

Cloud Classification Using Error-Correcting Output Codes*

David W. Aha
Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory
Washington, DC 20375
aha@aic.nrl.navy.mil

Richard L. Bankert
Marine Meteorology Division
Naval Research Laboratory
Monterey, CA 93943
bankert@nrlmry.navy.mil

October 30, 1996

Submission to *AI Applications: Natural Resources, Agriculture, and Environmental Science*.

Corresponding author: The first author is the corresponding author for this submission. David's phone number is (202) 767-9006 and FAX number is (202) 767-3172.

Suggested running head: "Cloud Classification Using Error-Correcting Output Codes"

* Available as NCARAI Technical Note AIC-96-024

Abstract

Novel artificial intelligence methods are used to classify 16x16 pixel regions (obtained from Advanced Very High Resolution Radiometer (AVHRR) images) in terms of cloud type (e.g., stratus, cumulus, etc.). We previously reported that intelligent feature selection methods, combined with nearest neighbor classifiers, can dramatically improve classification accuracy on this task. Our subsequent analyses of the confusion matrices revealed that a small number of confusable classes (e.g., cirrus and cirrostratus) dominated the classification errors. We conjectured that, if the class labels in the data were re-represented so that these cloud classes are more easily distinguished, then additional accuracy gains might result. We explored this hypothesis by replacing each class label with a set of *error-correcting output codes*, a general technique applicable to any classification algorithm for tasks with at least three classes. Our initial results are promising; error correcting codes significantly increased classification accuracy compared with using standard representations for class labels. To our knowledge, this is the first successful integration of k -nearest neighbor classifiers and error-correcting output codes (i.e., where k is, effectively, small). One conclusion is that environmental scientists should always select, for their classification tasks, a classifier that reduces *both* variance *and* learning bias.

1 Introduction

Several types of algorithms have been evaluated for their utility on various cloud classification tasks, including neural networks (Welch et al. 1992), Bayesian approaches (Bankert 1994), and k -nearest neighbor algorithms (Aha and Bankert 1996). A frequent goal of classification algorithms in the context of environmental science applications is to maximize accuracy when predicting classifications for previously unseen query *samples*. Thus, several innovative methods have been evaluated for their ability to enhance accuracy (e.g., clustering hierarchical neural networks (Welch 1996)).

One general way to increase accuracy involves re-representing samples so that their classes can be more easily distinguished. For example, principle components analysis can be used to re-represent numeric features used to describe samples; new features are combinations of the old ones, carefully selected to increase the distance between samples of different classes. Several similar methods, for use with either numeric or symbolic features, have been developed to automatically modify the representations of samples (e.g., Mohri and Tanaka 1994, Rendell and Seshu 1990) with the intent of increasing accuracy. This topic is often referred to as *constructive induction* by machine learning researchers (Birnbbaum and Collins 1991).

Although these methods re-represent samples, they do not re-represent the class labels associated with them. Recently, Dietterich and Bakiri (1991, 1995) introduced a technique that re-represents samples' class labels using *error-correcting output codes* (ECOCs). Like successful constructive induction techniques, ECOCs distinguish samples in different classes by making the class encodings themselves more dissimilar. Unlike constructive induction techniques, ECOCs do not require a carefully constrained search through the space of feature combinations, and thus are easier to use. Dietterich and Bakiri showed that, for some *multiclass* (i.e., more than two classes) classification tasks, ECOCs can significantly increase classification accuracy for both decision tree induction algorithms and neural networks trained using the error backpropagation algorithm (Rumelhart et al. 1986), although with substantially greater computational costs. These two

algorithms both generate *nonlocal* classifiers, which generate class predictions using all information in the set of training samples rather than only information local to the query sample. Kong and Dietterich (1995) explained why ECOCs work well for nonlocal learning algorithms: they reduce the variance of learning algorithms and correct for errors caused by learning biases. Thus, we surmised that they might work well for our cloud classification application.

However, we have been using *local* algorithms (i.e., variants of the nearest neighbor¹ classifier) for our cloud classification task, which generate class predictions by using only local information (i.e., one or more most similar samples) to the query. This causes a conflict: ECOCs usually do not work well in conjunction with local classifiers because local learning algorithms cannot correct errors caused by learning biases (i.e., their classification errors are correlated) (Kong and Dietterich 1995). We hypothesized that selecting different feature subsets for each bit in an ECOC code would allow ECOCs to perform well with a k -nearest neighbor classifier; this prevents errors across output bits from being correlated because different sets of k neighbors will be nearest for different output bits. This paper describes an empirical evaluation for this hypothesis.

The rest of this paper describes our application of error-correcting output code techniques to a cloud classification task. Section 2 describes the cloud data set we used and summarizes our previous findings. Section 3 then introduces the subject of error-correcting output codes, while Section 4 details the specific ECOC technique we applied. Our initial evaluation of this approach is summarized in Section 5. Section 6 discusses the implications of our findings for environmental scientists interested in applying AI techniques to classification tasks. We conclude and describe future research issues in Section 7.

2 Task Description and Background

Our work was motivated by research conducted at the Marine Meteorology Division (MMD) of the Naval Research Laboratory. One of their objectives is to create shipboard decision aid tools for assisting US Navy personnel in various meteorological analysis and prediction tasks. Tools developed or being developed at MMD that use AI include AESOP (an expert system for shipboard obscuration prediction (Peak and Tag 1989)), SIAMES (a satellite image analysis meteorological expert system (Fett et al. 1997)), ExperCAT (a clear air turbulence expert system (Ellrod et al. 1993)), and MEDEX (an expert system for forecasting Mediterranean gale force winds (Kuciauskus et al. 1996)), as well as classifiers of cloud types (Bankert 1994) and cloud systems (Bankert and Tag 1996) in satellite imagery.

With satellite imagery being a primary (and sometimes only) source of observational data for remote maritime regions, automated identification of cloud types within those images would be a useful tool for the operational forecaster. Because of the number of years required to develop satellite cloud analysis expertise, Bankert’s motivation was to provide navy personnel with automated cloud type classification similar to that displayed by the experts. Bankert (1994) describes two steps towards this goal: development of (1) an expertly-labeled cloud database, and (2) a cloud classification procedure that is trained using this database. Based on the work of Crosiar (1993), Bankert assimilated a data set containing 1633 samples (16x16 pixel areas) of

¹A nearest neighbor classifier (Dasarathy 1991) generates a class prediction for an unlabeled query q by using a real-valued distance function on all samples; it computes the query’s distance between q and all stored training samples, retrieving the stored sample x_i whose distance is smallest. The class of x_i is returned as q ’s predicted class.

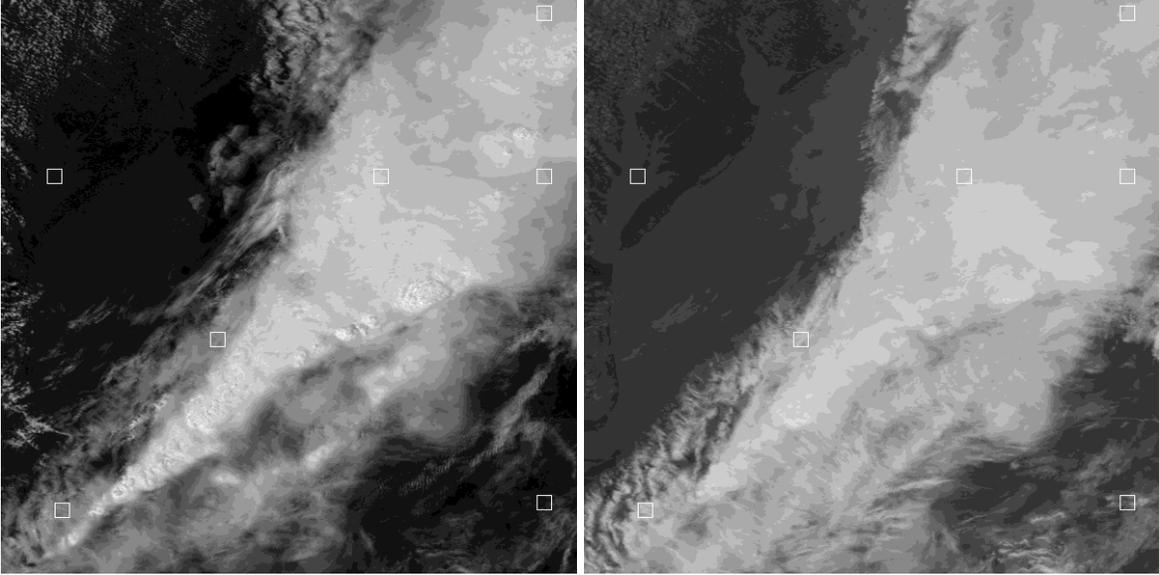


Figure 1: An AVHRR image with sample regions marked (visible channel, left; infrared channel, right).

cloud types taken from the National Oceanic and Atmospheric Administration (NOAA) Advanced Very High Resolution Radiometer (AVHRR) sensor aboard polar-orbiting satellites. Two channels of an AVHRR image with sample regions (16x16 pixel) marked by boxes is shown in Figure 1. Images were taken from various maritime regions throughout the Northern Hemisphere during a one-year period beginning in October, 1988. Adding to an initial collection of 610 samples labeled by experts at the Naval Postgraduate School, four experts independently labeled samples (Crosiar 1993) as one of nine cloud types, “clear” (see Table 1), or mixed clouds using image-displaying software.² Bankert then selected the subset of these 3625 cloud type samples for which there was a majority agreement (i.e., at least three of the four experts labelled it with the same class), and discarded the remaining samples. The experts had a majority agreement on only 2151 of the 3625 (59%) sample regions, which indicates that this task is difficult even for experts. He further decreased this number from 2151 to 1023 so as to reduce the number of samples from the “clear” class and delete the samples from the mixed clouds class. Combined with the 610 NPS samples, a data set of 1633 samples was generated.

To represent each of the 1633 samples for training and testing of the cloud type classifier, features were computed from the calibrated and scaled AVHRR data. Ideally, a complete domain theory would dictate which features to use to encode each image. Unfortunately, no such domain theory exists. Bankert, instead, encoded each image using a variety of numeric functions, yielding a set of 204 continuous spectral (e.g., maximum, minimum, mean pixel values), textural (i.e., spatial distribution of pixel values, including contrast and homogeneity), physical (e.g., cloud top temperature), and geographic (i.e., latitude) features.

There was little information on how, or even whether, these features relate to the ten classes. They were primarily chosen based on educated guesses and experience with similar applications.

²Experts labelled a sample as a particular cloud type only if they judged that at least 75% of the sample region corresponded to that cloud type (50% for stratocumulus).

Table 1: The ten cloud type classes in the cloud classification data set.

| Name | Code |
|---------------|------|
| Cirrus | Ci |
| Cirrocumulus | Cc |
| Cirrostratus | Cs |
| Altostratus | As |
| Nimbostratus | Ns |
| Stratocumulus | Sc |
| Stratus | St |
| Cumulus | Cu |
| Cumulonimbus | Cb |
| Clear | Clr |

Therefore, Bankert predicted that eliminating any irrelevant or redundant features would improve the classifier’s performance. Towards this goal, he used a *feature selection algorithm* (FSA) to preprocess the training data. Given a (usually large) set of features, an FSA searches for subsets that improve performance (e.g., classification accuracy) on a given task (Fu 1968). Since the space of feature subsets (i.e., the power set of all features) for this task is huge ($2^{204} \approx 2.6 \times 10^{61}$), Bankert (1994) selected a computationally inexpensive FSA named *forward sequential selection* (FSS) (Devijver and Kittler 1982) to search for feature subsets. It starts by locating the singleton subset of features that has highest accuracy and then adds the feature that maximizes accuracy. This process iterates with additional features until no accuracy improvements occur during an iteration. Bankert also selected an inexpensive function, the *Bhattacharya class separability index measure*, to evaluate and rank feature subsets. It ranks subsets based on their ability to cluster samples by their class. FSS, a deterministic algorithm in this context, selected only 15 of the 204 features. Bankert tested both the entire feature set and the selected subset on this data set using a probabilistic neural network (PNN) classifier (Specht 1990). PNN was chosen because it performed well on similar tasks (Welch et al. 1992). It uses a standard three-layer network of nodes and a Bayesian classification strategy to select the class with the highest value of the a posteriori class probability density function. The leave-one-out cross validation (LOOCV) accuracy³ was 75.3% for the complete feature set; this accuracy improved to 79.8% when using only the selected 15 features. Using only the 15 selected features also greatly reduced the variance of the individual class’s accuracy differences, which can be just as important as the overall accuracy when used operationally.

Although this FSA dramatically reduced the number of features used, reduced processing times correspondingly, and increased classification accuracy by 4.5% (i.e., compared with using all 204 features), it is not obvious whether enhanced performance can be obtained by using more sophisticated FSAs. Thus, we evaluated (Aha and Bankert 1994, 1996) a novel beam-search extension for sequential selection on this cloud classification task, coupled with a different evaluation function for feature subsets. Our extension, named *BEAM*, is a nondeterministic

³LOOCV is a process that yields the percentage of samples in a training set Tr whose class is correctly predicted when the classification for each $x \in Tr$ is obtained by classifying it using only the other training samples, namely $Tr - \{x\}$.

algorithm that maintains a queue (size n) of feature subsets, ordered by nonincreasing 10-fold classification accuracy. BEAM begins by evaluating m randomly-selected subsets of size s , retaining the best one in its queue. Assuming that FSS is used to select features, it then stochastically extracts a queue element and adds a randomly-selected feature (i.e., BEAM uses constraints to avoid creating previously-evaluated feature subsets). The newly constructed subset is evaluated and, if its accuracy is higher than at least one of the queue’s subsets, the queue is updated to include that subset. This select-and-evaluate process iterates until none of the remaining subsets can improve on the “best” subset found. The subset with the highest classification accuracy is chosen.

We also changed the way in which feature subsets were evaluated. Rather than use a clustering algorithm (e.g., the Bhattacharya index measure), we instead used the classifier itself to evaluate feature subsets (John et al. 1994). In this way, the classifier can provide feedback to the feature subset evaluation routine, and this practice can prevent a mismatch in the preference rankings between the subset evaluator and the classifier. That is, if we used any algorithm other than the classifier to evaluate feature subsets, then it might rank a feature subset differently than would the classifier. By using the same algorithm for both tasks, we avoid this conflict.

We used a variant of the nearest neighbor classifier, named IB1 (Aha et al. 1991), rather than PNN to evaluate feature subsets because, unlike PNN (which has a user-defined parameter) or backpropagation (which has many tunable parameters), IB1 does not require users or automated strategies to set parameter values. Given a query q and training set X , IB1 computes the *similarity*(q, x) for each $x \in X$, determines which stored case x is most similar to q , and then yields the class of x as its prediction for q ’s class. The definition of similarity we used, assuming each case is defined by a set F of features, is

$$\text{similarity}(q, x) = -\text{distance}(q, x) \tag{1}$$

$$\text{distance}(q, x) = \sqrt{\sum_{i \in F} (q_i - x_i)^2} \tag{2}$$

Nearest neighbor algorithms typically have a parameter k that determines how many of the nearest neighbors are used to predict classifications. In this study, we set $k = 1$. In Section 4, we will use a different variant of IB1 where, effectively, $k > 1$.

IB1’s 10-fold accuracy using all 204 features is 72.6%, which is similar to PNN’s accuracy. BEAM, using FSS to select feature subsets, located a subset of only eleven features that increase IB1’s accuracy to 87.0%. This accuracy was the highest we found while working with this data set (Bankert and Aha 1996).⁴

An analysis of the remaining errors lead us to question whether further improvements were possible. We found that three of the 45 pairs of classes were responsible for 42.3% of the (remaining) classification errors. We hypothesized that, if the samples in these classes could somehow be more easily distinguished, then classification accuracy will increase. This leads us to investigate whether error-correcting output codes could be used, in conjunction with our feature selection results, for this purpose. We introduce and describe how to use ECOCs in classification tasks in Section 3.

⁴We reported similar results for a cloud system (e.g., fronts, tropical cyclones) classification task (Aha and Bankert 1996), where a variant of BEAM located a feature subset that increased LOOCV from 62.3% to 95.7% while reducing the set of features from 98 to 16.

Table 2: Output representations for a hypothetical set of three classes

| Class Name | Output Representation | | |
|------------|-----------------------|---------------|-------------------------|
| | Monolithic | Distributed | |
| | | One-Per-Class | Error-Correcting (ECOC) |
| Earthling | 1 | 100 | 00000 |
| Martian | 2 | 010 | 11100 |
| American | 3 | 001 | 00111 |

3 Error-Correcting Output Codes

Multiclass classification tasks are classification tasks that involve at least three classes. For these tasks, error-correcting output codes (ECOCs) can help distinguish class labels by making them more dissimilar, which can increase classification accuracy. This section introduces ECOCs in the context of methods for solving multiclass tasks, and summarizes the motivation for using them.

At least three general strategies exist for designing classifiers to solve multiclass classification tasks. Table 2 exemplifies the three types of class label representations we describe here, and which we empirically compare in Section 5.

3.1 Monolithic output representations

The first approach, which we call the *monolithic* strategy, uses a unique monolithic encoding for each class label.⁵ For example, this corresponds to using the label “1” for the *Earthling* class in Table 2. Learning algorithms that use this approach induce a single concept description that distinguishes all class boundaries. For example, C4.5 (Quinlan 1993a) uses this approach; given a training set, C4.5 induces a decision tree where classification judgements are made at the leaves. The class prediction of a leaf is (usually) the majority class of all training samples at that leaf, where each class is represented by a unique symbolic label.

3.2 Distributed output representations

One distinguishing, and limiting, capability of monolithic output representations is that *there is no useful distance measure defined on monolithic class labels*. That is, the monolithic labels for two samples are either identical or different; no real-valued similarity function exists on monolithic class labels that can express that a new class label is more similar to one than some other existing class label.

In contrast, *distributed* output representations define a similarity function on class labels, and can thus define a distance function on class labels. Distributed codes represent each class label with a unique set of values called a *codeword*, typically in the form of a bit string. For example, in the

⁵Dietterich and Bakiri (1991, 1995) refer to this as the *direct multiclass* representation because only a single decision tree is required for classification decisions.

first entry of Table 2, this corresponds to using either the string “100” or “00000” to represent cases in the *Earthling* class. Each bit in the output string corresponds to a different partition of the classes and, thus, a different learning task. Given a query, classifiers with distributed output representations generate predictions for each output bit, and the class whose codeword has smallest Hamming distance (i.e., number of mismatching bit values) from the query’s prediction is predicted as the query’s class. We describe two examples of distributed output representations: *one-per-class* and *error-correcting*.

3.2.1 One-per-class output representations

The *one-per-class*⁶ encoding is a simple distributed output representation where there is a one-to-one correspondence between output bits and classes. That is, it involves learning a separate binary function for each class. For example, this corresponds to using the bit string “100” for cases in the *Earthling* class in the first entry of Table 2. For many learning algorithms, such as those that induce decision tree (Quinlan 1993a) or *k*-nearest neighbor classifiers (Aha 1989), one-per-class encoding is done by generating one concept description per class. For example, if C4.5 is used and there are n classes, samples for each tree T_i ($1 \leq i \leq n$) are labelled *positive* if they are members of class c_i and *negative* otherwise, and each tree T_i generates a prediction for how confident it is that the sample is member of class c_i . This strategy classifies a new sample according to the class of the tree with the highest confidence. For backpropagation-trained networks (Rumelhart et al. 1986), this representation instead uses n output nodes (i.e., one per class), and the classifier predicts the class corresponding to the output node with highest activation (i.e., a *winner-take-all* network). Thus, all samples in class c_i have a (binary) encoding of 1 for output node o_i and 0 for all other output nodes.

Hamming distance can be used to define the distance between codewords. Unfortunately, this representation is severely limited; it defines the Hamming distance between all codewords to be exactly 2, and only n possible encodings (i.e., one of the class’s codewords) can be generated in a classification attempt. This representation leaves little room for classification error, such as when one or more output bits are incorrectly predicted.

3.2.2 Error-correcting output representations

Distributed output codes do not need to have a one-to-one correspondence constraint between output bits and classes. For example, they can instead assign a set of $\log n$ nodes to represent the n classes, or use some other pre-determined number of bits to define codeword length (Sejnowski and Rosenberg 1987). Class encodings can be carefully designed with specific distinguishing properties.

ECOCs are examples of this approach: they are not restricted to having exactly n output bits and their codewords can have multiple output bits that are “on” (corresponding to values of 1). This lack of restriction supplies ECOC strategies with great freedom in how class encodings are assigned. Furthermore, each class’s encoding differs, in Hamming distance, from all other class encodings by (at least) a pre-determined amount (e.g., the ECOCs in Table 2, such as “11100” for

⁶Could also be called *one-bit-per-class* representations because each bit represents a different binary learning task.

the *Martian* class entry, have a minimum Hamming distance of 3). This encoding gives it an *error-correcting* capability. For example, suppose that the minimum Hamming distance between all classes is five. This example corresponds to an error-correcting capability of two. That is, if no more than two of the predicted output bit values for a given sample are incorrect, then the correct class’s encoding is still the “closest” to the predicted encoding, and will be predicted as the class for the query. More generally, class codewords that are separated by a minimum Hamming distance of d have an error-correcting capability of $\lfloor (d - 1)/2 \rfloor$. In contrast, the monolithic and one-per-class output representations have no error-correcting capabilities, even though the latter imposes a Hamming distance of 2 between each class’s codeword (i.e., if one output bit is incorrectly predicted, then more than one codeword may have the smallest Hamming distance, and the classification prediction is unclear).

Dietterich and Bakiri (1991, 1995) introduced the use of ECOCs for multiclass classification tasks. They detailed experiments showing that, versus other multiclass approaches, ECOCs can improve the classification accuracy of C4.5 (Quinlan 1993a) and a multilayer connectionist network trained by backpropagation (Rumelhart et al. 1986) on a set of multiclass tasks, although ECOCs are slower because they involve learning a function for each output bit (i.e., as opposed to learning a single, albeit multiclass, function). Dietterich and Bakiri also explore the robustness of their techniques and describe methods for designing error-correcting codes. They advocate two principles when designing ECOC codewords:

1. **row separation:** Each class’s codeword should be separated by a pre-determined (minimal) Hamming distance from the other codewords.
2. **column separation:** The function defined by each output bit should be uncorrelated with the other output nodes’ functions. Highly correlated bits are not useful for classification because they enforce similar decision boundaries on the data. This also entails these functions should not be correlated with the *complement* of other output bit functions.

Row separation is clearly crucial for distinguishing the class’s codewords. Column separation is crucial to ensure that the *errors* of the output bit predictions are uncorrelated.

Kong and Dietterich (1995) analyzed why ECOCs tend to increase classification accuracy on multiclass classification tasks, why column separation is crucial for designing effective error-correcting codes, and why learning algorithms that generate *nonlocal* classifiers, including decision trees and neural networks, should generally benefit from using ECOCs. In particular, they showed that ECOCs work well because they they reduce the number of errors caused by both the *variance* and *bias* of the learning algorithm.

Variance errors result from random variation and noise in the training set and from any random behaviors of the learning algorithm itself. These errors can be reduced by averaging the contributions of multiple predictions during a single prediction task, such as by voting among multiple runs of the same algorithm (Perrone and Cooper 1993, Breiman 1994). This also occurs when using ECOCs because a vote (among predictions for the different output bits) takes place when selecting the nearest codeword, as computed by predicting the class whose codeword has the smallest Hamming distance.

Bias errors instead refer to an algorithm’s systematic errors. These errors can also be reduced by a form of voting, but only when the individual predictions are uncorrelated. This can often be

accomplished by using *different* algorithms applied to the same learning task (Zhang et al. 1992, Quinlan 1993b). Alternatively, the same algorithm can be used multiple times, so long as the vote is on different subproblems that cause the algorithm to generate different bias errors. For example, C4.5 performed well when extended with ECOCs because each output bit function creates different partitions on the classes. Because these partitions encode different decision boundaries, C4.5 will yield different bias errors on each bit. This is the motivation for designing error-correcting codes with good column separation.

More generally, Kong and Dietterich (1995) predict that “nonlocal” learning algorithms (i.e., those that induce compact classifiers) such as backpropagation and C4.5 should benefit from using ECOCs, but not local learning algorithms (i.e., those that generate predictions based on information near the query sample), including nearest-neighbor classifiers. Their reasoning is that, when using only local information, the bias errors in different output bits will be correlated, which will prevent the ECOCs from reducing bias errors. Empirical evidence exists to support this claim (Wettschereck and Dietterich 1992).

The rest of this paper describes and evaluates a method for decorrelating the errors made by nearest-neighbor classifiers so that they can benefit from using ECOCs. Our approach relies on using feature selection for each output bit, so that similarity will be computed on different subsets of features for each bit. This will cause different stored samples to be retrieved for different output bits, and their class predictions should be independent. Thus, our strategy, while still depending on using local information during classification predictions, uses different, yet local, information for each output bit. The following section describes our algorithm, while Section 5 details its application to the cloud classification task.

4 Local Learning with Error-Correcting Output Codes

This section describes the AI approach we tested on the cloud classification task. A feature selection approach is integrated with error-correcting output codes for a k -nearest neighbor classifier. Because ECOCs require that the errors for each of the output bits be uncorrelated, we modified IB1 to use different features when computing distances for each of the ECOC output bits. We also varied IB1 in other ways for use with ECOCs.

More specifically, we used a variant of IB1 where, effectively, more than one nearest neighbor was used to generate classification predictions. But rather than optimizing for k during each feature subset evaluation, we instead summed the similarity of the query q to all stored features, weighting their similarity according to an exponentially decreasing function of their distance. That is, for a given output bit j , we defined

$$\text{similarity}(q, x, j) = \exp -s \text{ distance}(q, x, j) \tag{3}$$

where

$$\text{distance}(q, x, j) = \sqrt{\sum_{i \in F_j} (q_i - x_i)^2} \tag{4}$$

F_j is the set of features selected for bit j , and s is a scaling constant (set to 10 in our experiments) that determines the slope of the exponential curve.⁷ Then, for each class $c \in C$, we

⁷We found similar results when we set $s = 100$, making it behave more like 1-nearest neighbor.

k -NN-ECOC($Tr, Te, Cl, p(), h$)

Tr : Training set
 Te : Test (evaluation) set
 Cl : Set of classes
 $p()$: Class partition function
 h : Requested inter-class Hamming distance for codewords
 M : Confusion matrix
 O : Set of output bit functions (i.e., partitions on the classes)
 Co : Set of ECOC class codewords
 F : Matrix of feature subsets selected by FSS (one subset per output bit)
 p : Classification prediction vector
 e_i : Test sample i
 c : Class predicted by k -NN-ECOC for e_i
count: Number of correct class predictions (initially 0)

$M := \text{create_confusion_matrix}(\text{IB1}(), Tr)$
 $O := \textbf{create_output_bits}(Cl, p(), h, M)$
 $Co := \text{create_codewords}(O)$
FOREACH output bit $o_i \in O$ **DO**:
 $F_i := \text{FSS}(k\text{-NN}(), Tr, Co)$
FOREACH $e_i \in Te$ **DO**:
 FOR EACH output bit $o_j \in O$ **DO**:
 $p_j := \text{IB1}(Tr, e_i, o_j)$
 IF $\text{class}_{e_i} = \text{minimum_Hamming_class}(p, Co)$
 THEN count = count + 1
OUTPUT count/ $|Te|$

Figure 2: k -NN-ECOC: An extension of k -NN that uses error-correcting output codes.

summed the similarity of q to all training cases in c . Finally, the class with highest summed similarity was the class predicted for q .

Figure 2 lists the pseudocode for our approach, which we name k -NN-ECOC (i.e., k -nearest neighbor extended with ECOCs). **Boldfaced** function names are detailed in later figures.

k -NN-ECOC begins by generating a confusion matrix M , obtained by applying IB1 to the training set using LOOCV. This matrix is then used to help generate the ECOC codewords Co , for the classes Cl , so that their minimum Hamming distance is h . Function $p()$ achieves this goal by selecting the partition of classes to use for each output bit. k -NN-ECOC next computes the set of features to use when predicting bit values for each output bit. We again chose FSS as the FSA, and we chose the version of k -NN in Equation 4 to rank feature subsets (i.e., according to LOOCV on the training sets).⁸ Then, for each test sample e_i , k -NN-ECOC predicts a value for each output bit and compares the predicted output string with each codeword, yielding the most

⁸This definition of k -NN has two advantages: it is nonparametric, which means that we do not need to tune parameters in each feature subset evaluation, and it is as cheap to execute as IB1.

```

create_output_bits( $Cl, p(), h, M$ )
-----
 $Cl$ : Set of classes
 $p()$ : Class partition function
 $h$ : Requested inter-class Hamming distance for codewords
 $M$ : Confusion matrix
 $o$ : A partition on the set of classes, initially empty
 $O$ : A set of output bit partitions
-----
REPEAT
   $o := \mathbf{create\_class\_partition}(Cl, p(), M)$ 
  IF not_redundant( $o, O$ )
  THEN  $O := O + \{o\}$ 
UNTIL distinguished( $Cl, O, h$ )
RETURN Codes

```

Figure 3: Procedure for creating the output bit functions.

similar codeword's class as its prediction for e_i 's class. A count is maintained on the number of correct predictions and the average classification accuracy is output.

The function *create_output_bits*, which creates the class partitions corresponding to the output bit functions, is detailed in Figure 3. Each output bit is a binary partition function on the set of classes (Cl). Thus, *create_output_bits* builds a set of partitions, one per output bit, until the set of partitions distinguishes each pair of classes according to at least the requested Hamming distance h . Bits that are redundant with previously-created bits in terms of how they partition the classes are discarded.

Each partition (output bit) is created by *create_class_partition*, which is detailed in Figure 4. For a given output bit o_i , this procedure randomly selects two classes from Cl to initialize two subsets of classes. It then iteratively uses the partition function $p()$ to determine, for each of the remaining classes in Cl , which subset they will join. Afterwards, the codeword for each class in subset S_j is assigned value j for o_i .

Function *create_output_bits* constrains the partition function p to generate a different partition of the classes for each output bit. We describe experiments with three different functions for $p()$ in Section 5, motivated by our interest in determining whether the confusion matrix can be profitably exploited to bias the bit-creation process. The first (and simplest) definition ignores the confusion matrix and, instead, randomly assigns the remaining classes to each subset S_i . This corresponds to a top-level procedure call of

$$k\text{-NN-ECOC}(Tr, Te, Cl, \text{random_partition}(), h)$$

where *random_partition* is defined in Figure 5.

The *random_partition* function does not use information concerning how c relates to the classes already in subsets S_0 and S_1 . The other two partition functions bias the code-generation process by using the information in the confusion matrix M . The second definition of $p()$ that we will

```

create_class_partition( $Cl, p(), M$ )
-----
 $Cl$ : Set of classes
 $p()$ : Class partition function
 $M$ : Confusion matrix
 $S_0, S_1$ : Disjoint subsets of  $Cl$ 
 $c$ : A class in  $Cl$ 
-----
 $S_0 := \{ \text{random}(Cl) \}$ 
 $S_1 := \{ \text{random}(Cl - S_0) \}$ 
 $Cl := Cl - S_0 - S_1$ 
FOREACH  $c \in Cl$  DO:
    IF  $\mathbf{p}(c, M, S_0, S_1)$ 
    THEN  $S_0 := S_0 \cup \{c\}$ 
    ELSE  $S_1 := S_1 \cup \{c\}$ 
RETURN  $\{S_0, S_1\}$ 

```

Figure 4: Procedure for generating a class partition corresponding to an output bit.

```

random_partition( $c, M, S_0, S_1$ )
-----
 $c$ : A class, in neither  $S_0$  nor  $S_1$ 
 $M$ : Confusion matrix (not used here)
 $S_0, S_1$ : Disjoint subsets of  $C$  (not used here)
-----
IF  $\text{random}([0, 1]) \geq 0.5$ 
THEN RETURN TRUE
ELSE RETURN FALSE

```

Figure 5: Random partition function for partitioning classes.

evaluate is the `min_confusion` function, displayed in Figure 6. It computes the mean class confusion of the given class with the classes already in the two subsets, and stochastically favors assigning c to the subset with lower mean class confusion. Thus, classes with lower confusion are grouped together.

Our final partition function, `max_confusion` (Figure 7), is the reverse of `min_confusion` and tries to group maximally confusable classes together.

It is unclear how these three partition functions will compare. Our expectation is that the individual output bit functions of `min_confusion` will be difficult to learn because they distinguish highly confusable classes. In contrast, the bits created by `max_confusion` should be simpler to learn, but will not be as informative. However, we have not found any comparisons of random partition functions with those that bias the bit-creation process using class confusion tables.

```

min_confusion( $c, M, S_0, S_1$ )
-----
 $c$ : A class, in neither  $S_0$  nor  $S_1$ 
 $M$ : Confusion matrix
 $S_0, S_1$ : Disjoint subsets of  $Cl$ 
-----
 $C_0 := \text{mean\_confusion}(c, S_0)$ 
 $C_1 := \text{mean\_confusion}(c, S_1)$ 
IF  $\text{random}([0, 1]) < \frac{C_0}{C_0 + C_1}$ 
THEN RETURN TRUE
ELSE RETURN FALSE

```

Figure 6: Min-confusion partition function for partitioning classes.

```

max_confusion( $c, M, S_0, S_1$ )
-----
 $c$ : A class, in neither  $S_0$  nor  $S_1$ 
 $M$ : Confusion matrix
 $S_0, S_1$ : Disjoint subsets of  $Cl$ 
-----
 $C_0 := \text{mean\_confusion}(c, S_0)$ 
 $C_1 := \text{mean\_confusion}(c, S_1)$ 
IF  $\text{random}([0, 1]) < \frac{C_0}{C_0 + C_1}$ 
THEN RETURN FALSE
ELSE RETURN TRUE

```

Figure 7: Max-confusion partition function for partitioning classes.

The following section describes our experiments with applying error-correcting codes and other multiclass methods for the cloud classification task described in Section 2.

5 Evaluation

This section describes three experiments using ECOCs for our cloud classification task. First, we compared the ECOC approach versus alternative multiclass classification approaches, where the codewords were randomly generated (Figure 5) according to a given minimum Hamming distance constraint. Second, we varied the minimum Hamming distance between classes, and predicted that larger error-correcting capabilities would also have a positive effect on accuracy. Finally, we examined whether more sophisticated methods for generating error-correcting codewords (Figures 6 and 7) could further decrease classification error. Our results support our first two hypotheses, but not our third.

5.1 Empirical Methodology

All of the experiments followed the same empirical methodology. Due to the inherent complexity of this task (i.e., an expensive feature selection process repeated for each of several output bits), we did not work with the entire cloud classification data set, but with only subsets of it. That is, we created three subsets of size 100, 250, and 500 samples, respectively. We then ran each algorithm ten times on each data set, each time randomly splitting the data set into 70% training and 30% testing partitions. All algorithms were trained and tested on identical training and test sets. For the algorithms using ECOCs, the number of bits in the output code was determined by the function `create_output_bits` (Figure 3) using one of the three functions for the partition function $p()$ described in Section 3.2.2 (i.e., `random_partition`, `min_confusion`, or `max_confusion`). Features were selected for each output bit; feature subsets were selected by FSS and were evaluated by k -NN (Equation 4) on the complete training set.

5.2 Initial Comparison

In our first experiment, we compared the three approaches for solving multi-classification tasks described in Section 3, namely the monolithic, one-per-class, and error-correcting approaches. We used a single class label for the monolithic strategy that can take on one of ten discrete values corresponding to the ten cloud types in our data set. For the one-per-class strategy, we converted monolithic class labels to a set of ten output class bits, and ran k -NN once per bit. That is, the one-per-class strategy simply used the k -NN-ECOC (Figure 2) system where `create_output_bits` was modified to create one bit per class and the `create_codewords` created corresponding one-per-class codewords. k -NN-ECOC was also used for the ECOC strategy using the `random_partition` function for $p()$. However, in this experiment, we set h (Figure 2), to 1, thus requesting no error-correction capability. In response, `create_output_bits` distinguished the ten cloud type classes by generating ten codewords that were only five bits in length. The same codewords were used for each of the ten runs in this experiment.

We quickly found that the monolithic strategy performed poorly when feature selection was performed (i.e., its average accuracy for the subset of size 500 was only 21.0%). This poor performance occurred because FSS stopped after selecting only one feature in each run. Therefore, we abandoned feature selection when evaluating the monolithic strategy, which greatly improved its performance. We will refer to this strategy as *monolithic-all* (i.e., monolithic strategy using all features). In contrast, FSS performed adequately when using distributed output codes. For example, it retrieved, on average, 3.8 features when using `random_confusion` with Hamming distance 1 on the smallest data set size, and doubled this for the largest data set size (the average feature subset sizes were similar for other Hamming distance settings). Although this subset is small compared with the entire set of 204 features, it was adequate to yield good results.

Learning curves summarizing these results are shown in Figure 8. We generated learning curves rather than report a single classification result with the largest data set because they reveal more detailed behavior for distinguishing the algorithms' capabilities. We also display error bars representing ± 1 standard deviation in this figure, but only for the top-performing algorithm at each data subset size (showing all error bars yields a confusing graph). In this case, we see that the *monolithic-all* approach attains higher accuracy than one-per-class, and that the degenerate error-correcting encoding algorithm eventually surpasses both. According to a one-tailed t -test,

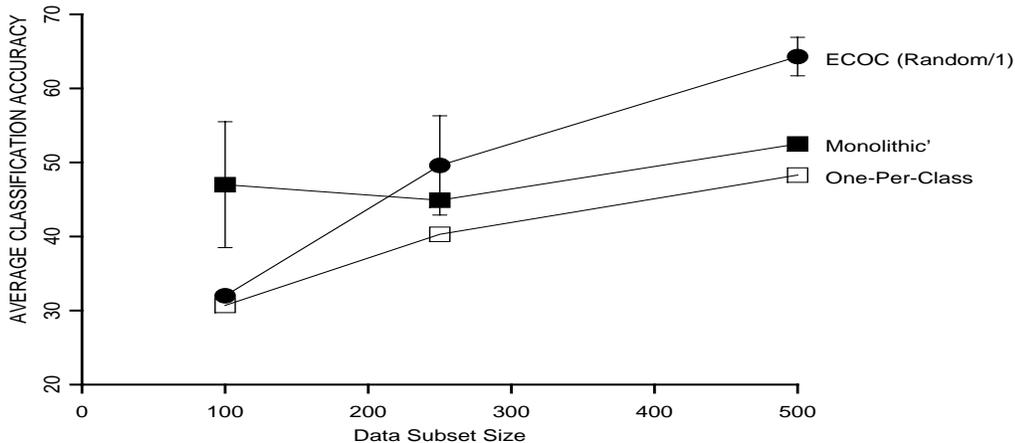


Figure 8: Comparing multi-classification approaches on the cloud classification task.

the higher accuracies recorded by this strategy for the data set of size 500 significantly differ from those recorded by the monolithic-all ($t = 2.7, p < 0.025$) and one-per-class ($t = 2.9, p < 0.01$) strategies. However, the benefit does not appear to exist for the smaller data set sizes for this type of distributed output codes. The question remains whether error-correcting codes (i.e., with inter-class Hamming distances of at least 3) can produce further benefits.

5.3 Varying the error-correction capability

In our second experiment, we again used the `random_partition` function (Figure 5), but this time we varied the requested minimum Hamming distance between codewords. In three runs, we set h to 1, 3, and 5, respectively. The results, shown in Figure 9, indicate that error-correcting codes (i.e., `Random/3` and `Random/5`) attain higher accuracies than when the distributed codes are not error-correcting (i.e., `Random/1`). For the 500-size data set, the differences between `Random/3` and `Random/1` are significant at the $p < 0.025$ level ($t = 2.4$), and the differences between `Random/5` and `Random/1` are even larger ($t = 2.7$). We also compared these size-500 results with the previous multiclass and one-per-class results. Their differences with the ECOC results were all significant at the $p < 0.01$ or $p < 0.005$ levels. Thus, the ECOC codes significantly increase classification accuracies on these subsets of the cloud classification data set.

5.4 Varying the partition function

In our final experiment, we varied the way in which the error-correcting codes were generated. In particular, we compared how the ECOC approach performs when $p()$ is assigned the `random_partition`, `min_confusion`, and `max_confusion` strategies described in Section 3.2.2. Whereas the `random_partition` function generates output bits corresponding to arbitrary partitions of the classes, the latter two instead generate bits that attempt to (respectively) maximize or minimize the Hamming distances of confusable classes.

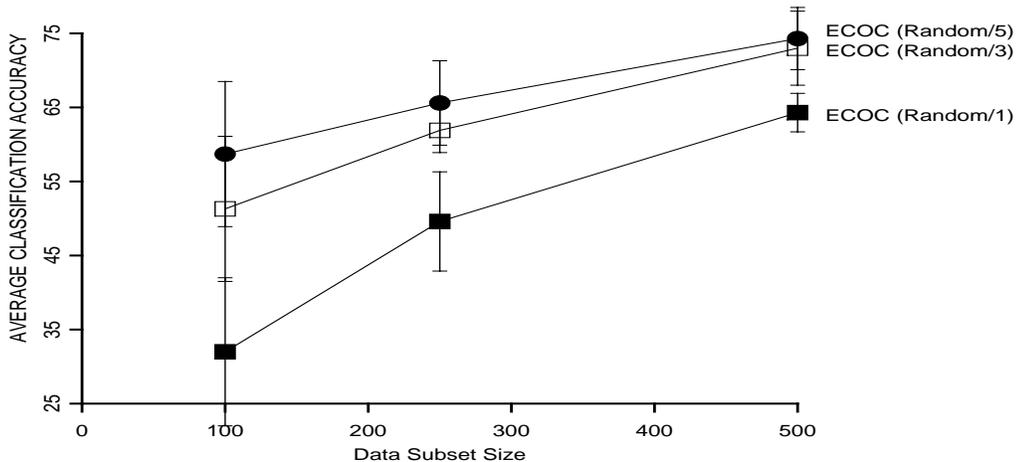


Figure 9: Comparing error-correcting codes versus distributed codes without error-correcting capabilities on the cloud classification task.

The latter two partition functions were chosen as foils to the `random_partition` function. We might expect `max_confusion` to perform well because it tends to separate highly confusable classes, and that `min_confusion` will perform poorly because it groups highly confusable classes. However, it is much more difficult to learn the individual output bit functions for `min_confusion` than `max_confusion` because it is harder to learn functions where highly confusable classes are in different halves of the partition. Thus, it is unclear how these methods will compare, and how they will compare with the `random_partition` function.

Table 3 summarizes the results of their comparison on the cloud classification data set. We see that `min_confusion` performs exceptionally well for the smallest-sized data set (i.e., for $h = 1$ and $h = 3$), but neither of the partition functions that exploit confusion matrix information outperforms the `random_partition` function for the larger data set sizes. Thus, while the error-correcting algorithms (i.e., distributed codes with $h > 1$) outperform the monolithic-all and one-per-class approaches, we found no strong benefit from these two simple methods for biasing the selection of class codewords.

5.5 Confusion matrices

As previously mentioned, one of the goals of this study was to determine whether ECOCs could reduce the number of errors among the most confusable classes. Our evidence confirms this hypothesis. Table 4 displays the confusion matrix for one run of k -NN (i.e., without feature selection, using the monolithic output representation). In this case, these results correspond to the first of the ten runs with 350 training and 150 test cases. Table 5 displays the confusion matrix for the same run when using ECOCs generated by `random_partition` ($h = 5$). In both cases, the numbers shown refer to the number of times, among the 150 test cases, where a test case according to the row was predicted to be in the class according to the column. In this particular run, k -NN’s accuracy was 54.7% while k -NN-ECOC’s accuracy was 70.0%.

Table 3: Average percentage accuracies and standard deviations of the multiclass approaches on the cloud classification task, where the minimum Hamming distance between codewords (H) and length of codewords (L) are shown for the distributed methods.

| Algorithm | H | L | Data Set Size | | |
|------------------|---|----|---------------|----------|----------|
| | | | 100 | 250 | 500 |
| Monolithic | 1 | 1 | 16.3±8.8 | 15.6±6.8 | 21.1±4.9 |
| Monolithic-all | 1 | 1 | 47.0±8.5 | 44.9±3.3 | 52.5±2.8 |
| One-per-class | 2 | 10 | 30.7±8.6 | 40.3±4.5 | 48.3±6.4 |
| Random_partition | 1 | 5 | 32.0±10.0 | 49.6±6.7 | 64.3±2.6 |
| | 3 | 21 | 51.3±9.8 | 61.9±3.0 | 73.0±5.0 |
| | 5 | 32 | 58.7±9.8 | 65.6±5.7 | 74.3±4.2 |
| Max_confusion | 1 | 6 | 34.0±6.8 | 51.2±4.6 | 62.5±6.0 |
| | 3 | 20 | 48.0±12.1 | 59.2±3.6 | 70.6±4.3 |
| | 5 | 29 | 54.0±10.7 | 64.8±4.6 | 73.9±4.5 |
| Min_confusion | 1 | 8 | 42.3±14.3 | 53.1±5.5 | 57.3±5.9 |
| | 3 | 19 | 59.7±8.8 | 59.6±4.9 | 69.5±5.2 |
| | 5 | 32 | 58.7±10.8 | 63.9±4.6 | 74.6±4.5 |

Table 4: Confusion matrix for one run of k -NN.

| Classes | Ci | Cc | Cs | As | Ns | Sc | St | Cu | Cb | Clr |
|---------|----|----|----|----|----|----|----|----|----|-----|
| Ci | 16 | 0 | 3 | 0 | 0 | 0 | 0 | 2 | 1 | 0 |
| Cc | 0 | 4 | 2 | 1 | 2 | 0 | 0 | 0 | 0 | 0 |
| Cs | 9 | 0 | 5 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| As | 1 | 0 | 1 | 10 | 0 | 4 | 4 | 0 | 0 | 0 |
| Ns | 0 | 1 | 0 | 0 | 2 | 0 | 0 | 0 | 1 | 0 |
| Sc | 0 | 0 | 0 | 2 | 0 | 15 | 4 | 1 | 0 | 0 |
| St | 0 | 0 | 0 | 2 | 0 | 2 | 9 | 0 | 0 | 0 |
| Cu | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 9 | 0 | 0 |
| Cb | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 12 | 0 |
| Clr | 0 | 0 | 0 | 0 | 0 | 1 | 16 | 0 | 0 | 0 |

The pair of classes most frequently confused by k -NN are Clr and St. This result is somewhat surprising in the sense that this confusion had not been seen in previous experiments, but considering the similarities in terms of texture and temperature, one can see how this misclassification could happen. This pair accounts for 16 of k -NN’s 68 errors. The next most frequently confused class pairing is, not surprisingly, Ci (Cirrus) and Cs (Cirrostratus), which accounts for another 12 errors by k -NN. Together they represent 41.2% of k -NN’s errors. Overall, the top six pairs of most frequently confused classes account for 51 (75.0%) of k -NN’s errors. k -NN-ECOC reduced this number to 25 (56% of the total error), eliminating all the errors of the most confusable pair of classes (St,Clr). Unfortunately, errors for the second most confusable pair, Ci and Cs, were not reduced. In future research, we plan to determine whether more other error

Table 5: Confusion matrix for one run of k -NN-ECOC (random_partition, $h = 5$).

| Classes | Ci | Cc | Cs | As | Ns | Sc | St | Cu | Cb | Clr |
|---------|----|----|----|----|----|----|----|----|----|-----|
| Ci | 12 | 2 | 5 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| Cc | 0 | 7 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Cs | 7 | 2 | 6 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| As | 1 | 0 | 0 | 13 | 0 | 2 | 3 | 0 | 1 | 0 |
| Ns | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 |
| Sc | 0 | 0 | 0 | 1 | 0 | 18 | 3 | 0 | 0 | 0 |
| St | 0 | 0 | 0 | 0 | 0 | 1 | 12 | 0 | 0 | 0 |
| Cu | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 5 | 0 | 0 |
| Cb | 0 | 0 | 2 | 1 | 0 | 1 | 0 | 0 | 12 | 0 |
| Clr | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 16 |

correcting codes, designed specifically for this pair of classes, can reduce their confusion rate.

These confusion matrices are typical of the ones we found while comparing these algorithms. While accuracy does not improve on all pairs of classes, the greatest accuracy increases typically occur with the most confusable pair of classes.

6 Discussion and Future Work

Although the max_confusion and min_confusion partition algorithms did not improve performance over random_partition on our task, performance improvements might result from other “smart” codeword selection strategies. For example, smart strategies could focus on distinguishing only the class pairs with highest confusions. However, these strategies might be elusive. While we could train an output bit o_i to focus specifically on a given pair of highly confusable classes c_j and $c_{j'}$ (i.e., by training them on samples drawn only from these two classes), it is unclear how o_i could be used during testing, when we do not know whether a test sample is among these two classes. Perhaps we could define an output bit $o_{i'}$ to distinguish samples in c_j or $c_{j'}$ from all others and then use $o_{i'}$ as a conditional filter before using o_i . However, this would greatly complicate the training and testing processes. Moreover, our preliminary attempts at using a cascading approach have not been promising (Bankert 1994).

While ECOCs can increase classification accuracy, they have two primary drawbacks. First, unlike the one-per-class approach, the output node functions no longer correspond to unique classes. Instead, output bits are functions that partition classes into (almost) arbitrary groupings. Thus, this limitation causes a loss of comprehensibility. Second, ECOCs increase computational complexity because they typically require far more output nodes than the one-per-class approach (e.g., $2x + 1$ output nodes are required for an error-correcting capability of x for (only) two classes). This can be expensive for some algorithms. For example, backpropagation’s training complexity is $O(mn^2)$, where m is the number of training samples and n is the maximum number of nodes at any layer (Bankman and Aha 1992). Thus, increases in the number of output nodes

can cause greater than linear increases in the training time. As an example from these experiments, the time required per run (on a Sun SparcStation Ultra 1) for data sets of size 500 was only about 20 seconds for monolithic-all, but the time increased to about five hours for runs with the largest number of output bits. The amount of time required was roughly a linearly increasing function of the number of output bits. For this investigation, we were willing to sacrifice comprehensibility and time for accuracy, but this will not be an appropriate sacrifice for many (e.g., time-critical) tasks.

As with other application areas, many environmental science tasks are posed as classification problems, in part because several robust classification tools exist. We have an important message for environmental scientists who are concerned about maximizing classification accuracy on their tasks: *select algorithms that are specifically designed to reduce both variance and bias errors*. As Kong and Dietterich (1995) explain, several algorithms reduce only errors caused by variance. These algorithms generate multiple hypotheses (e.g., by using different initial random weights in a neural network or different training sets to induce multiple decision trees or rule sets) and then vote among these hypotheses (Perrone 1994, Breiman 1994). They cannot reduce bias errors because the predictions among the multiple hypotheses are correlated. However, both variance and bias errors can be reduced by voting among multiple hypotheses produced by *different* learning algorithms applied to the same problem, assuming the bias errors differ among the different algorithms (Clemen 1989, Makridakis and Winkler 1983, Quinlan 1993b). Error correcting output codes also reduce variance and bias errors, but with a different form of voting (i.e., computing Hamming distance between the prediction and the codewords of each class). They also reduce bias by applying an algorithm multiple times, where each application is for a different learning task (i.e., output bit). Finally, local algorithms, such as the k -nearest neighbor classifier, require that different local information is selected when making predictions for each bit, which decorrelates the errors among the bit predictions.

In summary, environmental scientists should consider narrowing their choice of classifiers. Instead of using one that appeared to work well, but for unknown reasons, on some previous task, we instead encourage them to select (or design) an algorithm known to reduce both variance and bias errors. The question then becomes one of selecting which, from among these many algorithms, best matches their needs for other problem-solving aspects (e.g., training speed, space requirements).

7 Conclusion

We described an application of error correcting output codes (ECOCs) for a multiclass cloud classification task, where samples in the database were derived from AVHRR images. More precisely, we modified a nearest neighbor classifier by replacing each sample's class label with a set of "output" functions (bits) that each partition the set of classes differently. Furthermore, we used a simple sequential feature selection algorithm (FSS) to select a different set of features for each output function, implying that distances are computed differently for each bit. Our experiments showed that this method, which reduces both variance and bias errors, significantly outperforms other multiclass classification approaches on our cloud classification task. It performed especially well in increasing accuracy on the most confusable classes in the data set. However, using ECOCs can be more computationally expensive than using monolithic class

labels, and this cost increases with the number of bits used in the ECOC representation.

The primary contribution of this paper to environmental scientists is to provide additional evidence that learning algorithms which reduce both variance and bias errors are excellent choices for environmental science applications. The primary contribution for AI researchers is a useful algorithm for integrating k -nearest neighbor classifiers with ECOCs and its empirical evaluation on an interesting task. To our knowledge, this is the first successful integration of ECOCs with a local learning algorithm where, effectively, k is small (i.e., due to the high slope imposed by s in Equation 4). Selecting different features for each output bit is crucial to this integration's success.

Several avenues for future research remain. We plan to investigate the use of error-correcting output codes on other environmental science applications, and to also determine whether certain modifications of this approach are promising. This future work includes designing smarter partition functions for generating codewords, and using more sophisticated distributed error-correcting codes (e.g., with symbolic representations, of varying lengths).

Acknowledgements

Many thanks to professor Nick Flann of Utah State University, who generated the error-correcting codes used in this study. Thanks also to Francesco Ricci, John Grefenstette, Paul Tag, Jim Peak, and other colleagues at NRL/DC and NRL/Monterey for their continuing encouragement of and support for our joint research efforts.

References

- Aha, D. W. 1989. Incremental, instance-based learning of independent and graded concept descriptions. Pages 387–391 in: Proceedings, Sixth International Workshop on Machine Learning, Ithaca, New York, July 1989. Morgan Kaufmann, San Mateo, California.
- Aha, D. W., and R. L. Bankert. 1994. Feature selection for case-based classification of cloud types: An empirical comparison. Pages 106–112 in: Case-Based Reasoning: Papers from the 1994 Workshop (Technical Report WS-94-01), D. W. Aha, editor. AAAI Press, Menlo Park, California.
- Aha, D. W., and R. L. Bankert. 1996. A comparative evaluation of sequential feature selection algorithms. To appear in: Artificial Intelligence and Statistics V, D. Fisher and J.-H. Lenz, editors. Springer-Verlag, New York.
- Aha, D. W., D. Kibler, and M. K. Albert. 1991. Instance-based learning algorithms. *Machine Learning* 6: 37–66.
- Bankert, R. L. 1994. Cloud classification of AVHRR imagery in maritime regions using a probabilistic neural network. *Journal of Applied Meteorology* 33: 909–918.
- Bankert, R. L., and D. W. Aha. 1996. Improvement to a neural network cloud classifier. *Applied Meteorology* 35: 2036–2039.

- Bankert, R. L. and P. M. Tag. 1996. Automated extraction and identification of cloud systems in satellite imagery. Pages 373–376 in: Proceedings, Eighth Conference on Satellite Meteorology and Oceanography. American Meteorological Society, Atlanta, Georgia.
- Bankman, I. N., and D. W. Aha. 1992. Fast learning in feedforward neural networks by migrating hidden unit outputs. Pages 179–184 in: Intelligent Engineering Systems Through Artificial Neural Networks, Volume II, C. H. Dagli, editor. ASME Press, St. Louis, Missouri.
- Birnbaum, L. A., and G. C. Collins, editors. 1991. Proceedings, Eighth International Workshop on Machine Learning, Evanston, Illinois, June 1991. Morgan Kaufmann, San Mateo, California.
- Breiman, L. 1994. Bagging predictors (Technical Report 421). University of California, Department of Statistics, Berkeley, California.
- Clemen, R. T. 1989. Combining forecasts: A review and annotated bibliography. *International Journal of Forecasting* 5: 559–583.
- Crosiar, C. L. 1993. An AVHRR cloud classification database typed by experts (Technical Report NRL/MR/7531-93-7207). Naval Research Laboratory, Prediction Systems Branch, Marine Meteorology Division, Monterey, California.
- Devijver, P. A., and J. Kittler. 1982. Pattern recognition: A statistical approach. Prentice-Hall, Englewood Cliffs, New Jersey.
- Dietterich, T. G., and G. Bakiri. 1991. Error-correcting output codes: A general method for improving multiclass inductive learning programs. Pages 572–577 in: Proceedings, Ninth National Conference on Artificial Intelligence, Anaheim, California, August 1991. AAAI Press, Menlo Park, California.
- Dietterich, T. G., and G. Bakiri. 1995. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research* 2: 263–286.
- Ellrod, G. P., J. M. Peak, and R. L. Bankert. 1993. An expert system for the analysis of high altitude turbulence. Pages 617–620 in: Proceedings, Thirteenth Conference on Weather Analysis and Forecasting. American Meteorological Society, Vienna, Virginia.
- Fett, R. W., M. E. White, J. E. Peak, S. Brand, and P. M. Tag. 1997. Application of hypermedia and expert system technology to navy environmental satellite image analysis. Accepted for publication in *Bulletin of the American Meteorological Society*.
- Fu, K. S. 1968. Sequential methods in pattern recognition and machine learning. Academic Press, New York.
- John, G., R. Kohavi, and K. Pflieger. 1994. Irrelevant features and the subset selection problem. Pages 121–129 in: Proceedings, Eleventh International Machine Learning Conference, New Brunswick, New Jersey, August 1994. Morgan Kaufmann, San Mateo, California.
- Kong, E. B., and T. G. Dietterich. 1995. Error-correcting output coding corrects bias and variance. Pages 313–321 in: Proceedings, Twelfth International Conference on Machine Learning, Tahoe City, California, July 1995. Morgan Kaufmann, San Mateo, California.

- Kuciauskas, A. P., L. R. Brody, R. L. Bankert, P. M. Tag, and M. Hadjimichael. 1996. Automated forecasting of gale force winds in the mediterranean region. Pages 385–361 in: Proceedings, Fifteenth Conference on Weather Analysis and Forecasting. American Meteorological Society, Norfolk, Virginia.
- Makridakis, S., and R. L. Winkler. 1983. Averages of forecasts: Some empirical results. *Management Science* 29(9): 987–996.
- Mohri, T., and H. Tanaka. 1994. An optimal weighting criterion of case indexing for both numeric and symbolic attributes. Pages 123–127 in: *Case-Based Reasoning: Papers from the 1994 Workshop (Technical Report WS-94-01)*, D. W. Aha, editor. AAAI Press, Menlo Park, California.
- Peak, J. E., and P. M. Tag. 1989. An expert system approach for prediction of maritime visibility obscuration. *Monthly Weather Review* 117: 2641–2653.
- Perrone, M. P. 1994. Putting it all together: Methods for combining neural networks. Pages 1188–1190 in: *Advances in Neural Information Processing Systems, 6*, J. D. Cowan, G. Tesauro, and J. Alspector, editors. Morgan Kaufmann, San Mateo, California.
- Perrone, M. P., and L. N. Cooper. 1993. When networks disagree: Ensemble methods for hybrid neural networks. In *Neural Networks for Speech and Image Processing*, R. J. Mammone, editor. Chapman and Hall, Philadelphia, Pennsylvania.
- Quinlan, J. R. 1993a. C4.5: Programs for machine learning. Morgan Kaufmann, San Mateo, California.
- Quinlan, J. R. 1993b. Combining instance-based learning and model-based learning. Pages 236–243 in: *Proceedings, Tenth International Conference on Machine Learning*, Amherst, Massachusetts, July 1993. Morgan Kaufmann, San Mateo, California.
- Rendell, L., and R. Seshu. 1990. Learning hard concepts through constructive induction: Framework and rationale. *Computational Intelligence* 6: 247–270.
- Rumelhart, D. E., G. E. Hinton, and R. J. Williams. 1986. Learning internal representations by error propagation. Pages 318–362 in: *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, D. E. Rumelhart and J. L. McClelland, editors. MIT Press, Cambridge, Massachusetts.
- Sejnowski, T. J., and C. R. Rosenberg. 1987. Parallel networks that learn to pronounce English text. *Complex Systems* 1: 145–168.
- Specht, D. F. 1990. Probabilistic neural networks. *Neural Networks* 3: 109–118.
- Welch, R. M. 1996. Hierarchical neural networks with don't care nodes. Presentation at the Artificial Intelligence research in Environmental Sciences Workshop, Wakefield, Massachusetts, August 1996.
- Welch, R. M., S. K. Sengupta, A. K. Goroch, P. Rabindra, N. Rangaraj, and M. S. Navar. 1992. Polar cloud and surface classification using AVHRR imagery: An intercomparison of methods. *Journal of Applied Meteorology* 31: 405–420.

Wettschereck, D., and T. G. Dietterich. 1992. Improving the performance of radial basis function networks by learning center locations. Pages 1133–1140 in: *Neural Information Processing Systems 4*, J. Moody, S. Hanson, and R. Lippmann, editors. Morgan Kaufmann, San Mateo, California.

Zhang, X., J. Mesirov, and D. Waltz. 1992. Hybrid system for protein structure prediction. *Journal of Molecular Biology* 225: 1049–1063.

Biographical sketches

David W. Aha (UC Irvine 1990) joined the Naval Research Laboratory's Navy Center for Applied Research in Artificial Intelligence in 1993. His research interests lie at the intersection of machine learning (ML) and case-based reasoning (CBR). He serves both communities as a frequent conference program committee member, editing board member (*Machine Learning, Journal of Artificial Intelligence Research, Applied Intelligence*), workshop (co-)organizer, and WWW page maintainer (<http://www.aic.nrl.navy.mil/~aha>). He also is an AIRIES committee member, and recently edited a special issue for *Artificial Intelligence Review* on *Lazy Learning*.

Richard L. Bankert (Penn State, 1988) joined the Marine Meteorology Division of the Naval Research Laboratory in 1990 (then called the Naval Oceanographic and Atmospheric Research Laboratory). His research interests include the application of a variety of artificial intelligence techniques to meteorological and other environmental science problems. He has previously served on the AIRIES committee (1992) and is currently on the American Meteorological Society's Science and Technological Activities Committee on AI Applications to Environmental Science.