Round Robin Rule Learning

Johannes Fürnkranz
Austrian Research Institute for Artificial Intelligence
Schottengasse 3, A-1010 Wien, Austria
E-mail: juffi@ai.univie.ac.at

Technical Report OEFAI-TR-2001-02

Abstract

In this paper, we discuss a technique for handling multi-class problems with binary classifiers. The idea—learning one classifier for each pair of classes—is known as *pairwise classification* but—to our knowledge—has not yet been thoroughly investigated in the context of inductive rule learning. We present an empirical evaluation of the method as a wrapper around the Ripper rule learning algorithm on 20 multi-class datasets from the UCI database repository. Our results show that the method is very likely to improve Ripper's classification performance without having a high risk of decreasing it. The size of this improvement is similar to that obtained by boosting C5. In addition, we give a theoretical analysis of the complexity of the approach and show that its training time is within a small constant factor of the training time of the sequential class binarization technique that is currently used in Ripper

1 Introduction

Pairwise classification is a technique for reducing a multi-class problem to multiple 2-class problems by learning a classifier for each pair of classes. Its first formalization in the machine learning literature is due to Friedman (1996). It has been previously applied to various problems, mostly in the support vector machines community (see section 8), but we are not aware of an extensive experimental study, in particular not in the context of inductive rule learning algorithms. In this paper, we will show that this technique in fact gives significant improvements in predictive accuracy. We explain these improvements by pointing out that pairwise classification can be interpreted as an ensemble technique (Dietterich, 1997).

In addition, we provide a detailed analysis of the complexity of the approach and show that it is only within a constant factor of the conventional approaches that train each class against all other classes. For algorithms with super-linear asymptotic complexity, it may even be considerably faster. This is underlined by our experimental results.

2 Class Binarization

Many machine learning algorithms are inherently designed for binary (two-class) decision problems. Prominent examples are neural networks with single output nodes, support vector machines and separate-and-conquer rule learning. In addition, all regression algorithms can, in principle, be used for binary decision problems, but not for multi-class problems (unless, maybe, if the class values are ordered). However, real-world problems often have multiple classes. Fortunately, there exist several simple techniques for turning multi-class problems into a set of binary problems. We will call such techniques class binarization techniques.

Definition 2.1 (class binarization, decoding, base learner) *A* class binarization *is a mapping of a multi-class learning problem to several 2-class learning problems in a way that allows a sensible* decoding *of the prediction, i.e., allows to derive a prediction for the multi-class problem from the predictions of the set of 2-class classifiers. The learning algorithms used for solving the 2-class problems is called the* base classifier.

The most popular class binarization rule is the unordered or one-against-all class binarization, where one takes each class in turn and learns binary concepts that discriminate this class from all other classes. It has been independently proposed for rule learning (Clark & Boswell, 1991), neural networks (Anand et al., 1995), and support vector machines (Cortes & Vapnik, 1995).

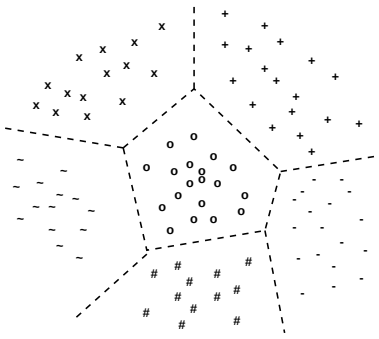
Definition 2.2 (unordered class binarization) *The* unordered class binarization *transforms a c-class problem into c 2-class problems. These are constructed by using the examples of class i as the positive examples and the examples of classes j, $j = 1 \ldots c, j \neq i$ as the negative examples.*

The name "unordered" originates from Clark and Boswell (1991), who proposed this approach as an alternative to the decision-list learning approach that was originally used in CN2 (Clark & Niblett, 1989; Rivest, 1987). In other fields, the strategy has different names, but as our main concern is rule learning, we stick with the terminology used there.

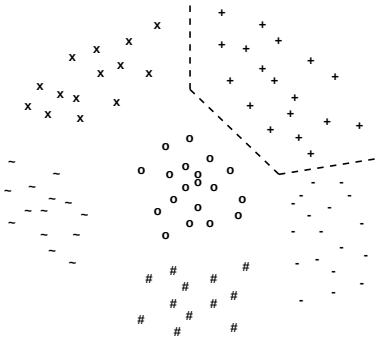
The rule learning algorithm Ripper (Cohen, 1995), which will be our main test object, also provides an option for inducing unordered rule sets. Its default mode of operation, however, is an ordered approach.

Definition 2.3 (ordered class binarization) *The* ordered class binarization *transforms a c-class problem into $(c-1)$ 2-class problems. These are constructed by using the examples of class i, $i = 1 \ldots c-1$ as the positive examples and the examples of classes j, $j = i+1 \ldots c$ as the negative examples.*

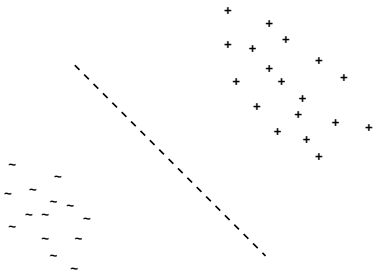
Note that ordered class binarization imposes an order on the induced classifiers, which has to be adhered to at classification time: the classifier learned for discriminating class 1 from classes $2 \ldots c$ has to be called first, and iff this classifier classifies the example as belonging to class 1, no other classifier is called. If not, the examples is subjected to to the next classifier. Unordered class binarization, on the other hand,



(a) *Multi-class learning*
one classifier that separates all classes.



(b) *Unordered learning*
c classifiers, each separates one class from all other classes. Here: + against all other classes.



(c) *Round robin learning*
$\frac{c(c-1)}{2}$ classifiers, one for each pair of classes. Here: $+$ against $\sim$.

Figure 1: Unordered and round robin binarization for a 6-class problem.

has to call each of its constituent binary classifiers and requires some external criterion for *decoding* the individual predictions into a final prediction. Typical decoding rules vote the predictions of the individual classifiers, possibly by taking into account the confidences of the predictions.

For a uniform view and an empirical evaluation of several class binarization and decoding techniques, we refer the reader to section 8 and (Allwein et al., 2000). In the following section, we discuss one particular technique in more detail.

3 Round Robin Classification

In this section, we will discuss a more complex class binarization procedure, the pairwise classifier. The basic idea is quite simple, namely to learn one classifier for each pair of classes. In analogy to round robin tournaments in sports and games, in which each participant is paired with each other participant once, we call this procedure *round robin binarization*.

Definition 3.1 (round robin, pairwise binarization) *The* round robin *or* pairwise *class binarization transforms a c-class problem into $\frac{c(c-1)}{2}$ 2-class problems, one for each unique pair of classes $<i, j>$ $(i = 1 \ldots c-1, j = i+1 \ldots c)$. The binary classifier is trained on the training examples of classes i and j. The examples of classes $\neq i, j$ are ignored for the binary classifier $<i, j>$.*

When classifying a new example, each of the learned base classifiers determines to which of its two classes the example is more likely to belong to. The winner is assigned a point, and in the end, the algorithm will predict the class that has accumulated the most points. In our version, ties are broken by preferring the class that is more frequent in the training set (or flipping a coin if the frequencies are equal), but more elaborate techniques are possible (see Section 7).

Round robin binarization is illustrated in Figure 1. For the 6-class problem shown in Figure 1(a), the round robin procedure learns 15 classifiers, one for each pair of classes. Figure 1(c) shows the classifier for the class pair $<+, \sim>$. In comparison, Figure 1(b) shows one of the classifiers for the unordered class binarization, namely the one that pairs class $+$ against all other classes. It is obvious that the round robin base classifier uses fewer examples and thus has more freedom for fitting a decision boundary between the two classes. In fact, in this problem, all binary classification problems of the round robin binarization could be solved with a simple linear discriminant, while neither the multi-class problem nor its unordered binarization have a linear solution.

Note that some examples will be forced to be classified erroneously by some of the binary base classifiers because each classifier must label all examples as belonging to one of the two classes it was trained on. Consider the classifier shown in Figure 1(c): it will arbitrarily assign all examples of class o to either $+$ or $\sim$ (depending on which side of the decision boundary they are). However, the votes of the 5 classifiers that contain examples of class o should be able to over-rule the votes of the 10 other classifiers, which more or less randomly assign one of the other 5 classes to each o example. In fact, if the 5 o classifiers unanimously vote for o, no other class can accumulate 4 votes (because it lost its direct match against o). Besides, it would be quite unlikely that all classifiers that did not contain any examples of class o would unanimously opt for the same class.

In the above definition, we assume that the problem of discriminating class i from class j is identical to the problem of discriminating class j from class i. This is the case if the base learner is *class-symmetric*. Rule learning algorithms, however, need not be class-symmetric. Many of them choose one of the two classes as the default class, and learn only rules to cover the other class. In such a case, $<i, j>$ and $<j, i>$ may be two

different classification problems, if j is used as the default class in the former, and i is used as the default class in the latter.

Ripper is such an algorithm. Unless specified otherwise, it will treat the larger class of a 2-class problem as the default class and learn rules for the smaller class. Although this procedure is class-symmetric (problem $<i, j>$ is converted to $<j, i>$ if $i > j$), we felt that it would not be fair. For example, the largest class in the multi-class problem would be used as the default class in all round robin problems. This may be an unfair advantage (or disadvantage) to this class.[1] One solution for avoiding this, is to play a so-called double round robin, in which separate classifiers are learned for both problems, $<i, j>$ and $<j, i>$.[2]

Definition 3.2 (double round robin) *The* double round robin *class binarization transforms a c-class problem into $c(c-1)$ 2-class problems, one for each pair of classes $<i, j>$ $(i, j = \ldots c, j \neq i$. The examples of class i are used as the positive examples and the examples of class j as the negative examples.*

In our experiments, we used Ripper with the option `-agiven` as the base classifier, which uses the classes as given in the specification. Hence, $<i, j>$ and $<j, i>$ are two different problems, and each class is the default class in exactly half of its binary classification problems. Note, that this procedure basically is identical to the one that is used by Ripper if it is used in unordered mode on a 2-class problem.

4 Accuracy

In this section, we will briefly present an experimental evaluation of round robin binarization in a rule learning context. We chose Ripper (Cohen, 1995) as the base classifier, which—in our view—is the most advanced algorithm of the family of separate-and-conquer (or covering) rule learning algorithms (Fürnkranz 1997; 1999).

The unordered and ordered binarization procedures were used as implemented within Ripper. The round robin binarization was implemented as a wrapper around Ripper, which provided it with the appropriate training sets. This was implemented in `perl` and had to communicate with Ripper by writing the training sets to and reading Ripper's results from the disk. This implementation is referred to as R^3 (Round Robin Ripper).

Table 1 shows the 20 datasets we used in this study. They were chosen arbitrarily among datasets with ≥ 4 classes available at the UCI repository (Blake & Merz, 1998).[3] The implementation of the algorithm was developed independently and not tuned on these datasets. On the six sets with a dedicated test set, we report the error

[1] The situation can be compared to a chess player that has to play all of his games with the same color. This can make a decisive difference and may invalidate the final result of the tournament.

[2] Another way to ensure a fair competition, which is frequently used in chess tournaments, is to assign the colors in a deterministic way so that each player has an equal number of white and black games in a single round robin tournament. This could also be adapted for our problem.

[3] The restriction to 4 or more classes was made because on 3-class problems, we would expect frequent 3-way ties, which are not yet handled very cleverly. The issue of ties is discussed further below in the paper (Section 7).

name	train	test	sym	num	classes	def. error
abalone	3133	1044	1	7	29	83.5
covertype	15,120	565,892	44	10	7	51.1
letter	16,000	4000	0	16	26	95.9
sat	4435	2000	0	36	6	76.2
shuttle	43,500	14,500	0	9	7	21.4
vowel	528	462	0	10	11	90.9
car	1728	—	6	0	4	30.0
glass	214	—	0	9	7	64.5
image	2310	—	0	19	7	85.7
lr spectrometer	531	—	1	101	48	89.6
optical	5620	—	0	64	10	89.8
page-blocks	5473	—	0	10	5	10.2
solar flares (c)	1389	—	10	0	8	15.7
solar flares (m)	1389	—	10	0	6	4.9
soybean	683	—	35	0	19	94.1
thyroid (hyper)	3772	—	21	6	5	2.7
thyroid (hypo)	3772	—	21	6	5	7.7
thyroid (repl.)	3772	—	21	6	4	3.3
vehicle	846	—	0	18	4	74.2
yeast	1484	—	0	8	10	68.8

Table 1: Data sets used. The first two columns show the training and test set sizes (as specified in the description of the datasets), the next three columns show the number of symbolic and numeric attributes as well as the number of classes. The last column shows the default error, i.e., the error one would get by always predicting the majority class.

rate on these test sets. On the other 14 sets, we estimated the error rate using paired, stratified 10-fold cross-validations. For *abalone*, *sat* and *vowel* we performed both a test set evaluation and a cross-validation.[4]

The right half of Table 4 shows the accuracies of Ripper (unordered and ordered) and R^3 on these datasets. On half of the 20 sets (not counting the cross-validated trials of the three sets in the middle) R^3 is significantly better ($p > 0.99$ on a McNemar test (Feelders & Verkooijen, 1995)) than Ripper's default mode (ordered binarization). There are only two sets (*thyroid (repl.)* and the test-set version of *vowel*), where R^3 is worse than Ripper, both differences being insignificant. The comparison to unordered Ripper is similar (the significance levels for this case are not shown).

We can safely conclude that round robin binarization may result in significant improvements over ordered or unordered binarization without giving a high risk of decreasing performance.

[4]For *abalone* and *sat*, this did not change the results, while for *vowel* the performance of all algorithms increased significantly, which may indicate that the 528 examples in the original training set are insufficient for learning a good concept. See Table 4.

	C5			Ripper				
dataset	default	boosted	ratio	unord.	ordered	R^3	ratio	$<$
abalone	78.45	77.59	*0.989*	81.03	82.18	72.99	*0.888*	++
covertype	31.17	26.18	*0.840*	35.37	38.50	33.20	*0.862*	++
letter	12.48	5.78	*0.463*	15.22	15.75	7.85	*0.498*	++
sat	15.65	10.00	*0.639*	14.25	17.05	11.15	*0.654*	++
shuttle	0.15	0.01	*0.045*	0.03	0.06	0.02	*0.375*	=
vowel	61.47	57.58	*0.937*	64.94	53.25	53.46	*1.004*	=
abalone (x-val)	78.48	77.88	*0.992*	81.64	81.18	74.34	*0.916*	++
sat (x-val)	14.02	9.20	*0.656*	13.18	13.04	10.35	*0.794*	++
vowel (x-val)	21.72	8.89	*0.409*	30.50	27.07	18.69	*0.690*	++
car	7.58	3.82	*0.504*	5.79	12.15	2.26	*0.186*	++
glass	35.05	27.57	*0.787*	35.51	34.58	25.70	*0.743*	++
image	3.20	1.60	*0.500*	4.15	4.29	3.46	*0.808*	+
lr spectrometer	51.22	46.70	*0.912*	64.22	61.39	53.11	*0.865*	++
optical	9.20	2.46	*0.267*	7.79	9.48	3.74	*0.394*	++
page-blocks	3.09	2.58	*0.834*	2.85	3.38	2.76	*0.816*	++
solar flares (c)	15.77	16.41	*1.041*	15.91	15.91	15.77	*0.991*	=
solar flares (m)	4.90	5.90	*1.206*	4.90	5.47	5.04	*0.921*	=
soybean	9.66	6.59	*0.682*	8.79	8.79	6.30	*0.717*	++
thyroid (hyper)	1.11	1.03	*0.929*	1.25	1.49	1.11	*0.749*	+
thyroid (hypo)	0.58	0.32	*0.545*	0.64	0.56	0.53	*0.955*	=
thyroid (repl.)	0.72	0.90	*1.259*	1.17	0.98	1.01	*1.026*	=
vehicle	26.24	24.11	*0.919*	28.25	30.38	29.08	*0.957*	=
yeast	43.26	41.85	*0.967*	44.00	42.39	41.78	*0.986*	=
average	20.55	17.95	*0.763*	21.80	21.90	18.52	*0.770*	

Table 2: *Error rates*: The first three columns show the error rates of C5 and C5-boost (all with default parameters) and their ratio (the improvement rate one gets by switching from C5 to C5-boost). The next three columns show the results of Ripper (in unordered and in default, ordered mode) and R^3. The last two columns show the improvement rate of R^3 over Ripper (default), and whether R^3 is significantly better (++ if $p > 0.99$, + if $p > 0.95$) better than Ripper, measured with a McNemar test. The last line shows the average of the columns above (the x-val versions of abalone, sat and vowel (middle part of the table) were ignored). For the accuracy-based columns, this measure is dominated by a few high-error datasets, which is the reason why the ratio of the average errors does not match the given value (which is the average of all ratios).

5 Round Robin Learning as an Ensemble Method

Recently, ensemble methods have received considerable attention within the machine learning literature (Dietterich, 1997). The basic idea is to obtain a diverse set of classifiers for a single learning problem. Averaging the predictions of these classifiers helps to reduce the variance and often increase the reliability of the predictions. There are several techniques for obtaining a diverse set of classifiers. The most common technique is to use subsampling to diversify the training sets as in bagging (Breiman,

1996) and boosting (Freund & Schapire, 1996). Other techniques include the use of different feature subsets (Bay, 1999) or to exploit the randomness of the base algorithms (Kolen & Pollack, 1991), possibly by artificially randomizing their behavior (Dietterich, 2000). A third approach for obtaining Dietterich and Bakiri (1995) investigated error-correcting output codes as a way of improving the predictive performance of classifiers.

Round robin classification may also be interpreted as an ensemble methods, and its performance gain may be seen in this context. Obviously, the final prediction is made by exploiting the redundancy provided by multiple models, each of them being constructed from a subset of the original data. However, contrary to subsampling approaches like bagging and boosting, these datasets are constructed deterministically.[5] In this respect it shares more similarities with error-correcting output codes, but differs from it through the fixed procedure for setting up the new binary problems and the fact that each of the new problems is smaller than the original problem. In particular the latter fact may often cause the sub-problems in pairwise classification to be conceptually simpler than the original problem (as illustrated in Figure 1).

If we compare the improvements in accuracy obtained by R^3 over Ripper to those obtained by C5-boost over C5 on the same problems (shown in the left half of Table 4), we see that the improvement rates are quite similar. Not only is the average of the improvement ratios almost equal, but the results on the individual datasets also seem to correlate with each other. Round robin binarization seems to work whenever boosting works, and vice versa (see, e.g, the results on *solar flares* where both don't work).

Figure 2 plots the error ratios of C5-boost/C5 and R^3/Ripper on the 20 datasets. The correlation coefficient r^2 between the ratios on these datasets is about 0.618. This is in the same range as correlation coefficients for bagging and boosting (Opitz & Maclin, 1999). There, it is also shown that worse base learners produce worse ensembles, which may explain why R^3 does not quite level the performance of C5-boost. It might be interesting to see whether C5's performance can be "boosted" to a similar level as C5-boost by using round robin binarization, in particular because (as we shall see in the next section) the round robin approach is considerably more efficient. We plan to conduct this experiment in the near future. We also plan to investigate whether additional additional gains can be achieved by using a combination of boosting and binarization. However, due to the correlation of the improvements achieved by the two techniques, we would not expect this to be the case.

6 Efficiency

At first sight, it appears to be a questionable idea to replace $O(c)$ binary learning tasks (unordered binarization) with $O(c^2)$ binary learning tasks (round robin binarization) because the quadratic complexity seems to be prohibitive for tasks with more than a few classes. This section will illustrate that this is not the case.

[5] Boosting is also deterministic if the base learner is able to use weighted example. Often, however, the example weights are interpreted as probabilities which are used for drawing the sample for the next boosting iteration.

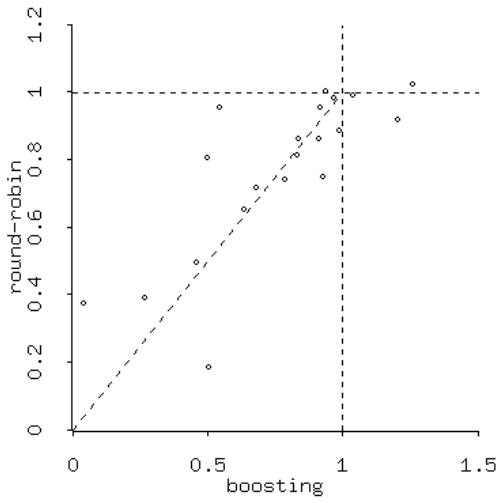


Figure 2: Error reductions ratios of boosting (x-axis) vs. round robin (y-axis).

6.1 Theoretical Considerations

In this section, we will see that although round robin classification turns a single c-class learning problem into $c(c-1)$ 2-class problems, the total learning effort is only linear in the number of classes, and may in some circumstances even be smaller than the effort needed for an unordered binarization. The analysis in this section is independent of the base learning algorithm used. Some of the ideas have already been sketched in a short paragraph by Friedman (1996), but we go into more detail here, and, in particular, focus on the comparison to conventional class binarization techniques.

Definition 6.1 (class penalty) *If the base learner has a complexity growth function $f(n)$ (i.e., the time needed for a n-example training set is $f(n)$), and the total time needed for the class binarized problems is $\pi(c)f(n)$, we call the function $\pi(c)$ the* class penalty.

The class penalty $\pi(c)$ measures the performance of an algorithm on a (class binarized) c-class problem relative to its performance on a single 2-class problem of equal size.

In the following, we will compare the class penalty π_r of a single round robin class binarization to class penalty π_u of an unordered binarization. Note that the class penalty for a double round robin is twice as high as the class penalty of a single round robin. Also, unordered binarization is more expensive than ordered binarization, but it is simpler to analyze because it does not depend on the class distribution. We would estimate that the ordered approach will take about half of the training time of the un-

ordered approach.[6] So, unless noted otherwise, the ratio of the penalty functions has to be adjusted by factor of 4 to get the results of ordered binarization versus double round robin binarization.

Lemma 6.2 *The total number of training examples produced by the unordered class binarization is greater than the total number of examples produced by a single round robin binarization (and more than half of the total number of examples produced by a double round robin binarization).*

Proof: We assume a learning problem with c classes and n examples. Each class i $(i = 1 \ldots c)$ contains n_i examples, $\sum_{i=1}^{c} n_i = n$.

First, we consider the unordered case: Each of the c classes is learned using all examples of the other $c-1$ classes as the negative examples. Thus we have c binary learning tasks, each of them using the entire training set. Hence, the learner processes a total of cn training examples.

Now consider a double round robin binarization: We have $c(c-1)$ problems, one for each pair of classes $<i,j>$ with $i,j = 1 \ldots c, i \neq j$. The problem corresponding to the pair $<i,j>$ has $n_i + n_j$ examples. Thus the total number of examples is

$$\begin{aligned}\sum_{i=1}^{c} \sum_{j=1, j\neq i}^{c} (n_i + n_j) &= \sum_{i=1}^{c} \sum_{j=1}^{c} (n_i + n_j) - \sum_{i=1}^{c} (n_i + n_i) \\ &= \sum_{i=1}^{c} (cn_i + n) - 2n = 2cn - 2n = 2(c-1)n\end{aligned}$$

As $2(c-1)n < 2cn$, the total number of examples is less than twice that of the unordered binarization.

The single round robin binarization uses each pair of classes only once instead of twice as in the double round robin. Hence the total number of examples is only $(c-1)n$ which is less than the cn for the unordered cases. □

There is a more intutitive proof for this fact: In the unordered class binarization, each example appears in each binary training task, i.e., in c tasks, resulting in a total of cn examples. In the (single) round robin binarization, each example appears in only $c-1$ tasks, namely in those tasks where its class is paired against one of the $c-1$ other classes.

How does the number of used training examples influence the training time? Let us, for the moment, assume that we have a linear growth function[7] $f(n) = \lambda n$, i.e., if we

[6] The ordered approach has only $c-1$ binary tasks, and the number of training examples decreases with each learned class. Our empirical results (Section 6.2) indicate that for Ripper, the actual gain is less than 50%.

[7] We also assume that the base algorithm has no constant overhead, which is, of course, a somewhat questionable assumption. Note however, some sigificant overhead costs, like reading in the training examples (and similar initialization steps) need, of course, only be performed once if the round robin procedure is performed in memory (which is not the case for our implementation). If there is nevertheless an overhead μ to be considered (i.e., $f(n) = \lambda n + \mu$), the total costs will be increased by $\frac{c(c-1)}{2}\mu$. Eventually (with growing c), these quadratic costs may be more expensive than the savings derived below, but under reasonable assumptions (e.g., $c^2 < n$) these additive costs should not matter.

double the training set size, the algorithm will take twice as long. Then the following theorem follows trivially from the above lemma.

Theorem 6.3 $\frac{\pi_r(c)}{\pi_u(c)} < 1$ *for algorithms with a linear complexity* $f(n) = \lambda n$.

Proof: The complexity of the round robin procedure is $\sum_{i=1}^{c} \sum_{j=1, j\neq i}^{c} f(n_i + n_j)$. A linear growth function with no constant overhead is $f(n) = \lambda n$. Substituting in the above formula and pulling out the λ results in a complexity of $\lambda(c-1)n$. An analogous derivation gives us a complexity of λcn for the unordered approach. Thus $\frac{\pi_r(c)}{\pi_u(c)} = \frac{c-1}{c} < 1$. □

The obvious question now is: what happens in the case of algorithms that have a super-linear growth function, i.e., where $f(n) = n^o$ for some $o > 1$? Note that for such functions $f(n_1) + f(n_2) < f(n_1 + n_2)$ holds ($n_1, n_2 > 0$).[8] This means that performing a single learning task on a training set of size n is more expensive than performing several training tasks on sets of size n_i where $\sum_{i=1}^{c} n_i = n$. This leads us to the following

Proposition 6.4 *The class penalty ratio* $\frac{\pi_r(c)}{\pi_u(c)}$ *of the round robin binarization over the unordered binarization decreases with increasing order of growth* o *of the base algorithm's growth function* $f(n) = n^o$. *Likewise, it decreases with increasing number of classes* c *for a fixed growth function with order* $o > 1$.

Unfortunately, we did not manage to find a strict formal proof for this proposition. However, as $\pi_r < \pi_u$ in the linear case (Theorem 6.3), and super-linear functions penalize larger training sets more heavily than smaller training sets ($\sum n_i^o < (\sum n_i)^o$ for $o > 1$), it seems to be evident that the smaller training sets of the round robin binarization will be penalized less by a more expensive algorithm than the larger sets of the conventional ordered or unordered binarization. This effect should be clearer for more classes and for growth functions of higher orders o.

As an illustration, however, it is easy to proof the following special case:

Theorem 6.5 *Proposition 6.4 holds for problems with a uniform class distribution.*

Proof: In the (single) round robin case, we have $\frac{c(c-1)}{2}$ problems with $2\frac{n}{c}$ examples each. Hence the total complexity is

$$\frac{c(c-1)}{2} f\left(2\frac{n}{c}\right) = \frac{c(c-1)}{2} \left(2\frac{n}{c}\right)^o = (c-1)\left(\frac{2}{c}\right)^{o-1} f(n)$$

and the penalty function $\pi_r(c) = (c-1)(\frac{2}{c})^{o-1}$. As for multi-class problems $\frac{2}{c} < 1$, $(\frac{2}{c})^{o-1}$ decreases with increasing o.

On the other hand, the unordered class binarization transforms the c-class problem into c 2-class problems, each of which has the complexity $f(n)$. Hence its class

[8] This is easy to see for integer values of o because the term on the left-hand side contains only all "pure" powers and leaves out all terms that contain powers of both n_1 and n_2.

penalty $\pi_u(c) = c$ is constant (with respect to o). As the numerator of the ratio $\frac{\pi_r(c)}{\pi_u(c)}$ decreases with increasing o and its denominator remains constant, the ratio decreases with increasing o.

Likewise, $\pi_u(c) = c$ grows linearly with c, while $\pi_r(c) = (c-1)(\frac{2}{c})^{o-1}$ grows sub-linearly because the factor $(\frac{2}{c})^{o-1}$ decreases with increasing c for $o > 1$. Hence the ratio $\frac{\pi_r(c)}{\pi_u(c)}$ decreases with increasing c for a constant $o > 1$. □

Note that this theorem implies that for sufficiently high c and/or o even the double round robin will be faster than the ordered binarization. Assume, e.g., an algorithm with a quadartic complexity on an eight-class problem (i.e., $o = 2$ and $c = 8$), the penalty ratio

$$\frac{\pi_r(c)}{\pi_u(c)} = \frac{(c-1)(\frac{2}{c})^{o-1}}{c} = \frac{7(\frac{1}{4})^1}{8} = \frac{7}{32} < \frac{1}{4}$$

Thus, under these circumstances, the single round robin is more than four times faster than the unordered approach. Considering that the double round robin is twice as slow, and assuming that the ordered approach is twice as fast as the unordered approach (see the following section for empirical values on that), a double round robin would be faster than the ordered approach. In the following section, this scenario will be empirically evaluated.

6.2 Empirical Evaluation

Contrary to the theoretical analysis in the previous section, where we focussed on the lenient case of pairing unordered binarization vs. single round robin, our empirical results show the performance of ordered binarization (Ripper's default mode) vs. double round robin binarization. This, as we have noted above, is the worst case. In the case of a linear algorithm complexity, the round robin should be about four times slower than the ordered binarization.

The right half of Table 6.2 shows the training times[9] in CPU secs. user time (measured on a Sparc Ultra-2 under Sun Solaris) of R^3 and its performance ratios against Ripper in unordered and ordered mode. On average, it is about 2.6 times slower than Ripper in unordered mode, and about 4 times slower than Ripper in default, ordered mode. Conincidentally, these numbers also show that the ordered mode is less than twice as fast as the unordered mode, which confirms the assumptions we made at the beginning of Section 6.1.

The left half of Table 6.2 shows the run-times of C5 and their ratios against C5-boostin default operation. C5-boost is about 8.75 times as slow as C5, which corresponds fairly well to the factor of 10 that can be expected from the fact that C5-boost performs 10 iterations of basic C5. In all cases but one (*solar flares (c)*), R^3 is more efficient in comparison to Ripper than C5-boost is in comparison to C5.

Moreover, there are several cases where R^3 is even faster than Ripper in unordered mode and comes close to Ripper in ordered mode. This is despite the fact that R^3 is

[9]Classification time is only included in the runs that had a separate test set. In general, it can be expected to be more expensive for R^3. See Section 7 for a brief discussion of this issue.

dataset	C5 -b	vs. C5	R^3	vs. unord	vs. order
abalone	23.34	*10.81*	193.0	*4.51*	*5.73*
covertype	—	—	—	—-	—-
letter	73.37	*6.64*	1250.0	*0.51*	*1.14*
sat	27.86	*9.10*	143.0	*0.85*	*1.51*
shuttle	35.98	*5.67*	277.0	*2.10*	*3.16*
vowel	2.66	*9.50*	14.0	*1.83*	*4.75*
abalone (x-val)	27.69	*10.80*	140.28	*3.14*	*3.27*
sat (x-val)	44.80	*9.31*	186.89	*0.69*	*1.25*
vowel (x-val)	5.02	*9.77*	16.22	*0.87*	*2.16*
car	0.38	*10.32*	6.71	*1.55*	*1.47*
glass	0.42	*8.72*	2.03	*2.26*	*3.80*
image	5.38	*6.81*	25.84	*0.90*	*1.98*
lr spectrometer	58.27	*10.61*	489.67	*4.40*	*6.93*
optical	55.75	*7.52*	275.69	*0.63*	*1.34*
page-blocks	7.83	*8.44*	36.66	*1.43*	*1.93*
solar flares (c)	0.17	*3.78*	6.65	*6.03*	*7.57*
solar flares (m)	0.27	*7.59*	3.98	*5.62*	*7.49*
soybean	0.78	*7.16*	21.07	*6.29*	*13.24*
thyroid (hyper)	2.81	*7.88*	19.71	*2.68*	*3.46*
thyroid (hypo)	2.14	*9.35*	14.91	*2.39*	*3.63*
thyroid (repl.)	1.90	*9.29*	15.35	*2.26*	*3.33*
vehicle	2.64	*10.82*	7.66	*1.22*	*2.10*
yeast	6.44	*10.65*	16.90	*1.77*	*3.12*
average		*8.75*		*2.60*	*4.00*

Table 3: *Runtime results:* columns 2 and 4 show the run-times (in CPU secs. user time) of C5-boost and R^3. The following columns show the ratio of C5-boost over C5 (col. 3) and R^3 over unordered (col. 5) and ordered (col. 6) Ripper. The first five lines are total run-times, i.e., training and test time, while the cross-validated results report training time only. We failed to measure the run-times for the covertype data set, where the situation was complicated because of the large test set, which had to be split into several pieces for the Ripper-based algorithms. The last line shows the average of the 17 cross-validated datasets.

implemented as a `perl`-script that communicates to Ripper by writing the training and test sets of the new tasks to the disk. Although this is somewhat compensated by the fact that we only report user time (which ignores time for disk access and system time),[10] a tight integration of round robin binarization into Ripper's C-code would certainly be more efficient.

The good performance of R^3 does not come entirely surprising, if we consider the super-linear run-time complexity of Ripper.[11] For more expensive base learning

[10] For example, on the 26-class *letter* dataset, where R^3 writes $26 \times 25 = 650$ files to the disk, its total run-time is about 15% higher than the reported user time, while there is almost no difference for Ripper.

[11] The complexity of Ripper's initial rule learning and pruning phase is $O(n \log^2(n))$ (Fürnkranz & Widmer, 1994; Cohen, 1995). The two phases of optimization that Ripper performs thereafter, only add a constant factor to these results according to the experimental evidence shown in (Cohen, 1995). However, in very large domains, like text domains, our own experience is that these two optimization phases can slow

algorithms (like support vector machines), the analysis in the previous section lets us expect even bigger savings.

7 Other Issues

In the following, we briefly discuss further important aspects of round robin binarization.

Parallel Implementations: It should be noted that—contrary to boosting, where the individual runs depend on each other and have to be performed in succession—pairwise classification can be entirely parallelized (as already noted by Friedman (1996) and Lu and Ito (1999)). As each binary task will be smaller than the original task, the total training time of a mutli-class problem of size n will be significantly below that of a binary problem of the same size, if each binary classifier can be trained on a separate processor.

Memory Requirements: It is also clear that each individual binary task in a round-robin binarization has less training examples than the original tasks. For multi-class tasks that are too large to be performed in memory, pairwise classification may provide a simple means to reduce the size of the learning task without resorting to subsampling. Note that this is not the case for unordered class binarization or error-correcting output codes.

Imbalanced Class Distributions: Although we have not yet evaluated this issue, we also believe that round robin binarization is able to better focus on minority classes, in particular in problems where several large classes appear next to a few small classes. In particular the fact that separate classifiers are trained to discriminate the small classes from each other (and not only from all remaining examples as would be the case for unordered binarization or for treating the multi-class problem as a whole) may help to improve the focus in the case of imbalanced class distributions.

Classification Efficiency: Our efficiency analysis is only concerned with training time. At testing time, we have to test a quadratic number of classifiers in order to make the final prediction. Although it might be the case that the constituent classifiers are simpler (which often means that they can make faster predictions) it can be expected that classification time would be considerably slower than in the unordered binarization case. This situation is particularly bad for lazy learners, which defer most of their training effort to the classification phase.

To cope with such situation, we could once more take a look a sports and game tournaments, where similar problems arise when the number of participants in the tournament prevents a round robin. The frequently used knock-out tournament (where players are paired randomly and only the winner advances into the next round) seems

down the algorithm considerably, and seem to dominate the run-time (but we have not performed a thorough analysis of this issue).

to be very brittle. An interesting alternative might be provided by swiss system tournaments, which are frequently used in chess, where all players play a fixed number of rounds. The trick is that in each round players of about equal strength (approximately equal ranking in the tournament) are paired against each other. Such schemes could improve classification time for problems with very high numbers of classes, in particular if classification is very expensive (like with lazy learners).

Tie Breaking: We have mostly ignored the issue of tie-breaking, which is necessary when several classes have an equal number of wins against other classes. In particular for a low number of classes, ties are more likely because the ensemble is much smaller (which was the reason why we restricted our study to problems with at least four classes). Our straight-forward approach of using the *a priori* more likely class, can certainly be improved upon. In addition to techniques known from the literature (see, e.g., (Hastie & Tibshirani, 1998; Allwein et al., 2000)), one could again think of exploring techniques that are commonly used for breaking ties in tournament cross tables in games and sports (such as the Sonneborn-Berger ranking in chess tournaments).

Comprehensibility: While boosting also provides similar gains in accuracy, the price to pay is that the learned ensemble of classifiers is no longer easy to comprehend.[12] While round robin rule learning also learns an ensemble of classifiers, we think that it has the advantage that each element of the ensemble has a well-defined semantics (separating two classes from each other). In fact, Pyle (1999) proposes a very similar technique called *pairwise ranking* in order to facilitate human decision-making in ranking problems. The basic claim is that it is easier for a human to determine an order between n items if one makes pairwise comparisons between the individual items and then adding up the wins for each item instead of trying to order the items right away.[13]

8 Related Work

Pairwise classification was introduced to the machine learning literature by Friedman (1996) but the idea seems to have been known for some time.[14] Friedman proposes the basic architecture and evaluates two versions of CART (Breiman et al., 1984) and a nearest neighbor algorithm on 50 randomly generated problems. He observed improvements for the CART version which uses a linear function for splitting a node and for the nearest neighbor rule. For CART with axis parallel splits, the performance of the

[12] An exception might be a system like Slipper (Cohen & Singer, 1999) which tightly integrates boosting into the rule learning algorithm with the effect that only a single set of rules is learned and the redundancy among these rules is exploited to obtain higher accuracies. A direct comparison between Slipper and R^3 would be very interesting, not only for this reason.

[13] The aspect of being able to rank the available classifications for each example (as an intermediate version between pedicting only a class value and providing a full probability distribution) is another interesting aspect of round robin binarization, which might be worth exploring in more depth.

[14] For example, Witten and Frank (2000) independently refer to it on p. 113 as an alternative technique for making linear regression applicable to multiclass problems. However, they do not give a source or an evaluation of the approach.

pairwise class binarization was similar to that of the standard techniques. The author also provided a brief discussion of the complexity of the approach.

Hastie and Tibshirani (1998) picked up the technique and introduced *pairwise coupling*, a tie-breaking technique which combines the class probability estimates from the binary classifiers into a joint probability distribution for the multi-class problem. They also suggested the use of this technique for support vector machines, where it subsequently gained some popularity. For example, Kreßel (1999) evaluated it on a handwritten digit classification problem. His results did not show a consistent improvement of the error rates (on only two datasets), but the pairwise approach outperformed the unordered binarization technique in terms of efficiency. This performance gain is explained by our theoretical considerations (Section 6.1). Pairwise support vector machines were also applied to the problem of speaker identification (Schmidt & Gish, 1996; Schmidt, 1996).

Angulo and Català (2000) suggest a related technique where multi-class problems are mapped to 3-class problems. Like with pairwise classification, the idea is to generate one training set for each pair of classes, but in addition to encoding the two class values with target values $+1$ and -1, examples of all other classes are added with a target value of 0, which seems to give up some of the advantages that result from the reduction of the training set sizes on the binary problems. We are not aware of an empirical evaluation of their approach.

Class binarization techniques were also investigated in the neural networks community. The motivation was that it is sometimes better to have a modular network, i.e., a network that consists of several independently trained sub-networks that are concerned with different aspects of the problem, than a single neural network with many output nodes, which usually requires a large hidden layer and significant training times. Unordered (Anand et al., 1995) and pairwise (Lu & Ito, 1999) binarization techniques have been investigated.

Error-correcting output codes (Dietterich & Bakiri, 1995) are a popular and powerful general class binarization technique. The basic idea is to encode a c-class problem as $\bar{c}$ binary problems ($\bar{c} > c$), where each binary problem uses a *subset* of the classes as the positive class and the remaining classes as a negative class. It may thus be viewed as a generalization of unordered binarization, where only single classes are used as positive examples. As a consequence, each original class is encoded as a $\bar{c}$-dimensional binary vector, one dimension for each prediction of a binary problem. New examples are classified by determining the binary vector that is closest to the binary vector obtained by submitting the example to the $\bar{c}$ classifiers. If the new classes are chosen in a way that the distance between the class vectors is large, the reliability of the classification can be significantly increased. Error-correcting output codes can also be easily parallelized, but each subtask requires the total training set. As the number of required binary problems is $> c$ for a c-class problems, its penalty function $\pi_{eoc} > c$, i.e., pairwise and unordered binarization are more efficient.

Most recently, Allwein et al. (2000) provide a unifying view of various class binarization techniques and derive some theoretical error bounds. They also provide a thorough experimental study of five different class binarization schemes with three different techniques for combining the predictions of the binary classifiers. Their results showed that for support vector machines the unordered binarization technique is infe-

rior to other techniques including pairwise classification. Among the alternatives, no clear winner emerged. However, for boosted decision trees, no consistent difference between the methods could be found (including unordered binarization). This opens the question, whether a combination of boosting and class binarization may increase the performance of either method. In our study, the similar performance improvements obtained by boosting and class binarization seem to suggest that although both techniques operate on different principles, their effect seems to be quite similar, and it is doubtful that their combination will result in additional performance gains. In the reign of rule learning, this question could be answered by a comparison of Slipper (a successor of Ripper that uses boosted rule sets (Cohen & Singer, 1999)) to class binarization. Alternatively, we plan to evaluate round robin binarization for C5 and C5-boost.

Finally, we note the relation of the approach to comparison training (Tesauro, 1989; Utgoff & Clouse, 1991), which was proposed as a training procedure in evaluation function learning. In this framework, the learner is not trained with the target values of the evaluation function in certain states, but instead is trained on state pairs where the preferred state is marked. Thus, this training procedure is somewhere between supervised learning (where it is trained on the target values) and reinforcement learning (where it only receives indirect feedback about the value of states). Tesauro (1989) demonstrated a particularly interesting technique, where he enforced a symmetric neural network architecture and showed that with this architecture, he only has to perform n network evaluations to determine the best of n moves. It is an interesting open question, whether a similar technique could be employed for speeding up pairwise classification.

9 Summary and Outlook

This paper has investigated the use of round robin binarization (or pairwise classification) as a technique for handling multi-class problems with separate-and-conquer rule learning algorithms (aka covering algorithms). Our experimental results show that, in comparison to conventional, ordered or unordered binarization, the round robin approach may yield significant gains in accuracy without risking a bad performance. The performance improvement seems to be similar to that achievable by boosting, although a thorough investigation of this issue is still pending. We think that the reason for its good performance lies on the one hand in the exploitation of diverse predictions in an ensemble of classifiers and, on the other hand, in the fact that these classifiers may be conceptually simpler and can be learned more reliably (Figure 1(c)). Moreover, we demonstrated both empirically and theoretically that the quadratic growth in the number of learning problems is compensated by the reduction in size for each of the individual problems. For algorithms with a super-linear run-time, round robin binarization may even be faster than the unordered or ordered techniques that are conventionally used in rule learning.

There are several issues that still need to be addressed. First, we want to investigate whether the correlation in the performance gains of boosting and round robin binarization prohibits an effective combination of these two techniques. To this end, we plan to evaluate round robin binarization using C5 and C5-boost as base learners. Alter-

natively, a direct comparison of R^3 to Slipper (Cohen & Singer, 1999) could provide evidence to answer this question.

The main disadvantage of the approach, of course, is its dependency on the number of classes. More classes result in a bigger ensemble which should produce better predictions. In particular in the limiting case, where only two or three classes are available, the benefits should be rather small. This trade-off also needs to be investigated in more detail.

Finally, a tight formal proof of Proposition 6.4 would close the gap in our theoretical argumentation (however, we think that our informal argumentation supports its validity). Likewise, an efficient, tight integration of round robin binarization into the Ripper rule learning algorithm would be highly desirable in order to facilitate a fair empirical evaluation of the effiency of the approach.

Acknowledgements

I wish to thank William Cohen, Eibe Frank, Stefan Kramer, Johann Petrak, Lupčo Todorovski, Gerhard Widmer, and the maintainers of and contributors to the UCI collection of databases for discussions, programs, databases, and pointers to relevant literature.

The Austrian Research Institute for Artificial Intelligence is supported by the Austrian Federal Ministry of Education, Science and Culture. The author was supported by the ESPRIT long-term project METAL (26.357).

References

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

Anand, R., Mehrotra, K. G., Mohan, C. K., & Ranka, S. (1995). Efficient classification for multiclass problems using modular neural networks. *IEEE Transactions on Neural Networks*, *6*, 117–124.

Angulo, C., & Català, A. (2000). K-SVCR. a multi-class support vector machine. *Proceedings of the 11th European Conference on Machine Learning (ECML-2000)* (pp. 31–38). Springer-Verlag.

Bay, S. D. (1999). Nearest neighbor classification from multiple feature subsets. *Intelligent Data Analysis*, *3*, 191–209.

Blake, C. L., & Merz, C. J. (1998). UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`. Department of Information and Computer Science, University of California at Irvine. Irvine, CA.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*, 123–140.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression trees.* Pacific Grove, CA: Wadsworth & Brooks.

Clark, P., & Boswell, R. (1991). Rule induction with CN2: Some recent improvements. *Proceedings of the 5th European Working Session on Learning (EWSL-91)* (pp. 151–163). Porto, Portugal: Springer-Verlag.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, *3*, 261–283.

Cohen, W. W. (1995). Fast effective rule induction. *Proceedings of the 12th International Conference on Machine Learning (ML-95)* (pp. 115–123). Lake Tahoe, CA: Morgan Kaufmann.

Cohen, W. W., & Singer, Y. (1999). A simple, fast, and effective rule learner. *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99)* (pp. 335–342). Menlo Park, CA: AAAI/MIT Press.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*, 273–297.

Dietterich, T. G. (1997). Machine learning research: Four current directions. *AI Magazine*, *18*, 97–136.

Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, *40*, 139–158.

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

Feelders, A., & Verkooijen, W. (1995). Which method learns most from the data? Methodological issues in the analysis of comparative studies. *Proceedings of the 5th International Workshop on Artificial Intelligence and Statistics*. Fort Lauderdale, Florida.

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. *Proceedings of the 13th International Conference on Machine Learning* (pp. 148–156). Bari, Italy: Morgan Kaufmann.

Friedman, J. H. (1996). *Another approach to polychotomous classification* (Technical Report). Department of Statistics, Stanford University, Stanford, CA. To appear in *Machine Learning*.

Fürnkranz, J. (1997). Pruning algorithms for rule learning. *Machine Learning*, *27*, 139–171.

Fürnkranz, J. (1999). Separate-and-conquer rule learning. *Artificial Intelligence Review*, *13*, 3–54.

Fürnkranz, J., & Widmer, G. (1994). Incremental Reduced Error Pruning. *Proceedings of the 11th International Conference on Machine Learning (ML-94)* (pp. 70–77). New Brunswick, NJ: Morgan Kaufmann.

Hastie, T., & Tibshirani, R. (1998). Classification by pairwise coupling. *Advances in Neural Information Processing Systems 10 (NIPS-98)* (pp. 507–513). MIT Press.

Kolen, J. F., & Pollack, J. B. (1991). Back propagation is sensitive to initial conditions. *Advances in Neural Information Processing Systems (NIPS-91)* (pp. 860–867). Morgan Kaufmann.

Kreßel, U. H.-G. (1999). Pairwise classification and support vector machines. In B. Schölkopf, C. Burges and A. Smola (Eds.), *Advances in kernel methods: Support vector learning*, chapter 15, 255–268. Cambridge, MA: MIT Press.

Lu, B.-L., & Ito, M. (1999). Task decomposition and module combination based on class relations: A modular neural network for pattern classification. *IEEE Transactions on Neural Networks, 10*, 1244–1256.

Opitz, D., & Maclin, R. (1999). Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research, 11*, 169–198.

Pyle, D. (1999). *Data preparation for data mining*. San Francisco, CA: Morgan Kaufmann.

Rivest, R. L. (1987). Learning decision lists. *Machine Learning, 2*, 229–246.

Schmidt, M. S. (1996). Identifying speakers with support vector networks. *Proceedings of the 28th Symposium on the Interface (INTERFACE-96)*. Sydney, Australia.

Schmidt, M. S., & Gish, H. (1996). Speaker identification via support vector classifiers. *Proceedings of the 21st IEEE International Conference Conference on Acoustics, Speech, and Signal Processing (ICASSP-96)* (pp. 105–108). Atlanta, GA.

Tesauro, G. (1989). Connectionist learning of expert preferences by comparison training. *Advances in Neural Information Processing Systems 1 (NIPS-88)* (pp. 99–106). Morgan Kaufmann.

Utgoff, P. E., & Clouse, J. (1991). Two kinds of training information for evaluation function learning. *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI-91)* (pp. 596–600). Anaheim, CA: AAAI Press.

Witten, I. H., & Frank, E. (2000). *Data mining — practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann Publishers.