A Multi-Class Extension to the Brownboost Algorithm

Ross A. McDonald
Imperial College London

Idris A. Eckley
Shell Research Ltd.

David J. Hand
Imperial College London

30th January 2003

Abstract

Brownboost is an adaptive, continuous time boosting algorithm based on the Boost-by-Majority (BBM) algorithm. Though it has been little studied at the time of writing, it is believed that it should prove especially robust with respect to noisy data sets. This would make it a very useful boosting algorithm for real-world applications. More familiar algorithms such as Adaboost, or its successor Logitboost, are known to be especially susceptible to overfitting the training data examples. This can lead to a poor generalization error in the presence of class noise, since weak hypotheses induced at later iterations to fit the noisy examples will tend to be given undue influence in the final combined hypothesis. Brownboost allows us to specify an expected base-line error rate in advance, corresponding to our prior beliefs about the proportion of noise in the training data, and thus avoid overfitting. The original derivation of Brownboost is restricted to binary classification problems. In this paper we propose a natural multi-class extension to the basic algorithm, incorporating error-correcting output codes and a multi-class *gain* measure. We test two-class and multi-class versions of the algorithm on a number of real and simulated data sets with artificial class noise, and show that Brownboost consistently outperforms Adaboost in these situations.

Keywords: Boost-by-majority, Error-Correcting Output Codes, BrownBoost, Adaboost, Brownian Motion, Multi-Class Problems.

1 Introduction

Brownboost [9] is a continuous-time adaptive boosting algorithm, developed by Freund and based on his non-adaptive Boost-by-Majority (BBM) algorithm [8]. It addresses a particular weakness of the widely-studied and popular Adaboost algorithm and its variants: that the adaptive property that is the key to these algorithms' success also makes them especially vulnerable to classification noise (where we take this to mean that a fixed proportion of the data have had their class labels reassigned at random).

Adaboost-type algorithms work by reweighting the training-set examples at each iteration in order to force the base learner to focus on those that prove the hardest to classify. Since these are very likely to be the noisy data, weak hypotheses induced at later iterations when such examples dominate will tend to be given undue influence in the final combined hypothesis, leading to overfitting and a poor generalization performance. In his empirical comparison of methods for constructing ensembles of decision trees [5], Dietterich concluded that 'the performance of Adaboost can be destroyed by classification noise'.

The great advantage of Brownboost is that it allows us to pre-specify an expected base-line error rate (the noise parameter), corresponding to our prior beliefs about the proportion of class noise in the training data. Because Brownboost knows in advance for how much time the algorithm will run, and is guaranteed to output a hypothesis with a given error rate, if it finds particular training examples especially hard to classify it will in effect 'give up' on them, and seek to gain advantage elsewhere. Given a close estimate of the true value of the noise parameter, Brownboost is likely to ignore the noisy examples entirely when constructing its output hypothesis.

Brownboost and Adaboost are not unrelated; Freund [9] has demonstrated that Adaboost may be viewed as the special case of Brownboost when the noise

parameter is allowed to tend to zero. Brownboost is also optimal in the sense that its output hypothesis will combine the smallest number of weak hypotheses for a given base learner.

Freund's original derivation of Brownboost is restricted to binary classification problems. A large proportion (perhaps the majority) of classification problems will be multi-class in nature. In this paper we discuss a method for extending Brownboost to the multi-class scenario (where by multi-class we mean problems with three or more discrete, categorical class labels, as distinct from binary problems where we have only two).

Multi-class extensions to binary algorithms often amount to breaking a particular multi-class problem down into a set of parallel binary sub-problems. Rather than boost each of these sub-problems separately, we propose that the reduction to binary instead be done at the level of the base-learner. We make the basic assumption that our base learner can return separate, independent, confidence-rated predictions for each class label. In this paper we introduce a novel version of the Brownboost algorithm that incorporates such predictions, based on a new multi-class measure of gain. As with the binary version of Brownboost, our output strong hypothesis is constructed using only a single set of hypothesis weights.

In Section 2 of this paper we review the Error Correcting Output Codes (ECOC) approach for reducing multi-class problems to binary. In Section 2.1 we briefly explain Hamming Decoding. Section 2.2 describes how a similar method can be used to help solve multi-class problems in machine learning, and introduce a new measure of gain (negative loss). Section 3 details the concept of a continuous-time boosting algorithm, and its relationship with Brownian Motion (from which the name Brownboost is derived). In Section 3.1 we recap Freund's insights in the binary case, while in Section 3.2 we recast this argument in the multi-class scenario, using our new gain function. In Section 4 we conduct some empirical experiments with Brownboost, adding artificial noise to both simulated

and real data. At present we are not aware of any other published experimental results relating to Brownboost. Section 4.1 details a simple simulated experiment, which graphically illustrates the noise-compensating properties of the binary version of Brownboost. In Section 4.2 we report the results of our experiments on real data. These results are discussed in Section 4.3. Finally, in Section 5 we summarize our conclusions and suggest directions for future work.

2 Error-Correcting Output Codes

2.1 Hamming Decoding

Error-Correcting Output Codes (ECOC) (see eg. [15]), were originally developed as a means of accurately transmitting a stream of data in binary form in situations where a degree of transmission error was expected to occur.

The original data stream consists of a sequence of symbols, drawn from a set, or *alphabet*, of labels of size k. For example, the alphabet might consist of all lower-case letters in the English language, which constitutes a set of size 26. Before transmission, each of the k labels in the original alphabet is replaced by a *binary codeword* of fixed length ℓ, which is a sequence of ℓ binary digits. This set of codewords may be written in the form of a $k \times \ell$ *coding matrix*, $\mathbf{M}$. For example, if we chose to replace two labels ($k = 2$) with codewords of length five ($\ell = 5$), we might construct the following coding matrix:

$$\mathbf{M} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

The *Hamming distance* between codewords is defined as the number of entries in which two codewords differ. So in the above example the distance between our labels is exactly 5. In order to correct errors efficiently, we aim to select codewords that are as widely spaced as possible.

Now suppose that we receive a corrupted stream of data containing the sequence 01000. Computing the Hamming distance between this vector and each of the rows of $\mathbf{M}$, we would conclude that the original transmitted sequence was more likely to signify the first class (Hamming distance 1) than the second (Hamming distance 4). This procedure of choosing the class label in our coding matrix that is the smallest Hamming distance from the observed data sequence is known as *Hamming decoding*. By replacing our class labels with codewords that are both a reasonable length and a suitable distance apart, we stand a fair chance of reconstructing the original data stream even in the presence of noise.

In 1965 Hamming decoding was used to transmit the first photographs of Mars from the Mariner 4 space probe back to Earth. Since then it has been used many times to transmit images from space.

2.2 Reducing Multi-Class Problems to Binary

The ECOC approach can also be adapted for multi-class problems in machine learning. In a typical learning problem, our goal is to find an approximation to an unknown function $f(\mathbf{x})$ given a *training set* of examples of the form $\langle \mathbf{x}_i, f(\mathbf{x}_i) \rangle$. The components of $\mathbf{x}$ are known as *attributes* and in a discrete problem $f(\mathbf{x})$ defines a probability mass function across the set of k distinct class labels.

A great many real-world problems may be formulated in this manner. One standard example is *credit scoring* (see eg [14], [25], [18]), employed by banks and other lending institutions seeking to predict the credit-worthiness of loan applicants based on information collected via the application procedure and a training data set comprising past customer records. Customers who default on repayments are classified *bad* (class 1), whereas those who repay the loan without default are classified *good* (class 0). In this simple form credit scoring amounts to a binary classification problem with discrete or continuous attribute vectors.

Credit scoring closely parallels problems of medical diagnosis, where the aim is to determine whether an illness is likely to be present using measurements based on symptoms and knowledge of previous cases.

At the learning stage, a binary model (such as ordinary logistic regression) or learning algorithm (such as the linear perceptron) uses the set of training data to construct a model for $f(\mathbf{x})$, known as a *hypothesis*. When presented with new data for which the true class labels are unknown, the hypothesis will output a single real-valued confidence prediction which is then translated into a class prediction by fixing a threshold value. By extension, the hypotheses generated by multi-class models or algorithms (such as the multi-layer perceptron, multi-class CART or C4.5 regression trees, neural nets or radial basis functions) will output a separate and independent confidence prediction for each of the k class labels. These predictions may be used to classify new examples by, for example, assigning them to the single class with the highest confidence level.

For notational convenience, we shall make the following assumption about the output of our hypotheses:

Assumption 1: *Hypotheses output confidence-rated predictions that take the form of real values in the range* $[-1, 1]$.

This assumption is not restrictive, since if necessary we can map the standard output of our hypotheses to the interval $[-1, 1]$ via a simple rescaling.

Thus we have two distinct possibilities, according to whether our learner is of binary type or multi-class type:

Case 1: *Our learner is of binary type. Given a binary learning problem it fits a hypothesis that outputs a single prediction in the range [-1,1]. The magnitude of the prediction is its confidence level, while its sign relates it to one of the two*

possible classes.

Case 2: *Our learner is of multi-class type. For each class label its hypothesis outputs a separate, independent confidence prediction in the range [-1,1].*

Using Error-Correcting Output Codes, we can transform a binary learner of the first type into a multi-class learner of the second. To illustrate this, suppose that we have a multi-class problem with k discrete class labels. Following the ECOC approach, we can replace each of these k class labels with a binary codeword of length ℓ. Here we modify our notation slightly so that a binary codeword is a string of symbols drawn from $\{-1, 1\}$. As before, we aim to select these codewords in such a way as to be as far apart from one another as possible under some chosen measure of distance. Thus each example in our training set becomes associated with a set of ℓ binary labels, rather than one single class label. We can now treat each column of the coding matrix as a separate two-class learning problem, equivalent to learning a binary partition of the set of all classes. We call the learner ℓ times, each time passing to it the complete set of training data attributes, along with the set of binary labels generated by the corresponding column of the coding matrix.

For example, suppose that we have a four-class problem. The following coding matrix corresponds to all contrasts between partitions of the class labels into subsets of size 2:

$$\mathbf{M} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{pmatrix}$$

Allwein et al. [2] have proposed a natural extension to the procedure, in which labels are also allowed to take the value 0. Training examples that are

labelled 0 are not passed on to the learner. This has the effect of increasing the range of possible contrasts.

There are many ways to construct a coding matrix. Perhaps the most direct choice is the *one-against-all* approach. In this case $\mathbf{M}$ is a $k \times k$ matrix whose diagonal entries are all +1 and all other entries -1. We have thus reduced the multi-class problem to the set of binary problems that correspond to distinguishing each class in turn from every other class. Alternatively, we might use the *all-pairs* approach. Here each column of the coding matrix corresponds to the binary problem that distinguishes a particular class pair. For this column $\mathbf{M}$ has +1 in the row corresponding to the first class, -1 in the row corresponding to the second class, and 0 in all other rows. $\mathbf{M}$ is therefore a $k \times \binom{k}{2}$ matrix. The columns of a *complete* coding matrix correspond to all non-trivial partitions of the set $1, ..., k$ into two subsets. Thus it is of size $k \times (2^{k-1} - 1)$.

Given a new example we proceed by calculating the distance between the assigned label set and each row of the coding matrix. But rather than use Hamming distance, which fails to take the confidence levels of the predictions into account, we propose taking the expected value across labels of a measure of *gain*. For a given coding matrix $\mathbf{M}$ and example with hypothesis outputs $h_j(\mathbf{x})$ for and $j = 1, ..., \ell$ we define the *gain* for class i and label j as follows:

$$G_{i,j} = M(i,j)h_j(\mathbf{x}) \tag{2.1}$$

Because of Assumption 1, the gain is a value in the range $[-1, 1]$ that is positive if and only if the prediction was on the correct side of 0, with magnitude equal to its confidence level.

We now define distance, d_i, to be the expected value of the gain across coding labels, taking care not to include labels in the coding matrix that take the value 0:

$$d_i = \frac{1}{\ell - \ell_i^0} \sum_{j=1}^{\ell} M(i,j)h_j(\mathbf{x}) \tag{2.2}$$

where

$$\ell_i^0 = \sum_{j=1}^{\ell} [[\mathbf{M}(i,j) = 0]]$$

and $[[\pi]]$ is the identity function that equals 1 when π is true, and 0 otherwise. The above measure has range $[-1, 1]$, so we can treat the k distances between the assigned labels and the rows of the coding matrix exactly as we would the predictions output by a multi-class learner. So to classify a new example, we would assign it to the single class with the *largest* prediction value.

For example, suppose that we were using the all-pairs coding matrix for a three-class problem:

$$\mathbf{M} = \begin{pmatrix} 1 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & -1 \end{pmatrix}$$

and for a given example our learner outputs the following confidence predictions, each one corresponding to a column of $\mathbf{M}$:

$$\begin{pmatrix} 0.36 & -0.74 & -0.98 \end{pmatrix}.$$

Then we can compute our distance values for each class using equation 2.2:

$$\frac{1}{2} \begin{pmatrix} 0.36 - 0.74 + 0 \\ -0.36 + 0 - 0.98 \\ 0 + 0.74 + 0.98 \end{pmatrix} = \begin{pmatrix} -0.19 \\ -0.67 \\ 0.86 \end{pmatrix}.$$

The largest distance value is the third, so we would assign this example to class 3.

Thus we have found a way of using a binary learner to solve a multi-class problem. In the following sections, we explain how this method can be used to help build a multi-class version of Brownboost.

3 Brownboost

3.1 Binary Version

A *weak learner* is a model or algorithm whose output hypotheses will with high probability outperform random guessing by a small amount over the training data under some loss measure. We note in passing that in the binary case we can always guarantee that our hypothesis performs no worse than random guessing in terms of error rate, since any hypothesis output $h(x)$ with error $\epsilon > \frac{1}{2}$ can be replaced by $-h(x)$ with error rate $1 - \epsilon < \frac{1}{2}$.

A *strong learner* is one that can learn the training data to an arbitrarily high degree of accuracy (for more rigorous definitions of weak and strong learner, see [22]).

A *boosting algorithm* is a means of transforming a weak learner into a strong learner by combining the hypotheses generated on reweighted versions of the training data. At every iteration of a binary boosting algorithm, the weak (or base) learner is trained on the weighted training data, and achieves an error rate (proportion of misclassified examples) of, say, $\frac{1}{2} - \gamma$ for some $\gamma > 0$. The example weights are then updated in such a way as to bring the error rate of the weak hypothesis output by the base learner close to $\frac{1}{2}$. The weak hypothesis, weighted appropriately, is added to the combined hypothesis, and the base learner is called again.

The Boost by Majority (BBM) algorithm [8] was one of the first to be developed and predates Adaboost. Though it is optimal in the sense that its output hypothesis will combine the smallest possible number of weak hypotheses, it is not adaptive, and is disadvantaged by the fact that it is necessary to know in advance that the weak learner will always achieve a *guaranteed* training error of $\frac{1}{2} - \gamma$ for some known $\gamma > 0$ under any possible example reweighting.

The weight assigned to a particular training example at iteration i is:

$$w_r^i = \left(\begin{array}{c} k_\gamma - i - 1 \\ \frac{k_\gamma}{2} - r \end{array} \right) (\frac{1}{2} + \gamma)^{\frac{k_\gamma}{2} - r} (\frac{1}{2} - \gamma)^{\frac{k_\gamma}{2} - i - 1 + r}, \tag{3.1}$$

where r is the number of correct predictions made so far for the example and k_γ is the total number of iterations, which is calculated in advance.

Another important quantity is the potential, which remains constant across iterations:

$$\beta_r^i = \sum_{j=0}^{\frac{k_\gamma}{2} - r} \left(\begin{array}{c} k_\gamma - i \\ j \end{array} \right) (\frac{1}{2} + \gamma)^j (\frac{1}{2} - \gamma)^{k_\gamma - i - j}. \tag{3.2}$$

Brownboost is an adaptive version of BBM, and does not presuppose that we know the value of γ. It is a true boosting algorithm in the sense that it outputs a strong hypothesis with a guaranteed training error rate. It is derived by considering what happens to the BBM algorithm if the example reweighting is assumed to happen in continuous time. We begin by imagining that at time 0 the base learner outputs a hypothesis with error rate $\frac{1}{2} - \gamma$ for some $\gamma > 0$ (not necessarily known), and that this hypothesis then *remains in play* until the process of continuous reweighting raises its error rate to (very close to) $\frac{1}{2}$. At this point we consider our hypothesis 'tested to destruction', weight its contribution to the combined hypothesis according to how long it survived, and call the base learner again. We define the error rate of the hypothesis under a particular reweighting as $\frac{1}{2} - \delta$ with $0 < \delta < \gamma$.

We can simulate the effect of the continuous reweighting on the weak hypothesis for a given value of δ by adding random noise to hypothesis $h(x)$ to construct a new hypothesis $h'(x)$ as follows:

$$h'(x) = \begin{cases} h(x), \text{with probability } \delta/\gamma \\ 0, \text{with probability } (1 - \delta/\gamma)/2 \\ 1, \text{with probability } (1 - \delta/\gamma)/2 \end{cases}$$

The expected error rate of $h'(x)$ with respect to the whole training set is $\frac{1}{2}-\delta$. We are interested in characterizing the behaviour of the sum of such randomly altered hypotheses as $\delta \to 0$.

If we set time $t = \delta^2 i$ and define 'location' as

$$r_\delta = \delta \sum_{j=1}^{[t/\delta^2]} h'_j(x),$$

then in the limit $\delta \to 0$ this behaviour approximates *Brownian Motion with Drift* with mean $\mu(t)$ and variance $\sigma(t)$ equal to

$$\mu(t) = \int_0^t \frac{1}{\gamma(s)} ds, \qquad \sigma^2(t) = t. \tag{3.3}$$

In the limit $\delta \to 0$, the weighting function 3.1 reduces to

$$w(t,r) = \exp\left(-\frac{(r(t)+c-t)^2}{c-t}\right)$$

where if k_δ is the total number of iterations run by the discrete algorithm, c is the limit as $\delta \to 0$ of $\delta^2 k_\delta$. The potential function 3.2 becomes:

$$\beta(t,r) = \frac{1}{2}\left(1 - \mathrm{erf}\left(\frac{r(t)+c-t}{\sqrt{c-t}}\right)\right)$$

where erf is the 'error function':

$$\mathrm{erf}(a) = \frac{2}{\pi}\int_0^a e^{-x^2} dx.$$

We use these definitions of $t, r(t), w(t,r)$ and $\beta(t,r)$ to construct the binary version of Brownboost quoted in figure 1.

There follows a step-by-step explanation of this algorithm.

Algorithm Brownboost (Binary Version)

Inputs:
Training Set: A set of m labelled examples: $T = (x_1, y_1), ..., (x_m, y_m)$ where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, +1\}$.
WeakLearn – A weak learning algorithm.
c – a positive real valued parameter.
$\nu > 0$ – a small constant used to avoid degenerate cases.

Data Structures:
prediction value: With each example we associate a real valued *margin*. The margin of example (x, y) on iteration i is denoted $r_i(x, y)$. The initial prediction values of all examples is zero: $r_1(x, y) = 0$.

Initialize 'remaining time' $s_1 = c$.
Do for $i = 1, 2, ...$

1. Associate with each example a positive weight
$$W_i(x, y) = e^{-(r_i(x,y)+s_i)^2/c}$$

2. Call **Weaklearn** with the distribution defined by normalizing $W_i(x, y)$ and receive from it a hypothesis $h_i(x)$ which has some advantage over random guessing $\sum_{(x,y)} W_i(x, y)h_i(x)y = \gamma_i > 0$.

3. Let γ, α and $\mathbf{t}$ be real valued scalar variables that obey the following differential equation:
$$\frac{d\mathbf{t}}{d\alpha} = \gamma = \frac{\sum_{(x,y)\in T} \exp(-\frac{1}{c}(r_i(x,y)+\alpha h_i(x)y+s_i-\mathbf{t})^2)h_i(x)y}{\sum_{(x,y)\in T} \exp(-\frac{1}{c}(r_i(x,y)+\alpha h_i(x)y+s_i-\mathbf{t})^2)}$$
where $r_i(x, y_j)$, $h_i(x)y$ and s_i are constants in this context. Given the boundary conditions $\mathbf{t} = 0, \alpha = 0$ solve the set of equations to find $t_i = \mathbf{t}^* > 0$ and $\alpha_i = \alpha^*$ such that either $\gamma^* \leq \nu$ or $\mathbf{t}^* = s_i$.
4. Update the prediction value of each example to
$$r_{i+1}(x, y) = r_i(x, y) + \alpha_i h_i(x)y.$$

5. Update 'remaining time' $s_{i+1} = s_i - t_i$.

Until $s_{i+1} \leq 0$
Output the final hypothesis,

$$\text{if } p(x) \in [-1, +1] \text{ then } p(x) = \text{erf}\left(\frac{\sum_{i=1}^N \alpha_i h_i(x)}{\sqrt{c}}\right).$$

$$\text{if } p(x) \in \{-1, +1\} \text{ then } p(x) = \text{sign}\left(\sum_{i=1}^N \alpha_i h_i(x)\right).$$

Figure 1: The two-class Brownboost Algorithm (quoted from [9])

Inputs

The algorithm takes as input a training data set of size m and a weak learner, as already discussed. It also inputs a small constant, $\nu > 0$, used to avoid degenerate cases when solving the differential equation, and a parameter c, which is equal to the total time for which the algorithm is to be run. As explained in [9], this constant also relates to Brownboost's ability to ignore noisy training examples. If we define the *loss* of a particular training example as $1 - h(x)y$ then the expected loss of the strong hypothesis over all examples in the training data set can be written as

$$1 - \frac{1}{m}\sum_{j=1}^{m} p(x_j)y_j$$

Note that the loss is a value in the range $[0, 2]$, though we do not expect it to exceed 1 in practice since this would imply that we have made no improvement on average over random guessing. If we prefer to measure error in the range $[0, 1]$ we simply have to divide by 2.

It can be shown that the parameter c can be related to the training loss ϵ of the strong hypothesis by

$$\epsilon = 1 - \mathrm{erf}(\sqrt{c}) \tag{3.4}$$

Thus by selecting a value of c we are also specifying the proportion of class noise that we expect to find in the training data. Since the noisy examples should generally be the hardest to learn, it is likely that the Brownboost algorithm will not attempt to fit them.

Data Structures

For each example in the training data set, Brownboost maintains a cumulative quantity known as the margin, defined as follows:

$$r_1(x, y) = 0,$$
$$r_{i+1}(x, y) = r_i(x, y) + \alpha_i h_i(x)y$$

where α_i is the weight that the boosting algorithm assigns to the weak hypothesis at iteration i. The margin is therefore a positive or negative real value which tells us, for a particular example, whether that example will be correctly classified under the current strong hypothesis. A positive margin value tells us that the example is being correctly classified, a negative value signifies the converse. A margin of 0 tells us that our hypothesis is no more informative for this example than random guessing. The magnitude of our margin value can therefore be viewed as a measure of how much better (or worse) our hypothesis is performing than random guessing. The margins are used in the weight update step.

Step 1

This is the weight update step as defined by equation 3.1.

Step 2

This is the pivotal step of the algorithm, where the weak learner is passed the training data and the vector of example weights, and returns a hypothesis function with weighted gain γ_i.

Step 3

At this step we compute the hypothesis weight α_i and the time step size t_i. The hypothesis weight is taken to be the mean of the Brownian motion, as defined in 3.3, across the time interval. This means that to determine α_i and t_i we must solve the following differential equation:

$$\frac{d\mathbf{t}}{d\alpha} = \gamma = 0.$$

given the boundary condition $\alpha(0) = 0$. This equation can be solved via numerical methods. For suggested means of doing so, see Section 7 in [9].

Step 4

This is the margin update step.

Step 5

The parameter s keeps track of the remaining time. It is initially set to c, and at each iteration i an amount equal to t_i is subtracted from it.

Output

The output takes the form of either a confidence measure in the range $[-1, 1]$ or a label drawn from $\{-1, 1\}$.

In the first case, the exact form of $p(\mathbf{x})$ is chosen such that the predictions on the training data have an expected loss of exactly ϵ, where ϵ is defined in terms of c as in 3.4. This result is derived by equating the potential at the start and end of the algorithm (see Theorem 2 in [9] for more details). This explains why the error function emerges in the definition of $p(x)$ (the error function is in effect simply a non-linear mapping back to the range $[-1, 1]$).

In the second case, we return a straightforward binary prediction by taking the sign of the sum of the weighted weak hypotheses.

3.2 A Multi-Class Extension to Brownboost

In this section we bring together the ECOC approach and the binary version of Brownboost to create a multi-class extension to the basic algorithm.

Recall that there are two distinct possibilities, depending on whether we are boosting a binary or a multi-class base learner:

Case 1: *Our base learner is of binary type. Given a binary learning problem it fits a hypothesis that outputs a single prediction in the range [-1,1]. The magnitude of the prediction is its confidence level, while its sign relates it to one of the two possible classes.*

Case 2: *Our base learner is of multi-class type. For each class label its hypothesis outputs a separate, independent confidence prediction in the range [-1,1].*

In the first case, as already discussed we can construct a $k \times \ell$ coding matrix to break the multi-class problem down into a set of binary discrimination problems.

In the latter case we would expect a perfect hypothesis to assign a confidence value of +1 to the correct class, and -1 to the others. We can therefore associate our output with the coding matrix that corresponds to the one-against-all approach, that is, the $k \times k$ matrix whose diagonal entries are +1 with all other entries -1. The length of the codewords is $\ell = k$.

In either case, our boosting algorithm communicates with the base learner via a $k \times \ell$ coding matrix which maps each of k discrete *classes* to a set of ℓ binary *labels*.

The binary version of Brownboost maintains a weight for each of the m examples in the training data set. In our multi-class extension, we maintain a separate weight for each example and for each binary label associated with that example. Thus we could write the full set of example weights in the form of an $m \times \ell$ matrix.

We also have to introduce a new measure of the *gain* of a hypothesis over random guessing, γ. We take this to be the weighted expectation, across all examples and labels, of the gain as defined in equation 2.1. So if the true labels for a particular example are denoted λ_j, $j = 1, ..., \ell$, and the hypothesis prediction for label λ_j is denoted $h_j(x)$, then we define the overall gain as follows:

$$\gamma = \mathrm{E}_W h_j(x) \lambda_j \tag{3.5}$$

where E_W means expectation across the full matrix of example weights. This is a value in the range $[-1, 1]$ which measures how informative our hypothesis is with respect to random guessing.

We define the *pseudo-loss* of a set of hypotheses to be $1 - \gamma$ and the *pseudo-error* (analogous to the binary error in the 2-class case) to be $\frac{1}{2} - \frac{1}{2}\gamma$.

Under 3.5, an *uninformative hypothesis* (one with zero gain and pseudo-error

$\frac{1}{2}$) is no better than random guessing. Otherwise, we can always ensure that the hypothesis output by the base learner will be more informative than random guessing, since any hypothesis $h(x)$ with pseudo-error $\epsilon > \frac{1}{2}$ may be replaced by $-h(x)$ with pseudo-error $1 - \epsilon < \frac{1}{2}$.

By analogy with the binary argument, we can suppose that we are combining hypotheses of the form:

$$h'(x) = \begin{cases} h(x), \text{with probability } \delta/\gamma \\ \text{Any uninformative hypothesis with probability } (1 - \delta/\gamma) \end{cases}$$

so that now $h'(x)$ has *pseudo-error* of exactly $\frac{1}{2} - \delta$.

Our multi-class version of the Brownboost algorithm is shown in figure 2. For details of how various theorems and proofs in [9] can be modified to accommodate this extension, see Appendix B.

There follows a step-by-step explanation of the multi-class version of BrownBoost.

Inputs

The multi-class extension is based on the coding matrix $\mathbf{M}$ with entries in $\{-1, 0, 1\}$. The rows of $\mathbf{M}$ correspond to the original set of classes, while the columns represent the binary partitions of this set that are to be distinguished by the base learner. There are many possible choices for $\mathbf{M}$, ranging from the *one-against-all approach* to the *complete* coding matrix. For more details, see [6] and [2].

Algorithm Brownboost (Multi-Class Version)

Inputs:
ECOC Matrix: The $k \times \ell$ coding matrix $\mathbf{M}$.
Training Set: A set of m labelled examples: $T = (x_n, y_n), n = 1, ..., m$ where $x_n \in \mathbb{R}^d$ and $y_n \in \{y_1, y_2, ..., y_k\}$. Each y_n is associated via the matrix $\mathbf{M}$ with a set of ℓ binary labels $\{\lambda_1^n, ..., \lambda_\ell^n\}$, where $\lambda_j^n \in \{-1, 1\}, j = 1, ..., \ell$.
WeakLearn – A weak learning algorithm.
c – a positive real valued parameter.
$\nu > 0$ – a small constant used to avoid degenerate cases.

Data Structures:
prediction value: With each example we associate a set of real valued margins. The margin of example (x_n, y_n) for label λ_j^n on iteration i is denoted $r_{i,j}(x_n, y_n)$. The initial value of all margins is zero: $r_{1,j}(x_n, y_n) = 0, n = 1, ..., m, j = 1, ..., \ell$.

Initialize 'remaining time' $s_1 = c$.
Do for $i = 1, 2, ...$

1. Associate with each example and label a positive weight
$$W_{i,j}(x_n, y_n) = e^{-(r_{i,j}(x_n, y_n) + s_i)^2 / c}, n = 1, ..., m, j = 1, ..., \ell,$$

2. **Binary base learner:** Call **Weaklearn** ℓ times $j = 1, ..., \ell$, each time passing it the weight distribution defined by normalizing $W_{i,j}(x_n, y_n)$ for fixed j, and the training data set alongside the binary labels defined by column j of the matrix $\mathbf{M}$.
Multi-class base learner: Call **Weaklearn**, passing it the training data and the full set of weights.

   Receive from **Weaklearn** a set of ℓ hypotheses $h_{i,j}(x)$ which have some advantage over random guessing
$$\frac{\sum_{n=1}^{m} \sum_{j=1}^{\ell} W_{i,j}(x_n, y_n)(h_{i,j}(x_n)\lambda_j^n)}{\sum_{n=1}^{m} \sum_{j=1}^{\ell} W_{i,j}(x_n, y_n)} = \gamma_i > 0.$$

3. Let γ, α and $\mathbf{t}$ be real valued variables that obey the following differential equation:
$$\frac{d\mathbf{t}}{d\alpha} = \gamma = \frac{\sum_{n=1}^{m} \sum_{j=1}^{\ell} \exp(-\frac{1}{c}(r_{i,j}(x_n, y_n) + \alpha h_{i,j}(x_n)\lambda_j^n + s_i - \mathbf{t})^2) h_{i,j}(x_n)\lambda_j^n}{\sum_{n=1}^{n} \sum_{j=1}^{\ell} \exp(-\frac{1}{c}(r_{i,j}(x_n, y_n) + \alpha h_{i,j}(x_n)\lambda_j^n + s_i - \mathbf{t})^2)}$$
   where $r_{i,j}(x_n, y_n)$, $h_{i,j}(x_n)$ and s_i are constants in this context.
   Given the boundary conditions $\mathbf{t} = 0, \alpha = 0$ solve the set of equations to find $t_i = \mathbf{t}^* > 0$ and $\alpha_i = \alpha^*$ such that either $\gamma^* \leq \nu$ or $\mathbf{t}^* = s_i$.

4. Margin update: $r_{i+1,j}(x_n, y_n) = r_{i,j}(x_n, y_n) + \alpha_i h_{i,j}(x_n)\lambda_j^n$.

5. Update 'remaining time' $s_{i+1} = s_i - t_i$.

Until $s_{i+1} \leq 0$
Output Final hypotheses: $p_j(x) = \text{erf}\left(\frac{\sum_{i=1}^{N} \alpha_i h_{i,j}(x)}{\sqrt{c}}\right), j = 1, ..., \ell.$

Figure 2: The Multi-class Brownboost Algorithm

If our base learner already outputs multi-class predictions, we assume that our coding matrix is the $k \times k$ matrix whose diagonal entries are 1, with all other entries -1.

We also input c, the noise parameter estimate related to the accuracy of the output hypothesis via equation 3.4, and ν, a small constant used to avoid degenerate cases in solving the differential equation.

Data Structures

We now associate a margin with every example and label in the training data set. The margins may be written in the form of an $m \times \ell$ matrix.

Step 1

The weight update step. We associate a separate weight with each example and label. The weights could thus also be written in the form of an $m \times \ell$ matrix.

Step 2

At this step we either call the weak learner ℓ times if it is binary, or once if it is multi-class. In either case we receive from it a hypothesis that given a new example outputs a prediction in the range $[-1, 1]$ for each of the ℓ labels.

Step 3

Here the differential equation is solved to determine the values of the hypothesis weight α and the time step $\mathbf{t}$. To solve this equation, we can use slightly modified versions of the same numerical methods as in the binary case (see Appendix B for more details).

Step 4

The margin update step.

Step 5

The 'remaining time' update step.

Output

The outputs take the form of a set of label predictions in the range $[-1, 1]$. We can use these predictions to classify a new example by calculating the expected

gain across each row of the ECOC matrix as in equation 2.2. We predict the class with the highest gain.

4 Experiments

4.1 Simulated Data

In this first example we aim graphically to demonstrate the noise-compensatory properties of the two-class version of the Brownboost algorithm on a simulated two-dimensional data set.

To construct the initial data, a set of 300 two-dimensional uniform random variates, $\{\mathbf{x_1}, \mathbf{x_2}\}$, were generated in the range $[-1, 1] \times [-1, 1]$. The class division is defined via a smooth decision surface as follows:

$$y^n = \begin{cases} 0 \text{ if } \frac{3}{4} \sin(\pi x_1^n) < x_2^n \\ 1 \text{ if } \frac{3}{4} \sin(\pi x_1^n) \geq x_2^n \end{cases}, n = 1, ..., 300.$$

Note that in this idealized example the partition between our classes is clearly delineated, without any ambiguity. We do not expect this to be reflected in many real-world situations.

This initial data are shown plotted in Figure 3, panel (1).

Next we fit Adaboost to these data, allowing the algorithm to run until the training error rate is exactly 0, at which point it terminates naturally. The version of Adaboost used in all our experiments is detailed in Appendix A. Our chosen base learner is a simple algorithm that outputs confidence-rated binary stumps (these are decision trees with only one split - see [4] for more details). The combined hypothesis output by Adaboost unambiguously maps every point in the example space to one of the two classes, as illustrated in Figure 3, panel (2). We observe that the fitted partition is a fair approximation to the shape of the true decision boundary.

Now we introduce class noise, which we define as follows (note that some authors adopt a different definition):

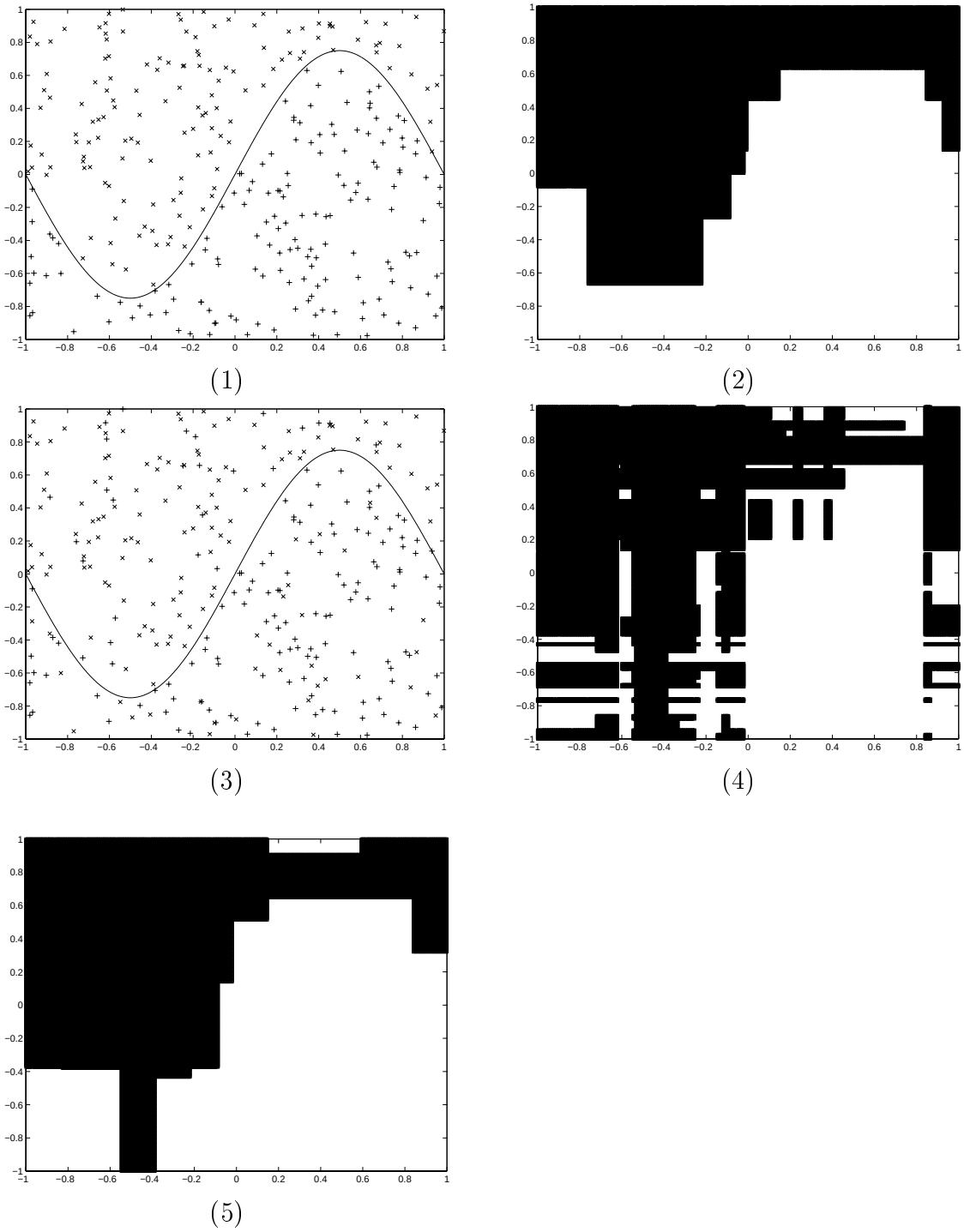

Figure 3: Plots for the sine wave example. (1) 300 data points in 2 dimensions mapped to 2 classes via a decision boundary. (2) Adaboost fitted to the non-noisy data. (3) The same data with 20% class noise. (4) Adaboost fitted to the noisy data. (5) Brownboost fitted to the noisy data. See Section 4.1 for a fuller explanation of panels.

Definition: *x% class noise means that x% of the examples are assigned a randomly chosen incorrect label. In a binary problem, this means that x% of examples have their class label swapped.*

We construct a data set with 20% class noise by swapping the class labels of one in five of examples in the original data set. This new data are shown plotted in Figure 3, panel (3).

Since we know *a priori* that 20% of the class assignments in the new data are inherently misleading, we run Adaboost only until it yields a training error rate of 0.2. The partition fitted by the output hypothesis is that shown in Figure 3, panel (4). Here we see a clear illustration of Adaboost's vulnerability to class noise. The 'patchiness' of the strong hypothesis is the result of overfitting. This happens because the Adaboost algorithm is designed in such a way as to focus the attention of the base learner at an early stage on those examples that are the most difficult to classify. This means that the algorithm attempts to fit the noisy data first, to the overall detriment of the combined output hypothesis.

Finally, we fit Brownboost to the noisy data. In order to do this, we require a good estimate of the noise parameter, c. Since we know that 20% of class assignments are misleading, we aim for a combined classifier with a training error rate of 0.2. This corresponds to a pseudo-loss of 0.4 (since in the binary case the pseudo-loss is simply twice the error rate). This leads us, via equation 3.4, to the choice of $c = 0.3541$. In a situation where the noise level was not known in advance, we would need to search for an appropriate choice of c, possibly via the binary search method described in [9].

The combined hypothesis output by Brownboost with this choice of noise parameter is shown in Figure 3, panel (5). Note that except at the peaks of the sine curve, where a high concentration of noisy data seems to have led to some distortion, the final classifier strongly resembles the original decision surface.

This is because Brownboost effectively assigns diminishing weight to examples that it is unlikely to classify in the time available, and so proves robust to the class noise.

Cross-validatory tests were carried out on the original and noisy versions of this data to determine approximate test error rates for the two algorithms. The results are detailed in the next section.

4.2 Real Data, Artificial Noise

We conducted a series of tests using the six data sets summarized in Table 1. The *Sine* data is the simulated example described in the previous section. All of the remaining data sets, with the exception of *Credit* were taken from the *UCI Machine Learning Repository* [1].

King-Rook vs King-Pawn (KRKP) is a two-class data set based on chess endgames.

The *Credit* data set is a credit scoring data set of the type described in Section 2.2. Details of these data have been omitted from Table 1 for commercial reasons.

Wisconsin is the well-known diagnostic data set for breast cancer compiled by Dr William H. Wolberg, University of Wisconsin Hospitals [17].

The *Wine* data, based on a chemical analysis of Italian wines, and *Balance* data, which records the results of a psychological experiment, are three-class data sets which have been included to test our multi-class extension to Brownboost.

In order to ensure algorithmic convergence in the time available, the *KRKP* and *Credit* data sets were curtailed to 500 examples apiece. Indicator variables were substituted for categorical variables in both of these data sets (see [13], Section 9.7 for more details).

We constructed a noisy version of each data set by assigning a randomly chosen, incorrect class label to 20% of training examples.

Each experiment consisted of 100 trials. At each trial, one third of the data

examples were selected at random and set aside as a test set. The remaining two thirds of examples were used to train the algorithm. We recorded the final pseudo-loss and error rates of the output hypothesis on both the training and test data.

We trained Adaboost on the original data sets to give us a benchmark for our comparison. Both algorithms were trained on the noisy data, using the same randomly chosen training and test sets. For each of the 100 trials performed for each algorithm on the noisy data, we noted which algorithm achieved the lowest test error rate. These results are set out in Table 2.

In all trials Adaboost was allowed to run until its training loss matched the expected training loss rate of that data set, or for a maximum of 100 iterations. The version of Adaboost used was that detailed in Appendix A.

For Brownboost, we calculated an appropriate value for c using equation 3.4. For the two-class data (*Sine, KRKP, Wisconsin* and *Credit*) an error rate of 20% corresponds to a loss rate of 0.4, since loss is measured in the range $[0, 2]$. For the three-class data (*Wine* and *Balance*) we have to be more careful, since the expected pseudo-loss will be dependent on our coding matrix. We used the *one-against-all* approach on these data sets, so our coding matrix was as follows:

$$\mathbf{M} = \begin{pmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{pmatrix}$$

Since each row of the coding matrix has one label in common with every other, randomly reassigning 20% of class assignments only amounts to changing two thirds that proportion of coding labels. Thus our expected pseudo-loss rate will be 0.2667, and this leads us to the choice of $c = 0.6169$.

The training and test loss and error rates for each trial are recorded for a 95% confidence interval in Tables 3, 4 and 5. Figures 4 and 5 are box plots of the test error rates in Tables 4 and 5.

Data Set	Entries	Total Attributes	Categorical Attributes	Continuous Attributes	Binary Attributes	Classes	Class Distribution
Sine	300	2	0	2	0	2	143,157
KRKP	500	36	2	34	0	2	276,224
Wisconsin	699	9	0	9	0	2	241,458
Credit	500	–	–	–	–	2	–
Wine	178	13	0	13	0	3	59,71,48
Balance	625	4	0	4	0	3	288,49,288

Table 1: Summary table for the data sets used in experiments.

Data Set	BrownBoost - Ties - Adaboost
Sine	87 – 8 – 5
KRKP	86 – 3 – 11
Wisconsin	57 – 9 – 34
Credit	67 – 8 – 25
Wine	65 – 13 – 22
Balance	98 – 1 – 1

Table 2: Number of cross-validatory samples on which each algorithm achieved the lowest test error rate (Total = 100).

Adaboost - 0% Class Noise				
Data Set	Training Loss	Training Error	Test Loss	Test Error
Sine	0.00110 ± 0.00084	0.00055 ± 0.00042	0.08840 ± 0.00765	0.04420 ± 0.00383
KRKP	0.06066 ± 0.00365	0.03033 ± 0.00182	0.10132 ± 0.00580	0.05066 ± 0.00290
Wisconsin	0.06747 ± 0.00334	0.03373 ± 0.00167	0.09579 ± 0.00511	0.04790 ± 0.00255
Credit	0.15647 ± 0.00379	0.07823 ± 0.00189	0.21084 ± 0.01084	0.10542 ± 0.00542
Wine	0.00000 ± 0.00000	0.00000 ± 0.00000	0.06942 ± 0.00512	0.04136 ± 0.00472
Balance	0.13944 ± 0.00235	0.07619 ± 0.00164	0.66475 ± 0.01315	0.18490 ± 0.00516

Table 3: Loss and error rates for Adaboost on the original data sets, 95% confidence intervals, 0% class noise.

Adaboost - 20% Class Noise				
Data Set	Training Loss	Training Error	Test Loss	Test Error
Sine	0.41270 ± 0.00674	0.20635 ± 0.00337	0.62180 ± 0.01646	0.31090 ± 0.00823
KRKP	0.43270 ± 0.00708	0.21635 ± 0.00354	0.53711 ± 0.01030	0.26855 ± 0.00515
Wisconsin	0.44695 ± 0.00439	0.21618 ± 0.00309	0.47614 ± 0.01067	0.23807 ± 0.00533
Credit	0.47820 ± 0.00643	0.23910 ± 0.00321	0.63193 ± 0.02018	0.31596 ± 0.01009
Wine	0.19555 ± 0.00071	0.08815 ± 0.00321	0.46701 ± 0.00900	0.28661 ± 0.00838
Balance	0.43615 ± 0.00394	0.21365 ± 0.00299	0.81912 ± 0.00579	0.33707 ± 0.00605

Table 4: Loss and error rates for Adaboost on the noisy data sets, 95% confidence intervals, 20% class noise.

Brownboost - 20% Class Noise				
Data Set	Training Loss	Training Error	Test Loss	Test Error
Sine	0.40613 ± 0.00108	0.17335 ± 0.00170	0.55601 ± 0.01159	0.26130 ± 0.00628
KRKP	0.40840 ± 0.00050	0.18213 ± 0.00135	0.51195 ± 0.00815	0.24349 ± 0.00448
Wisconsin	0.40025 ± 0.00043	0.18824 ± 0.00094	0.47419 ± 0.00684	0.22961 ± 0.00412
Credit	0.39898 ± 0.00087	0.17686 ± 0.00156	0.59015 ± 0.00861	0.28910 ± 0.00488
Wine	0.25919 ± 0.00132	0.16513 ± 0.00279	0.41989 ± 0.01038	0.25492 ± 0.01056
Balance	0.27005 ± 0.00020	0.15832 ± 0.00198	0.62421 ± 0.00712	0.28029 ± 0.00574

Table 5: Loss and error rates for Brownboost on the noisy data sets, 95% confidence intervals, 20% class noise.

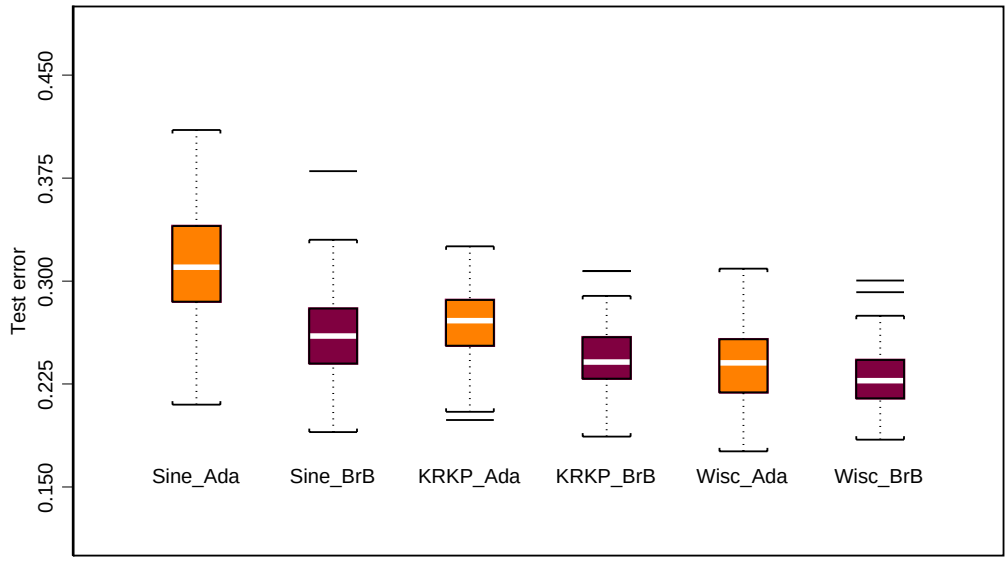

Figure 4: Box plot of the test error rates in Tables 4 and 5 for *Sine, KRKP* and *Wisconsin* data.

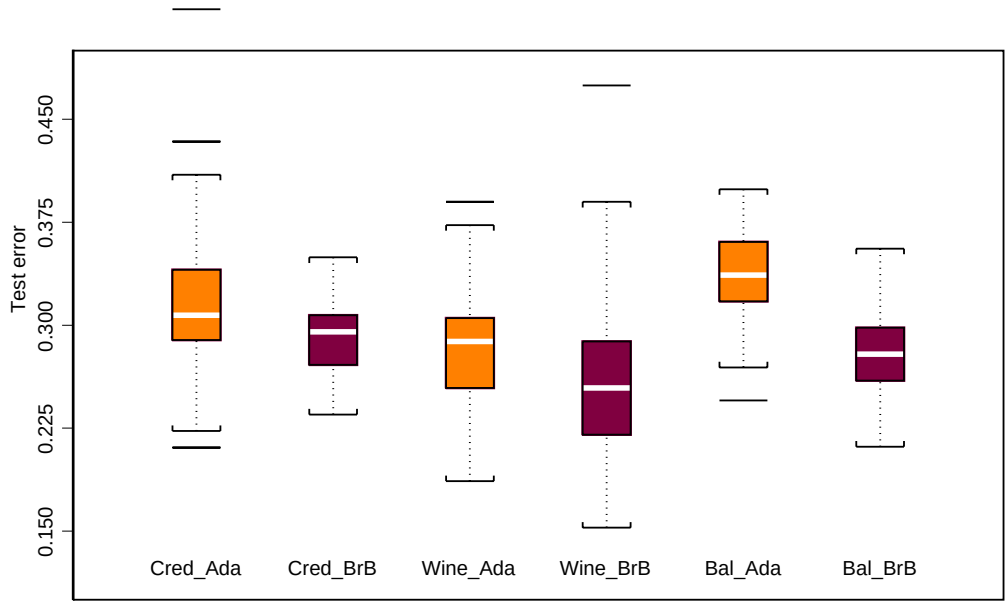

Figure 5: Box plot of the test error rates in Tables 4 and 5 for *Credit, Wine* and *Balance* data.

4.3 Discussion of Experimental Results

Broadly speaking, the results of the previous section bear out claims that Adaboost is especially susceptible to class noise, while providing strong evidence that Brownboost is particularly robust in such situations.

It is immediately evident from the test error rates in Tables 3 and 4 that the introduction of class noise to real data seriously impairs the generalization performance of Adaboost. This would appear to tally with the observations made by Dietterich in [5].

We find that Brownboost is much better able to cope with the presence of noise. The tally counts in Table 2 show that on all the data sets in the experiment, Brownboost yielded a superior test error rate in the majority of trials. The box–plots of in Figures 4 and 5 tell us that in two cases, for the *Sine* and *Balance* data sets, there was no overlap between the 95% confidence intervals on the test error. This allows us to say with good justification that Brownboost proved the better option in these particular instances. However, we can see fair evidence that Brownboost is consistently outperforming Adaboost on all the data sets in our trials.

We can speculate that the fact that the greatest improvements were seen in the *Sine* and *Balance* data sets may well reflect the fact that both had a comparatively very small number of attribute vectors. This may indicate that the 'patchiness' due to overfitting (as seen in Figure 3, panel (4)), is more disparate in high-dimensional domains, and so has less of an adverse effect on generalization performance.

Of course, in a real situation we would be very unlikely to know the level of class noise in advance. In our experiments we were able to calculate the value of c directly given our prior knowledge. It remains to be seen how difficult it would prove to estimate c in practice.

5 Conclusions and Future Work

In this paper we have shown how the method of Error-Correcting output codes can be used to derive a natural and workable multi-class extension to the Brownboost algorithm. This puts it on a level-footing with Adaboost, for which many multi-class variants already exist, and greatly broadens its range of potential applications.

In our experimental trials, we were able to provide good evidence that Brownboost does indeed outperform Adaboost in the presence of systematic class noise, while in all other respects retaining the advantages of that algorithm.

One potential weakness of the Brownboost algorithm at present is its lack of a cost-sensitive variant. There exist certain classes of problem involving unequal initial distributions and assymetric loss functions for which the overall error rate is not necessarily the best measure of loss. For example, a credit-scoring data set will typically contain fewer than 10% defaults. Thus a learning algorithm based only on the error rate can achieve a 90% success rate simply by assigning all examples to class 0. To overcome this, we must reward the algorithm more for correctly identifying the defaulters. Additionally, we may wish to distinguish between the two different types of error that the classifier can make. The incorrect classification of a good customer is sometimes referred to as a Type 1 error or *false negative*, while the misclassification of a defaulter is designated a Type 2 error or *false positive*. From the lender's point of view, a Type 2 error is generally more serious in terms of the loss incurred from a bad debt than a Type 1 error, which results only in the loss of potential profit. Thus the algorithm should assign higher loss to Type 2 errors. In practice the lender will often fix a proportion of the overall population to accept, and then seek to minimize the Type 2 error rate. Cost-sensitive variants of Adaboost already exist (see [7]), but so far there is no analogous version of Brownboost.

In our experiments, we have thus far tested Brownboost on artificial data with artificial class noise, and real data with artificial class noise. It remains to be seen how the algorithm would respond to real data with real class noise, though it seems very probable that Brownboost would perform well in these situations.

It would also be interesting to know whether Brownboost's ability to learn a data set to a fixed level of accuracy could be used to account for *attribute* noise. In the real world we very rarely encounter data sets with clearly defined class boundaries. From a statistical perspective, we would generally model the uncertainty at the interface between classes in terms of overlapping distributions. In the machine learning community, the uncertainty at the overlap is designated attribute noise, and its effects are often ignored. But it is possible that the choice of parameter c would compensate for this in the same way as class noise. If this proves to be the case, then Brownboost's built-in ability to avoid overfitting should prove advantageous in a wide variety of situations.

Acknowledgements

We would like to acknowledge Yoav Freund's work in developing the two-class version of Brownboost, and the enormous contributions of Yoav Freund and Robert Schapire to the field of boosting in general.

RAM was supported in this work by Shell Research Ltd. and research grant number 0130322X from the Engineering and Physical Sciences Research Council.

Appendix A - Adaboost

The following algorithm is the version of Adaboost used in the experiments of Section 4, and is equivalent to Adaboost.MH as described in [24].

Algorithm Adaboost (Multi-Class Version)

Inputs:
ECOC Matrix: The $k \times \ell$ coding matrix $\mathbf{M}$.
Training Set: A set of m labelled examples: $T = (x_n, y_n), n = 1, ..., m$ where $x_n \in \mathbb{R}^d$ and $y_n \in \{y_1, y_2, ..., y_k\}$. Each y_n is associated via the matrix $\mathbf{M}$ with a set of ℓ binary labels $\{\lambda_1^n, ..., \lambda_\ell^n\}$, where $\lambda_j^n \in \{-1, 1\}, \quad j = 1, ..., \ell$.
Weights: An $m \times \ell$ vector of initial weights, say, $W_{1,j}(x_n, y_n) = \frac{1}{m\ell}, \quad n = 1, ..., m, \quad j = 1, ..., \ell$
WeakLearn – A weak learning algorithm.

Do for $i = 1, 2, ..., T$

1. **Binary base learner:** Call **Weaklearn** ℓ times $j = 1, ..., \ell$, each time passing it the weight distribution defined by normalizing $W_{i,j}(x_n, y_n)$ for fixed j, and the training data set alongside the binary labels defined by column j of the matrix $\mathbf{M}$.
   Multi-class base learner: Call **Weaklearn**, passing it the training data and the full set of weights.

   Receive from **Weaklearn** a set of ℓ hypotheses $h_{i,j}(x)$ which have some advantage over random guessing

   $$\frac{\sum_{n=1}^{m} \sum_{j=1}^{\ell} W_{i,j}(x_n, y_n)(h_{i,j}(x_n)\lambda_j^n)}{\sum_{n=1}^{m} \sum_{j=1}^{\ell} W_{i,j}(x_n, y_n)} = \gamma_i > 0, \quad n = 1, ..., m, \quad j = 1, ..., \ell.$$

2. Select $\alpha_i = \frac{1}{2} \ln \left(\frac{1+\gamma_i}{1-\gamma_i} \right)$.
3. Weight update: $W_{i+1,j}(x_n, y_n) = \frac{W_{i,j}(x_n, y_n) \exp(-\alpha_i \ell_j^n h_{i,j}(x_n))}{\sum_{n=1}^{m} \sum_{j=1}^{\ell} W_{i,j}(x_n, y_n)}$.

Output Final hypotheses: $p_j(x) = \text{sign} \left(\sum_{i=1}^{N} \alpha_i h_{i,j}(x) \right), \quad j = 1, ..., \ell.$

Figure 5: A Multi-class Adaboost Algorithm

Appendix B - Theorems and Proofs

The section headings refer to the lemmas, theorems and proofs in [9].

Lemma 1

We can still write our differential equation in the form shown, provided we re-define our index. So for instance, we could sum across $l = (n-1)\ell + j$, for $l = 1, ..., m\ell$. The proof holds as before.

Theorem 2

For a single example and class, within an iteration, we get

$$\frac{d}{d\mathbf{t}}\beta_{(x,j)}(\mathbf{t}) = \frac{2}{c\pi}W_{(x,j)}(\mathbf{t})\left[h_{i,j}(x)\lambda_j\frac{d\alpha}{d\mathbf{t}} - 1\right]$$

We know that

$$\frac{d\alpha}{d\mathbf{t}} = \frac{1}{\gamma}.$$

Plugging in the definition of γ, and averaging over all examples and classes, we get:

$$\begin{aligned}
\frac{d}{d\mathbf{t}}\frac{1}{m\,\ell}\sum_{(x,y)}\sum_{j=1}^{\ell}\beta_{(x,j)}(\mathbf{t}) \quad &= \frac{2}{cm\ell\pi}\left[\frac{1}{\gamma}\sum_{(x,y)}\sum_{j=1}^{\ell}(W_{(x,j)}h_{i,j}(x)\lambda_j) - \sum_{(x,y)}\sum_{j=1}^{\ell}W_{(x,j)}(\mathbf{t})\right] \\
&= \frac{2}{cm\ell\pi}\left[\frac{\sum_{(x,y)}\sum_{j=1}^{\ell}W_{(x,j)}(\mathbf{t})}{\sum_{(x,y)}\sum_{j=1}^{\ell}(W_{(x,j)}(\mathbf{t})h_{i,j}(x)\lambda_j)}\right. \\
&\left.\sum_{(x,y)}\sum_{j=1}^{\ell}(W_{(x,j)}(\mathbf{t})h_{i,j}(x)\lambda_j) - \sum_{(x,y)}\sum_{j=1}^{\ell}W_{(x,j)}(\mathbf{t})\right] \\
&= 0.
\end{aligned}$$

Thus we have shown that Brownboost's key property, the guaranteed final error rate, still holds in the multi-class case for the equivalent pseudo-error.

Theorem 3 and Corollary 4

The proof of this theorem still holds, provided we bear in mind that expectation is now taken across examples *and* labels. This means that where previously a summation was written $\sum_{i=1}^{m}$, we now write $\sum_{n=1}^{m}\sum_{j=1}^{\ell}$, and instead of dividing by m, we divide by $m\ell$.

Theorem 5

Begin by considering the potential for a single example *and* label and remember to sum as for Theorem 3 when calculating the average potential. Remember to use the definition of γ_i as given in 3.5, and to replace $h_i(x)y$ with $h_i(x)\lambda_j$.

Solving the Differential Equation

We can still use either method, provided we make the same modifications as for Theorem 3.

References

[1] *UCI Machine Learning Repository.* `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

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

[3] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting and variants. *Machine Learning*, 36:105 – 142, 1999.

[4] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, U.S., 1984.

[5] T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *AI Magazine*, 18:97 – 136, 1997.

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

[7] W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan. Adacost: Misclassification cost-sensitive boosting. In *16th International Conference on Machine Learning*, 1999.

[8] Y. Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121, 1995.

[9] Y. Freund. An adaptive version of the boost by majority algorithm. *Machine Learning 43*, 3:293 – 318, 2001.

[10] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Second European Conference on Computational Learning Theory*, 1995.

[11] Y. Freund and R. E. Schapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14:771 – 780, 1999.

[12] J. H. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28:337 – 374, 2000.

[13] D. J. Hand. *Construction and Assessment of Classification Rules.* John Wiley & Sons, Chichester, 1997.

[14] D. J. Hand and W. E. Henley. Statistical classification methods in consumer credit scoring: A review. *Journal of the Royal Statistical Society, Series A*, 160:523 – 541, 1997.

[15] R. Hill. *A First Course in Coding Theory.* Clarendon Press, Oxford, 1986.

[16] W. Jiang. Some results on weakly accurate base learners for boosting regression and classification. In *Proceedings of the First International Workshop on Multiple Classifier Systems*, pages 87 – 96, 2000.

[17] O. L. Mangasarian and W. H. Wolberg. Cancer diagnosis via linear programming. *SIAM News*, 23(5):1 – 18, 1990.

[18] E. Mays, editor. *Credit Risk Modeling, Design and Application.* GPCo, 1998.

[19] J. R. Quinlan. The effect of noise on concept learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2, San Mateo, CA, 1986. Morgan Kauffmann.

[20] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[21] J. R. Quinlan. Bagging, boosting and c4.5. *AAAI/IAAI*, 1:725 – 730, 1996.

[22] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197 – 227, 1990.

[23] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26:1651 – 1686, 1998.

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

[25] L. C. Thomas. A survey of credit and behavioural scoring; forecasting financial risk of lending to consumers. *Journal of Forecasting*, 16:149 – 172, 2000.