# Boosting Nearest Neighbor Classifiers for Multiclass Recognition

Boston University Computer Science Tech. Report No. 2004-006, February 5, 2004.

# Boosting Nearest Neighbor Classifiers for Multiclass Recognition

**Vassilis Athitsos** ATHITSOS@CS.BU.EDU
**Stan Sclaroff** SCLAROFF@CS.BU.EDU
Computer Science Department, Boston University, 111 Cummington Street, Boston, MA 02215, USA

## Abstract

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 into 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, in some domains, it may lead to better results than existing methods.

## 1. Introduction

K-nearest neighbor classification and boosting are two popular methods for multiclass recognition. 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 data sets 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 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 (Allwein et al., 2000). Recognition rates are sensitive to the choice of output code, and choosing the right code can be a challenging task.

This paper introduces a new method for combining 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, and also has very well understood theoretical properties, including resistance to overfitting the training data (Schapire & Singer, 1999).

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 problems with a very large number of classes. We also contribute a first application of this idea: learning a distance measure for k-nn classification using boosting.

## 2. Related Work

The basic AdaBoost algorithms (Freund & Schapire, 1996; Schapire & Singer, 1999) 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.

Schapire and Singer (1999) propose an algorithm, called AdaBoost.MO, in which the multiclass problem is partitioned into a set of binary problems, using the idea of error correcting output codes (ECOC) proposed in Dietterich and Bakiri (1995). Allwein et al. (2000) provide an extensive experimental evaluation of

AdaBoost.MO using different output codes, and conclude 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 (Crammer & Singer, 2002; Rätsch et al., 2003). Dekel and Singer (2002) replace binary output codes with continuous codes, which are optimized using an iterative method.

In the field of k-nn classification, different approaches have been proposed for constructing a good distance measure. Short and Fukunaga (1981) and Blanzieri and Ricci (1999) propose distance metrics that are based on estimates of class probability densities around objects. However, such estimates can be hard to obtain, especially in high dimensions.

In Lowe (1995), a variable interpolation kernel is used for classification. The kernel size and the similarity metric are optimized using training data. Paredes and Vidal (2000) use Fractional Programming to optimize an asymmetric distance measure between test objects and training objects, which depends on the class label of the training object.

In Hastie and Tibshirani (1996) and Domeniconi et al. (2002), a local measure is learned for the area around a given test point. These methods are iterative and require choosing an initial distance measure.

## 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 $X$. 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 higher k-nn classification accuracy than the individual distance measures in $\mathbb{D}$. We also want to estimate a good value for the number $k$ of neighbors used by the k-nn classifier.

### 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 $y(q) \in \{y(a), y(b)\}$. 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 $\{a, b\}$. The 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, with the only constraint that $y(q) = y(a), y(q) \neq y(b)$. Intuitively, if the output of AdaBoost is a good classifier of triples, the corresponding distance measure should be good for k-nn classification.

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.

Obtaining a family of distance measures $\mathbb{D}$, to use as input to our algorithm, can be achieved in various ways in practice. If the objects in $X$ are represented as vectors of attributes, each attribute can be used to define a distance measure. We can also define distance measures based on non-linear combinations of attributes.

## 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 $p(q, a, b) \in \{-1, 0, 1\}$:

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

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 $y(q) \neq y(b)$. 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 (Schapire & Singer, 1999). The error rate of $\tilde{D}$ is defined to be the error rate of the corresponding $\bar{D}$.

## 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 $X$.

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 (Schapire & Singer, 1999) on the training set $S'$ of triples. 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: $H_1 = \sum_{j=1}^{d} \alpha_j \tilde{D}_j$.

Using $H_1$ we define a distance $D^1_{\text{out}}$ as follows:

$$D^1_{\text{out}}(x_1, x_2) = \sum_{j=1}^{d} \alpha_j D_j(x_1, x_2) \ , \tag{4}$$

where $x_1, x_2$ are objects of $X$.

We want to claim that AdaBoost essentially constructed $D^1_{\text{out}}$ by learning the corresponding binary classifier $\tilde{D}^1_{\text{out}}$. To make that claim, we should show that $\tilde{D}^1_{\text{out}} = H_1$. This is straightforward to show, but not a trivial thing to check: if $H_1$ were a linear combination of discrete-output classifiers $\bar{D}_j$, as opposed to continuous-output classifiers $\tilde{D}_j$, then we would not be able to define a distance measure $D^1_{\text{out}}$ such that $H_1 = \tilde{D}^1_{\text{out}}$ or $H_1 = \bar{D}^1_{\text{out}}$.

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

**Proof:**

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

Given $D^1_{\text{out}}$, we could use it directly for k-nn classification. However, we can improve accuracy by refining this measure in an iterative way, and this is what we discuss next.

### 5.1. Iterative Refinement

$\tilde{D}^1_{\text{out}}$, 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 $S_o$, 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 $y(b)$.

Based on these considerations, given distance measure $D^1_{\text{out}}$, 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 $D^2_{\text{out}}$. 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 $x \in S_o$ 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:

$$\begin{aligned} T(D,r) &= \{(q, N_{y(q)}(q,r,D), N_w(q,r,D)) : \\ &\quad q \in S_o, w \in (Y - \{y(q)\})\} \ . \quad (5) \\ T'(D,r) &= \bigcup_{i=1}^{r} T(D,i) \ . \quad (6) \end{aligned}$$

$T(D)$ 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 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 $D^1_{\text{out}}$. In the experimental results, we use an initial implementation where we manually set $r_{\max} = 2$, regardless of the value we found for $k$. In the short term we plan to get results using an implementation where $r_{\max}$ is set automatically to $\lfloor k/2 \rfloor + 1$.

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 $D^n_{\text{out}}$ from those triples using AdaBoost.

At the end of the n-th iteration, based on $D^n_{\text{out}}$, for all possible values of $k$, we measure the error of k-nn classifiers on the set $S_o$ of training objects. 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 $k_{\text{out}}$ is the $k$ we will use for k-nn classification.

## 6. Theoretical Considerations

### 6.1. Connecting Error on Triples to Nearest Neighbor Classification Error

In the previous section we established that if, given a distance measure $D$ and an integer $k$, 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 $S_o$. Here we establish a tighter connection between the error on set $T(D, 1)$ and 1-nn classification error 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 $T(D, 1)$, $f \leq f' \leq (|Y| - 1)f$. Dividing $f'$ by $|T(D, 1)|$ and $f$ by $|S_o|$, and taking into account that $|T(D, 1)| = (|Y| - 1)|S_o|$, we get $e'(\tilde{D}) \leq e(D) \leq (|Y| - 1)e'(\tilde{D})$. $\square$

### 6.2. What Is Missing

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 $D^n_{\text{out}}$ by minimizing an upper bound on the error on the current training set of triples (Schapire & Singer, 1999). That training set is not related to the set $T(D^n_{\text{out}}, 1)$, 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.

A potential path towards establishing a connection between AdaBoost optimization and 1-nn error is as follows: we could use, after the n-th iteration of our algorithm, $T(D^n_{\text{out}}, 1)$ (or a random subset of it) as the new training set of triples. If, at the end of the next iteration, we get that $T(D^{n+1}_{\text{out}}, 1) = T(D^n_{\text{out}}, 1)$, then we get that $T(D^{n+1}_{\text{out}}, 1)$ is actually the training set that AdaBoost used to construct $\tilde{D}^{n+1}_{\text{out}}$. Therefore, the error rate of $\tilde{D}^{n+1}_{\text{out}}$ on $T(D^{n+1}_{\text{out}}, 1)$ would be the AdaBoost training error. Since Proposition 2 connects that error to the 1-nn error on actual objects, we could claim that the $(n+1)$ application of AdaBoost minimized an upper bound on the 1-nn error on training objects.

Such a proof can be completed if we show convergence of the sequence of distance measures produced during the iterative refinement process. However, we have no theoretical or experimental evidence of such convergence. This means that the optimization of the distance measure, with our current formulation, is heuristic. However, we should stress that, in practice, the first iterations do reduce training and test error, and afterwards those errors fluctuate in a small range, so the iterative refinement does improve accuracy over the initial distance measure $D^1_{\text{out}}$.

## 7. 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 (Indyk, 2000; Yianilos, 1993).

## 8. Relation to Output Codes

Our algorithm uses training triples $(q, a, b)$ such that $q$ and $a$ belong to the same class and $b$ belongs to a different class. Each such triple is an instance of one of the each-versus-each binary problems, which are defined by choosing any pair of classes. This establishes a conceptual connection between our approach and the each-versus-each output code. However, our method is not equivalent to AdaBoost.MO with the each-versus-each output code, because in our case the output is a single distance measure and a value for the $k$ to be used in k-nn classification. With the each-versus-each output code, the output is a set of binary classifiers, and the size of that set is quadratic to the number of classes.

A key feature of our algorithm is that we can control time complexity by using sampling to create the training set of triples, i.e. by setting parameter $m'$. Of course, sampling of the training set can also be applied in AdaBoost.MO. However, our method learns far fewer parameters than AdaBoost.MO: we just learn a weight for each distance measure, and a value of $k$ for k-nn classification. In AdaBoost.MO the number of parameters, and training time, are proportional to the number of binary classifiers, which is $O(t^2)$ for the each-versus-each code, where $t$ is the number of classes. It will be interesting to evaluate experimentally whether the small number of learned parameters can make our algorithm more amenable to sampling, and therefore more applicable to problems with a large number of classes.

Another feature of our method is that, while learning relatively few parameters, we still obtain a classifier that inherits the ability of nearest neighbor classifiers to model complex, non-parametric distributions using a set of training objects. In AdaBoost.MO, the capability to model complex distributions is obtained by learning a large number of parameters, which are needed to specify all the binary classifiers.

K-nn classifiers are often considered inefficient in terms of storage and time requirements. However, classifiers obtained with AdaBoost.MO can be, in some cases, comparably or more inefficient at classification time, because they require storage and application to the

*Table 1.* Information about the UCI datasets used in the experiments, largely copied from (Allwein et al., 2000).

| 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 Allwein et al. (2000) 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 | $24.4 \pm 1.7$ | 25.2 | 26.8 |
| isolet | 6.5 | 5.3 | 7.6 |
| letter | $3.5 \pm 0.2$ | 7.1 | 4.5 |
| pendigits | $3.9 \pm 0.6$ | 2.9 | 2.2 |
| satimage | $9.6 \pm 0.3$ | 11.2 | 9.1 |
| segmentation | $1.8 \pm 0.2$ | 0.0 | 2.6 |
| vowel | $41.9 \pm 1.6$ | 49.8 | 44.3 |
| yeast | $41.7 \pm 0.6$ | 41.6 | 40.9 |

test object of a number of binary classifiers. For some output codes, the number of binary classifiers can be linear or quadratic to the number of classes.

## 9. Experiments

We applied our algorithm to eight datasets from the UCI repository (Blake & Merz, 1998). Some information about these datasets is given in Table 1. Allwein et al. (2000) evaluate 15 different variations of AdaBoost.MO 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 Allwein et al. (2000) 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; this is a shortcoming we plan to address soon. For the remaining datasets, we compare our results to those cited by Allwein et al. (2000), 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 Allwein et al. (2000).

*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 Allwein et al. (2000). 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 Allwein et al. (2000) 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 |

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 Allwein et al. (2000). 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, $m' = 10,000$, 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, multiple training data were collected by each of a set of human subjects. 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. 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 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 with 2GB of memory. In general, the number of iterations per trial (each iteration consisting of forming training triples followed by an application of AdaBoost) 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 Allwein et al. (2000) and those of "naive" k-nn. In those tables we refer to our method as Boost-NN (for Boosted Nearest Neighbors). Overall, the results show that 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.

We should mention that, in the Allwein et al. (2000) implementation of AdaBoost.MO, the weak classifiers were decision stumps. For a more appropriate comparison of our method to AdaBoost.MO we need an implementation of AdaBoost.MO in which the weak classifiers are nearest neighbor classifiers based on single attributes, so that they resemble as much as possible the weak classifiers we use in our algorithm.

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.

## 10. 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 some standard k-nn classification methods, and some implementations of boosting based on output codes. At the same time, we believe that the following experiments are needed to better evaluate our method:

- Analyze dependence of error rate on the number $m'$ of training triples.
- Analyze whether the sequence of distance measures produced in the iterative refinement stage exhibits some kind of convergence.
- Compare our method to existing methods for learning distance measures from data (Lowe, 1995; Paredes & Vidal, 2000).
- Evaluate different variations of AdaBoost.MO, using k-nearest neighbors as weak binary classifiers. That would provide a more appropriate comparison between AdaBoost.MO and our method.
- Experiment with datasets with large numbers of classes, in the hundreds or thousands. Domains where datasets with a very large number of classes can exist include recognition of faces, fingerprints, articulated body pose, and speech. We should test the conjecture that, in such datasets, our method may be more tolerant to sampling and therefore more efficient than methods using output codes.

Also, there are several directions that we are interested in exploring, that may lead to better classification rates:

- Training an ensemble of k-nn classifiers, and combining them using majority voting.
- Extending our formulation so as to learn class-specific asymmetric distance measures, as proposed in Paredes and Vidal (2000).

- Constructing a hierarchy of k-nn classifiers. For example, based on the confusion matrix of the classifier learned by our current algorithm, we can train specialized k-nn classifiers that focus on specific sets of classes that the top-level k-nn classifier tends to confuse with each other.

Also, in terms of the theoretical foundation of our method, it is an open question whether we can provide a connection between the training error on triples and the training and generalization error on actual objects. Such a connection would allow us to extend known theoretical properties of AdaBoost to the resulting k-nn classifier, and it would provide a theoretical validation for the claim that our algorithm constructs a distance measure optimized for k-nn classification.

## 11. Conclusions

We have proposed a novel algorithm that uses boosting to learn a weighted distance measure for k-nn classification. Our algorithm is also a new method for applying boosting to multiclass recognition problems, that does not use output codes, and that may be a more efficient alternative in problems with very large numbers of classes. Preliminary experiments show that, in some domains, our algorithm provides better results than some related approaches. More experiments are needed to better understand our approach and compare it to existing approaches.

At the core of our formulation lies the reduction of multiclass recognition of objects to binary classification of triples of objects. In a sense, the algorithm we propose in this paper is a first attempt to apply this idea to boosting multiclass recognition. In the future work section we propose to use this idea to train classifiers of a more sophisticated structure, that will hopefully be more powerful and lead to better recognition rates.

## References

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

Blake, C., & Merz, C. (1998). UCI repository of machine learning databases. http://www.ics.uci.edu/∼mlearn/mlrepository.html.

Blanzieri, E., & Ricci, F. (1999). A minimum risk metric for nearest neighbor classification. *ICML* (pp. 22–31).

Crammer, K., & Singer, Y. (2002). On the learnability and design of output codes for multiclass problems. *Machine Learning*, *47*, 201–233.

Dekel, O., & Singer, Y. (2002). Multiclass learning by probabilistic embeddings. *NIPS*.

Dietterich, T., & Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, *2*, 263–286.

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

Freund, Y., & Schapire, R. (1996). Experiments with a new boosting algorithm. *ICML* (pp. 148–156).

Hastie, T., & Tibshirani, R. (1996). Discriminant adaptive nearest-neighbor classification. *PAMI*, *18*, 607–616.

Indyk, P. (2000). *High-dimensional computational geometry*. Doctoral dissertation, MIT.

Lowe, D. (1995). Similarity metric learning for a variable-kernel classifier. *Neural Computation*, *7*, 72–85.

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

Rätsch, G., Smola, A., & Mika, S. (2003). Adapting codes and embeddings for polychotomies. *NIPS*.

Schapire, R., & Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, *37*, 297–336.

Short, R., & Fukunaga, K. (1981). The optimal distance measure for nearest neighbor classification. *IEEE Transactions on Information Theory*, *27*, 622–627.

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