To appear in Proceedings of IEEE Workshop on Learning in Computer Vision and Pattern Recognition, June 2005.

Boosting Nearest Neighbor Classifiers for Multiclass Recognition

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

Abstract

Nearest neighbor classifiers are a popular method for multiclass recognition in a wide range of computer vision and pattern recognition domains. At the same time, the accuracy of nearest neighbor classifiers is sensitive to the choice of distance measure. This paper introduces an algorithm that uses boosting to learn a distance measure for multiclass k-nearest neighbor classification. Given a family of distance measures as input, AdaBoost is used to learn a weighted distance measure, that is a linear combination of the input measures. The proposed method can be seen both as a novel way to learn a distance measure from data, and as a novel way to apply boosting to multiclass recognition problems that does not require output codes. In our approach, multiclass recognition of objects is reduced to a single binary recognition task, defined on triples of objects. Preliminary experiments with eight UCI datasets yield no clear winner among our method, boosting using output codes, and k-nn classification using an unoptimized distance measure. Our algorithm did achieve lower error rates in some of the datasets, which indicates that it is a method worth considering for nearest neighbor recognition in various pattern recognition domains.

1 Introduction

Nearest neighbor classification is a popular method for multiclass recognition in general pattern recognition domains. In computer vision, nearest neighbor classification has been applied to a wide range of problems, including face recognition [21, 24], articulated pose estimation [18] and optical character recognition [3]. K-nearest neighbor (k-nn) classifiers are appealing because of their simplicity, ability to model a wide range of parametric and non-parametric distributions, and theoretical optimality as the training size goes to infinity. At the same time, k-nn recognition rates in real datasets are sensitive to the choice of distance measure. Choosing a good distance measure is particularly challenging when the dimensionality of the data is large.

Boosting is another very popular learning method, that has been successfully applied to many vision problems, including face detection [23] and image retrieval [20]. Boosting can be very effective with high-dimensional data, by combining many weak classifiers in a way that they complement each other. On the other hand, the natural setting for boosting is binary classification, and applying boosting methods to a multiclass recognition task typically requires partitioning the multiclass problem into multiple binary problems using output codes [1]. Recognition rates are sensitive to the choice of output code, and choosing the right code can be a challenging task.

This paper introduces a method that combines boosting with k-nn classification. From a k-nn perspective, the main contribution is a method for using boosting to learn, from training data, a distance measure for k-nn classification. Compared to other methods for optimizing a distance measure, boosting offers the capability of feature selection, an efficient training process, and the ability to handle data of very high dimensionality. From a boosting perspective, the key contribution is a strategy for associating a multiclass recognition problem with a single binary recognition problem, which is defined on triples of objects. We believe that this idea can facilitate applying boosting to multiclass problems, without having to use output codes.

We report results on three computer vision datasets from the UCI repository [4], and five non-vision datasets from the same repository. In the experiments we compare our method to two widely used methods: AdaBoost with output codes (using the results reported in [1]), and k-nn classification using an unoptimized distance measure. For each of these three methods there were two datasets where that method gave better results than the two other methods. The results suggest that our method is worth evaluating in systems that currently use AdaBoost with output codes or nearest neighbor classification with an unoptimized distance

This work was supported by NSF grants IIS-0308213 and IIS-0133825, and by ONR grant N00014-03-1-0108.

measure.

2 Related Work

The basic AdaBoost algorithms [10, 17] construct a classifier as a linear combination of weak classifiers. Each weak classifier is assumed to achieve an error rate lower than 0.5, as measured on a training set that has been weighted based on the results of previously chosen classifiers. An error rate lower than 0.5 is easy to achieve for binary classifiers, but becomes increasingly harder as the number of classes increases. As a consequence, the standard AdaBoost algorithms are not easily applied to multiclass problems.

To address this problem, AdaBoost.MO is proposed in [17]. In AdaBoost.MO, the multiclass problem is partitioned into a set of binary problems, using the idea of error correcting output codes (ECOC) proposed in [8]. An extensive experimental evaluation of AdaBoost.MO using different output codes is provided in [1], with the conclusion that no output code is clearly better, and the choice of the best code depends on the domain.

A poor choice of output code can lead to unnatural binary problems that are hard to learn. A possible remedy is to include the selection of the output code in the learning process, so that the code is learned from the data [6, 15]. In [7], binary output codes are replaced with continuous codes, which are optimized using an iterative method.

A general problem with output codes is that they essentially convert a single multiclass problem into multiple binary problems. The number of binary problems is typically at least linear to the number of classes, and it can even be quadratic in the case of the all-pairs output coding scheme. In problems with very large numbers of classes, the time and memory needed to construct and store all the resulting binary classifiers can be prohibitive. The problems of choosing output codes and scaling to a large number of classes are also shared by support vector machines [22], which are also primarily formulated for binary classification problems.

K-nearest neighbor classification can easily be applied to multiclass problems, regardless of the number of classes. However, to obtain good accuracy, we need to find an appropriate distance measure. Several methods have been proposed for learning a distance measure from data. In [5, 19], distance metrics are constructed based on estimates of class probability densities around objects. However, such estimates can be hard to obtain, especially in high dimensions. Furthermore, if those estimates were available, then we could simply use Bayesian classification, which is optimal.

In [9] and [11], a local measure is learned for the area around a given test point. These methods assume that an initial distance measure is available. Given a test point, these methods iteratively modify the initial distance measure, based on the nearest neighbors of the test point given the current distance measure. The method we propose is complementary to these methods, because it can be used to obtain an initial distance measure, on top of which those methods can be applied.

In [13], a variable interpolation kernel is used for classification. The kernel size and the similarity metric are learned from training data using conjugate gradient optimization. In [14] Fractional Programming is used to construct an asymmetric distance measure between test objects and training objects. This distance measure depends on the class label of the training object. Our method is similar to these methods, in that it constructs, using training data, a new similarity measure for k-nn classification. The main difference of our method is that it uses AdaBoost to optimize the similarity measure. This way, our method inherits the advantages of AdaBoost, including good generalization properties [16], efficient training algorithm, and ability to handle data of very high dimensionality.

The standard AdaBoost algorithm has been formulated for binary classification problems. Naturally, nearest neighbor classification can involve an arbitrary number of classes. To convert our problem into a binary problem, we use an idea introduced in the BoostMap method [2]. Given a space of objects with a computationally expensive distance measure, BoostMap constructs an embedding F that maps objects into high-dimensional real vectors. The embedding F is optimized based on the following criterion: given any three objects (q, a, b), such that q is closer to a than to b in the original space, we want as often as possible F to preserve the relative order, so that $F(q)$ is closer to $F(a)$ than to $F(b)$.

Our proposed method solves a different problem than BoostMap: BoostMap takes as input a computationally expensive distance measure, and outputs an efficient approximation of that distance measure. In our method, we take as input a (possibly large) number of distance measures, and the output is a weighted linear combination of those distance measures. The output distance measure is optimized for nearest neighbor classification accuracy. What our method shares with BoostMap is the "trick" of defining an optimization cost based on triples of objects. This trick will allow us to reduce the problem of learning a good similarity measure into a single binary classification problem, on which standard AdaBoost can be applied.

3 Problem Definition and Overview

Let X be a space of objects, Y be a finite set of classes, and $\mathbb{D}$ be a set of distance measures defined on X. For example, each distance measure can be based on a single coordinate of each object (if X is a vector space), or on a single feature when thousands of features are available (in the style of [23]). Each object $x \in X$ belongs to a class $y(x) \in Y$. We are given a training set S of m objects from X and their associated class labels: $S = \{(x_1, y(x_1)), \ldots, (x_m, y(x_m))\}$. We want to combine the distance measures in $\mathbb{D}$ into a single weighted distance measure that leads to good k-nn classification accuracy. We also want to estimate a good value for the number k of neighbors used by the k-nn classifier.

A note on terminology: in this paper, we use the terms "similarity measure" and "distance measure" almost interchangeably. Sometimes (but not always) the term "similarity measure" is used in the literature to indicate a measure where zero indicates maximum dissimilarity between objects, and increasing values indicate increasing similarity. In that sense, our algorithm produces a distance measure, i.e. a measure where zero indicates absolute similarity and increasing values indicate increasing dissimilarity.

3.1 Overview of the Algorithm

AdaBoost is good at combining binary weak classifiers, but is hard to apply directly to multiclass problems. In order to use AdaBoost to combine different distance measures, we will establish a one-to-one correspondence between distance measures and a family of binary classifiers that classify triples of objects. In particular, suppose we have a triple (q, a, b), where $q, a, b \in X$, $y(a) \neq y(b)$, and $y(q) \in \{y(a), y(b)\}$. In words, q is *either* of the same class as a or of the same class as b. The binary classification task is to decide whether $y(q) = y(a)$ or $y(q) = y(b)$.

A distance measure D defines a binary classifier, which compares the distances $D(q, a)$ and $D(q, b)$ and assigns to q the label of its nearest neighbor in the set $\{a, b\}$. This one-to-one correspondence that we establish between distance measures and binary classifiers allows us to convert the distance measures in $\mathbb{D}$ to weak classifiers, apply AdaBoost to combine those weak classifiers into a strong classifier, and then convert the strong classifier into a distance measure. At first, the training set used by AdaBoost is a random set of triples (q, a, b) of training objects. Each triple has to obey the constraint that $y(q) = y(a)$ and $y(q) \neq y(b)$. Intuitively, if the output of AdaBoost is a good classifier of triples, the corresponding distance measure should be good for k-nn classification, because it tends to give smaller distances to objects of the same class than to objects of different classes.

Given the distance measure that was constructed using AdaBoost, we define a new training set of triples, by imposing the additional constraint that a and b should be among the nearest neighbors of q in their respective classes. The error of a binary classifier on these triples is more closely related to the k-nn error of the distance measure that corresponds to that binary classifier. Then, we iterate between learning a new distance measure, by applying AdaBoost on the current training triples, and choosing new training triples using the current distance measure. In practice, this iterative refinement improves k-nn classification accuracy over the initial distance measure returned by the first application of AdaBoost.

4 Defining Binary Classifiers from Distances

In this section we formally define how to associate distances with binary classifiers. We use notation from the problem definition. First, we assign to each triple $(q, a, b) \in X^3$ a class label $p(q, a, b) \in \{-1, 0, 1\}$ as follows:

$$p(q,a,b) = \begin{cases} 1 & \text{if } (y(q) = y(a)) \wedge (y(q) \neq y(b)) \ . \\ 0 & \text{if } (y(q) = y(a)) \wedge (y(q) = y(b)) \ . \\ 0 & \text{if } (y(q) \neq y(a)) \wedge (y(q) \neq y(b)) \ . \\ -1 & \text{if } (y(q) \neq y(a)) \wedge (y(q) = y(b)) \ . \end{cases} \tag{1}$$

Note that $p(q, a, b)$ can take only three possible values, and assigns a class label to a *triple* of objects, based on the class labels of the individual objects. In contrast $y(x)$ denotes the class label of a single object and the possible values for $y(x)$ are as many as the number of classes defined on the space X.

Every distance measure D on X defines a discrete-output classifier $\bar{D}(q, a, b)$ and a continuous-output classifier $\tilde{D}(q, a, b)$, as follows:

$$\tilde{D}(q,a,b) \quad = \quad D(q,b) - D(q,a) \ . \tag{2}$$

$$\bar{D}(q,a,b) \quad = \quad \begin{cases} 1 & \text{if } D(q,a) < D(q,b) \ . \\ 0 & \text{if } D(q,a) = D(q,b) \ . \\ -1 & \text{if } D(q,a) > D(q,b) \ . \end{cases} \tag{3}$$

$\bar{D}$ is essentially a discretization of $\tilde{D}$, and $\tilde{D}$ can be considered to give a confidence-rated prediction [17]. The error rate of $\tilde{D}$ is defined to be the error rate of the corresponding $\bar{D}$.

Triples of class label 1 or -1 are relevant to how good the distance measure D is for k-nearest neighbor classification. For example, if $\tilde{D}$ correctly classifies all such triples, then D defines a perfectly accurate nearest-neighbor classifier. Without loss of generality, we will limit our attention to classifying triples (q, a, b) for which $p(q, a, b) = 1$, i.e. where $y(q) = y(a)$ and $y(q) \neq y(b)$. Any triple (q, a, b) with class label -1 corresponds to a triple (q, b, a) with class label 1.

5 Learning a Weighted Distance Measure with AdaBoost

The inputs to our algorithm are the following:

- A training set $S = \{(x_1, y(x_1)), \ldots, (x_m, y(x_m))\}$ of m objects of X, and their class labels $y(x_i)$. Given S we also define the set S_o of training objects to be $S_o = \{x_1, ..., x_m\}$, i.e. the set of all objects appearing in S.
- A set $\mathbb{D}$ of distance measures defined on X. Each distance measure $D \in \mathbb{D}$ will be used to define a classifier $\tilde{D}$, that will be evaluated by AdaBoost.

Since we want AdaBoost to combine classifiers of triples of objects, we construct a training set S' of m' triples of objects, where m' is a manually set parameter. The i-th triple (q_i, a_i, b_i) is chosen as follows:

- Pick an object $q_i \in S_o$ at random.
- Pick an object $a_i \in S_o$ such that that $y(q_i) = y(a_i)$ and $q_i \neq a_i$.
- Pick an object $b_i \in S_o$ such that $y(q_i) \neq y(b_i)$.

We run the generalized AdaBoost algorithm [17] with S' as the training set. AdaBoost evaluates all weak classifiers $\tilde{D}$ that correspond to distances $D \in \mathbb{D}$, and outputs a linear combination H_1 of some of those weak classifiers:

$$H_1 = \sum_{j=1}^{d} \alpha_j \tilde{D}_j \ . \tag{4}$$

Our end goal is to construct a distance measure that is good for nearest neighbor classification. We can easily construct this distance measure using the distances D_j and weights α_j that occur in the definition of H_1 (that was output by AdaBoost). In particular, we define a distance D^1_{out} as follows:

$$D^1_{\text{out}}(x_1, x_2) = \sum_{j=1}^{d} \alpha_j D_j(x_1, x_2) \ , \tag{5}$$

where x_1, x_2 are objects of X.

A natural question to ask is why we propose that D^1_{out} is a good distance measure for nearest neighbor classification. In order to answer that question, we first need to establish an important equivalence between D^1_{out} and the classifier H_1 constructed by AdaBoost: for any $q, a, b \in X$ such that $p(q, a, b) = 1$, H_1 classifies triple (q, a, b) correctly if and only if $D^1_{\text{out}}(q, a) < D^1_{\text{out}}(q, b)$. In other words, if we use Eq. 3 to define a classifier $\tilde{D}^1_{\text{out}}$ using distance measure D^1_{out}, then $\tilde{D}^1_{\text{out}} = H_1$. The proof is straightforward, but we include it for completeness:

Proposition 1 $\tilde{D}^1_{\text{out}} = H_1$.

Proof:

$$\begin{aligned}
\tilde{D}^1_{\text{out}}(q, a, b) &= D^1_{\text{out}}(q, b) - D^1_{\text{out}}(q, a) \\
&= \sum_{j=1}^{d} \alpha_j D_j(q, b) - \sum_{j=1}^{d} \alpha_j D_j(q, a) \\
&= \sum_{j=1}^{d} \alpha_j (D_j(q, b) - D_j(q, a)) \\
&= \sum_{j=1}^{d} \alpha_j \tilde{D}_j(q, a, b) = H_1(q, a, b) \ . \quad \square
\end{aligned}$$

Now we can provide some intuition as to why we expect D^1_{out} to be a good distance measure for nearest neighbor classification: by optimizing H_1, we have also optimized D^1_{out} so that objects of the same class tend to be closer to each other than to objects of other classes. Overall, this property is desirable for nearest neighbor classification. If the optimization has been perfect (i.e., if AdaBoost has constructed an H_1 that classifies correctly all triples (q, a, b) with class label $p(q, a, b) \neq 0$) then the nearest neighbor classifier defined using D^1_{out} will also give perfect accuracy.

While a zero error rate for H_1 guarantees a zero error rate for nearest neighbor classification using distance measure D^1_{out}, when the error rate of H_1 is non-zero there is very little we can say about the error of the nearest neighbor classifier. The remainder of this section discusses how to define a set of training triples such that the classification error on those triples is more directly related to the error rate of the nearest neighbor classifier, and how to use that set of triples to iteratively refine the distance measure.

5.1 Iterative Refinement

Distance measure $\tilde{D}^1_{\text{out}}$ has been optimized by AdaBoost with respect to binary classification of a random training set of triples. However, for accurate k-nn classification of object $q \in X$ using some distance measure D, it does not have to hold that all training objects of the same class as q are closer to q than all training objects of other classes (which would correspond to $\tilde{D}$ perfectly classifying all triples (q, a, b) with $a, b \in S_o, y(a) = y(q), y(b) \neq y(q)$). It suffices that, among the k nearest neighbors of q in training set S_o, objects of class $y(q)$ achieve a simple majority. Therefore, it is sufficient (and not even necessary) that $\tilde{D}$ classifies correctly all triples (q, a, b) such that a and b are among the $(\lfloor k/2 \rfloor + 1)$ nearest neighbors of q among training objects of their respective classes $y(a)$ (which equals $y(q)$) and $y(b)$.

The problem with using a random training set of triples is that even if AdaBoost has produced a classifier with a very low error rate on such triples, that error is not tightly related to the error of the resulting nearest neighbor classifier. Therefore, we cannot really argue that our learning algorithm optimizes nearest neighbor classification in a principled way. Based on these considerations, given distance measure D^1_{out}, we want to define a new set of training triples, which is more related to k-nn classification error, and use that new training set to learn a new distance measure D^2_{out}.

To define the new training set of triples, first we define $N_w(q, r, D)$, the w-class r-th nearest neighbor of an object $q \in X$ as follows: $N_w(q, r, D)$ is the r-th nearest neighbor of q based on distance measure D, among all objects $x \in S_o - \{q\}$ that belong to class w. If q itself is a training object with class label w, it is not considered to be a w-class r-th nearest neighbor of itself for any r. Also, given a distance D, the set Y of all classes, and an integer r, we define sets of triples $T(D, r)$ and $T'(D, r)$, as follows:

$$T(D,r) = \{(q, N_{y(q)}(q,r,D), N_w(q,r,D)) : q \in S_o, w \in (Y - \{y(q)\})\} . \quad (6)$$

$$T'(D,r) = \bigcup_{i=1}^{r} T(D,i) . \quad (7)$$

$T(D)$ is the set of all triples (q, a, b) we can define by choosing a training object q, its same-class r-th nearest neighbor a, and its w-class r-th nearest neighbor b for all classes $w \neq y(q)$.

If we knew the right value of k for k-nn classification, we could set $r_{\max} = \lfloor k/2 \rfloor + 1$, and build a new set of training triples by randomly sampling m' triples from $T'(D^1_{\text{out}}, r_{\max})$, since classifying such triples correctly is related to the k-nn classification error. We can actually estimate a value for k by trying different values of k and evaluating the k-nn error on the set S_o of training objects, or on a validation set, based on distance measure D^1_{out}. In the experimental results, we use an initial implementation where we manually set $r_{\max} = 2$, regardless of the value we found for k.

We construct the new training set of m' triples by sampling from $T'(D^1_{\text{out}}, r_{\max})$. Now, we can start the iterative refinement process. In general, for $n > 1$, the n-th iteration consists of choosing a set of training triples by sampling m' triples from $T'(D^{n-1}_{\text{out}}, r_{\max})$, and then learning a new distance measure D^n_{out} from those triples using AdaBoost.

At the end of the n-th iteration, based on D^n_{out}, we measure the error of k-nn classifiers on the set S_o of training objects for all possible values of k. We set k_n to be the k that leads to the smallest training error, and we define e_n to be that error. When, for some n, we get $e_n \geq e_{n-1}$, then we stop the learning algorithm altogether, and we give the final output: $D_{\text{out}} = D^{n-1}_{\text{out}}$, and $k_{\text{out}} = k_{n-1}$. The number k_{out} is the k we will use for k-nn classification of the test objects.

5.2 Theoretical Considerations

As mentioned in the previous subsection, given a distance measure D and an integer k, if the classifier $\tilde{D}$ perfectly classifies triples on the set $T'(D, \lfloor k/2 \rfloor + 1)$, then D and k define a perfect k-nn classifier on the training set S_o. Even when the error of $\tilde{D}$ is non-zero, we still intuitively expect D to give good nearest neighbor classification accuracy, since it has been optimized to give smaller distances between an object and its same-class nearest neighbors than between the same object and its nearest neighbors from other classes.

At the same time, intuition is not a substitute for theoretically provable guarantees. At this point we still do not have a theoretical framework that tightly associates the error of $\tilde{D}$ with the k-nearest neighbor classification error using D. In the next paragraphs we describe some bounds that we can actually prove, and we outline directions that we are exploring for obtaining the desired theoretical justification for our method.

First, we establish a tighter connection between the error of $\tilde{D}$ on set $T(D,1)$ and the 1-nn classification error of measure D on training objects:

Proposition 2 *Given a distance measure D, if the corresponding classifier $\tilde{D}$ has error rate $e'(\tilde{D})$ on the set $T(D,1)$, and the 1-nn classifier defined using D has error $e(D)$ on the training set S_o, then $e'(\tilde{D}) \leq e(D) \leq (|Y|-1)e'(\tilde{D})$.*

Proof: For each $q \in S_o$, $T(D,1)$ has a subset $T_q(D,1)$ of $|Y|-1$ triples of the form $(q, N_{y(q)}(q,1,D), N_w(q,1,D))$. $T_q(D,r)$ contains one triple for each class $w \neq y(q)$. Object q is classified incorrectly by the 1-nn classifier if and only if some number of triples (between one and $|Y|-1$) in $T_q(D,1)$ are classified incorrectly by $\tilde{D}$. Therefore, if f is the number of misclassified training objects, and f' is the number of misclassified triples in $T(D,1)$, $f \leq f' \leq (|Y|-1)f$. Since $f \leq f'$, it follows that $(|Y|-1)f \leq (|Y|-1)f'$. Therefore, $f' \leq (|Y|-1)f \leq (|Y|-1)f'$. Dividing f', $(|Y|-1)f$, and $(|Y|-1)f'$ by $|T(D,1)|$, and taking into account that $|T(D,1)| = (|Y|-1)|S_o|$, $e'(\tilde{D}) = \frac{f'}{|T(D,1)|}$, and $e(D) = \frac{f}{|S_o|}$, we get that $e'(\tilde{D}) \leq e(D) \leq (|Y|-1)e'(\tilde{D})$.

□

Proposition 2 establishes a connection between the error of a classifier $\tilde{D}$ on a special set of triples $T(D,1)$ and the corresponding 1-nn error of the distance measure D on training objects. However, at this point, we do not have an equally compact formula that associates the error on some set of triples with k-nn error when $k > 1$.

Even in analyzing 1-nn error, an important issue is that AdaBoost, at the n-th iteration, constructs a distance measure D^n_{out} by optimizing a classifier on the current training set of triples. That training set is chosen using distance measure D^{n-1}_{out}, and is not necessarily related to the set $T(D^n_{\mathrm{out}}, 1)$, which is the set linked to the 1-nn classification error by Proposition 2. Therefore, from a theoretical standpoint, we cannot claim that AdaBoost optimizes a quantity directly related to 1-nn or k-nn classification error. We have a chicken-and-egg problem, where in order to choose the right set of training triples we need to know the distance measure, but at the same time we need the training triples in order to construct the distance measure using AdaBoost.

Our iterative refinement method is a heuristic way to address that problem: hopefully after a few iterations the distance measure will stop changing that much. Ideally, we would like to get to a point where round $n-1$ produces distance measure D^{n-1}_{out}, we choose training triples based on that measure, and applying AdaBoost to those triples in the n-th round produces again D^{n-1}_{out} (i.e. $D^n_{\mathrm{out}} = D^{n-1}_{\mathrm{out}}$). In that case we can argue that the output distance measure D^n_{out} has been optimized with respect to the right set of training triples. At this point we have no theoretical guarantees that our algorithm will indeed converge. We are currently investigating potential modifications to our algorithm that can lead to provable convergence. At the same time, iterative refinement does improve the classification accuracy of the resulting distance measure in practice.

6 Complexity

The storage requirements of our method for training and testing are dominated by the need to store all training objects, as is typical in k-nn classifiers. For high-dimensional data, our algorithm sometimes effectively performs feature selection, by outputting a distance measure that only depends on some of the features. That allows for a more compact representation of the training objects in the actual k-nn classifier.

The training time depends on the number $|\mathbb{D}|$ of distance measures in $\mathbb{D}$, the total number of iterations n, the average number of steps d it takes each invocation of AdaBoost to complete its training, the number of training triples m' used at each iteration, and a number g, which we define to be the maximum number of objects that belong to a single class in the training set S_o. If m is the number of training objects in S_o and t is the number of classes, if there is an equal number of objects for each class, then $g = m/t$.

At each iteration, we need to choose m' training triples. Choosing each triple involves finding two w-class r-th nearest neighbors of q, for two classes $w \in Y$, some integer r, and some training object q. This takes time $O(g)$ per triple. Then we need to invoke AdaBoost, whose training takes time $O(m'd|\mathbb{D}|)$. So, the overall training time of the algorithm is $O(nm'(g + d|\mathbb{D}|))$.

In our current implementation, we use k-nn training error as a stopping criterion. To compute that at each iteration, we need to compare each training object to all other training objects, which takes time $O(m^2)$. For large training sets, we can estimate the k-nn training error statistically using sampling, or we can measure error on a smaller validation set, so that we can eliminate this quadratic dependency on m.

The recognition time, in the worst, case, involves computing distances from the test object to all training objects. However, several efficient methods proposed for finding nearest neighbors or approximate nearest neighbors may be applicable in some domains [12, 25].

Table 1: Information about the UCI datasets used in the experiments, largely copied from [1].

Dataset	Train	Test	Attributes	Classes
glass	214	-	9	6
isolet	6238	1559	617	26
letter	16000	4000	16	26
pendigits	7494	3498	16	10
satimage	4435	2000	36	6
segmentation	2310	-	19	7
vowel	528	462	10	11
yeast	1484	-	8	10

Table 2: The error rate achieved by our method (denoted as Boost-NN) in each dataset, compared to the *best* result attained among the 15 AdaBoost.MO variations evaluated in [1] and the *best* result attained among the 6 variations of "naive" k-nn classification. For our method we also provide the standard deviation across multiple trials, except for the isolet dataset where we only ran one trial of our algorithm.

Dataset	Boost-NN	Allwein	Naive k-nn
glass	24.4 ± 1.7	25.2	26.8
isolet	6.5	5.3	7.6
letter	3.5 ± 0.2	7.1	4.5
pendigits	3.9 ± 0.6	2.9	2.2
satimage	9.6 ± 0.3	11.2	9.1
segmentation	1.8 ± 0.2	0.0	2.6
vowel	41.9 ± 1.6	49.8	44.3
yeast	41.7 ± 0.6	41.6	40.9

7 Experiments

We applied our algorithm to eight pattern recognition datasets from the UCI repository [4]. Some information

Table 3: For each dataset, we count how many variations of AdaBoost.MO gave lower (<), equal (=), and higher (>) error rates than our algorithm, based on the results in [1]. For example, in the segmentation dataset, all 15 variations of AdaBoost.MO did better than our algorithm. We also give the same information for the six variations of the naive k-nn algorithm. We consider an error rate equal to the error rate of our algorithm if it is within one standard deviation of the error rate of our algorithm. Note that in [1] some of the 15 variations were not tried on all datasets, because of their complexity.

Dataset	Allwein			Naive k-nn		
	<	=	>	<	=	>
glass	0	1	14	0	0	6
isolet	2	0	7	0	0	6
letter	0	0	12	0	0	6
pendigits	3	0	12	3	3	0
satimage	0	0	15	2	2	2
segmentation	15	0	0	0	0	6
vowel	0	0	15	0	0	6
yeast	0	6	9	1	5	0

about these datasets is given in Table 1. The *letter*, *satimage* and *segmentation* datasets are computer vision datasets, where we need to classify objects or regions based on some pre-extracted image features. The remaining datasets are from non-vision domains, and we used them to get a more complete picture about the behavior of our algorithm.

In [1], 15 different variations of AdaBoost.MO are evaluated on 13 UCI datasets. The fifteen variations were obtained by trying five different output codes, and three different ways to assign a class to a test object based on the outputs of the binary classifiers. Our goal was to compare our algorithm to the results given by [1] on the same datasets. We ended up using only eight of those datasets. We did not use four datasets (dermatology, soybean, thyroid, audiology) because they have missing attributes, which our current formulation cannot handle. One dataset (ecoli) contains a nominal attribute, which our current implementation cannot handle in practice (although the theoretical formulation is fully compatible with nominal attributes). For the remaining datasets, we compare our results to those cited by [1], using in each dataset the same training and test set that were used in that publication. For datasets where no independent test set was available, we used 10-fold cross-validation, again as in [1].

We also compared our algorithm to a "naive" k-nn algorithm, that does not learn a distance measure, but instead computes a standard L_1 or Euclidean (L_2) distance. We applied both those distances to data that was normalized using three normalization schemes: the null scheme (no normalization at all), normalizing the range of each attribute to be between 0 and 1, and normalizing the standard deviation of each attribute to be equal to 0.5. This gives us a total of 6 variations. For each variation, the best k was chosen to be the one that minimized the classification error on the training set. In datasets where cross-validation was needed, we ran three full cross-validation trials and averaged the results. It is interesting that, in some datasets, the "naive" k-nn algorithm had actually lower error rates compared both to our method and the methods evaluated in [1]. This is not entirely unexpected, since neither AdaBoost nor our formulation guarantee learning globally optimal values for the classifier parameters.

The family $\mathbb{D}$ of distance measures used as input to our algorithm was constructed by using each attribute to define a distance measure based only on that attribute. In all experiments, the number of training triples m' was set to $10,000$, except for isolet where $m' = 100,000$. We noticed that larger values of m' did not make much difference on the resulting error rate.

In some UCI datasets, training and test objects were collected from a set of human subjects, and more than one object was collected from each human subject. In forming training triples, given a training object q, we exclude objects a and b coming from the same subject. For the UCI pendigits dataset, we could not find any subject ID information, so we could not apply this criterion.

Since our algorithm relies on random sampling in constructing training triples, we ran at least 22 trials on each dataset, in order to estimate the standard deviation of the error rate. The only exception was the isolet dataset, where we ran a single trial because training was significantly slower than in the other datasets. In datasets where cross-validation was used, each trial consisted of a full cross-validation, where the dataset was split into 10 subsets and each subset was used once as a test set. The running time for each trial, which included learning the distance measure and evaluating nearest neighbor accuracy on the test data, ranged from about 30 seconds for the vowel dataset, to about one hour for the letter dataset and two days for the isolet dataset (the only dataset where we used 100,000 training triples). The remaining datasets took a few minutes per trial. The running time was measured on an Athlon 1.2GHz PC. In general, the number of iterations per trial (each iteration consisting of forming training triples followed by an application of AdaBoost to construct a distance measure) was between three and eight for all datasets.

Tables 2 and 3 compare the results of our method to the results of the variations of AdaBoost.MO cited in [1] and those of "naive" k-nn. In those tables we refer to our method as Boost-NN (for Boosted Nearest Neighbors). Overall, for each method there are two datasets where that method does better than the other methods. There are also two datasets (glass and yeast) where the results of our algorithm and the best results from ECOC-based boosting and naive k-nn classification are quite similar.

Overall, the results yield no clear winner among our method, AdaBoost.MO, and naive k-nn classifiers. At the same time, we believe that the results offer evidence that, at least in some domains, our method may provide better classifiers than other standard methods. Clearly, more results are needed in order to evaluate the strengths and weaknesses of our method versus AdaBoost.MO and existing methods for learning distance measures for k-nn classification.

8 Discussion and Future Work

Our algorithm combines the ability of AdaBoost to select and combine different weak classifiers in a way that they complement each other with the ability of k-nn classifiers to model complex, non-parametric distributions. The experiments show that, in some domains, our method can lead to lower error rates than the alternative AdaBoost.MO and k-nn classification methods that we compared our method to. At the same time, it is important to experimentally compare our algorithm to alternative methods for learning distance measures from data [13, 14] in terms of classification accuracy and efficiency of the training algorithm.

One direction for future work is to try to establish a tighter theoretical connection between the training error of the classifier optimized by AdaBoost (which operates on triples of objects) and the training error of the nearest neighbor classifier (which operates on actual objects). Another direction is to apply our algorithm in problems with a very large number of classes, like sign language recognition, or articulated body pose estimation. Boosting with output codes is hard to apply in such problems, because typically the number of binary classifiers that need to be learned is at least as large as the number of classes. Our formulation converts the multiclass problem into a single binary problem, defined on triples of objects, and therefore can easily be applied to problems with a large number of classes.

References

[1] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: a unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

[2] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios. BoostMap: A method for efficient approximate similarity rankings. In *CVPR*, volume 2, pages 268–275, 2004.

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

[4] C. L. Blake and C. J. Merz. UCI repository of machine learning databases. http://www.ics.uci.edu/∼mlearn/mlrepository.html, 1998.

[5] E. Blanzieri and F. Ricci. A minimum risk metric for nearest neighbor classification. In *ICML*, pages 22–31, 1999.

[6] K. Crammer and Y. Singer. On the learnability and design of output codes for multiclass problems. *Machine Learning*, 47(2-3):201–233, 2002.

[7] O. Dekel and Y. Singer. Multiclass learning by probabilistic embeddings. In *NIPS*, 2002.

[8] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

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

[10] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *ICML*, pages 148–156, 1996.

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

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

[13] D. G. Lowe. Similarity metric learning for a variable-kernel classifier. *Neural Computation*, 7(1):72–85, 1995.

[14] R. Paredes and E. Vidal. A class-dependent weighted dissimilarity measure for nearest neighbor classification problems. *Pattern Recognition Letters*, 21(12):1027–1036, 2000.

[15] G. Rätsch, A. Smola, and S. Mika. Adapting codes and embeddings for polychotomies. In *NIPS*, 2003.

[16] R. E. Schapire. The boosting approach to machine learning: An overview. In *MSRI Workshop on Nonlinear Estimation and Classification*, 2002.

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

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

[19] R. D. Short and K. Fukunaga. The optimal distance measure for nearest neighbor classification. *IEEE Transactions on Information Theory*, 27(5):622–627, 1981.

[20] K. Tieu and P. Viola. Boosting image retrieval. In *CVPR*, pages 228–235, 2000.

[21] M. Turk and A. P. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1), 1991.

[22] V. N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., 1995.

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

[24] L. Wiskott, J.-M. Fellous, N. Krüger, and C. von der Malsburg. Face recognition by elastic bunch graph matching. *PAMI*, 19(7):775–779, 1997.

[25] 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.