ICANN'99, September, Edinburgh, UK.

Learning Error-Correcting Output Codes from Data

Ethem Alpaydın*
Dept of Computer Engineering, Boğaziçi University, TR-80815 Istanbul Turkey
alpaydin@boun.edu.tr

Eddy Mayoraz
IDIAP, CP 592, CH-1920 Martigny Switzerland
mayoraz@idiap.ch

Abstract

A *polychotomizer* which assigns the input to one of $K \geq 3$ classes is constructed using a set of *dichotomizers* which assign the input to one of two classes. Defining classes in terms of the dichotomizers is the binary decomposition matrix of size $K \times L$ where each of the $K \geq 3$ classes is written as *error-correcting output codes (ECOC)*, i.e., an array of the responses of binary decisions made by L dichotomizers. We use *linear* dichotomizers and by combining them suitably, we build *nonlinear* polychotomizers, thereby reducing complex decisions into a group of simpler decisions. We propose a new method to learn the error-correcting codes from data based on *soft weight sharing* which forces parameters to take one of a set (here two: $-1/+1$) values. Simulation results on eight datasets indicate that compared with a *linear one-per-class polychotomizer* and ECOC proper, these methods generate more accurate classifiers, using less dichotomizers than *pairwise classifiers*.

1 Introduction

In classification, we have a multidimensional input $\boldsymbol{x}$ that we want to assign to one of a set of mutually exclusive and exhaustive classes $\mathcal{C}_1, \ldots, \mathcal{C}_K$. When $K = 2$, the classifier is a *dichotomizer* and when $K \geq 3$, we implement K *polychotomizers*, one for each class.

Implementing a one-per-class polychotomy directly by separating a class from all other classes may be difficult and require a complex discriminant function. The alternative is to define a set of simpler classifiers, each specializing on one aspect thereby solving a simpler problem and combining these simpler classifiers we get the final classifier.

In our case, the base classifier is a linear dichotomizer with small number of free parameters. That is we use as the building block a hyperplane that separates a set of positive examples from a set of negative examples. To define a polychotomy in terms of dichotomizers, from the original multiclass training set, we generate a set of two-class training sets of positive and negative examples each of which trains one dichotomizer. The $K \geq 3$ classes are defined as a function of the response of an array of binary decisions with enough redundancy added for tolerance to faults.

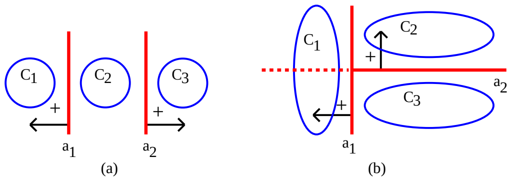
*Supported by Boğaziçi University Research Funds 98HA0101.

2 Writing a Polychotomy in terms of Dichotomies

2.1 Decomposition Matrix

We have a set of dichotomizers $a_l, l = 1, \ldots, L$ where for each, we define a sample of positive examples $\mathcal{X}_l^+$ and a sample of negative examples $\mathcal{X}_l^-$. Any method can be used to train a_l; our approach of using linear dichotomizers is discussed in Section 3. The *decomposition matrix* $D = [d_{kl}]$ of size $K \times L$ associates classes, $\mathcal{C}_k, k = 1, \ldots, K$, to the training samples of the dichotomizers $\mathcal{X}_l^+$ and $\mathcal{X}_l^-, l = 1, \ldots, L$

$$d_{kl} = \begin{cases} +1 & \text{means } \mathcal{C}_k \subset \mathcal{X}_l^+ \\ -1 & \text{means } \mathcal{C}_k \subset \mathcal{X}_l^- \\ 0 & \text{means } \mathcal{C}_k \cap (\mathcal{X}_l^+ \cup \mathcal{X}_l^-) = \emptyset \end{cases}$$



$$D = \begin{bmatrix} +1 & -1 \\ -1 & -1 \\ -1 & +1 \end{bmatrix} \qquad D = \begin{bmatrix} +1 & 0 \\ -1 & +1 \\ -1 & -1 \end{bmatrix}$$

Figure 1: Two examples where linear polychotomizers do not work but linear dichotomizers do with the given decomposition matrices.

Rows of D correspond to the definition of a class as a vector of responses of the L dichotomizers; this is the "code" of class $\mathcal{C}_k$ using the alphabet of dichotomizer outputs. The columns of D define the tasks of the dichotomizers. For example with four dichotomizers, if row k is $[-1, +1, 0, +1]$, for us to say the input belongs to $\mathcal{C}_k$, we should have $a_1 = -1, a_2 = +1, a_4 = +1$ and we do not care for the value of a_3. Whereas with three classes, if column l is $[+1, -1, +1]^T$ then we see that a_l should separate $\mathcal{C}_2$ from $\mathcal{C}_1$ and $\mathcal{C}_3$. We see in Fig. 1 how two problems that are not linearly separable can be solved using linear dichotomizers with the given decomposition matrices.

Once we have a_l and D, given a pattern to classify, all a_l compute their outputs and we assign the pattern to the class having the closest representation (row of D). When the vectors are $-1/+1$, this can be done by taking a dot product and choosing the maximum

$$o_k = \sum_l d_{kl} a_l \qquad (1)$$

$$c = \arg\max_k o_k \qquad (2)$$

2.2 One-per-Class Scheme

In *one-per-class* classification, each polychotomizer separates the examples of its class from the examples of all other classes. This corresponds to using a square ($L = K$) matrix D where $d_{kk} = +1, k = 1, \ldots, K$ and $d_{kl} = -1, k \neq l$ otherwise. That is $\mathcal{X}_k^+ = \{\boldsymbol{x} | \boldsymbol{x} \in \mathcal{C}_K\}$ and $\mathcal{X}_k^- = \{\boldsymbol{x} | \boldsymbol{x} \in \mathcal{C}_l, k \neq l\}$.

For example for $K = 4$, we have

$$D = \begin{bmatrix} +1 & -1 & -1 & -1 \\ -1 & +1 & -1 & -1 \\ -1 & -1 & +1 & -1 \\ -1 & -1 & -1 & +1 \end{bmatrix}$$

2.3 Pairwise Classifiers

Another approach frequently used [4] is to build $K(K-1)/2$ *pairwise classifiers* each being trained to separate examples of one class from examples of one other class, not using the examples of other classes except these two. Matrix D is of size $K \times K(K-1)/2$. For example for $K = 4$, we have

$$D = \begin{bmatrix} +1 & +1 & +1 & 0 & 0 & 0 \\ -1 & 0 & 0 & +1 & +1 & 0 \\ 0 & -1 & 0 & -1 & 0 & +1 \\ 0 & 0 & -1 & 0 & -1 & -1 \end{bmatrix}$$

One problem is that the number of dichotomizers is $\mathcal{O}(K^2)$. Another is that each dichotomizer is trained with data from a smaller training set of only two classes and thus has higher variance. Related to this fact is that the response of dichotomizer separating $\mathcal{C}_i$ from $\mathcal{C}_j$ gives no information at all when the example does not belong to $\mathcal{C}_i$ or $\mathcal{C}_j$. To alleviate these, a smaller number of dichotomizers which are all trained with the whole dataset is desirable.

2.4 Error-Correcting Output Codes

To get robustness in case the dichotomizers fail, we should have rows of D as different as possible in terms of Hamming distance by adding redundancy; this is the idea of *error-correcting output codes* (ECOC) [2] and is used to decompose polychotomies into dichotomies [6, 8]. With K classes, we have up to $2^{(K-1)} - 1$ dichotomizers. For example, for $K = 4$, the D matrix is

$$D = \begin{bmatrix} +1 & +1 & +1 & +1 & +1 & +1 & +1 \\ -1 & -1 & -1 & -1 & +1 & +1 & +1 \\ -1 & -1 & +1 & +1 & -1 & -1 & +1 \\ -1 & +1 & -1 & +1 & -1 & +1 & -1 \end{bmatrix}$$

When K is large, all $2^{(K-1)} - 1$ dichotomizers cannot be built. Instead, a subset of size L is chosen where the Hamming distances between the codes are chosen as large as possible to minimize error.

Dietterich and Bakiri [2] report that with C4.5 trees, the dichotomizers trained to learn the ECOC dichotomizers are larger and when multilayer perceptrons are used as dichotomizers, there may be a problem of convergence. These imply that the tasks defined by the columns of D may be more difficult than the original problem of separating one class from all others. For a three-class problem as those in Fig. 1, ECOC generates the following matrix

$$D_3 = \begin{bmatrix} +1 & +1 & +1 \\ -1 & -1 & +1 \\ -1 & +1 & -1 \end{bmatrix}$$

In Fig. (1-a), the second column which indicates separating $\mathcal{C}_2$ from $\mathcal{C}_1$ and $\mathcal{C}_3$ is not linearly separable. In (b), the second and third columns are nonrealizable with linear dichotomizers.

3 ML Training of a Linear Dichotomizer

We are given a labelled sample $\mathcal{X} = \{\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)}\}$ where

$$y_i^{(t)} = \begin{cases} 1 & \text{if } \boldsymbol{x}^{(t)} \in \mathcal{C}_i \\ 0 & \text{otherwise} \end{cases}$$

Let us assume that for each dichotomizer a_l, using one of the techniques above, the K classes are partitioned into two, as given by the decomposition matrix D. We define the set of positive classes: $\mathcal{C}_l^+ = \cup_{k:D_{kl}=+1}\mathcal{C}_k$, and the set of negative classes: $\mathcal{C}_l^- = \cup_{k:D_{kl}=-1}\mathcal{C}_k$. Similarly we define the set of positive examples $\mathcal{X}_l^+$ containing all $\boldsymbol{x}^{(t)}$ whose class is in $\mathcal{C}_l^+$ and $\mathcal{X}_l^-$ contains all $\boldsymbol{x}^{(t)}$ whose class is in $\mathcal{C}_l^-$; if for any class i, $D_{il} = 0$ then its examples are neither in $\mathcal{X}_l^+$ nor $\mathcal{X}_l^-$. We associate labels with examples in $\mathcal{X}_l^+$ and $\mathcal{X}_l^-$ such that

$$y_l^{(t)} = \begin{cases} 1 & \text{if } \boldsymbol{x}^{(t)} \in \mathcal{C}_l^+ \\ 0 & \text{if } \boldsymbol{x}^{(t)} \in \mathcal{C}_l^- \end{cases}$$

We want dichotomizer l to approximate $o_l^{(t)} \equiv P(\mathcal{C}_l^+|\boldsymbol{x}^{(t)})$, the probability of the positive class given that $\boldsymbol{x}^{(t)}$ is observed, which, in the case of a linear model is

$$o_l = \frac{1}{1+\exp[-\boldsymbol{w}_l^T\boldsymbol{x}]} \qquad (3)$$

Table 1: Properties of the datasets used.

	d inputs	K classes	N size
`glass`	9	7	214
`vowel`	10	11	990
`thyroid`	21	3	5,200
`satimage`	36	7	6,435
`letter`	16	26	20,000
`digsma`	256	10	1200
`optdigits`	64	10	3,731
`pendigits`	16	10	7,246

where $\boldsymbol{w}_l$ includes a bias term and is learned to minimize the $-\log$ likelihood [1]

$$E = -\sum_t y_l^{(t)} \ln o_l^{(t)} + (1-y_l^{(t)})\ln(1-o_l^{(t)})$$

In later sections, we will assume that a dichotomizer gives an output in the range $[-1,+1]$ instead of $[0,1]$ and we postprocess

$$a_l = 2o_l - 1 \qquad (4)$$

4 First Results

We compare the techniques we discussed on eight datasets (Table 1) from the UCI repository [7]. We first compare the three techniques we give above using linear dichotomizers: These are one-per-class separation (OPC), pairwise separation (PW), and Error-Correcting Output Codes (ECOC). Results are given in Table 2. We use the 5x2cv t test [3] for statistically significant difference, except on `letter` where only one run is done and the McNemar test [3] is used.

Note that ECOC does not perform as well as OPC. It may be that in the high dimensional space, classes lie in far away corners and it is possible to linearly separate one class from others. But when they are grouped, they cannot be separated linearly. In five of the eight datasets, the accuracy of PW is significantly superior to that of OPC. This is not due to having larger L however; ECOC with as large L is not as accurate. This indicates that classes are pairwise linearly separable; adding other classes, as ECOC does, is harmful. Instead of assuming codes a priori, we need to find a way of "learning" them taking into account the hypothesis bias of our dichotomizers, e.g., linear separability.

Table 2: Comparison of linear one-per-class polychotomizer, ECOC-linear and pairwise linear dichotomizers. Values are L: average, stdev accuracies of ten independent runs on the test set, except on `letter` where only one run is made. '+' indicates statistically significantly (0.95) better than the one-per-class scheme. '–' means worse.

	One-per-cla	ECOC-lin	Pairwise
g	7: 61.9, 3.9	7: 58.9, 3.3	21: 61.9, 3.6
v	11: 44.2, 1.0	11: 24.8, 4.7 -	55: 66.4, 2.5 +
t	3: 95.5, 0.3	3: 94.7, 0.4 -	3: 95.1, 0.4
s	7: 85.4, 0.3	7: 81.0, 2.8 -	21: 86.4, 0.3 +
l	26: 76.4	26: 39.8 -	325: 77.9 +
d	10: 95.3, 0.8	20: 90.8, 0.7 -	45: 97.5, 0.8
o	10: 96.2, 0.5	20: 91.2, 0.9 -	45: 97.6, 0.2 +
p	10: 91.0, 1.4	20: 84.3, 1.2 -	45: 94.6, 0.4 +

5 Learning ECOC

We have a set of dichotomizers $a_l, l = 1, \ldots, L$ generated by ECOC (or any other method like decision trees). The output for class k, as given in Eq. (1) is a weighted sum of the dichotomizers with entries of matrix D as weights. This combination can be seen as another linear model where a_l are the "inputs" and d_{kl} are the "weights" and thus d_{kl} can be trained given a sample of a_l and the corresponding desired outputs. So given a sample of $\{\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)}\}$, for each $\boldsymbol{x}^{(t)}$, we can compute the outputs of the dichotomizers and generate the vector $\boldsymbol{a}^{(t)}$. Then using the sample $\{\boldsymbol{a}^{(t)}, \boldsymbol{y}^{(t)}\}$, we can learn the weights d_{kl}. Note that what we are training is a *polychotomizer*. Using softmax instead of a sigmoid, we get the estimators for the posteriors $o_k \equiv P(\mathcal{C}_k|\boldsymbol{x})$

$$o_k = \frac{\exp[\sum_{l=0}^{L} d_{kl} a_l]}{\sum_{i=1}^{K} \exp[\sum_{l=0}^{L} d_{il} a_l]} \tag{5}$$

with biases, d_{k0}, added. With $K \geq 3$, we minimize

$$E = -\sum_t \sum_k y_k^{(t)} \ln o_k^{(t)} \tag{6}$$

We can now use e.g., gradient-descent, to learn d_{kl}. However here there is no restriction on the values that d_{kl} may take. If we want to have them binary, i.e., $-1/+1$, there are two ways we can do this.

5.1 Soft Weight Sharing

We assume that d_{kl} (nonbias weights only) are sampled from a bimodal density, e.g., mixture of two Gaussians [9, 1] centered at -1 and $+1$

$$p(d) = \alpha_1 p_1(d) + \alpha_2 p_2(d) \tag{7}$$

where $\alpha_j, j = 1, 2$ are priors and p_j are Gaussian components $p_j(d) \sim \mathcal{N}(m_j, s_j^2)$

$$p_j(d) = \frac{1}{\sqrt{2\pi} s_j} \exp\left[-\frac{(d - m_j)^2}{2s_j^2}\right] \tag{8}$$

with $m_1 = -1$ and $m_2 = +1$. We want to maximize the likelihood that d_{kl} are drawn from this bimodal distribution

$$l(d) = \prod_{k,l} \sum_{j=1}^{2} \alpha_j p_j(d_{kl}) \tag{9}$$

Defining $\Omega = -\ln l(d)$, we convert this to an error term to be minimized, named the *regularization term* which we add to Eq. (6)

$$\begin{aligned} E_{\text{reg}} &= E + \nu\Omega \\ &= -\sum_{t,k} y_k^{(t)} \ln o_k^{(t)} - \nu \sum_{k,l} \ln \sum_j \alpha_j p_j(d_{kl}) \end{aligned} \tag{10}$$

ν is the *regularization coefficient* and gives the relative importance of the two terms. Eq. (10) can be justified from a Bayesian perspective [9, 1], which implies that d_{kl} that minimize Eq. (10) is the MAP estimate. Using gradient-descent, we get

$$\begin{aligned} \Delta d_{kl} &= \eta \sum_t (y_k^{(t)} - o_k^{(t)}) a_l^{(t)} \\ &- \eta\nu \sum_{j=1}^{2} \pi_j(d_{kl}) \frac{(d_{kl} - m_j)}{s_j^2} \end{aligned}$$

where the posteriors are given as

$$\pi_1(d) = \frac{\alpha_1 p_1(d)}{\alpha_1 p_2(d) + \alpha_2 p_2(d)} \tag{11}$$

and $\pi_2(d) = 1 - \pi_1(d)$. s_j and α_j are also learned to minimize Eq. (10); see [9, 1] for details. We can also use three Gaussians instead of two, with the third centered at 0; this has the advantage that the dichotomizers whose d converge to 0 are effectively eliminated from the class description.

5.2 Mixture of Dichotomizers

Another possibility that is simpler to implement is to have auxiliary variables z_{kl} and

$$d_{kl} = \tanh(\beta z_{kl}) \tag{12}$$

where β defines the sharpness. tanh forces d_{kl} to be in $[-1, +1]$ saturating in the two extremes where the derivative is zero. Plugging Eq. (12) into Eq. (5), we can learn z_{kl} by backpropagating to minimize Eq. (6)

$$\Delta z_{kl} = \eta \sum_t (y_k^{(t)} - o_k^{(t)}) a_l^{(t)} (1 - {d_{kl}}^2) \beta$$

A further possibility which we can name *Mixture of Dichotomizers*, is to make d_{kl} input dependent, just like the gating network of a *Mixture of Experts* architecture [5].

$$d_{kl} = \tanh(\boldsymbol{v}_{kl}^T \boldsymbol{x}) \tag{13}$$

This architecture allows us to have different D in different parts of the input space. We can again backpropagate with respect to $\boldsymbol{v}_{kl}$ and train them using labelled data.

$$\Delta \boldsymbol{v}_{kl} = \eta \sum_t (y_k^{(t)} - o_k^{(t)}) a_l^{(t)} (1 - {d_{kl}}^2) \boldsymbol{x}$$

5.3 Finetuning of Dichotomizers

Placing Eqs. (3, 4) into Eq. (5) and "backpropagating," we can also learn the dichotomizer parameters, $\boldsymbol{w}_l$. Instead of Eqs. (3, 4), let us define

$$a_l = \tanh(\boldsymbol{w}_l^T \boldsymbol{x}) \tag{14}$$

and we have $a_l \in [-1, +1]$. Note that now what we have is a multilayer perceptron (MLP) where a_l are the "hidden units," $\boldsymbol{w}_l$ are the "first layer weights" and d_{kl} are the "second layer weights." Each hidden unit is a linear dichotomizer and MLP is a combiner of linear dichotomizers to construct a polychotomizer. If the MLP has two hidden layers or more, the hidden units in the last hidden layer can be viewed as *nonlinear* dichotomizers.

Though this structure can be trained from random initial $\boldsymbol{w}_l$ and d_{kl} (Note that there is no prior on the $\boldsymbol{w}_l$ so their update equations are as in backprop proper), we propose to use a method like ECOC to initialize the structure. In our simulations, we noticed that the three schemes of guaranteeing binary d_{kl} have similar accuracies and

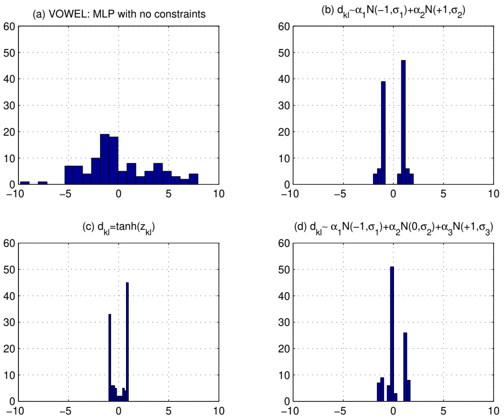


Figure 2: For one run on `vowel` with 10 hidden units (dichotomizers), the histograms of weights are given. (a) Second layer weights of a normal MLP, (b) Second layer weights of an MLP with two Gaussians at -1 and 1, (c) Like (b) except that weights filtered through tanh, (d) Like (b) except there are three Gaussians with the third at 0.

preferred Eq. (12) due to its simplicity. To initialize the auxiliary variables from d values set by ECOC, we use an approximate inverse of tanh

$$z_{kl} = \begin{cases} +1.5 & \text{if } d_{kl} = +1 \\ -1.5 & \text{if } d_{kl} = -1 \end{cases} \tag{15}$$

We take values a little bit off the saturation regions for the derivative to be nonzero. Having now constructed a bona fide MLP, we can now finetune the dichotomizers and the decomposition matrix by training them in a coupled manner as the first and second layer weights. During this finetuning, we use a small learning factor η to not destroy the structure generated by ECOC.

The advantage of this structure is that, when compared with pairwise classification, it uses less dichotomizers and all dichotomizers are trained with the whole sample. And compared with ECOC proper, the partitioning of classes is not done a priori but is coupled with the training of the dichotomizers such that any deficiency in one can be corrected by other dichotomizers.

6 Simulation Results

In Figure 2, we see that with the constraints added, the second layer weights of an MLP do converge to a bimodal distribution without losing accuracy. We notice that the approach given in Eq. (12) where auxiliary parameters are filtered through tanh converge better than soft weight sharing with two Gaussians. Our method is able to learn with arbitrary L, which may be much less than $K(K-1)/2$ as required by pairwise separation, without losing from accuracy.

As shown in Table 3, finetuning significantly improves accuracy in four out of eight datasets (Table 3). We also see that though ECOC proper was worse than OPC, ECOC finetuned is better in five of the eight datasets. Similarly, ECOC finetuned is more accurate than PW in three of the eight datasets and worse in three; this is despite the fact that PW uses much larger number of dichotomizers.

Though we use linear dichotomizers trained with maximum likelihood, any other method, e.g., decision trees, can be used to learn a_l and our method can be used to learn the error-correcting codes, i.e., the right way to combine them. When K is large, there are many different sets of error-correcting codes of length L possible. Some of them may fit the hypothesis bias of our learners, e.g., linear separability in the case of linear a_l, and some may not. By finetuning these codes from data, we make the codes better suited to the bias of the a_l we have at hand.

Note that this approach is also interesting in that ECOC defines a target value for the hidden units of the MLP in a classification problem. This allows dividing the task of training a two-layer perceptron into the training of several one-layer perceptrons for which faster convergence can be attained, and the training can also be parallelized if several processors are available. When the MLP is viewed this way, it is no longer a black box that is difficult to interpret but each hidden unit is a dichotomizer with a well-defined task and the second layer is the definition of the classes in terms of these dichotomizers.

One possible direction for future work is generalizing for the decomposition matrix to be a function of the input. This allows having different ways of combining the dichotomizers in different parts of the input space and would allow representing more complicated discriminants.

Table 3: Accuracy with ECOC-linear before and after finetuning. '+' in Column 4 indicates statistically significant improvement during finetuning. Columns 5 and 6 are comparisons with one-per-class and pairwise separation results in terms of significance.

	ECOC-linear				
	L: Raw	Finetuned		1pc	ps
g	7: 58.9, 3.3	67.0, 3.8		+	+
v	11: 24.8, 4.7	47.7, 4.2		+	−
t	3: 94.7, 0.4	94.8, 0.1			
s	7: 81.0, 2.8	88.2, 0.5	+	+	+
l	26: 39.8	89.4	+	+	+
d	20: 90.8, 0.8	90.6, 1.4		−	−
o	20: 91.2, 0.9	93.2, 0.5	+	−	−
p	20: 84.3, 1.2	94.9, 0.6	+	+	

References

[1] Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*, Oxford.

[2] T. G. Dietterich, G. Bakiri (1995). "Solving Multiclass Learning Programs via Error-Correcting Output Codes," *J. Artificial Intell. Res.*, **2**, 263–286.

[3] T. G. Dietterich (1998). "Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms," *Neural Comp*, **10**, 1895–1923.

[4] R. Duda, P. Hart (1973). *Pattern Classification and Scene Analysis*, Wiley.

[5] R. Jacobs, M. Jordan, S. Nowlan, G. Hinton (1991) "Adaptive Mixtures of Local Experts," *Neural Comp*, **3**, 79–87.

[6] E. Mayoraz, M. Moreira (1997). "On the Decomposition of Polychotomies into Dichotomies", *14th Int'l Conf on Machine Learning*, 219–226.

[7] C. Merz, P. Murphy (1998). UCI Repository of ML Databases.

[8] M. Moreira, E. Mayoraz (1998). "Improved Pairwise Coupling Classification with Correcting Classifiers", *10th Euro Conf Machine Learning*, 160–171.

[9] S. Nowlan, G. Hinton (1992). "Simplifying Neural Networks by Soft Weight Sharing," *Neural Comp*, **4**, 473–493.