Available online at www.sciencedirect.com
ScienceDirect
Pattern Recognition Letters 28 (2007) 1747–1758
Pattern Recognition Letters
www.elsevier.com/locate/patrec

Approximating the multiclass ROC by pairwise analysis

Thomas C.W. Landgrebe *, Robert P.W. Duin

Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, P.O. Box 5031 2600 GA Delft, Mekelweg 4 2628CD Delft, The Netherlands

Received 1 September 2006; received in revised form 18 March 2007
Available online 16 May 2007

Communicated by K. Tumer

Abstract

The use of Receiver Operator Characteristic (ROC) analysis for the sake of model selection and threshold optimisation has become a standard practice for the design of two-class pattern recognition systems. Advantages include decision boundary adaptation to imbalanced misallocation costs, the ability to fix some classification errors, and performance evaluation in imprecise, ill-defined conditions where costs, or prior probabilities may vary. Extending this to the multiclass case has recently become a topic of interest. The primary challenge involved is the computational complexity, that increases to the power of the number of classes, rendering many problems intractable. In this paper the multiclass ROC is formalised, and the computational complexities exposed. A pairwise approach is proposed that approximates the multi-dimensional operating characteristic by discounting some interactions, resulting in an algorithm that is tractable, and extensible to large numbers of classes. Two additional multiclass optimisation techniques are also proposed that provide a benchmark for the pairwise algorithm. Experiments compare the various approaches in a variety of practical situations, demonstrating the efficacy of the pairwise approach.

Keywords: ROC analysis; Multiclass ROC; Cost sensitive; Threshold optimisation

1. Introduction

In pattern recognition, the goal is to choose/optimise representations and classifiers that provide acceptable discriminability between the various classes. Typically one output exists per class, with a new incoming object being assigned to the highest output. The outputs can be weighted[1] in order to vary the trade-offs that exist between classes. These weights are called the operating weights[2].

Once a suitable classifier model has been found, the next step is typically to optimise these operating weights to suit the given problem. For example in the equal/minimum-error rate situation (Duda et al., 2001), e.g. face detection (Pham et al., 2002), errors should be the same for all classes. In detection problems it is often the case that some errors should be fixed, and the others minimised (see, e.g. Edwards et al., 2004; Landgrebe et al., 2005). In cost-sensitive problems, different classification outcomes have associated costs/penalties (Bishop, 1995), and so the optimisation problem is to find a set of operating weights that result in the lowest overall loss/risk. Other optimisation scenarios exist, such as the necessity to choose and optimise the best classifier when class priors/costs are unknown (see Provost and Fawcett, 2001; Adams and Hand, 1999), or varying (Landgrebe et al., 2006). At this point we emphasise that of interest in this case is the optimisation of classifier operating weights, and not on internal parameters of the model. Any internal adjustment of the model e.g. varying the

* Corresponding author. Tel.: +31 (0) 15 27 88433; fax: +31 (0) 15 27 81843.
E-mail addresses: t.c.w.landgrebe@ewi.tudelft.nl (T.C.W. Landgrebe), r.p.w.duin@ewi.tudelft.nl (R.P.W. Duin).

[1] Even non-probabilistic classifiers, e.g. support vector classifiers can be weighted in this manner – outputs in this case are distances to support vectors, which are analogous to probabilities (Li and Sethi, 2006).

[2] Conceptually similar to the two-class concept of "thresholds". In the multiclass case, weighting of the output allows for a generalisation of the ROC, with the final discrimination decision based on the highest weighted output.

0167-8655/$ - see front matter
doi:10.1016/j.patrec.2007.05.001

thresholds in a one-vs-all multiclass scheme, results in a new model, and a new operating characteristic.

Receiver Operator Characteristic (ROC) analysis (Metz, 1978; Fawcett, 2005) was developed for the two-class case, allowing the classifier threshold to be studied for all possible combinations of the two respective classification errors involved, namely the false positive rate, and the false negative rate (with each combination known as the *operating point*). As such, this tool has become standard in the aforementioned optimisation scenarios, but applying it to the multiclass case is more challenging, and poorly understood. Recently some work has begun in this area, such as an extension to the three-class case in (Mossman, 1999), a study of the feasibility of the extension in (Srinivasan, 1999), and investigating some formalisations in (Ferri et al., 2003; Edwards et al., 2004). For the imprecise case in which priors/costs are unknown, the work in (Hand and Till, 2001) presents an approximate method for evaluating classifiers in these conditions, extending the well-known Area Under the ROC (AUC) measure (Bradley, 1997), resulting in the approximate VUS (Volume Under the ROC Surface).

However, even though several works have investigated multi-class operating characteristics, no known methods exist that result in the practical construction of a multiclass ROC that extends to problems involving large numbers of classes, which is necessary in order to perform the optimisations and analyses as discussed earlier. This is attributed primarily to the computational complexity of the analysis. Recently two approaches emerged that are useful for multiclass cost-sensitive optimisation, capable of using input costs and priors to optimise operating weights. The first is a hill-climbing approach as presented in (Lachiche et al., 2003) that can easily be adapted to the cost-sensitive case, and the second is an evolutionary approach (Everson et al., 2005), attempting to find a global solution to the minimisation problem. The problem with the former approach is that it is very susceptible to local minima, dealing with interactions between classes sub-optimally. The algorithm optimises the classifier operating weights successively, which is not ideal for arbitrary interactions, but nevertheless improves performance in many cases. The latter approach has only been demonstrated in problems with low numbers of classes. The approach involves sampling of operating points, which becomes less tractable as the number of classes increases, due to an exponential computational complexity (shown later).

In this paper the multiclass ROC approach is formalised, and computational complexity investigated as a function of both the number of classes, and the resolution of the operating characteristic. This shows that computing the multiclass ROC is feasible for low numbers of classes (C), but rapidly becomes intractable as C increases. A new multiclass ROC approach is proposed, in which ROC curves are generated between each type of classification error, characterising the interaction between each pair, but ignoring other interactions. In the cost-sensitive scenario, these characteristics are interrogated to obtain the most optimal operating weight pairs to suit the priors and costs. This involves exhaustively searching pair combinations in order to select the most appropriate weight pairs, followed by a normalisation procedure that "calibrates" weights against each other. The number of weight pairs is reduced for large C problems by excluding weight pairs exhibiting little/no interaction to reduce computation required. Even though this algorithm is not optimal, it will be shown that it is both computationally tractable, and performs well over many real problems. In addition to the pairwise multiclass ROC method, this paper also presents two simple approaches to the cost-sensitive optimisation problem, consisting of a naive approach that ignores interactions between operating weights, and a greedy-search approach that accounts for interactions (to an extent). These methods provide a benchmark with which to compare the pairwise approach. As a further benchmark, the algorithm in (Lachiche et al., 2003) is also included.

The paper is structured as follows: a notational overview and formalism is presented in Section 2, followed by a formalisation of multiclass ROC construction and an analysis of computational requirements in Section 3. The naive and greedy multiclass cost-sensitive optimisation algorithms are presented in Section 4, followed by the proposed all-pairs approach in Section 5. A variety of experiments are discussed in Section 6, demonstrating the various algorithms in cost-sensitive scenarios. Conclusions are given in Section 7.

2. Notation and formalisation

Consider a C-class problem, with each class denoted $\omega_1, \omega_2, \ldots, \omega_C$, and new observations characterised by vector $\boldsymbol{x}$, with dimensionality d. The class conditional probability of ω_i is denoted $p(\boldsymbol{x}|\omega_i)$, with prior probability $p(\omega_i)$. Class assignment is based on the highest posterior output, denoted $p(\omega_i|\boldsymbol{x})$, for the ith class, formalised as

$$\operatorname{argmax}_{i=1}^{C} p(\omega_i|\boldsymbol{x}) \tag{1}$$

The posterior is computed via Bayes formula:

$$p(\omega_i|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\omega_i)p(\omega_i)}{p(\boldsymbol{x}|\omega_1)p(\omega_1) + p(\boldsymbol{x}|\omega_2)p(\omega_2) + \cdots + p(\boldsymbol{x}|\omega_C)p(\omega_C)} \tag{2}$$

A multiclass classifier is evaluated by inspection of a $C \times C$ dimensional confusion rate matrix $\boldsymbol{\xi}$ as defined in Table 1. Each element is referenced as $\xi_{i,j}$. In order to compute each confusion $\xi_{i,j}$ (the fraction of ω_i classified as ω_j weighted by priors), the following integration is performed:

$$\xi_{i,j} = p(\omega_i) \int p(\boldsymbol{x}|\omega_i) I_j(\boldsymbol{x}) \mathrm{d}x \tag{3}$$

The indicator function $I_j(\boldsymbol{x})$ specifies the relevant domain:

$$I_j(\boldsymbol{x}) = \begin{cases} 1 & \text{if } p(\omega_j|\boldsymbol{x}) > p(\omega_k|\boldsymbol{x}) \quad \forall k, k \neq j \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Table 1
Defining the multi-class confusion rate matrix $\boldsymbol{\xi}$

True	Estimated			
	ω_1	ω_2	$\cdots$	ω_C
ω_1	$\xi_{1,1}$	$\xi_{1,2}$	$\cdots$	$\xi_{1,C}$
ω_2	$\xi_{2,1}$	$\xi_{2,2}$	$\cdots$	$\xi_{2,C}$
$\vdots$	$\vdots$		$\ddots$	
ω_C	$\xi_{C,1}$	$\xi_{C,2}$	$\cdots$	$\xi_{C,C}$

Table 2
Defining the multi-class cost-matrix $\boldsymbol{s}$

	ω_1	ω_2	$\cdots$	ω_C
ω_1	$s_{1,1}$	$s_{1,2}$	$\cdots$	$s_{1,C}$
ω_2	$s_{2,1}$	$s_{2,2}$	$\cdots$	$s_{2,C}$
$\vdots$	$\vdots$		$\ddots$	
ω_C	$s_{C,1}$	$s_{C,2}$	$\cdots$	$s_{C,C}$

Eq. (3) allows any confusion matrix output to be computed, generalised for both diagonal elements (performances), and off-diagonal elements (errors). In the practical case where distributions are unknown, and only representative examples per class are available, a confusion matrix ***cm*** is constructed, generated via application of a representative independent test set. These ***cm*** outputs are normalised by the absolute number of objects N_i per class ω_i, $\boldsymbol{N} = [N_1, N_2, \ldots, N_C]^{\mathrm{T}}$, resulting in the confusion rate matrix, where each element $\xi_{i,j} = \frac{\boldsymbol{cm}_{i,j}}{N(i)}$.

Each posterior output $p(\omega_i|\boldsymbol{x})$ can be weighted by the scalar ϕ_i, $\phi_i \geqslant 0$, in order to control trade-offs between the various classification errors (note that all classifier outputs are scaled between [0, 1], irrespective of the classifier type). The classifier weight vector $\boldsymbol{\Phi} = [\phi_1, \phi_2, \ldots, \phi_C]$ is thus the mechanism for manipulating a classifier's decision boundary. Note that there are $C - 1$ degrees of freedom, and thus in the two-class case, $\boldsymbol{\Phi} = [\phi_1, (1 - \phi_1)]$, since there is only one degree of freedom. Similarly, where $C > 2$, one of the weights is generally set to an arbitrary positive value, e.g. for a five-class problem $\boldsymbol{\Phi} = [1, \phi_2, \phi_3, \phi_4, \phi_5]$. Class assignment can thus be modified as

$$\operatorname{argmax}_{i=1}^{C} \phi_i p(\omega_i|\boldsymbol{x}) \tag{5}$$

In the cost-sensitive case, the optimisation goal is to minimise the various classification errors, using a framework in which the importance of each error is weighted via a classification *cost*, and the respective prior probability. The cost-sensitive situation is typically evaluated by considering the overall system *loss*, L, given a matrix of costs $\boldsymbol{s}$ (see Table 2), and prior probabilities. Now the loss can be defined (in this paper the profits (diagonals) are set to zero):

$$L = \sum_{i=1}^{C} p(\omega_i) \left(\sum_{j=1, i \neq j}^{C} \xi_{i,j} s_{i,j} \right) - \sum_{i=1}^{C} p(\omega_i) \xi_{i,i} s_{i,i} \tag{6}$$

In the left of Fig. 1, a four-class example is shown for two different operating points, consisting of Gaussian-distributed classes $\omega_1, \omega_2, \ldots, \omega_4$, with means occurring at $\mu_1 = -3$, $\mu_2 = 0$, $\mu_3 = 3$, $\mu_4 = 6$, respectively, with unit variance. The left plot shows an operating point involving equal priors and $\boldsymbol{\Phi} = [1.0, 1.0, 1.0, 1.0]$, and the right plot shows another operating point $\boldsymbol{\Phi} = [1.0, 0.3, 1.0, 1.0]$, in which $\xi_{1,2}$ decreases at the expense of $\xi_{2,1}$.

3. Multiclass ROC

3.1. Implementation

Referring to Fig. 1, the plots show only two operating points, corresponding to two different operating weight settings. In fact, any combination of weightings results in a different operating point. The challenge in multiclass optimisation is in understanding the relation between a weight modification and the corresponding alteration of the confusion matrix, which depicts the consequences of the new operating point. Multiclass operating weight (analogous to threshold) optimisation is thus the process by which the optimal set of weights $\boldsymbol{\Phi}^*$ is found to suit

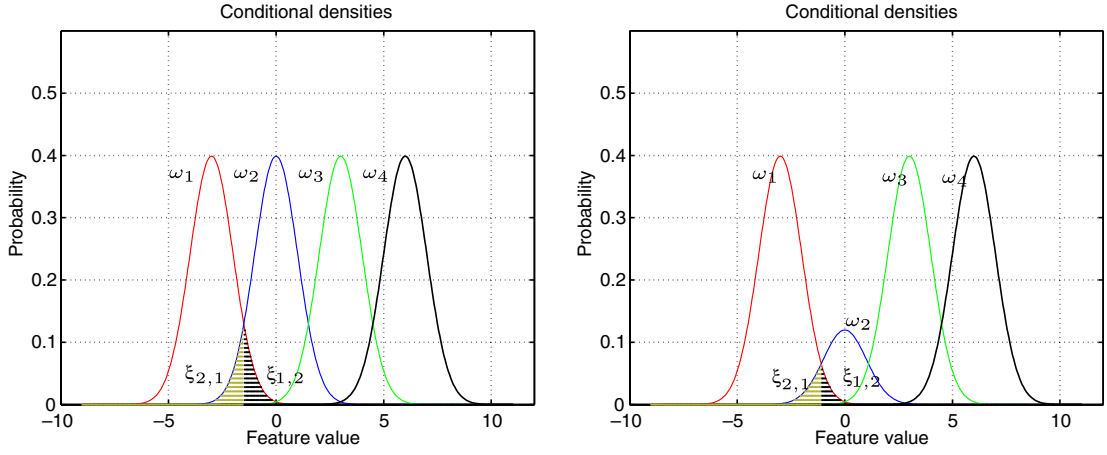

Fig. 1. Probability density functions for the four-class example with known distributions. Two operating points are shown, with the left plot involving equal output weighting, and the right with a higher ϕ_2 and lower ϕ_1.

the problem at hand. Note that there may be multiple optimal solutions for $C > 2$. Multiclass ROC analysis involves the generation of a hypersurface consisting of all possible combinations of Φ, allowing these various possible confusion matrices to be characterised. The dimensionality of the hypersurface is $C^2 - C$ (diagonal elements are superfluous), which can be constructed by adapting Eq. (3). In this case, each output between class i and j is weighted by ϕ_i as follows:

$$\xi_{i,j}(\Phi) = \phi_i p(\omega_i) \int p(\boldsymbol{x}|\omega_i) I_j(\boldsymbol{x}|\Phi) \mathrm{d}x \tag{7}$$

The indicator function $I_j(\boldsymbol{x}|\Phi)$ is as in 4, except each posterior is multiplied by the corresponding class weight:

$$I_j(\boldsymbol{x}|\Phi) = \begin{cases} 1 & \text{if } \phi_j p(\omega_j|\boldsymbol{x}) > \phi_k p(\omega_k|\boldsymbol{x}) \\ & \quad \forall k,\ k = 1, 2, \ldots C, k \neq j \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

Since the classifier has $(C-1)$ degrees of freedom, the ROC is constructed by generating a $(C-1)$ dimensional grid of all possible operating weights $\boldsymbol{\Phi}$, with resolution r. This results in a set of confusion rate matrices, corresponding to each weight combination, denoted $\boldsymbol{\xi}(\boldsymbol{\Phi})$. Different elements of the confusion matrices as a function of Φ are the dimensions of the ROC. An important consideration for practical implementation is the choice of the resolution r and scale Θ of each weight. The resolution must be fine enough, and the scale adequately chosen to ensure the operating characteristic is well sampled. Experiments in this paper consider a logarithmic scale, sampled 80 times, with $10^{-3} \leqslant \Theta \leqslant 10^{3}$.

The full multiclass ROC has been computed for the analytic example (see Fig. 1), of which 3 of the 12 $(4 \times 4 - 4)$ ROC dimensions are plotted in Fig. 2, illustrating the respective interactions. The full ROC in this case results in $80^{4-1} = 512 \times 10^3$ operating points.

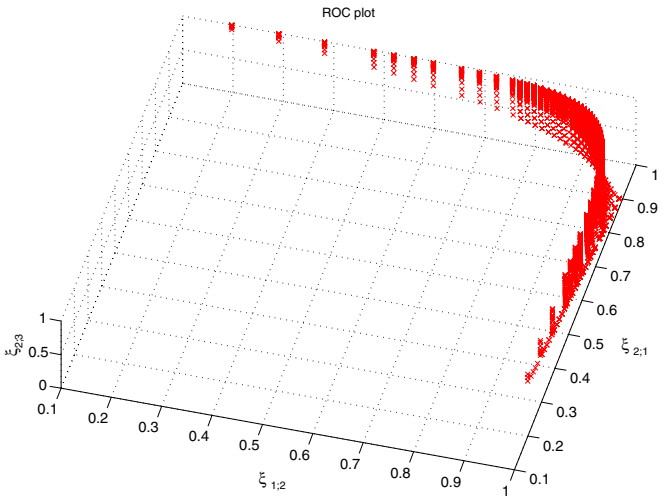

Fig. 2. Illustrating the ROC dimensions $\xi_{1,2}$, $\xi_{2,1}$, and $\xi_{2,3}$ for the example.

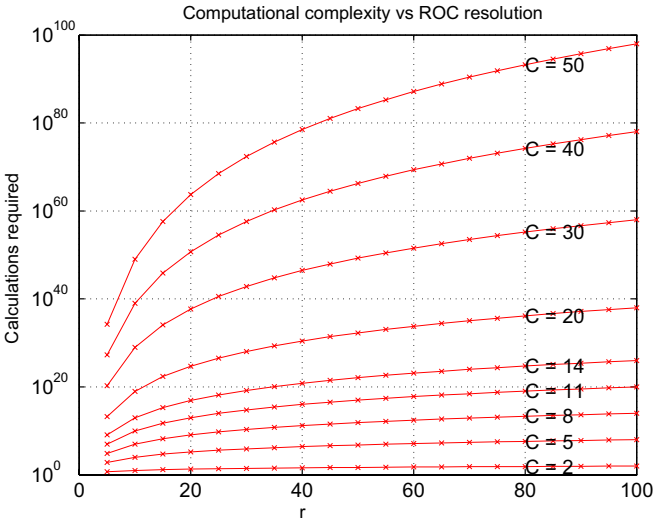

Fig. 3. The relationship between computational complexity, ROC resolution, and C.

3.2. Computational considerations

In the two-class case, ROC construction involves generating a one-dimensional grid of weights (since there are $C-1$ weights, there is only 1 DOF in this case), with r steps across the output range. In the multiclass case $(C > 2)$, computing the ROC involves the generation of a $C-1$ dimensional grid with r steps, resulting in r^{C-1} operating points. This increase to the power of the number of classes minus 1 ($O(r^{C-1})$) explodes the computational complexity with increasing C, becoming infeasible to compute for all but low C problems. To illustrate the severity of this problem, Fig. 3 plots the number of calculations (and number of memory slots for storage of the hypersurface) as a function of the ROC resolution r, for a number of C-values. It is clear that for high C, computational complexity becomes prohibitive. Use of a lower r does reduce the computation required, but this is at the expense of poorer sampling, which could lead to a poorly sampled ROC surface[3]. To illustrate, consider the following real problems:

- *Satellite* (obtained from ELENA (2004)), which consists of six classes of multi-spectral remote sensing data. An ROC resolution of 80 requires $80^{6-1} = 3.28 \times 10^9$ calculations.
- *Letter* (obtained from Murphy et al. (1992)) consists of 26 different classes of hand-written digits, requiring $80^{26-1} = 3.78 \times 10^{47}$ calculations for $r = 80$, which is clearly prohibitive.

Thus, even though it is theoretically possible to construct any multiclass ROC, only problems with low C are

[3] Different classifier models, e.g. a support vector classifier or a Bayes linear classifier, typically result in outputs (e.g. posteriors) with different scales, and thus choosing a scale suitable to the classifier used could improve the computational situation by allowing for a lower r.

feasible to compute. This is the justification for seeking approximate techniques for performing multiclass ROC analysis.

4. Naive and greedy cost-sensitive optimisation algorithms

In the cost-sensitive case, obtaining a set of operating weights to suit new costs $\boldsymbol{s}$ and priors $\boldsymbol{p}$ can be viewed as an optimisation problem (Everson et al., 2005). In this Section two simple algorithms are presented that are suitable for this task. We stress that these approaches simply result in a single set of weights, and not an operating characteristic, and are thus unable to fulfil several functions that are made possible by ROC analysis. However, they are still useful and relatively straightforward in the cost-sensitive scenario, allowing for an adaptation of an unoptimised classifier (the "default" classifier). These algorithms also provide a basis for comparison with respect to the pairwise ROC approach.

The objective of both algorithms is to obtain the most appropriate weight vector Φ^*, achieved by searching for a solution that reduces an initial system loss (using Eq. (6)) by varying individual classifier weights. These approaches cannot guarantee a global minimum, but should always improve on an initial "default" classifier that was trained to a typically equal-error operating point (and accounting for class priors). This initial classifier results in a baseline confusion rate matrix used to obtain the initial loss for a given set of conditions (equivalent to operating weights set to unity).

4.1. Naive multiclass cost-sensitive optimisation

The *Naive* algorithm attempts to optimise operating weights to given cost $\boldsymbol{s}$ and prior probability $\boldsymbol{p}$ conditions by varying each operating weight successively and independently, and inspecting overall system loss (Eq. (6)). This approach ignores interaction between different weights, and simply "optimises" each one in turn.

The following steps are taken: a threshold vector Θ is computed, $10^{-3} \leqslant \Theta \leqslant 10^{3}$, with resolution r, using a logarithmic scale. In the first step the first weight ϕ_1 is optimised, resulting in ϕ_1^*. This is obtained by varying ϕ_1 according to Θ, while other weights are fixed, and computing the loss. The minimum loss for this weight is denoted $L^*(\phi_1|\phi_i = 1\ \forall i, i \neq 1)$, which is then used to obtain ϕ_1^*, computed using Eq. (6):

$$L^*(\phi_1|\phi_i = 1\ \forall i, i \neq 1) = \min(L(\phi_1 = \Theta)), \quad \phi_1^* = \Theta(L_1^*) \tag{9}$$

This "optimises" the first weight independently of the others. The same procedure is then followed for all other weights, resulting in an "optimal" weight value in each case, ignoring all interactions:

$$\begin{aligned}
&L^*(\phi_2|\phi_i = 1\ \forall i, i \neq 2) = \min(L(\phi_2 = \Theta)), \quad \phi_2^* = \Theta(L_2^*)\\
&L^*(\phi_3|\phi_i = 1\ \forall i, i \neq 3) = \min(L(\phi_3 = \Theta)), \quad \phi_3^* = \Theta(L_3^*)\\
&\vdots\\
&L^*(\phi_k|\phi_i = 1\ \forall i, i \neq k) = \min(L(\phi_k = \Theta)), \quad \phi_k^* = \Theta(L_k^*)\\
&\vdots\\
&L^*(\phi_C|\phi_i = 1\ \forall i, i \neq C) = \min(L(\phi_C = \Theta)), \quad \phi_C^* = \Theta(L_C^*)
\end{aligned} \tag{10}$$

Even though the *Naive* algorithm is sub-optimal, it has a computational complexity of $O(rC)$ (where r is the resolution of Θ), extending linearly with C, and is thus scalable to high C problems. The experiments (Section 6) show that this approach is generally better than an un-optimised approach, but is usually outperformed by other more sophisticated algorithms that do account for interactions.

4.2. Greedy multiclass cost-sensitive optimisation

The *Greedy* multiclass optimisation algorithm is quite similar to the *Naive* approach, except that some degree of interaction between weights is accounted for. This is achieved by searching for a (local) optimal set of operating weights by optimising weights with respect to each other in a greedy manner. This involves randomly selecting a weight to update, followed by optimisation relative to other weights. Subsequently another operating weight is randomly selected, and optimised while taking into account previously optimised weights. This process is repeated until all weights are accounted for. In an attempt to avoid local minima, the algorithm is typically run a number of times with different random initialisations. The algorithm is as follows: a threshold vector Θ is computed as in the *Naive* case, and the weight vector Φ is randomly ordered, resulting in Φ^R, with the ith weight denoted ϕ_i^R. The first weight ϕ_1^R is optimised, resulting in ϕ_1^{R*}, by considering all possible Θ, while other weights are fixed, and computing the loss (Eq. (6)). The minimum loss for this weight is denoted $L^*(\phi_1^R|\phi_i^R = 1\ \forall i, i \neq 1)$, which is then used to obtain ϕ_1^{R*}, computed using Eq. (6):

$$L_1^*(\phi_1^R|\phi_i^R = 1\ \forall i, i > 1) = \min(L(\phi_1^R = \Theta)), \quad \phi_1^{R*} = \Theta(L_1^*) \tag{11}$$

The *Greedy* algorithm then proceeds to optimise other weights in the order as per Φ^R. The primary difference to the *Naive* algorithm is that weights are now updated *dependent* on previously updated weights, as follows:

$$
\begin{aligned}
&L^*(\phi_2^R|\phi_1^R = \phi_1^{R*}, \phi_i^R = 1\ \forall i, i > 2)\\
&\quad = \min(L(\phi_2^R = \Theta)), \phi_2^{R*} = \Theta(L_2^*)\\
&L^*(\phi_3^R|\phi_1^R = \phi_1^{R*}, \phi_2^R = \phi_2^{R*}, \phi_i^R = 1\ \forall i, i > 3)\\
&\quad = \min(L(\phi_3^R = \Theta)), \phi_3^{R*} = \Theta(L_3^*)\\
&\vdots\\
&L^*(\phi_k^R|\phi_j^R = \phi_j^{R*}\ \forall j, j < k, \phi_i^R = 1\ \forall i, i > k)\\
&\quad = \min(L(\phi_k^R = \Theta)), \phi_k^{R*} = \Theta(L_k^*)\\
&\vdots\\
&L^*(\phi_C^R|\phi_j^R = \phi_j^{R*}\ \forall j, j < C) = \min(L(\phi_C^R = \Theta)), \phi_C^{R*} = \Theta(L_C^*)
\end{aligned}
\tag{12}
$$

The *Greedy* algorithm has a computational complexity of $O(N_r r\ C)$, where N_r is the number of algorithm repetitions. This algorithm is thus also extensible to large C. A similar algorithm was also recently proposed in (Everson et al., 2005), using a more sophisticated search approach.

5. Pairwise multiclass ROC analysis

The pairwise ROC algorithm investigates interactions between each pair of operating weights by analysing ROC plots between each pair. Only the respective pair operating weights are varied (with other weights held constant), and the resultant confusion rate matrices stored. For a given problem, the pairs that are most suitable are then chosen to be used. This results in a much more efficient algorithm than the full multiclass ROC, since only individual pairings are considered. The algorithm is simplified further by discounting pairs which are approximately separable based on the Area Under the ROC (AUC) criterion. The limitation of this pairwise approach is that the pairs discount interactions not included in the pair, leading to sub-optimality, but we argue that accounting for the most important interactions may result in a good approximation. In an attempt to cater for some degree of further interaction, the algorithm is followed by a post-processing step, using the *Greedy* algorithm in Section 4.

Given a C-class problem, the algorithm proceeds as follows: In the first step a vector of weight pair indices PI is constructed, consisting of $\frac{C^2-C}{2}$ pairs:

$$
\begin{aligned}
\mathrm{PI} = [&[\phi_1, \phi_2], [\phi_1, \phi_3], [\phi_1, \phi_4], \ldots, \quad [\phi_1, \phi_C], [\phi_2, \phi_3],\\
&[\phi_2, \phi_4], \ldots, [\phi_2, \phi_C], \ldots, [\phi_{C-1}, \phi_C]]
\end{aligned}
\tag{13}
$$

The next step involves computing an ROC curve corresponding to each pair $[\phi_i, \phi_j]$, $j > i$, denoted $\mathrm{ROC}(\phi_i, \phi_j)$. This is performed via Eq. (7), varying weights ϕ_i and ϕ_j only, and weights $\phi_k = 1,\ k = 1, 2, \ldots, C,\ k \neq i, j$. This process results in $\frac{C^2-C}{2}$ ROC plots that can be analysed or interrogated to suit a given problem. In the full multiclass case, the operating characteristic can be analysed directly, but in this case, a secondary step is required to amalgamate information from the most relevant ROC pairs.

Consider for example the cost-sensitive case given a cost $\boldsymbol{s}$ and prior $\boldsymbol{p}$. The task is to select the best ROC pairs to suit the new situation, but there is an ambiguity in the pairwise case, since the same operating weight is optimised $(C-1)$ times e.g. ROC (ϕ_2, ϕ_3) and ROC (ϕ_2, ϕ_4) both result in an optimised ϕ_2. The pairwise algorithm selects the pair best suited to the problem by considering all possible pairings. This is sub-optimal because interactions between other classifier weights not in the pair are now ignored, but in some problems certain interactions may be more significant than others. In addition, some pairs may involve more costly implications than others, in which case these pairs should be favoured. Thus the philosophy of the pairwise algorithm is to consider both the degree of interaction, and the severity of an interclass error, resulting in the most optimal pair selection for the given scenario. This is practically achieved by considering all feasible combinations of pairs of results, and selecting the combination with the lowest overall loss. The number of possible unique combinations of pairings is denoted N_{PC}, computed as follows (not to be confused with the total number of pairs):

$$
N_{\mathrm{PC}} = (C-1)(C-3)(C-5)\ldots 1 \tag{14}
$$

For example, in the six-class case, the following PI results ($N_{\mathrm{PC}} = (6-1)(6-3)(6-5) = 15$ in this case):

$$
\begin{aligned}
\mathrm{PI} = [&[\phi_1, \phi_2], [\phi_1, \phi_3], [\phi_1, \phi_4], [\phi_1, \phi_5], [\phi_1, \phi_6],\\
&[\phi_2, \phi_3], [\phi_2, \phi_4], [\phi_2, \phi_5], [\phi_2, \phi_6], [\phi_3, \phi_4],\\
&[\phi_3, \phi_5], [\phi_3, \phi_6], [\phi_4, \phi_5], [\phi_4, \phi_6], [\phi_5, \phi_6]]
\end{aligned}
\tag{15}
$$

Next, a matrix PI_c is constructed from the PI pairs. Each row consists of a different complete set of class pairs covering all classes. For example, in the six-class case, PI_c is

$$
\begin{bmatrix}
\mathrm{PI}(1) & \mathrm{PI}(10) & \mathrm{PI}(15)\\
\mathrm{PI}(1) & \mathrm{PI}(11) & \mathrm{PI}(14)\\
\mathrm{PI}(1) & \mathrm{PI}(12) & \mathrm{PI}(13)\\
\mathrm{PI}(2) & \mathrm{PI}(7) & \mathrm{PI}(15)\\
\mathrm{PI}(2) & \mathrm{PI}(8) & \mathrm{PI}(14)\\
\mathrm{PI}(2) & \mathrm{PI}(9) & \mathrm{PI}(13)\\
\mathrm{PI}(3) & \mathrm{PI}(6) & \mathrm{PI}(15)\\
\mathrm{PI}(3) & \mathrm{PI}(8) & \mathrm{PI}(12)\\
\mathrm{PI}(3) & \mathrm{PI}(9) & \mathrm{PI}(11)\\
\mathrm{PI}(4) & \mathrm{PI}(6) & \mathrm{PI}(14)\\
\mathrm{PI}(4) & \mathrm{PI}(7) & \mathrm{PI}(12)\\
\mathrm{PI}(4) & \mathrm{PI}(9) & \mathrm{PI}(10)\\
\mathrm{PI}(5) & \mathrm{PI}(6) & \mathrm{PI}(13)\\
\mathrm{PI}(5) & \mathrm{PI}(7) & \mathrm{PI}(11)\\
\mathrm{PI}(5) & \mathrm{PI}(8) & \mathrm{PI}(10)
\end{bmatrix}
\tag{16}
$$

In the odd case, one weight will not be included, since it cannot be paired. This weight is assigned a finite positive value (e.g. 1), and is excluded from the optimisation, chosen based on lack of importance (e.g. low cost, or low degree of interaction).

Now we have obtained a set of pairwise ROC curves $\mathrm{ROC}(\phi_i, \phi_j)$, $i = 1, 2, \ldots, C, j > i$, and the PI_c matrix, which indicates how the ROC curves can be interrogated based on

feasible pairings. We now focus on the cost-sensitive case to illustrate how this ROC approximation can be used. Each pair is optimised using the given $\boldsymbol{s}$ and $\boldsymbol{p}$, resulting in pairs of optimal weights. These values are stored in a matrix with the same size as PI according to the PI pairing, denoted PI^{Φ}. In the next step, all the possible optimised weight pairings are assembled according to PI_c, and re-ordered to correspond with the operating weight ordering, denoted Φ^M.

At this stage an additional step is included to "calibrate" operating weight pairs with respect to other pairs. This is necessary because the pairwise weight optimisation considers the loss associated with the two respective weights, resulting in an optimal weighting between them, but the overall weighting may differ in scale with other groups. Thus it is important to adjust/"calibrate" the various weight pairs with respect to each other. Importantly, the relative values in a weight pair are held constant, but the overall weighting is adjusted in a greedy fashion using an approach similar to the *Greedy* algorithm, with a computational complexity of $\mathrm{O}(\frac{r}{2}(C^2-C))$. The "calibrated" weight pairs are denoted Φ^{Mc}.

Subsequent to the computation of Φ^{Mc}, the best weight vector entry is chosen (each row in Φ^{Mc} is one possible operating weighting) by computing the overall system loss according to Eq. (6) for each entry, and choosing the weight vector resulting in the minimum loss, denoted Φ^*. A final optional post-processing step attempts to overcome the assumption of pairwise interactions only by adjusting Φ^* according to the *Greedy* algorithm in Section 4, resulting in Φ^{**}.

Computationally, the pairwise algorithm increases in complexity as follows: firstly $(\frac{1}{2}(C^2-C))$ ROC pairs must be computed, each of which have a complexity $\mathrm{O}(r)$, resulting in an $\mathrm{O}(\frac{r}{2}(C^2-C))$ calculation, which is tractable even for large C. The next step is to compute all possible pairings, resulting in N_{PC} pairs according to Eq. (14), followed by the "calibration" step, resulting in an $\mathrm{O}(\frac{rN_{\mathrm{PC}}}{2}(C^2-C))$. While problems with lower C are quite tractable (e.g. $N_{\mathrm{PC}} = 48$ for $C=7$, and $N_{\mathrm{PC}}=105$ for $C=8$), larger C problems become less so (e.g. $N_{\mathrm{PC}}=10{,}395$ for $C=12$). A useful approach to reduce the computational complexity for large C problems is to remove pairs that are considered unimportant. Fewer pairs implies a smaller N_{PC}, reducing the complexity radically for high C problems. These pairs are chosen based on the degree of interaction – little or no interaction implies that these pairs will play an insignificant role in the optimisation process. The AUC criterion is applied here $\mathrm{AUC}(\phi_i,\phi_j) = 1-\int(\xi_{i,j})\mathrm{d}\xi_{j,i}$, measuring the degree of separability for each pair, with high AUC values implying a low degree of interaction. A threshold t_p is used to eliminate these pairs.

In summary, the pairwise algorithm proceeds as follows (for the cost-sensitive case):

- Weight pair indices NI are computed according to Eq. (13), resulting in $\frac{C^2-C}{2}$ pairs.
- ROC curves are produced for each pair in NI.
- For large C problems, the AUC criterion is applied to each ROC pair, eliminating pairs demonstrating little interaction according to a chosen threshold t_p.
- All possible pairwise combinations PI_c are calculated, resulting in N_{PC} possibilities (or fewer if pairs were removed in the previous step).
- Given a new cost/prior situation, each ROC is optimised, resulting in a set of optimised weight pairs.
- Optimised weight pairs are assembled and ordered to create Φ^M, according to PI_c.
- Each weight pair is "calibrated" to normalise the overall scaling of pair groups with each other, resulting in Φ^{Mc}.
- Each candidate weight vector in Φ^{Mc} is used to compute the overall loss according to Eq. (6), with best solution chosen as Φ^*.
- An optional *Greedy* optimisation (Section 4) is applied to Φ^*, resulting in Φ^{**}, attempting to overcome the limitation of the pairwise assumption.

5.1. Pairwise optimisation example

An example of the cost-sensitive pairwise ROC analysis approach is discussed, referring to the synthetic problem in the left of Fig. 4 (called the *PRTools8* dataset), generated by the PRTools pattern recognition toolbox (Duin, 2000), consisting of eight classes, with balanced priors. In this experiment, a Bayes quadratic discriminant has been trained on 500 independent training examples, as depicted in the left of Fig. 4, showing the resultant default decision boundary on an independent test set with 500 examples. In this equal-cost, equal-prior case, it can be seen that the classifier attempts to maintain the equal-error state. A cost-sensitive scenario is considered, in which balanced priors are present, but the following cost matrix $\boldsymbol{s}$ occurs:

0.000	0.003	0.113	0.115	0.169	0.133	0.113	0.101
0.044	0.000	0.101	0.140	0.007	0.189	0.183	0.084
0.063	0.136	0.000	0.102	0.061	0.045	0.092	0.159
0.090	0.273	0.235	0.000	0.078	0.016	0.113	0.002
0.023	0.013	0.049	0.048	0.000	0.048	0.049	0.009
0.114	0.091	0.054	0.081	0.029	0.000	0.016	0.090
0.002	0.059	0.057	0.037	0.000	0.006	0.000	0.041
0.030	0.047	0.089	0.068	0.102	0.060	0.087	0.000

The default classifier results in a loss of 0.9359 in this example. The pairwise ROC algorithm attempts to reduce this by finding a new set of operating weights, resulting in a loss of 0.6250 (with no post-processing), which is an improvement over the default case. The following operating weights result:

$$\Phi^* = [0.1000, 0.9000, 0.7875, 0.2125, 0.9250, 0.0750, 0.7875, 0.2125] \quad (17)$$

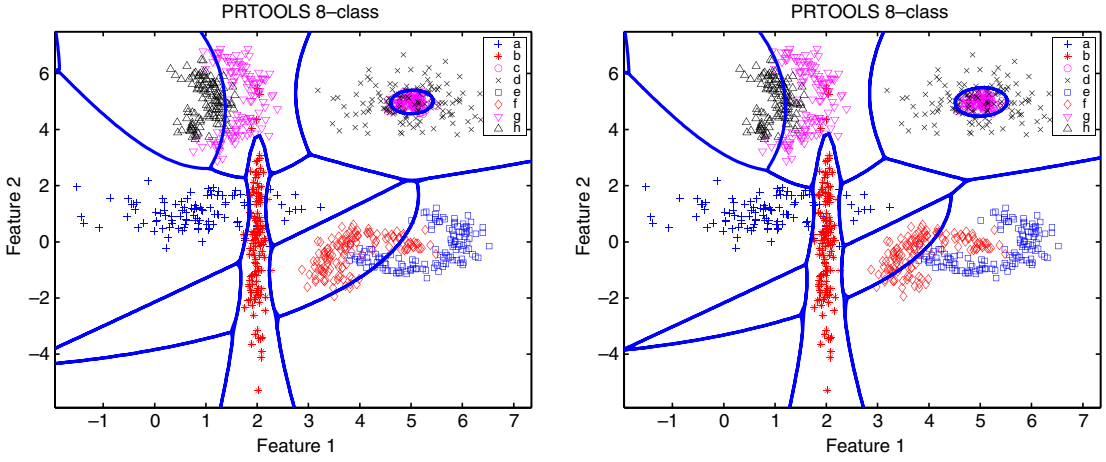

Fig. 4. Comparing decision boundaries for an eight-class problem in the balanced prior/cost case (left), and in the imbalanced case optimised using the pairwise algorithm in the right plot.

These operating weights are applied to the trained classifier, resulting in the decision boundary depicted in the right plot. For comparative purposes, the *Naive* algorithm resulted in a loss of 0.8081 which is better than the default case, but worse than the pairwise algorithm and the *Greedy* algorithm resulted in a loss of 0.6347, which is similar to the pairwise algorithm result.

6. Experiments

A number of experiments are undertaken over a variety of datasets in order to assess the applicability and efficacy of the pairwise multiclass ROC approach, as well as the *Naive* and *Greedy* approaches in the cost-sensitive scenario. The hill-climbing algorithm from Lachiche et al. (2003) is also compared, called *Lachiche*. Two sets of experiments are performed, consisting of *synthetic* and *real* datasets. The *synthetic* experiments analyse performance of the various algorithms in synthetic scenarios suited to the pairwise algorithm, and also investigate the impact of a growing number of classes. The *real* experiments compare performance in realistic scenarios. A trained classifier which is unoptimised (the "default" classifier) is used as the basis for the comparison. The various problems are studied over many different misallocation cost scenarios in order to assess the strengths, weaknesses, and generality of the various algorithms over many situations.

6.1. Methodology

Each dataset is analysed, and an appropriate classifier trained (independently), and tested on an independent test set. The chosen classifiers are not necessarily optimal, since the objective is to demonstrate how a classifier can be tuned to a new operating point, rather than how to design a classifier.

The experimental methodology involves the generation of 50 different random cost-matrices (sampled uniformly, and zero diagonal elements), with priors held constant, and comparing the results of the various multiclass cost-sensitive optimisation algorithms. Each different cost-matrix is a new scenario, weighting both inter- and intra-class errors in a variety of different combinations, with the experimental objective of obtaining a diverse set of scenarios with which to fairly compare different algorithms. The results of the various algorithms are benchmarked against the default unoptimised classifier, with the loss measure used to evaluate performance (Eq. (6)). For each new set of costs, the following algorithms are compared:

- *Default* algorithm, in which no optimisation occurs, and the overall loss is simply computed given the default confusion matrix.
- *Naive* algorithm, as described in Section 4.
- *Greedy* algorithm, as described in Section 4, with $N_r = 3$.
- *Lachiche* algorithm, implemented according to Lachiche et al. (2003).
- *Pairs* algorithm, which is the multiclass pairwise algorithm as described in Section 5.
- *PairsG* algorithm, which is the same as the *Pairs* algorithm, but the post-processing *Greedy* optimisation step is included, using the first stage to initialise the search, attempting to overcome the pairwise limitation.

Even though loss (or cost) is a useful measure to observe relative differences between algorithms, it is dependent on the number of classes, and the priors, and scales according to the costs. This makes it difficult to assess the absolute benefit of an optimisation, and how well the classifier is performing for a given set of costs and priors. In (McDonald, 2006), the L measure is rescaled using $\boldsymbol{s}$ and $\boldsymbol{p}$ into the Mean Subjective Utility Score (MSU), that overcomes these issues, resulting in a performance measure (higher scores imply improvement) that scales between 0 and 1:

$$\mathrm{MSU} = \sum_{i=1}^{C}\left(\sum_{j=1, i\neq j}^{C} \xi_{i,j} s'_{ij}\right) \tag{18}$$

$$
\begin{aligned}
S' &= -\beta_1 S + \beta_2 \\
&- \beta_1 \left(\sum_{i=1}^{C} p(\omega_i) s_{i,i} \right) + \beta_2 = 1 \\
&- \beta_1 \left(\frac{1}{C} \sum_{i=1}^{C} p(\omega_i) \sum_{j=1, j\neq i}^{C} s_{i,j} \right) + \beta_2 = 0
\end{aligned} \tag{19}
$$

This measure can now be used to gauge the percentage improvement that an optimisation may have, instead of using an arbitrary scale. For example, in Section 5.1, the *Default* classifier resulted in a loss of 0.9359, and the pairwise optimisation reduced this to 0.6250. The degree of improvement is not known, but the MSU scores in this case are 0.8016 and 0.8675, respectively, implying that the pairwise approach has improved performance by 6.59%. Thus the MSU score is chosen to compare results since we can immediately assess the impact an optimisation has on performance.

In summary, for each experiment, 50 different cost-sensitive scenarios are considered for the various approaches, each of which results in a new operating weight vector. This is used to weight the classifier outputs, resulting in a new confusion matrix based on the independent test set. Each approach is compared to the default unoptimised classifier, with the MSU measure showing the percentage improvement. The experiments consider the overall improvement the various approaches give over the various scenarios. Note that if an optimisation results in a performance worse than the default case, the improvement in performance is considered to be zero.

6.2. Dataset descriptions

The various algorithms are evaluated by considering a wide variety of problems. In Table 3, the experimental datasets are summarised in terms of training/test sizes, numbers of classes, and dimensionality. Also included are the number of pairs involved in each case, followed by the AUC threshold t_p used (if any), which reduces the numbers of pairs used, shown in the 'Pairs2' column. The table then shows which classifier has been chosen in each case, as well as the mean classification error on the default classifier $\epsilon = \text{mean}(\sum_{i=1}^{C}\sum_{j=1}^{C}\xi_{i,j}, i \neq j)$ based on the test set. The classifier 'qdc' is a Bayes quadratic classifier, 'mogc' is a Bayes mixture of Gaussians classifier followed by the number of mixtures, 'pca' is a principal components feature extraction, followed by the number of components used, and 'fisherm' is a Fisher projection on the original data. The first two entries are the *synthetic* datasets, with the remainder consisting of *real* datasets. The *PRTools8*, *Letter*, and *Satellite* datasets have been introduced earlier. The *PRTools16* dataset is a second synthetic dataset based on the *PRTools8* dataset, with a repetition of the eight-classes in feature space, resulting in 16 classes (simply by adding a value of 10 to both features). The *CBands* dataset consists of chromosome band profiles (Houtepen, 1994). The *Digits* dataset consists of examples of 10 hand-written digits, originating from Dutch utility maps (available from Murphy et al., 1992). In this dataset, Fourier components have been extracted from the original images, resulting in a 76-dimensional representation of each digit. Comparing t_p for the *Letter* and *Cbands*, it can be seen that the *Cbands* appears to contain more interactions, since more pairs result even with a lower threshold, suggesting that there is a higher "intrinsic complexity" than in the *Letter* case. Computationally, the pairwise algorithms for the *PRTools16*, *Letter*, *CBands*, and *Digits* datasets used a reduced number of pairs by eliminating the least interacting pairs. This reduced the N_{PC} in each case, radically reducing the computation required, since this becomes the determining factor with respect to computational complexity when C is large.

6.3. Results

In Table 4, the results of the various algorithms are summarised over the 50 different cost-sensitive scenarios for each dataset. Each result subtracts the MSU score of the respective algorithm with the MSU result (%/100) of the default unoptimised classifier. Higher results thus indicate larger improvements by the respective algorithm. The table attempts to result in an overall analysis of the use of each algorithm, showing the minimum, median, maximum, best, and mean performance over the 50 different conditions.

The *PairsG* and *Greedy* algorithms are the two most promising approaches, which are compared directly via Table 5. These results show the number of experiments (out of 50) that the *PairsG* algorithm was superior.

In order to illustrate the experimental process in detail, the full results of the *Letter* dataset are shown in Fig. 5,

Table 3
Important dataset statistics, showing number of train/test objects, the number of classes, dimensions, number of pairwise ROC curves required, the pairwise AUC threshold t_p, followed by the actual number of pairs used ('Pairs2'), the classifier model, and the mean classification error ϵ of the unoptimised classifier

Dataset	Train/test	C	d	Pairs	t_p	Pairs2	Classifier	ϵ (%)
PRTools8	500/500	8	2	28	–	28	qdc	13.05
PRTools16	1000/1000	16	2	120	0.996	17	qdc	13.05
Letter	16,000/4000	26	16	325	0.980	22	fisherm qdc	12.50
Cbands	6000/6000	24	30	276	0.975	34	pca10 mogc2	17.63
Digits	1000/1000	10	76	45	0.980	10	ldc	20.50
Satellite	4435/2000	6	17	15	–	15	fisherm mogc2	12.50

Table 4
Comparing the various algorithms for each dataset, summarising results over 50 different cost-sensitive scenarios

Dataset	Algorithm	Min	Median	Best	Mean
PRTools8	Naive	0.000	0.027	0.108	0.030
	Greedy	0.021	0.054	0.147	0.058
	Lachiche	0.000	0.000	0.071	0.002
	Pairs	**0.023**	0.057	**0.148**	0.060
	PairsG	**0.023**	**0.058**	**0.148**	**0.061**
PRTools16	Naive	0.000	0.022	0.094	0.022
	Greedy	0.017	0.047	0.132	0.055
	Lachiche	0.000	0.000	0.000	0.000
	Pairs	**0.019**	0.050	**0.135**	**0.057**
	PairsG	**0.019**	**0.051**	**0.135**	**0.057**
Letter	Naive	0.003	0.015	0.026	0.015
	Greedy	0.018	0.027	0.039	0.028
	Lachiche	0.000	0.000	0.000	0.000
	Pairs	0.014	0.024	0.036	0.024
	PairsG	**0.021**	**0.031**	**0.042**	**0.032**
Cbands	Naive	0.000	0.017	0.045	0.019
	Greedy	0.013	0.030	0.060	0.031
	Lachiche	0.000	0.000	0.000	0.000
	Pairs	0.009	0.025	0.054	0.026
	PairsG	**0.014**	**0.032**	**0.061**	**0.032**
Digits	Naive	0.005	0.056	0.184	0.064
	Greedy	0.033	0.081	0.199	0.090
	Lachiche	0.000	0.000	0.164	0.011
	Pairs	0.034	0.082	0.199	0.094
	PairsG	**0.037**	**0.086**	**0.201**	**0.098**
Satellite	Naive	0.000	0.009	0.084	0.007
	Greedy	0.000	0.025	0.120	0.035
	Lachiche	0.000	0.000	0.082	0.005
	Pairs	0.000	0.033	**0.135**	0.040
	PairsG	0.000	**0.034**	0.134	**0.041**

Each value represents the difference between the algorithm MSU performance and the default classifier performance, showing the minimum (Min), median (Median), maximum (Best), and mean (Mean) performance across the 50 runs. Larger differences are favourable. The best performers per statistic are highlighted in bold.

Table 5
Comparing the Pairwise algorithm with post-processing, with the Greedy algorithm, indicating the percentage of experiments in which the *PairsG* algorithm was superior

Dataset	*PairsG* > *Greedy* (%)
PRTools8	80
PRTools16	96
Letter	98
Cbands	70
Digits	88
Satellite	74

comparing the various MSU results for 50 different cost-sensitive scenarios. It can be seen that the default unoptimised classifier is consistently inferior. The *Naive* algorithm is consistently better than the default classifier, but is inferior to the *Greedy* and *PairsG* algorithms. The *PairsG* algorithm is superior in most cases, suggesting that this is generally the best approach. The *Pairs* algorithm (omitted) did not work well by itself, suggesting higher order (i.e. >2) interactions. However, the superior performance of the

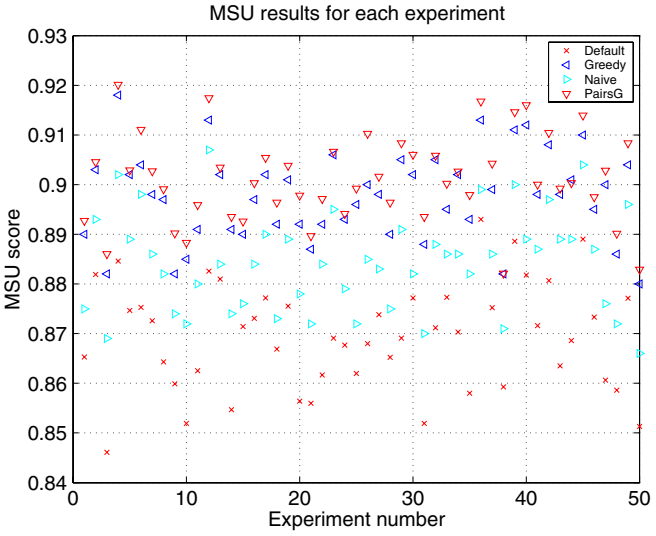

Fig. 5. Detailed results for the *Letter* dataset, comparing MSU performance (larger scores are better) across 50 different experiments, involving randomly varying cost specifications.

PairsG algorithm shows that the *Pairs* algorithm did provide a good starting point for the post-processing, by considering the most important pairs. The *Lachiche* results are omitted because no improvements resulted under any conditions.

6.3.1. Synthetic datasets

The first two entries in Table 4 summarise the performance on the two *synthetic* datasets. The *Pairs* and *PairsG* algorithm perform best in both cases, with little difference in performance between them. This is because these datasets suit the *Pairs* algorithm since there are only pairwise interactions. The *Lachiche* algorithm results in no improvement in most cases for the first dataset, and none at all for the second. The *Naive* algorithm generally results in an improvement in performance, but is outperformed by the *Greedy* and *Pairs* algorithms. This is because of the limitation of the *Naive* approach, which is prone to local minima. The *Pairs* algorithm outperforms the *Greedy* approach here because all interactions are accounted for. Referring to Table 5, it can be seen that the *PairsG* algorithm outperforms the *Greedy* algorithm more often as the number of classes increases. This shows that the pairwise approach scales with increasing C, whereas the *Greedy* approach becomes more susceptible to local minima.

6.3.2. Real datasets

Referring to Table 4, in general, the *PairsG* algorithm performs best overall. In the *Digits* and *Satellite* cases, the *Pairs* algorithm performs well (better than the *Greedy* approach), showing that in some cases the raw algorithm is robust. This suggests that these datasets may be dominated by pairwise interactions. In the case of the *Letter* and *Cbands* datasets, the *Pairs* approach is inferior to the *Greedy* algorithm,

suggesting important higher-order interactions. The inclusion of the post-processing (*PairsG*) results in superior performance, showing that the initial weighting from the pairwise algorithm is in fact useful. The reason is that the initial *Pairs* algorithm optimises the most important interactions for the given conditions, which is a good starting point. As in the case of the *synthetic* experiments, the *Naive* algorithm improves performance rather consistently, but as expected is inferior to the more sophisticated *Greedy* approach. The *Lachiche* algorithm occasionally results in a significant improvement in performance, but most scenarios result in none at all. The *Cbands* dataset shows a scenario where the *Greedy* algorithm competes in many cases with the *PairsG* algorithm. This dataset may have many higher order interactions that are not dealt with effectively due to the pairwise limitation.

Considering Table 5, it can be seen that the *PairsG* algorithm is generally better than the *Greedy* approach. The *Letter* dataset results in the best *PairsG* performance, with 49 out of 50 scenarios favouring this algorithm, whereas only 70% of experiments show superiority in the *CBands* case. Perhaps the *PairsG* algorithm would perform better if more interactions were considered here (i.e. increasing the AUC threshold t_p), but this would significantly increase the computational burden.

7. Conclusion

In this paper, a practical framework for generalised multiclass ROC analysis was presented, providing an extension of techniques and analyses commonly used in two-class ROC analysis. The computational complexity of multiclass ROC analysis was discussed as a function of the number of classes and ROC resolution, exhibiting a computational burden that increases to the power of the number of classes. This severely limits its practical use to problems with a small number of classes. This limitation was used as the argument for the development of approximate techniques. For cost-sensitive applications, two simple algorithms were proposed that use a search paradigm, in which a new cost and prior is used to direct a search, resulting in new "optimal" operating weights. The first uses a simple approach that optimises operating weights independently, ignoring interactions, and the second uses a greedy search with random initialisations, attempting to account for interactions. Both algorithms extend linearly with the number of classes C, and are thus tractable for even high C. However, these approaches do not result in a multiclass ROC, losing the various benefits, and are susceptible to local minima. The paper then presented an approximation of the multiclass ROC, called the *Pairwise Multiclass ROC*, which is tractable for high C problems. As the name suggests, this algorithm investigates interactions between operating weight pairs (two-class ROC's). In the cost-sensitive case, each ROC pair is optimised to the new priors and costs, followed by construction of the final operating weight vector. However several possible pair combinations exist, so the pairwise algorithm considers the various feasible pairings, and chooses the best one, based on minimum loss. Conceptually, this results in considering the most interacting pairs, and the most costly errors, which impact the new situation most significantly. A variety of experiments compared the various approaches, showing consistent benefits (except the *Lachiche* algorithm) of the various algorithms over a default unoptimised classifier in the cost-sensitive case. The naive approach is usually inferior to the greedy approach, and the pairwise algorithm outperformed the other algorithms in most cases, indicating that it is well formulated, and generally works well.

It is anticipated that this study will encourage new algorithms and approaches to tackling the area of multiclass ROC analysis, which will further diversify the field of statistical pattern recognition into new applications.

Acknowledgements

This research is/was supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Ministry of Economic Affairs, The Netherlands.

References

Adams, N., Hand, D., 1999. Comparing classifiers when misallocation costs are uncertain. Pattern Recognition 32 (7), 1139–1147.

Bishop, C., 1995. Neural Networks for Pattern Recognition, first ed. Oxford University Press Inc., New York.

Bradley, A., 1997. The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern Recognition 30 (7), 1145–1159.

Duda, R., Hart, P., Stork, D., 2001. Pattern Classification, second ed. Wiley-Interscience.

Duin, R., January 2000. PRTools, A Matlab Toolbox for Pattern Recognition. Pattern Recognition Group, TU Delft.

Edwards, D., Metz, C., Kupinski, M., 2004. Ideal observers and optimal ROC hypersurfaces in N-class classification. IEEE Trans. Med Imaging 23 (7), 891–895.

ELENA, 2004. European ESPRIT 5516 project. phoneme dataset. URL ftp://ftp.dice.ucl.ac.be/pub/neural-nets/ELENA/databases.

Everson, R., Fieldsend, J., 2005. Multi-class ROC analysis from a multi-objective optimisation perspective. Pattern Recognition Lett., Special issue on ROC analysis 27.

Fawcett, T., 2005. An introduction to ROC analysis. Pattern Recognition Lett. 27, 861–874 (Special issue on ROC analysis).

Ferri, C., Hernandez-Orallo, J., Salido, M., 2003. Volume under the roc surface for multi-class problems. In: Proc. 14th European Conf. on Machine Learning, pp. 108–120.

Hand, D., Till, R., 2001. A simple generalisation of the area under the ROC curve for multiple class classification problems. Mach. Learn. 45, 171–186.

Houtepen, J., 1994. Chromosome Banding Profiles: How Many Samples are Necessary for a Neural Network Classifier? Masters Thesis, Delft University of Technology, pp. 1–120.

Lachiche, N., Flach, P., 2003. Improving accuracy and cost of two-class and multi-class probabilistic classifiers using ROC curves. In: Proc. 20th Int. Conf. on Machine Learning (ICML-2003), Washington DC, pp. 416–423.

Landgrebe, T., Duin, R., November 2005. On Neyman-Pearson optimisation for multiclass classifiers. In: Proc. 16th Annual Symposium of the Pattern Recognition Assoc. of South Africa.

Landgrebe, T., Duin, R., 2006. Combining accuracy and prior sensitivity for classifier design under prior uncertainty. Structural and Syntactic Pattern Recognition, Proc. SSPR2006 (Hong Kong, China). In: Lecture Notes in Computer Science, vol. 4109. Springer Verlag, Berlin, pp. 512–521.

Li, M., Sethi, I., 2006. Confidence-based classifier design. Pattern Recognition 39, 1230–1240.

McDonald, R., 2006. The mean subjective utility score, a novel metric for cost-sensitive classifier evaluation. Pattern Recognition Lett. 27 (13), 1472–1477.

Metz, C., 1978. Basic principles of ROC analysis. Semin. Nucl. Med. 3 (4).

Mossman, D., 1999. Three-way roc's. Med. Decision Making 19, 78–89.

Murphy, P., Aha, D., 1992. UCI repository of machine learning databases. University of California, Department of Information and Computer Science. ftp://ftp.ics.uci.edu/pub/machine-learning-databases.

Pham, T.V., Worring, M., Smeulders, A.W.M., 2002. Face detection by aggregated bayesian network classifiers. Pattern Recognition Lett. 23 (4), 451–461.

Provost, F., Fawcett, T., 2001. Robust classification for imprecise environments. Mach. Learning 42, 203–231.

Srinivasan, A., October 1999. Note on the location of optimal classifiers in N-dimensional ROC space. Oxford University Computing Laboratory Technical report PRG-TR-2-99.