pivot object for each node was selected randomly. We evaluated these methods using the smaller training set of 20,000 objects. Both methods achieved significantly worse tradeoffs between accuracy and efficiency compared to our method. CNN selected 1060 out of the 20,000 training objects, speeding up classification time by approximately a factor of 20. However, the error rate using CNN increased from 0.63% to 2.40%. With vp-trees the error rate was 0.66%, but the attained speed up was only a factor of 2.3; an average of 8594 exact distances needed to be measured per test object.

Method	Distances per query object	Speed-up factor	Seconds per query object	Error rate
brute force	20,000	1	1232	0.63%
vp-trees [24]	8594	2.3	572	0.66%
CNN [10]	1060	18.9	70.6	2.40%
Zhang [25]	50	400	3.3	2.55%
BoostMap	800	25	53.3	0.74%
BoostMap-C	800	25	53.3	0.72%
Cascade	149	134	9.9	0.75%
Cascade-C	92.5	216	6.2	0.74%

Table 2: Speeds and error rates achieved by different methods on the MNIST dataset, using 10,000 test objects and 20,000 training objects. We also show the number of exact shape context distance evaluations per query object for each method.

Table 2 summarizes the results of all the different methods on the smaller training set of 20,000 objects. As we can see from those results, vp-trees, BoostMap and the cascade methods are the only methods that achieve accuracy comparable to brute force search. The speed-up obtained using vp-trees is pretty minor compared to using BoostMap or using a cascade. The cascade methods, and especially Cascade-C, achieve by far the best trade-offs between accuracy and efficiency.

6. Discussion

We have presented two improvements to the state of the art with respect to efficient nearest neighbor classification under computationally expensive similarity measures. The first contribution is BoostMap-C, a modified version of the BoostMap embedding algorithm, that constructs embeddings using an optimization cost that is more relevant to nearest neighbor classification accuracy. The second contribution of this paper is a method for constructing a cascade of approximations of the nearest neighbor classifier corresponding to the original similarity measure. This cascade method can be applied on top of BoostMap or any other appropriate embedding method that may well work in a particular domain, like FastMap [7] or SparseMap [13].

The method proposed in this paper is domain-independent, i.e., its formulation is appropriate for arbitrary spaces with computationally expensive distance measures. This is in contrast to methods like LSH [14] that can only be applied to specific distance measures. At the same time, no theoretical guarantees can be provided that the proposed method, or any other alternative domain-independent method (including the methods proposed in [6, 7, 8, 10, 13, 22, 24]) will actually give useful results in an arbitrary domain, in terms of accuracy and efficiency on previously unseen test objects. Some knowledge about the domain, like the presence of Euclidean or metric properties, is necessary in order to provide such guarantees.

A limitation of the cascade method, at least as formulated in this paper, is that it cannot be applied if there is only one training object per class. Overall, the more training examples per class we have, the more we expect the cascade to improve classification efficiency. A direction for future work is to formulate alternative cascade methods that overcome this limitation.

In our experiments with the MNIST database of handwritten digits and shape context matching as the underlying distance measure, our method yielded a speed-up of about two to three orders of magnitude with respect to brute-force search, and yielded significant speed-ups over using BoostMap without a cascade, with negligible loss in accuracy. Given the good experimental results and the generality of the formulation, we believe that the proposed method can be useful in a variety of classification tasks employing large databases of training objects and computationally expensive distance measures.

References

[1] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios. BoostMap: A method for efficient approximate similarity rankings. In *CVPR*, 2004.

[2] H.G. Barrow, J.M. Tenenbaum, R.C. Bolles, and H.C. Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. In *IJCAI*, pages 659–663, 1977.

[3] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *PAMI*, 24(4):509–522, 2002.

[4] 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*, 33(3):322–373, 2001.

[5] T.J. Darrell, I.A. Essa, and A.P. Pentland. Task-specific gesture analysis in real-time using interpolated views. *PAMI*, 18(12), 1996.

[6] V. S. Devi and M. N. Murty. An incremental prototype set building technique. *Pattern Recognition*, 35(2):505–513, 2002.

[7] C. Faloutsos and K. I. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *ACM SIGMOD*, pages 163–174, 1995.

[8] G. W. Gates. The reduced nearest neighbor rule. *IEEE Transactions on Information Theory*, 18(3):431–433, 1972.

[9] K. Grauman and T.J. Darrell. Fast contour matching using approximate earth mover's distance. In *CVPR04*, pages I: 220–227, 2004.

[10] P. E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14(3):515–516, 1968.

[11] G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems*, 28(4):517–580, 2003.

[12] G.R. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *PAMI*, 25(5):530–549, 2003.

[13] G. Hristescu and M. Farach-Colton. Cluster-preserving embedding of proteins. Technical Report 99-50, CS Department, Rutgers University, 1999.

[14] P. Indyk. *High-dimensional Computational Geometry*. PhD thesis, Stanford University, 2000.

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

[16] S. Z. Li and Z. Q. Zhang. Floatboost learning and statistical face detection. *PAMI*, 26(9):1112–1123, 2004.

[17] G. Mori, S. Belongie, and J. Malik. Shape contexts enable efficient retrieval of similar shapes. In *CVPR*, volume 1, pages 723–730, 2001.

[18] G. Mori and J. Malik. Estimating human body configurations using shape context matching. In *ECCV*, volume 3, pages 666–680, 2002.

[19] E.J. Ong and R. Bowden. A boosted classifier tree for hand shape detection. In *Face and Gesture Recognition*, pages 889–894, 2004.

[20] R.E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[21] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, volume 1, pages 511–518, 2001.

[22] 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*, 2(2):161–184, 2000.

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

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

[25] H. Zhang and J. Malik. Learning a discriminative classifier using shape context distances. In *CVPR*, volume 1, pages 242–247, 2003.