Noname manuscript No.
(will be inserted by the editor)

Efficient Prediction Algorithms for Binary Decomposition Techniques

Sang-Hyeun Park · Johannes Fürnkranz

the date of receipt and acceptance should be inserted later

Abstract Binary decomposition methods transform multiclass learning problems into a series of two-class learning problems that can be solved with simpler learning algorithms. As the number of such binary learning problems often grows super-linearly with the number of classes, we need efficient methods for computing the predictions. In this paper, we discuss an efficient algorithm that queries only a dynamically determined subset of the trained classifiers, but still predicts the same classes that would have been predicted if all classifiers had been queried. The algorithm is first derived for the simple case of pairwise classification, and then generalized to arbitrary pairwise decompositions of the learning problem in the form of ternary error-correcting output codes under a variety of different code designs and decoding strategies.

Keywords binary decomposition, pairwise classification, ternary ECOC, multiclass classification, aggregation, efficient decoding, efficient voting

1 Introduction

Many learning algorithms can only deal with two-class problems. For multiclass problems, they have to rely on *binary decomposition* (or *binarization)* procedures that transform the original learning problem into a series of binary learning problems. A standard solution for this problem is the *one-against-all* approach, which constructs one binary classifier for each class, where the positive training examples are those belonging to this class and the negative training examples are formed by the union of all other classes.

S. Park · J. Fürnkranz
Knowledge Engineering Group, Department of Computer Science, TU Darmstadt, Germany
E-mail: park@ke.tu-darmstadt.de

J. Fürnkranz
E-mail: juffi@ke.tu-darmstadt.de

An alternative approach, known as *pairwise classification* or *round robin classification* has recently gained attention (Fürnkranz 2002; Wu et al. 2004). Its basic idea is to transform a k-class problem into $k(k-1)/2$ binary problems, one for each pair of classes. This approach has been shown to produce more accurate results than the one-against-all approach for a wide variety of learning algorithms such as support vector machines (Hsu and Lin 2002) or rule learning algorithms (Fürnkranz 2002). Moreover, Fürnkranz (2002) has also proved that despite the fact that even though the number of binary classifiers is quadratic in the number of classes, the ensemble can in fact be *trained* faster than the conventional one-against-all technique. However, in order to obtain a final prediction, we still have to combine the predictions of all $k(k-1)/2$ classifiers, which can be very inefficient for large values of k.

Our first contribution is a novel solution for this problem. Unlike previous proposals (such as (Platt et al. 1999); cf. Section 3.3) our approach is not heuristic but is guaranteed to produce exactly the same prediction as the full pairwise classifier, which in turn has been shown to optimize the Spearman rank correlation with the target labels (Hüllermeier and Fürnkranz 2004a). In essence, the algorithm selects and evaluates iterative pairwise classifiers using a simple heuristic to minimize the number of used pairwise classifiers that are needed to determine the correct *top rank* class of the complete (weighted) voting.

Pairwise classification may be viewed as a special case of *ternary error-correcting output codes* (Allwein et al. 2000) which are a general framework for describing different decompositions of a multiclass problem into a set of binary problems. They extend conventional *error-correcting output codes (ECOCs)* (Dietterich and Bakiri 1995) with the possibility of representing that some classifiers may not be trained on all available examples. Although not strictly necessary, the number of the generated binary classification problems typically exceeds the number of class values ($n > k$), for many common general encoding techniques by several orders of magnitude. For example, for the above-mentioned pairwise classification, the number of binary classifiers is quadratic in the number of classes. Thus, the increase in predictive accuracy comes with a corresponding increase in computational demands at classification time.

In this paper, we generalize the previously mentioned algorithm to allow for quick decoding of arbitrary ternary ECOC ensembles. The resulting predictions are guaranteed to be equivalent to the original decoding strategy except for ambiguous final predictions. We show that the algorithm is applicable to various decoding techniques, including Hamming distance, Euclidean distance, their attenuated counter-parts, loss-based decoding, and the Laplace decoding strategy.

Besides pairwise classification and ECOCs, a variety of other decomposition-based approaches have been proposed for the multiclass classification task. It is not the goal of this paper to contribute to the discussion of their respective virtues—for a recent survey on this subject we refer to (Lorena et al. 2008). Our contribution this on-going debate is to solve one of the most severe problems with two of the most popular decomposition methods, namely

by improving their classification efficiency without changing their predictive quality.

In Section 2, we will briefly recapitulate pairwise classification and ECOC with an overview of typical code designs and decoding methods. In Section 3, the fast voting algorithm for pairwise classification is presented and in Section 4 subsequently generalized for use with general ternary ECOCs. The performance of these algorithms is evaluated and discussed in Section 5. Finally, we will conclude and elaborate on possible future directions.

Parts of this paper have previously appeared as (Park and Fürnkranz 2007a, 2009).

2 Preliminaries

A multiclass classification problem consists of a set of instances $X = \{x_i \mid i = 1 \dots l\}$ and a set of classes $K = \{c_i \mid i = 1 \dots k\}$, where each instance is exactly associated to one class. Typically, a subset of instances $X_t \subseteq X$ along with their corresponding true class associations is given, which are used to learn an classifier $f(.) : X \to K$. This classifier is supposed to predict for previously unseen instances x_i to the correct class, such that some performance criterion (e.g. classification accuracy) is maximized.

In the following, we recapitulate two basic approaches for tackling multiclass problems by reducing them to an ensemble of binary problems, namely the pairwise classification and error-correcting output codes.

2.1 Pairwise Classification

The key idea of pairwise classification is to learn one classifier for each pair of classes. At classification time, the prediction of these classifiers are then combined into an overall prediction.

2.1.1 Training Phase

A pairwise or round robin classifier trains a set of $k(k-1)/2$ *binary classifiers* $C_{i,j}$, one for each pair of classes $(c_i, c_j), i < j$. We will refer to the learning algorithm that is used to train the classifiers $C_{i,j}$ as the *base learner*. Each binary classifier is only trained on the subset of training examples belonging to classes c_i and c_j, all other examples are ignored for the training of $C_{i,j}$.

It is important to note that the total effort required to train the entire ensemble of the $k(k-1)/2$ classifiers is only linear in the number of classes k, and, in fact, cheaper than the training of a one-against-all ensemble. It is easy to see this, if one considers that in the one-against-all case each training example is used k times (namely in each of the k binary problems), while in the round robin approach each example is only used $k-1$ times, namely only

in those binary problems, where its own class is paired against one of the other $k-1$ classes.

Typically, the binary classifiers are class-symmetric, i.e., the classifiers $C_{i,j}$ and $C_{j,i}$ are identical. However, for some types of classifiers this does not hold. For example, rule learning algorithms will always learn rules for the positive class, and classify all uncovered examples as negative. Thus, the predictions may depend on whether class c_i or class c_j has been used as the positive class. As has been noted in (Fürnkranz 2002), a simple method for solving this problem is to average the predictions of $C_{i,j}$ and $C_{j,i}$, which basically amounts to the use of a so-called *double round robin* procedure, where we have two classifiers for each pair of classes. We will use this procedure for our results with Ripper.

2.1.2 Prediction Phase

At classification time, each binary classifier $C_{i,j}$ is queried and issues a vote (a prediction for either c_i or c_j) for the given example. This can be compared with sports and games tournaments, in which each player plays each other player once. In each game, the winner receives a point, and the player with the maximum number of points is the winner of the tournament. In this paper, we will assume binary classifiers that return class probabilities $p(c_i|c_i \vee c_j)$ and $p(c_j|c_i \vee c_j)$. These can be used for *weighted voting*, i.e., we predict the class that receives the maximum weighted number of votes:

$$c^* = \arg\max_{c \in K} \sum_{c' \in K \setminus c} p(c \,|\, c \vee c')$$

Other choices for decoding pairwise classifiers are possible (cf., e.g., (Hastie and Tibshirani 1997; Wu et al. 2004)), but voting is surprisingly stable. For example, one can show that weighted voting, where each binary vote is split according to the probability distribution estimated by the binary classifier, minimizes the Spearman rank correlation with the correct ranking of classes, provided that the classifier provides good probability estimates (Hüllermeier et al. 2008). Also, empirically, weighted voting seems to be a fairly robust method that is hard to beat with other, more complex methods (Hüllermeier and Fürnkranz 2004b).

2.2 Error-Correcting Output Codes (ECOC)

Error-correcting output codes (ECOCs) (Dietterich and Bakiri 1995) are another well-known technique for handling multiclass classification problems by reducing the k-class classification problem to a series of n binary problems. The method has its origin in coding and information Theory (MacWilliams and Sloane 1983), where it is used for detecting and correcting errors in suitably encoded signals. In the context of classification, we *encode* the class variable

with a n-dimensional binary *code word*, whose entries specify whether the example in question is a positive or a negative example in the corresponding binary classifier.

Formally, each class c_i $(i = 1 \dots k)$ is associated with a so-called *code word* $\mathbf{cw}_i \in \{-1, 1\}^n$ of length n. We denote the j-th bit of $\mathbf{cw}_i$ as $b_{i,j}$. In the context of ECOC, all relevant information is summarized in a so-called *coding matrix* $(m_{i,j}) = M \in \{-1, 1\}^{k \times n}$, whose i-th row describes code word $\mathbf{cw}_i$, whereas the j-th column represents a classifier f_j. The set of all such classifiers is denoted as $\mathcal{C} = \{f_1, \dots, f_n\}$.

Furthermore, the coding matrix implicitly describes a decomposition scheme of the original multiclass problem. In each column j the rows contain a (1) for all classes whose training examples are used as positive examples, and (-1) for all negative examples for the corresponding classifier f_j.

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

The above example shows a coding matrix for 4 classes, which are encoded with 6 classifiers. The first classifier uses the examples of classes 1 and 2 as positive examples, and the examples of classes 3 and 4 as negative examples.

Typically, the number of classifiers exceeds the number of classes, i.e., $n > k$. This allows longer code words, so that the mapping to the closest code word is not compromised by individual mistakes of a few binary classifiers. Thus, ECOCs not only make multiclass problems amenable to binary classifiers, but may also yield a better predictive performance than conventional multiclass classifiers.

At prediction time, all binary classifiers are queried, and collectively predict an n-dimensional vector, which must be *decoded* into one of the original class values, e.g., by assigning it to the class of the closest code word. More precisely, for the classification of a test instance x, all binary classifiers are evaluated and their predictions, which form a *prediction vector* $\mathbf{p} = [f_1(x), f_2(x), \dots, f_n(x)]$, are compared to the code words. The class c^* whose associated code word $\mathbf{cw}_{c^*}$ is "nearest" to $\mathbf{p}$ according to some distance measure $d(.)$ is returned as the overall prediction, i.e.

$$c^* = \arg\min_c \, d(\mathbf{cw}_c, \mathbf{p})$$

For computing the similarity between the prediction vector and the code word, the most common choice is the Hamming Distance, which measures the number of bit positions in which the prediction vector $\mathbf{p}$ differs from a code word $\mathbf{cw_i}$.

$$d_H(\mathbf{cw_i}, \mathbf{p}) = \sum_{j=1}^{n} \frac{|m_{i,j} - p_j|}{2} \tag{1}$$

The good performance of ECOCs has been confirmed in subsequent theoretical and practical work. For example, it has been shown that ECOCs can

to some extent correct variance and even bias of the underlying learning algorithm (Kong and Dietterich 1995). An exhaustive survey of work in this area is beyond the scope of this paper, but can be found in (Windeatt and Ghaderi 2003).

2.3 Ternary ECOC

Conventional ECOCs as described in the previous section always use all classes and all training examples for training a binary classifier. Thus, binary decompositions which use only parts of the data (such as pairwise classification) can not be modeled in this framework.

Allwein et al. (2000) extended the ECOC approach to the ternary case, where code words are now of the form $\mathbf{cw}_i \in \{-1, 0, 1\}^n$. The additional code $m_{i,j} = 0$ denotes that examples of class c_i are ignored for training classifier f_j. We will sometimes also denote a classifiers f_j as C_{P_j,N_j}, where P_j is the set of labels that are used as positive examples, and N_j is the set of all labels that are used as negative examples. We will then say a classifier $f_j = C_{P_j,N_j}$ is *incident* to a class c_i, if the examples of c_i are either positive or negative examples for f_j, i.e., if $c_i \in P_j$ or $c_i \in N_j$, which implies that $m_{i,j} \neq 0$.

This extension increases the expressive power of ECOCs, so that now nearly all common multiclass binarization methods can be modelled. For example, pairwise classification (Section 2.1), where one classifier is trained for each pair of classes, could not be modeled in the original framework, but can be modeled with ternary ECOCs. Its coding matrix has $n = k(k-1)/2$ columns, each consisting of exactly one positive value (+1), exactly one negative value (−1), and $k-2$ zero values (0). Below, we show the coding matrix of a pairwise classifier for a 4-class problem.

$$M = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 1 & 0 \\ 0 & -1 & 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 0 & -1 & -1 \end{pmatrix} \tag{2}$$

The conventionally used Hamming decoding can be adapted to this scenario straight-forwardly. Note that while the code word can now contain 0-values, the prediction vector is considered as a set of binary predictions which can only predict either −1 or 1. Thus, a zero symbol in the code word ($m_{i,j} = 0$) will always increase the distance by 1/2 (independent of the prediction).

Many alternative decoding strategies have been proposed in the literature. Along with the generalization of ECOCs to the ternary case, Allwein et al. (2000) proposed a loss-based strategy. Escalera et al. (2006) discussed the shortcomings of traditional Hamming distance for ternary ECOCs and presented two novel decoding strategies, which should be more appropriate for dealing with the zero symbol. We considered all these decoding strategies in our work. In Section 4.5 we show how our method can be adapted to different decoding functions.

2.4 Code Design for (Ternary) ECOCs

A well-known theorem from coding theory states that if the minimal Hamming Distance between two arbitrary code words is h, the ECC framework is capable of correcting up to $\lfloor \frac{h}{2} \rfloor$ bits. This is easy to see, since every code word x has a $\lfloor \frac{h}{2} \rfloor$ neighborhood, for which every code in this neighborhood is nearer to x than to any other code word. Thus, it is obvious that good error correction crucially depends on the choice of a suitable coding matrix.

Unfortunately, some of the results in coding theory are not fully applicable to the machine learning setting. For example, the above result assumes that the bit-wise error is independent, which leads to the conclusion that the minimal Hamming Distance is the main criterion for a good code. But this assumption does not necessarily hold in machine learning. Classifiers are learned with similar training examples and therefore their predictions tend to correlate. Thus, a good ECOC code also has to consider, e.g., column distances, which may be taken as a rough measure for the independence of the involved classifiers.

In the machine-learning literature, a considerable amount of research has been devoted to code design for ternary ECOCs (see, e.g., Crammer and Singer 2002; Pimenta et al. 2008), but without reaching a clear conclusion. We want to emphasize that our work does not contribute to this discussion, because we will not be concerned with comparing the predictive quality of different coding schemes. Our goal is to show that, irrespective of the selected coding scheme, we can achieve a substantial reduction in prediction time, without changing the predicted outcome.

Nevertheless, we will briefly review common coding schemes, because we will later demonstrate that our algorithm is applicable to different types of coding schemes. Essentially, one can distinguish between four code families:

Exhaustive Ternary Codes Exhaustive ternary codes cover all possible classifiers involving a given number of classes l. More formally, a (k, l)-exhaustive ternary code defines a ternary coding matrix M, for which every column j contains exactly l non-zero values, i.e., $\sum_{i \in K} |m_{i,j}| = l$. Obviously, in the context of multiclass classification, only columns with at least one positive $(+1)$ and one negative (-1) class are useful. The following example shows a $(4, 3)$-exhaustive code.

$$M = \begin{pmatrix} 1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 1 & -1 & 1 & 0 & 0 & 0 & 1 & 1 & -1 \\ -1 & 1 & 1 & 0 & 0 & 0 & 1 & -1 & 1 & 1 & -1 & 1 \\ 0 & 0 & 0 & -1 & 1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 \end{pmatrix} \tag{3}$$

The number of classifiers for a (k, l) exhaustive ternary code is $\binom{k}{l}(2^{l-1} - 1)$, since the number of *binary* exhaustive codes is $2^{l-1} - 1$ and the number of combinations to select l row positions from k rows is $\binom{k}{l}$. These codes are a straightforward generalization of the exhaustive binary codes, which

were considered in the first works on ECOC (Dietterich and Bakiri 1995), to the ternary case. Note that $(k, 2)$-exhaustive codes correspond to pairwise classification.

In addition, we define a *cumulative* version of exhaustive ternary codes, which subsumes all (k, i)-exhaustive codes with $i = 2, 3, \ldots, l$ up to a specific level l. In this case, we speak of (k, l)-cumulative exhaustive codes, which generate a total of $\sum_{i=2}^{l} \binom{k}{i}(2^{i-1} - 1)$ columns. For a dataset with k classes, (k, k)-cumulative exhaustive codes represent the set of all possible binary classifiers.

Random Codes We consider two types of randomly generated codes. The first variant allows to control the probability distribution of the set of possible symbols $\{-1, 0, 1\}$ from which random columns are drawn. By specifying a parameter $r \in [0, 1]$, the probability for the zero symbol is set to $p(\{0\}) = r$, whereas the remainder is equally subdivided to the other symbols: $p(\{1\}) = p(\{-1\}) = \frac{1-r}{2}$. This type of code allows to control the *sparsity* of the coding matrix, which will be useful for evaluating which factors determine the performance of our algorithms.

The second random code generation method selects randomly a subset from the set of all possible classifiers $\mathcal{C}$. This set of classifiers $\mathcal{C}$ equals the cumulative ternary code matrix where the used level l equals the number of classes k. Obviously, this variant guarantees that no duplicate classifiers are generated, whereas it can occur in the other variant. We do not enforce this, because we wanted to model and evaluate two interpretations of randomly generated codes: randomly filled matrices and randomly selected classifiers.

Coding Theory, BCH-Codes Many different code types were developed within coding theory. We pick the so-called BCH Codes (Bose and Ray-Chaudhuri 1960) as a representative, because they have been studied in depth and have properties which are favourable in practical applications. For example, the desired minimum Hamming distance of M can be specified, and fast decoding methods are available. Note, however, that efficient decoding in coding theory has the goal to minimize the complexity of finding the nearest code word given the received *full* code word. In our setting, we are interested in minimizing the classifier evaluations, and this relates to using the minimum number of *bits* of the receiving code word to estimate the nearest code word respectively class. Although some concepts of efficient decoding in coding theory seem to be transferable to our setting, they lack the capability to be a general purpose decoding method for arbitrary coding matrices.

A detailed description of this code family is beyond the scope of this paper, but we refer to (Bose and Ray-Chaudhuri 1960; MacWilliams and Sloane 1983) for a detailed description and further information regarding BCH-Codes. In our evaluation, we considered binary BCH codes of lengths $7, 15, 31, 63, 127$, and 255. Similarly to (Dietterich and Bakiri 1995), we randomly selected k code words from the set of codes, if k is the number of classes.

Domain-Dependent Codes This type of codes project data-specific relationships or expert knowledge explicitly to the coding matrix. For example, the knowledge of an inherent hierarchy or order among the classes can be used to model classifiers which exploit this information (e.g., Melvin et al. 2007; Cardoso and da Costa 2007). Another interesting direction of generating a data-based code is considered by Pujol et al. (2006). Their proposed algorithm DECOC tries to generate a coding matrix, whose columns consist of the best discriminating classifiers on the considered dataset. By applying only classifiers with the maximum discriminative ability, they expect to maximize the overall prediction accuracy. Also, it seems to be rather efficient, since they restrict the length of the coding matrix.

For the work reported in this paper, we did not consider domain-dependent codes, because they need to be fit to each individual dataset.

3 Efficient Prediction for Pairwise Classification

Although the training effort for the entire ensemble of pairwise classifiers is only linear in the number of examples, at prediction time we still have to query a quadratic number of classifiers. In this section, we discuss an algorithm that allows to significantly reduce the number of classifier evaluation without changing the prediction of the ensemble.

3.1 Key Idea

Weighted or unweighted voting predicts the top rank class by returning the class with the highest accumulated voting mass after evaluation of all pairwise classifiers. During such a procedure there exist many situations where particular classes can be excluded from the set of possible top rank classes, even if they reach the maximal voting mass in the remaining evaluations. Consider following simple example: Given k classes with $k > j$, if class c_a has received more than $k - j$ votes and class c_b lost j votings, it is impossible for c_b to achieve a higher total voting mass than c_a. Thus further evaluations with c_b can be safely ignored for the comparison of these two classes.

To increase the reduction of evaluations we are interested in obtaining such exploitable situations frequently. Pairwise classifiers will be selected depending on a *loss* value, which is the amount of potential voting mass that a class has *not* received. More specifically, the loss l_i of a class c_i is defined as $l_i := p_i - v_i$, where p_i is the number of evaluated incident classifiers of c_i and v_i is the current vote amount of c_i. Obviously, the loss will begin with a value of zero and is monotonically increasing. The class with the current minimal loss is one of the top candidates for the top rank class.

Algorithm 1 QWeighted

Require: pairwise classifiers $C_{i,j}$ with $1 \leq i < j \leq k$, testing instance $\mathbf{x} \in X$

1: $\mathbf{l} \in \mathbb{R}^k \leftarrow 0$ ▷ loss values vector
2: $c^* \leftarrow$ NULL
3: $G \leftarrow \{\}$ ▷ keep track of evaluated classifiers
4: **while** $c^* =$ NULL **do**
5: $c_a \leftarrow \arg\min_{c_i \in K} l_i$ ▷ select top candidate class
6: $c_b \leftarrow \arg\min_{c_j \in K \setminus \{c_a\}, C_{a,j} \notin G} l_j$ ▷ select second
7: **if** no c_b exists **then**
8: $c^* \leftarrow c_a$ ▷ top rank class determined
9: **else** ▷ evaluate
10: $v_{ab} \leftarrow C_{a,b}(x)$ ▷ one vote for c_a ($v_{ab} = 1$) or c_b ($v_{ab} = 0$)
11: $l_a \leftarrow l_a + (1 - v_{ab})$ ▷ update voting loss for c_a
12: $l_b \leftarrow l_b + v_{ab}$ ▷ update voting loss for c_b
13: $G \leftarrow G \cup C_{a,b}$ ▷ update already evaluated classifiers
return c^*

3.2 The QWeighted Algorithm

Algorithm 1 shows the QWeighted algorithm, which implements this idea. First the pairwise classifier $C_{a,b}$ will be selected for which the losses l_a and l_b of the relevant classes c_a and c_b are minimal, provided that the classifier $C_{a,b}$ has not yet been evaluated. In the case of multiple classes that have the same minimal loss, there exists no further distinction, and we select a class randomly from this set. Then, the losses l_a and l_b will be updated based on the evaluation returned by $C_{a,b}$ (recall that v_{ab} is interpreted as the amount of the voting mass of the classifier $C_{a,b}$ that goes to class c_a and $1 - v_{ab}$ is the amount that goes to class c_b). These two steps will be repeated until all classifiers for the class c_m with the minimal loss has been evaluated. Thus the current/estimated loss l_m is the correct loss for this class. As all other classes already have a greater loss and considering that the losses are monotonically increasing, c_m is the correct *top rank* class.

Theoretically, a minimal number of comparisons of $k - 1$ is possible (*best case*). Assuming that the incident classifiers of the correct top rank c^* always returns the maximum voting amount ($l^* = 0$), c^* is always in the set $\{c_j \in K | l_j = \min_{c_i \in K} l_i\}$. In addition, c^* should be selected as the first class in step 1 of the algorithm among the classes with the minimal loss value. It follows that exactly $k - 1$ comparisons will be evaluated, more precisely all incident classifiers of c^*. The algorithm terminates and returns c^* as the correct top rank.

The *worst case*, on the other hand, is still $k(k-1)/2$ comparisons, which can, e.g., occur if all pairwise classifiers classify randomly with a probability of 0.5. In practice, the number of comparisons will be somewhere between these two extremes, depending on the nature of the problem. This trade-off will be evaluated in section 5.1.

3.3 Related Work

The loss l_i, which we use for selecting the next classifier, is essentially identical to the voting-against principle introduced by Cutzu (2003a,b), who also observed that it allows to reliably conclude a class when not all of the pairwise classifiers are present. For example, Cutzu claims that using the voting-against rule one could correctly predict class c_i even if none of the incident pairwise classifiers $C_{i,j}$ ($j = 1 \ldots k, j \neq i$) are used. However, this argument is based on the assumption that all base classifiers classify correctly. Moreover, if there is a second class c_j that should ideally receive $k-2$ votes, voting-against could only conclude a tie between classes c_i and c_j, as long as the vote of classifier $C_{i,j}$ is not known. The main contribution of his work, however, is a method for computing posterior class probabilities in the voting-against scenario. Our approach builds upon the same ideas as Cutzu's, but our contribution is the algorithm that exploits the voting-against principle to effectively increase the prediction efficiency of pairwise classifiers without changing the predicted results.

The voting-against principle was already used earlier in the form of DDAGs (Platt et al. 1999), which organize the binary base classifiers in a decision graph. Each node represents a binary decision that rules out the class that is not predicted by the corresponding binary classifier. At classification time, only the classifiers on the path from the root to a leaf of the tree (at most $k-1$ classifiers) are consulted. While the authors empirically show that the method does not lose accuracy on three benchmark problems, it does not have the guarantee of our method, which will always predict the same class as the full pairwise classifier. Intuitively, one would also presume that a fixed evaluation routine that uses only $k-1$ of the $k(k-1)/2$ base classifiers will sacrifice one of the main strengths of the pairwise approach, namely that the influence of a single incorrectly trained binary classifier is diminished in a large ensemble of classifiers (Fürnkranz 2003). Our empirical results (presented in Section 5.1) will confirm that DDAGs are only slightly more efficient but less accurate than the QWeighted approach.

4 Efficient ECOC Decoding

In this section, we will generalize the QWeighted algorithm to arbitrary ternary ECOC matrices. We will discuss the three key modifications that have to be made: first, Hamming decoding has to be reduced to a voting process (Section 4.1), second, the heuristic for selecting the next classifier has to be adapted to the case where multiple classifiers can be incident with a pair of classes (Section 4.2), and finally the stopping criterion can be improved to take multiple incidences into account (Section 4.3). We will then present the resulting QuickECOC algorithm for Hamming decoding in Section 4.4. Finally, we will discuss how QuickECOC can be adapted to different decoding techniques (Section 4.5).

4.1 Reducing Hamming Distances to Voting

Obviously, pairwise classification may be considered as a special case of ternary ECOCs, where each column of the coding matrix contains exactly one positive, one negative, and $k-2$ ignore values, as shown in (2). Thus, it is natural to ask the the question whether the QWeighted algorithm can be generalized to arbitrary ternary ECOCs.

To do so, we first have to consider that ECOCs typically use Hamming distance for decoding, whereas pairwise classification typically uses a simple voting procedure. In voting aggregation, the class that receives the most votes from the binary classifiers is predicted, i.e.,

$$\tilde{c} := \arg\max_{i \in K} \sum_{j \neq i, j \in K} f_{i,j}$$

where $f_{i,j}$ is the prediction of the pairwise classifier that discriminates between classes c_i and c_j.

Traditional ECOC with Hamming decoding predicts the class c^* whose code word $\mathbf{cw}_{c^*}$ has the minimal Hamming Distance $d_H(\mathbf{cw}_{c^*}, \mathbf{p})$ to the prediction vector $\mathbf{p} = (p_1, \ldots, p_n)$. A certain analogy between both methods can be seen easily and was further examined by Kong and Dietterich (1995) and has a relation to *correlation decoding* from coding theory (Gallager 1968). However, we briefly repeat with following lemma that the minimization of Hamming distances reduces to voting aggregation:

Lemma 1 *Let* $v_{i,j} := \left(1 - \frac{|m_{i,j} - p_j|}{2}\right)$ *be the vote that classifier* f_j *gives to class* c_i*, then*

$$\arg\min_{i=1\ldots k} d_H(\mathbf{cw_i}, \mathbf{p}) = \arg\max_{i=1\ldots k} \sum_{j=1}^{n} v_{i,j}$$

Proof Recall that

$$d_H(\mathbf{cw_i}, \mathbf{p}) = \sum_{a=1}^{n} \frac{|cw_{i_a} - p_a|}{2} = \sum_{a=1}^{n} \frac{|m_{i,a} - p_a|}{2}$$

Let $b_{i,a} := \frac{|m_{i,a} - p_a|}{2}$. Since $b_{i,a} \in [0, 1]$,

$$\arg\min_{i=1\ldots k} \sum_{a=1}^{n} b_{i,a} = \arg\max_{i=1\ldots k} \sum_{a=1}^{n} (1 - b_{i,a}) = \arg\max_{i=1\ldots k} \sum_{a=1}^{n} v_{i,a}$$

holds and we obtain the proposition.

4.2 Next Classifier Selection

The QWeighted algorithm always pairs the class with the least amount of voting loss l_i with the class that has the least amount of voting loss among all classes with which it has not yet been paired, and evaluates the resulting classifier. This choice of is deterministic because, obviously, there is only one classifier that is incident with any given pair of classes. General ECOC coding matrices, on the other hand, can have more then two non-zero entries per column. As a result, a pair of classes may be incident to multiple binary classifiers. This has the consequence that the selection of the next classifier to evaluate has gained an additional degree of freedom. For example, consider a 4-class problem (c_1, c_2, c_3, c_4) using 3-level ternary exhaustive codes, as shown in (3). If classes c_1 and c_2 are those with the current minimum voting loss, we could select any of four different classifiers that discriminate the classes c_1 and c_2, namely $f_2 = C_{\{1,3\},\{2\}}$, $f_3 = C_{\{1\},\{2,3\}}$, $f_5 = C_{\{1,4\},\{2\}}$, and $f_6 = C_{\{1\},\{2,4\}}$.

QuickECOC uses a selection process which conforms to the key idea of QWeighted: Given the current favorite class c_{i_0}, we select all incident classifiers $\mathcal{C}_{i_0}$, i.e.,

$$\mathcal{C}_{i_0} = \{C_{P_j,N_j} \in \mathcal{C} \mid c_{i_0} \in P_j \vee c_{i_0} \in N_j\} \tag{4}$$

Let K_j denote the set of classes, which are involved in the binary classifier $f_j = C_{P_j,N_j}$, but with a different sign than c_{i_0}, i.e.,

$$K_j = \begin{cases} P_j & \text{if } c_{i_0} \in N_j \\ N_j & \text{if } c_{i_0} \in P_j \end{cases}$$

In other words, K_j contains all rows i of column j in the coding matrix M, for which $m_{i,j} \neq m_{i_0,j}$ and $m_{i,j} \neq 0$ hold. We then compute a score

$$s(j) = \sum_{i \in K_j} k - r(i)$$

for every classifier $c_j \in \mathcal{C}_{i_0}$, where $r(i)$ denotes the rank of class c_i when all classes are increasingly ordered by their current votings (or, equivalently, ordered by decreasing distances). Finally, we select the classifier f_{j_0} with the maximal score $s(j_0)$. Roughly speaking, this amounts to selecting the classifier which discriminates c_{i_0} to the greatest number of currently highly ranked classes.

We experienced that this simple score-based selection was superior among other tested methods, whose presentation and evaluation we omit here. One point to note is, that for the special case of pairwise codes, this scheme is identical to the one used by QWeighted.

4.3 Stopping Criterion

The key idea of the algorithm is to stop the evaluation of binary classifiers as soon as it is clear which class will be predicted, irrespective of the outcome of

all other classifiers. Thus, the QuickECOC algorithm has to check whether c_{i_0}, the current class with the minimal Hamming distance to $\mathbf{p}$, can be caught up by other classes at the current state. If not, c_{i_0} can be safely predicted.

A straight-forward adaptation of the QWeighted algorithm for pairwise classification would simply compute the maximal possible Hamming distance for c_{i_0} and compare this distance to the current Hamming distances l_i of all other classes $c_i \in K \setminus \{c_{i_0}\}$. The maximal possible Hamming distance for c_{i_0} can be estimated by assuming that all outstanding evaluations involving c_{i_0} will increase its Hamming distance by 1 and all remaining outstanding (non-incident) classifiers will increase its distance by 0.5 (according to the definition of hamming distance for ternary code words). Thus, we simply add the number of remaining incident classifiers of c_{i_0} and one half of the number of remainder classifiers to its current distance l_{i_0}.

Note, however, that this simple method makes the assumption that all binary classifiers only increase the Hamming distance of c_{i_0}, but not of the other classes. This is unnecessarily pessimistic, because each classifier will always equally increase the Hamming distance for *all* (or none) of the incident classes that have the same sign in the coding matrix (positive or negative). Thus, we can refine the above procedure by computing a separate upper bound of l_{i_0} for each class c_i. This bound does not assume that all remaining incident classifiers will increase the distance for c_{i_0} by 1, but only those where c_i and c_{i_0} are on different sides of the training set. For the cases where either c_i or c_{i_0} was ignored in the training phase, $\frac{1}{2}$ is added to the distance. If there exist no class which can overtake c_{i_0}, the algorithm returns c_{i_0} as the prediction.

Note that the stopping criterion can only test whether no class can surpass the current favorite class. However, there may be other classes with the same Hamming distance. As the QuickECOC algorithm will always return the first class that cannot be surpassed by other classes, this may not be the same class that is returned by the full ECOC ensemble. Thus, in the case, where the decoding is not unique, QuickECOC may return a different prediction. However, in all cases where the code word minimal Hamming distance is unique, QuickECOC will return exactly the same prediction as ECOC.

We also defined a second criterion, which simply stops when all classifiers of the favorite class c_{i_0} have already been evaluated. Strictly speaking, this is a special case of the first stopping criterion and could be removed. However, we found that making this distinction facilitated some of our analyzes (presented in the appendix), so we leave it in the algorithm.

4.4 Quick ECOC Algorithm

Algorithm 2 shows the pseudocode of the QuickECOC algorithm. The algorithm maintains a vector $\mathbf{l} = (l_1, l_2, \ldots, l_k) \in \mathbb{R}^k$, where l_i is the current accumulated Hamming distance of the prediction vector $\mathbf{p}$ to the code word $\mathbf{cw}_i$ of class c_i. The l_i can be seen as lower bounds of the distances $d_H(\mathbf{cw}_i, \mathbf{p})$,

Algorithm 2 QuickECOC

Require: ECOC Matrix $\mathbf{M} = (m_{i,j}) \in \{-1, 0, 1\}^{k \times n}$, binary classifiers $\mathcal{C} = \{f_1, \ldots, f_n\}$, testing instance $\mathbf{x} \in X$

1: $\mathbf{l} \in \mathbb{R}^k \Leftarrow 0$ ▷ Hamming distance vector
2: $c^* \leftarrow$ NULL
3: $\mathcal{C}' \leftarrow \mathcal{C}$
4: **while** $c^* =$ NULL **do**
5: $f_j \leftarrow$SelectNextClassifier$(\mathbf{M}, \mathbf{l})$
6: $p \leftarrow f_j(\mathbf{x})$ ▷ Evaluate classifier
7: **for all** $i \in K$ **do**
8: $l_i \leftarrow l_i + \frac{|m_{i,j} - p|}{2}$
9: $\mathcal{C}' \leftarrow \mathcal{C}' \setminus \{f_j\}$
10: $\mathbf{M} \leftarrow \mathbf{M} \setminus M_j$
11: $i_0 = \arg\min_{i \in K} l_i$
12: ▷ First stopping criterion
13: abort $\leftarrow$ **true**
14: **for all** $i \in K \setminus \{i_0\}$ **do**
15: $n_{Full} \leftarrow |\{f_j \in \mathcal{C}' \mid m_{i,j} \cdot m_{i_0,j} = -1\}|$
16: $n_{Half} \leftarrow |\{f_j \in \mathcal{C}' \mid m_{i,j} \cdot m_{i_0,j} = 0$ **and** $m_{i,j} + m_{i_0,j} \neq 0\}|$
17: **if** $l_{i_0} + n_{Full} + \frac{1}{2} n_{Half} > l_i$ **then**
18: abort $\leftarrow$ **false**
19: ▷ Second stopping criterion
20: **if** abort **or** $\forall f_j \in \mathcal{C}' : m_{i_0,j} = 0$ **then**
21: $c^* \leftarrow c_{i_0}$
return c^*

which are updated incrementally in a loop which essentially consists of four steps:

(1) Selection of the Next Classifier:
First, the next classifier is selected. Depending on the current Hamming distance values, the routine SelectNextClassifier returns a classifier that pairs the current favorite $i_0 = \arg\min_i l_i$ with another class that is selected as described in Section 4.2. In the beginning all values l_i are zero, so that SelectNextClassfier returns an arbitrary classifier f_j.

(2) Classifier Evaluation and Update of Bounds l:
After the evaluation of f_j, $\mathbf{l}$ is updated using the Hamming distance projected to this classifier (as described in Section 4.1) and f_j is removed from the set of possible classifiers.

(3) First Stopping Criterion:
Starting with line 12, the first stopping criterion is checked. It checks whether the current favorite class i_0 can already be safely determined as the class with the maximum number of votes, as described in Section 4.3.

(4) Second Stopping Criterion:
Starting with line 19, the algorithm stops when all incident classifiers of c_{i_0} have been evaluated. In this case, since it holds that $l_{i_0} \leq l_i$ for all classes c_i with l_{i_0} fixed and considering that l_i can only increase monotonically, we can safely ignore all remaining evaluations.

4.5 Adaption to Different Decoding Strategies

As briefly discussed in Section 2.3, various decoding strategies have been proposed as alternatives to Hamming decoding. In this section, we show how QuickECOC can be adapted to a variety of domain-independent encoding strategies via small modifications to the basic algorithm.

In general, there are two locations where adaptations are needed. First, the statistics update step and the first stopping criteria have to be adapted according to the used distance measure. Second, some decoding strategies require a special treatment of the zero symbol, which can, in general, be modeled as a preprocessing step.

In the following, we review some important decoding strategies and show how QuickECOC can be adapted to deal with each strategy.

Euclidean Distance The Euclidean Distance d_E computes the distance between the code-word and the prediction vector Euclidean space.

$$d_E(\mathbf{cw_i}, \mathbf{p}) = ||\mathbf{cw_i} - \mathbf{p}||_2 = \sqrt{\sum_{j=1}^{n}(m_{i,j} - p_j)^2} \tag{5}$$

For minimizing this distance we can ignore the root operation and, instead, minimize $d_E(\mathbf{cw_i}, \mathbf{p})$. This can again be computed incrementally, by substituting the update statement of the pseudocode (line 8) with:

$$l_i \leftarrow l_i + (m_{i,j} - p)^2$$

Consequently, in the sum in line 17, the weight for n_{Half} is changed to 1 and the one for n_{Full} to 4.

Attenuated Euclidean/Hamming Distance These measures work analogously to the Hamming Distance and the Euclidean distance, but distances to zero symbols in the coding vector are ignored (which is equivalent to weighting the distance with $|m_{i,j}|$). The attenuated Euclidean distance is thus defined as

$$d_{AE}(\mathbf{cw_i}, \mathbf{p}) = \sqrt{\sum_{j=1}^{n}|m_{i,j}|(m_{i,j} - p_j)^2} \tag{6}$$

The analogue version for the Hamming distance is:

$$d_{AH}(\mathbf{cw_i}, \mathbf{p}) = \sum_{j=1}^{n}|m_{i,j}|\,\frac{|m_{i,j} - p_j|}{2} \tag{7}$$

The modifications to lines 8 and 17 are analogous to the previous case.

Loss-based Decoding In loss-based decoding Allwein et al. (2000) we assume a score-based base classifier, and want to take the returned score $f(.)$ into consideration. The similarity function d_L is then defined as

$$d_L(\mathbf{cw_i}, \mathbf{p}) = \sum_{j=1}^{n} l(m_{i,j} \cdot f_j) \tag{8}$$

where $l(.)$ is a loss function, such as $l(s) = -s$ or the exponential loss $l(s) = e^{-s}$.

For both loss functions, we assume that we have given a normalizing function $w(.)$ which projects $f_j(x)$ into the interval $[-1, 1]$, e.g.,[1]

$$w(z) = \begin{cases} \frac{z}{\max z} & z \geq 0 \\ \frac{z}{|\min z|} & z < 0 \end{cases}$$

For the linear loss, we substitute line 6 with

$$p \leftarrow w(f_j(x))$$

and the update in line 8 with

$$l_i \leftarrow l_i + \frac{1 - p \cdot m_{i,j}}{2}$$

and remove the occurrences of n_{Half}.

For the exponential loss, we have to change line 6 as above and the update step with

$$l_i \leftarrow l_i + e^{-p \cdot m_{i,j}}.$$

In addition, the weights in line 17 are set to e^1 for n_{Full} and to $e^0 = 1$ for n_{Half}.

Laplace Strategy This measure interprets the zero symbol in a different way: If a code word $\mathbf{cw}_1$ consists of more zero symbols than $\mathbf{cw}_2$, the number of "reasonable" predictions is smaller, so every non-zero symbol prediction of $\mathbf{cw}_1$ should be given a higher weight.

$$d_{LA}(\mathbf{cw_i}, \mathbf{p}) = \frac{E + 1}{E + C + T} = \frac{d_{AH}(\mathbf{cw_i}, \mathbf{p}) + 1}{\sum_{j=1}^{n} |m_{i,j}| + T} \tag{9}$$

where C is the number of bit positions in which they are equal and E in which they differ. So, roughly speaking, depending of the number of zero symbols of $\mathbf{cw_i}$, every bit agreement contributes more or less to the distance measure. T

[1] Note that we did not use such a normalizing function in our actual evaluation since we used a decision tree learner which already returns scores in the right range. Although the normalization of score-based functions, such as SVMs, is not a trivial task, the sketched function $w(.)$ could be possibly determined by estimating $\min f(x)$ and $\max f(x)$ during training time (e.g. saving the largest distances between instances to the hyperplane for each classifier).

is the number of involved classes, in our case $T = 2$, since we employ binary classifiers. Thus, the default value of $d_{LA}(.)$ is $\frac{1}{2}$.

This strategy can be used by incorporating a class- respectively row-based incrementer. Note that each error bit between a code word **cw** and the prediction vector **p** contributes $\frac{1}{b+T}$ to the total distance $d_{LA}(\mathbf{cw}, \mathbf{p})$, where b is the number of non-zero bits of **cw**. This incrementer denoted by I_i for class c_i can be computed as a preprocessing step from the given ECOC Matrix. So, the update step in line 8 has to be changed to

$$l_i \leftarrow l_i + I_i$$

and the weight of n_{Full} in the sum 17 changes to I_i. Besides, n_{Half} can be removed.

Beta Density Distribution Pessimistic Strategy This measure assumes that the distance is a Beta-distributed random variable parameterized by C and E of two code words. The Beta distribution is here defined as

$$\psi(z, E, C) = \frac{1}{T} z^E (1 - z)^C$$

Its expected value is $E(\psi_i) = \frac{E}{E+C}$.
Let $Z_i := \arg \max_{z \in [0,1]} (\psi_i(z))$ and $a_i \in [0, 1]$ such that

$$\int_{Z_i}^{Z_i + a_i} \psi_i(z) = \frac{1}{3}$$

then we define

$$d_{BD}(x, y) = Z_i + a_i \tag{10}$$

The value a_i is regarded as a pessimistic value, which incorporates the uncertainty of Z_i into the distance measure.

Here, we use an approximation of the original strategy. First, similar to the Laplace Strategy, an incrementer is used to determine $Z_i = \frac{E}{E+C}$. And second, instead of using a numerical integration to determine $Z_i + a_i$, its standard deviation is added, which is in compliance with the intended semantic of this overall strategy to incorporate the uncertainty. The incrementer I_i is again set during a preprocessing step and we change the update step (line 8) to

$$l_i \leftarrow l_i + \min(1, (I_i + \sigma_i)).$$

The weight for n_{Full} has to be changed to I_i and n_{Half} has to be removed. In practice, this approximation yielded in all our evaluations the same prediction as the original strategy.

The above decoding techniques were just a few examples, which we have empirically tested. Other techniques can be adapted in a similar fashion. In general, a distance measure is compatible to QuickECOC if the distance can be determined bit-wise or incremental, and the iterative estimate of l_i has to be monotonically increasing, but must never over-estimate the true distance.

5 Experimental Results

In this section, we show the results of the empirical evaluation of our algorithms. We focus on the number of classifier evaluations that have to be performed in order to compute a prediction. We typically do not compare our results in terms of predictive accuracy, because our algorithms make the same prediction as their respective versions that use all classifiers. Nevertheless, unless mentioned otherwise, the reported results are averages of a 10-fold cross-validation, in order to get more reliable estimates.

5.1 Pairwise Classification - Evaluation of QWeighted

5.1.1 UCI Datasets

We start with a comparison of the QWeighted algorithm with the full pairwise classifier and with DDAGs on seven arbitrarily selected multiclass datasets from the UCI database of machine learning databases (Asuncion and Newman 2007). We used four commonly used learning algorithms as base learners (the rule learner Ripper,[2] a Naive Bayes algorithm, the C4.5 decision tree learner, and a support vector machine) all in their implementations in the Weka machine learning library (Witten and Frank 2005). Each algorithm was used as a base classifier for QWeighted, and the combination was run on each of the datasets. A mild parameter tuning was applied to each base algorithm, which does not necessarily help to answer the question of the choice for best predictive combination on these datasets because of its non-exhaustive conducting, but were rather applied to take the fact into account that the predictive performance has an impact on the efficiency of the QWeighted algorithm. Inner 5-fold cross-validation tuning[3] was applied for following base learners and parameters:

- NB:
  - with or without kernel density estimators
- SMO:
  - complexity $\{0.1, 0.2, \ldots, 1\}$
  - exponent of polynomial kernel $\{0.5, 1, 1.5, 2\}$
- J48:
  - confidence factor $\{0.02, 0.04, \ldots, 0.5\}$
  - minimum number instances per leaf $\{1, 2, 3, 4\}$
- JRip:
  - folds for pruning $\{2, 3, 4\}$
  - minimum total weight of instances per rule $\{1, 2, 3\}$

[2] As mentioned above, we used a double round robin for Ripper for both, the full pairwise classifier and for QWeighted. In order to be comparable to the other results, we, in this case, divide the observed number of comparisons by two.

[3] The *CVParameterSelection* method implemented in WEKA was used for parameter tuning.

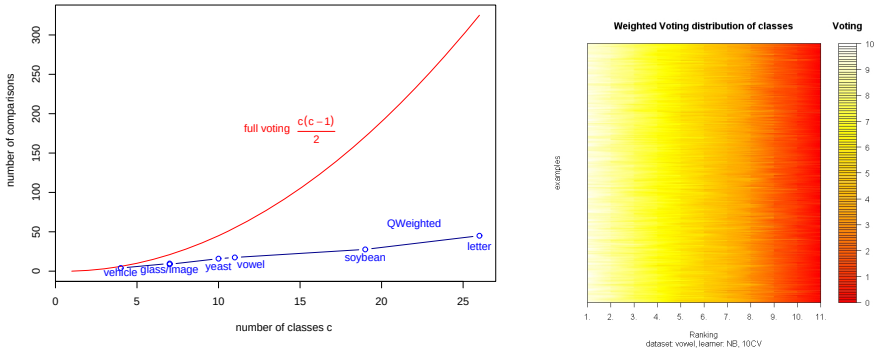
– number of optimization runs $\{1, 2, 3\}$

All results are obtained via a 10-fold cross-validation except for *letter*, where the supplied testset was used. The same experiments were already performed without parameter tuning and can be found in (Park and Fürnkranz 2007a).

Table 1: Comparison of QWeighted and DDAGs with different base learners on seven multiclass datasets. The right-most column shows the number of comparisons needed by a full pairwise classifier ($k(k-1)/2$). Next to the average numbers of comparisons $\hat{n}$ for QWeighted we show their trade-off $\frac{\hat{n}-(k-1)}{\max -(k-1)}$ between best and worst case (in brackets).

dataset	k	learner	Accuracy QWeighted	Accuracy DDAG	∅ Comparisons QWeighted	∅ Comparisons DDAG	∅ Comparisons full
vehicle	4	NB	61.24 ± 4.66	61.12 ± 4.75	4.11 (*0.370*)	3	6
		SMO	80.62 ± 4.30	81.09 ± 4.57	3.70 (*0.233*)		
		J48	71.40 ± 3.06	70.70 ± 2.64	3.94 (*0.314*)		
		JRip	69.63 ± 7.30	69.99 ± 6.36	4.00 (*0.335*)		
glass	7	NB	51.88 ± 7.77	51.43 ± 7.11	9.68 (*0.245*)	6	21
		SMO	62.66 ± 5.89	63.59 ± 6.29	10.03 (*0.269*)		
		J48	70.09 ± 7.43	68.25 ± 5.60	9.81 (*0.254*)		
		JRip	68.27 ± 10.39	66.43 ± 10.11	9.77 (*0.251*)		
image	7	NB	85.76 ± 1.20	85.76 ± 1.20	8.95 (*0.197*)	6	21
		SMO	96.58 ± 1.14	96.62 ± 1.13	8.04 (*0.136*)		
		J48	95.84 ± 1.29	96.36 ± 1.08	8.65 (*0.177*)		
		JRip	96.67 ± 1.32	96.45 ± 1.48	8.77 (*0.185*)		
yeast	10	NB	59.16 ± 3.58	58.96 ± 3.46	15.96 (*0.193*)	9	45
		SMO	58.28 ± 4.05	58.15 ± 3.83	15.48 (*0.180*)		
		J48	58.96 ± 3.58	58.29 ± 3.75	15.61 (*0.184*)		
		JRip	58.89 ± 3.59	57.81 ± 3.31	15.68 (*0.185*)		
vowel	11	NB	71.41 ± 5.57	71.31 ± 5.43	17.19 (*0.160*)	10	55
		SMO	98.59 ± 1.28	98.38 ± 1.66	14.88 (*0.108*)		
		J48	82.73 ± 3.48	78.79 ± 4.07	16.99 (*0.155*)		
		JRip	83.64 ± 5.28	77.88 ± 6.01	17.64 (*0.170*)		
soybean	19	NB	92.96 ± 1.83	92.96 ± 1.83	27.70 (*0.063*)	18	171
		SMO	93.40 ± 2.63	93.11 ± 2.70	28.36 (*0.068*)		
		J48	93.55 ± 2.63	91.36 ± 2.55	28.98 (*0.072*)		
		JRip	93.70 ± 1.97	92.67 ± 2.30	29.79 (*0.077*)		
letter	26	NB	73.93	73.88	43.77 (*0.063*)	25	325
		SMO	91.70	91.13	41.49 (*0.055*)		
		J48	91.10	85.90	47.86 (*0.076*)		
		JRip	90.23	85.83	47.33 (*0.068*)		

Table 1 shows the results. With respect to accuracy, there are only 5 cases in a total of 28 experiments (4 base classifiers × 7 datasets) where DDAGs outperformed QWeighted, whereas QWeighted outperformed DDAGs in 20 cases (3 experiments ended in a tie). Even according to the very conservative sign test, this difference is significant with $p = 0.004$. This and the fact that, to the best of our knowledge, it is not known what loss function is optimized by DDAGs, confirm our intuition that QWeighted is a more principled approach than DDAGs.



(a) (b)

Fig. 1: a) Efficiency of QWeighted in comparison to a full pairwise classifier, b) Distribution of votes for *vowel* (11-class problem, base learner NB). The x-axis describes the ranking positions.

With respect to the number of comparisons, it can be seen that the average number of comparisons needed by QWeighted is much closer to the best case than to the worst case. Next to the absolute numbers, we show the trade-off between best and worst case (in brackets). A value of 0 indicates that the average number of comparisons is $k-1$, a value of 1 indicates that the value is $k(k-1)/2$ (the value in the last column). As we have ordered the datasets by their respective number of classes, we can observe that this value has a clear tendency to decrease with the number of the classes. For example, for the 19-class *soybean* and the 26-class *letter* datasets, only about $6-7\%$ of the possible number of additional pairwise classifiers are used, i.e., the total number of comparisons seems to grow only linearly with the number of classes. This can also be seen from Figure 1a, which plots the datasets with their respective number of classes together with a curve that indicates the performance of the full pairwise classifier.

Finally, we note that the results are qualitatively the same for all base classifiers. QWeighted does not seem to depend on a choice of base classifiers.

5.1.2 Simulation Experiment

For a more systematic investigation of the complexity of the algorithm, we performed a simulation experiment. We assume classes in the form of numbers from $1 \ldots k$, and, without loss of generality, 1 is always the correct class. We further assume pairwise base pseudo-classifiers $C^{\epsilon}_{i,j}$, which, for $i < j$, return *true* with a probability $1-\epsilon$ and *false* with a probability ϵ. For each example, the QWeighted algorithm is applied to compute a prediction based on these pseudo-classifiers. The setting $\epsilon = 0$ (or $\epsilon = 1$) corresponds to a pairwise

Table 2: Average number $\hat{n}$ of pairwise comparisons for various number of classes and different error probabilities ϵ of the pairwise classifiers using QWeighted, and for the full pairwise classifier. Below, we show their trade-off $\frac{\hat{n}-(k-1)}{\max-(k-1)}$ between the best and worst case, and an estimate of the growth ratio $\frac{\log(\hat{n}_2/\hat{n}_1)}{\log(k_2/k_1)}$ of successive values of $\hat{n}$.

	$k=5$	$k=10$	$k=25$	$k=50$	$k=100$
$\epsilon=0.0$	5.43	14.11	42.45	91.04	189.51
	0.238 | —	*0.142* | **1.378**	*0.067* | **1.202**	*0.036* | **1.101**	*0.019* | **1.058**
$\epsilon=0.05$	5.72	16.19	60.01	171.53	530.17
	0.287 | —	*0.200* | **1.501**	*0.130* | **1.430**	*0.104* | **1.515**	*0.089* | **1.628**
$\epsilon=0.1$	6.07	18.34	76.82	251.18	900.29
	0.345 | —	*0.259* | **1.595**	*0.191* | **1.563**	*0.172* | **1.709**	*0.165* | **1.842**
$\epsilon=0.2$	6.45	21.90	113.75	422.58	1,684.21
	0.408 | —	*0.358* | **1.764**	*0.325* | **1.798**	*0.318* | **1.893**	*0.327* | **1.995**
$\epsilon=0.3$	6.90	25.39	151.19	606.74	2,504.54
	0.483 | —	*0.455* | **1.880**	*0.461* | **1.974**	*0.474* | **2.005**	*0.496* | **2.045**
$\epsilon=0.4$	6.93	27.73	182.58	776.98	3,265.56
	0.488 | —	*0.520* | **2.000**	*0.575* | **2.057**	*0.619* | **2.089**	*0.653* | **2.071**
$\epsilon=0.5$	7.12	28.74	198.51	868.25	3,772.45
	0.520 | —	*0.548* | **2.013**	*0.632* | **2.109**	*0.697* | **2.129**	*0.757* | **2.119**
full	10	45	300	1,225	4,950

classifier where all predictions are consistent with a total order of the possible class labels, and $\epsilon = 0.5$ corresponds to the case where the predictions of the base classifiers are entirely random.

Table 2 shows the results for various numbers of classes ($k = 5$, 10, 25, 50, 100) and for various settings of the error parameter ($\epsilon = 0.0$, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5). Each data point is the average outcome of 1000 trials with the corresponding parameter settings. We can see that even for entirely random data, our algorithm can still save about 1/4 of the pairwise comparisons that would be needed for the entire ensemble. For cases with a total order and error-free base classifiers, the number of needed comparisons approaches the number of classes, i.e., the growth appears to be linear.

To shed more light on this, we provide two more measures below each average: the lower left number (in italics) shows the trade-off between best and worst case, as defined above. The result confirms that for a reasonable performance of the base classifiers (up to about $\epsilon = 0.2$), the fraction of additional work reduces with the number of classes. Above that, we start to observe a growth. The reason for this is that with a low number of classes, there is still a good chance that the random base classifiers produce a reasonably ordered class structure, while this chance is decreasing with increasing numbers of classes. On the other hand, the influence of each individual false prediction of a base classifier decreases with an increasing number of classes, so that the true class ordering is still clearly visible and can be better exploited by the QWeighted algorithm.

This can also be seen in Figure 1b, which shows the distribution of the votes produced by the SVM base classifier for the dataset *vowel*. As shown

on the scale to right, different color codes are used for encoding different numbers of received votes. Each line in the plot represents one example, the left shows the highest number of votes, the right the lowest number of votes. If all classes receive the same number of votes, the area should be colored uniformly. However, here we observe a fairly clear change in the color distribution, the bright areas to the left indicating the the top-rank class often receives nine or more votes, and the areas to the right indicating that the lowest ranking class typically receives less than one vote (recall that we use weighted voting).

We tried to directly estimate the exponent of the growth function of the number of comparisons of QWeighted, based on the number of classes k. The resulting exponents, based on two successive measure points, are shown in bold font below the absolute numbers. For example, the exponent of the growth function between $k = 5$ and $k = 10$ is estimated (for $\epsilon = 0$) as $\frac{\log(14.11/5.43)}{\log(10/5)} \approx 1.378$. We can see that in the growth rate starts almost linearly (for a high number of classes and no errors in the base classifiers) and approaches a quadratic growth when the error rate increases.[4]

5.1.3 Datasets with Large Number of Classes

In addition to the small datasets from Table 1, we evaluated the QWeighted algorithm on three more real world datasets with a relative high number of classes:

Uni-label RCV1-v2
: RCV1-v2 (Lewis et al. 2004) is a dataset consisting of over 800,000 categorized news articles from Reuters, Ltd. For the category *topic* multiple labels from a total of 103 hierarchically organized labels are assigned to the instances. We transformed this original multilabel dataset to a multiclass dataset by selecting the assigned label with the greatest depth in the hierarchical tree as the class label. We applied this procedure on the provided trainset and testset no. 0 by Lewis et al. resulting to a multiclass dataset with 100 classes, 23,149 train- and 199,328 test-instances, with at least one positive example for each of the 100 classes. We selected 2,000 features according to a χ^2-based feature selection (Yang and Pedersen 1997). We will refer to this created dataset as *urcv1-v2*.

ASTRAL 2 & 3
: These datasets describe protein sequences retrieved from the SCOP 1.71 protein database (Murzin et al. 1995). We used ASTRAL (Brenner et al. 2000) to filter these sequences so that no two sequences share greater than 95% identity. The class labels are organized in a 3-level hierarchy, consisting of protein folds, superfamilies and families (in descending order). *astral3* consists of 1,588 classes and contains the original hierarchy. To fill the gap

[4] At first sight, it may be surprising that some of the numbers are greater than 2. This is a result of the fact that $k(k-1)/2 = k^2/2 - k/2$ is quadratic *in the limit*, but for low values of c, the subtraction of the linear term $k/2$ has a more significant effect. Thus, e.g., the estimated growth of the full pairwise classifier from $k = 5$ to $k = 10$ is $\frac{\log(45/10)}{\log(10/5)} \approx 2.17$.

Table 3: Results of QWeighted for datasets with a relative high number of classes. Below, we show their trade-off $\frac{\hat{n}-(k-1)}{\max-(k-1)}$ between the best and worst case (base learner J48).

dataset	k	QW	full	Accuracy
urcv1-v2	100	312.62 (*0.0440*)	4,950	62.1
astral2	971	9,490.81 (*0.0018*)	470,935	24.8
astral3	1,588	28,476.20 (*0.0213*)	1,260,078	20.8

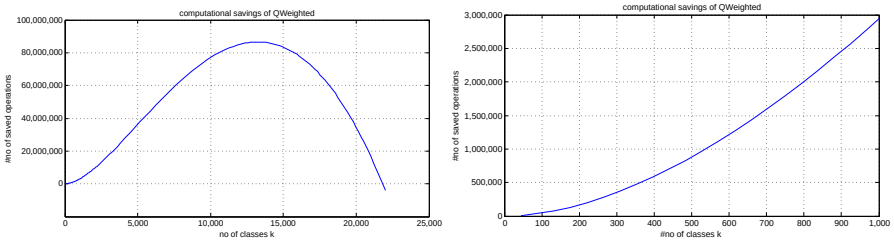
between datasets *urcv1-v2* and *astral3* in terms of number of classes, we constructed a second dataset *astral2* by limiting the hierarchical depth to 2. So, two instances which previously shared the same superfamily x are now assigned to superfamily x as new class label. By decreasing the depth, the number of classes were reduced to 971. Both datasets have 13,006 instances and 21 numeric attributes (20 amino acids plus selenocysteine).

Table 3 shows the results of these experiments. For *astral2* and *astral3*, 66 percent of all instances were used for training and the rest for testing. Once again, trade-off values were estimated for the average number of pairwise comparisons. As these values show, QWeighted uses only a fairly small amount compared to a full voting aggregation and is much closer to the best case than to the worst case ($k(k-1)/2$ comparisons). One can see an increasing growth of the trade-off values between *astral2* and *astral3*. However, this effect can be explained with the general poor classification accuracy of protein sequences. According to the simulation results, there exist a correlation between performance of QWeighted and performance of the underlying base classifiers. The decreased accuracy on *astral3* compared to *astral2* (right-most column) indicates weaker base classifiers, which leads to a increasing number of needed pairwise comparisons.

In summary, our results indicate that the QWeighted algorithm always increases the efficiency of the pairwise classifier: for high error rates in the base classifiers, we can only expect improvements by a constant factor, whereas for the practical case of low error rates we can also expect a significant reduction in the asymptotic algorithmic complexity.

5.1.4 Overall Complexity

Besides the complexities for the comparisons, the overall complexity of the algorithm including the inherent overhead of the algorithm, e.g. estimating the next classifier $C_{a,b}$ and so on, can be stated as $g(k) \cdot (k+p)$ operations, where $g(k)$ denotes the number of comparisons in dependance of k and p describes the cost of one comparison (prediction) in terms of basic operations. The summand k is here understood as the operations for the overhead involving additions, value associations and $\arg min$ operations, which can be implemented in an



(a) full range of advantageous k (b) focus on $k = 1..1000$

Fig. 2: Approximate computational savings for $p = 20$ and assumed average complexity of $k \cdot log\, k$ of QWeighted

incremental manner with $O(k)$ by maintaining a sorted vector of classes (for the limits l_i) updated after each comparison and using a 2-dimensional boolean array which maintains the status of the classifiers (evaluated/not evaluated).

Given the empirical evidences, lets assume $g(k) = k \cdot \log k$. This results in a total of $k^2 \cdot log\, k + p \cdot k \cdot \log k$ operations. Obviously, the overall asymptotic complexity of QWeighted is worse than the complexity for standard voting of $O(k^2)$, such that for very large k the complexity of the standard voting is favoured. But, in practice for reasonable assumptions, e.g $p \gg 1$ (keep also in mind that p can increase for some learning schemes, for e.g. in dependance of the number of training instances), there exist an upper limit $\hat{k} \in \mathbb{N}$ such that $k^2 \cdot \log k + p \cdot k \cdot \log k$ is significantly smaller than $p \cdot \frac{k \cdot (k-1)}{2}$ for $k < \hat{k}$.

To give a clearer picture, consider figure 2, where the difference of both quantities, i.e. $p \cdot \frac{k \cdot (k-1)}{2} - (k \cdot \log k \cdot (k + p))$ is plotted for $p = 20$ (20 atomar operations needed for one prediction). The graph shows the saved number of operations by using QWeighted in contrast to standard voting procedure in dependance of k. The left figure shows the savings up to the critical class count $\hat{k} \approx 22,000$ and the right figure shows the same plot for the selected range $k = [1, 1000]$, which corresponds approximately to the typical range in real-world datasets.

5.2 ECOC Classification - Evaluation of QuickECOC

In this section, we evaluate the performance of QuickECOC for a variety of different codes. In addition, we were interested to see if it works for all decoding methods and whether we can gain insights on which factors determine the performance of QuickECOC. In particular, we investigated the effects of the sparsity and length of the codes.

5.2.1 Experimental Setup

In contrast to the results presented in the previous section, we only used the decision tree learner J48 with default parameters as a base learner, to

restrict the already large number of the experiments. Besides, the results in Section 5.1 gave no indication that the performance in terms of the number of needed comparisons depends on the choice of the base classifier. Thus, we are quite confident that the presented results are representative for other base classifiers.

Our setup consisted of

- **5 encoding strategies**: BCH Codes and two versions each of exhaustive and random codes.
- **7 decoding methods**: Hamming, Euclidean, attenuated Euclidean, linear loss-based, exponential loss-based, Laplacian Strategy and Beta Density Probabilistic Pessimistic
- **7 multiclass datasets** selected from the UCI Machine Learning Repository.

For the encoding strategies, we also tried several different parameters. Regarding the exhaustive codes, we evaluated all (k, l) codes ranging from $l = 2$ to $l = k$ per dataset and analogously for the cumulative version. For the generation of the first type of random codes the zero symbol probability was parametrized by $r = 0.2, 0.4, 0.6, 0.8$ and the dimension of the coding matrix was fixed to 50 % of the maximum possible dimension with respect to the number of classes. The second type of random codes was generated by randomly selecting 20 %, 40 %, 60 % and 80 % from the set of all valid classifiers respectively columns (all columns of an (k, k) cumulative ternary coding matrix) without repetition. Regarding BCH Codes, we generated $7, 15, 31, 63, 127$ and 255-bit BCH codes and randomly selected n rows matching the class count of the currently evaluated dataset. For the datasets *machine* and *ecoli* where the number of classes is greater than 7, we excluded the evaluation with 7-bit BCH codes.

For the evaluation of QuickECOC, the seven datasets were selected to have a rather low number of different classes. The main reason for this limitation was that for some considered code types the number of classifiers grows exponentially. Especially for the datasets with the maximum number of eight classes (*machine* and *ecoli*), the cumulative ternary exhaustive codes generates up to 3025 classifiers. In addition, we evaluated all possible combinations of decoding methods, code types with various parameters, which we can not present here completely (in total 1246 experiments) because of lack of space. Nevertheless, we performed experiments with a few of more efficient codes on datasets with larger number of classes as well. These will be shown in Section 5.2.6.

Because of the high number of experiments, we cannot present all results in detail, but will try to focus on the most interesting aspects. In addition to assess the general performance of QuickECOC, we will analyze the influence of the sparsity of the code matrix, of the code length, and of different decoding strategies.

5.2.2 Reduction in Number of Evaluations

Table 4 shows the reduction in the number of classifier evaluations with QuickECOC on all evaluated datasets with Hamming decoding and ternary exhaustive codes. In every column, the average number of classifier evaluations is stated with its corresponding ratio to the number of generated classifiers in italics (the lower the better). The datasets are ordered from left to right by ascending class-count. As the level parameter l is bounded by the class-count k, some of the cells are empty.

One can clearly see that QuickECOC is able to reduce the number of classifier evaluations for all datasets. The percentage of needed evaluations ranges from about 81 % (*vehicle*, $l = 4$) to only 35 % (*machine*, $l = 3$). At first glance, these improvements may not seem striking, because a saving of a little less than 40% for the small datasets does not appear to be such a large gain. However, one must put these results in perspective. For example, for the *vehicle* dataset with a $(4, 3)$-exhaustive code, QuickECOC evaluated 65.9% of all classifiers. A $(4, 3)$-exhaustive code has 12 classifiers, and each individual class is involved in 75% of these classifiers (cf. the example in Section 2.4). Thus, on average, QuickECOC did not even evaluate all the classifiers that involve the winning class before this class was predicted.

Furthermore, one can observe a general trend of higher reduction by increasing class-count. This is particularly obvious if we compare the reduction on the exhaustive codes (the last line of each column, where $l = k$), but can also be observed for individual code sizes (e.g., for $l = 3$). Although we have not performed a full evaluation on datasets with a larger amount of classes because of the exponential growth in the number of classifiers, a few informal and quick tests supported the trend: the higher the class-count, the higher the reduction.

Another interesting observation is that except for dataset *vehicle* and *auto* the exhaustive ternary codes for level $l = 3$ consistently lead to the best QuickECOC performance over all datasets. A possible explanation based on a "combinatorial trade-off" is briefly described in the appendix.

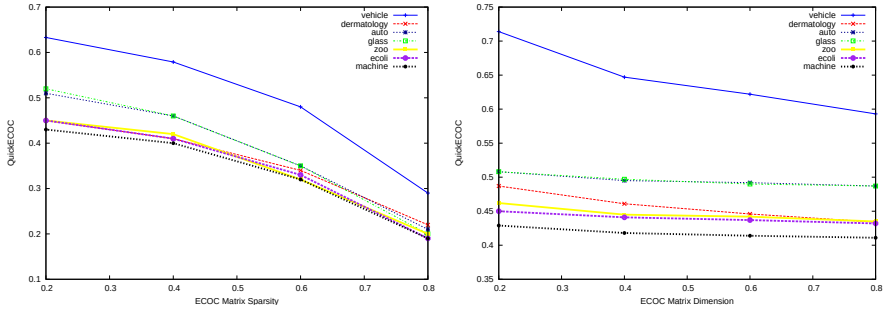
The results for BCH codes are shown in Table 5. Again, we can observe an improved performance in all cases. This result is particularly interesting because for BCH-codes, all coding matrices are dense, i.e., they do not have

Table 4: QuickECOC performance using Hamming Decoding and Exhaustive Ternary Codes. The maximal relative standard deviation for all values is 8.65% with mean 3.88%.

	vehicle	derm.	auto	glass	zoo	ecoli	machine
$l = 2$	3.82 *63.7*	7.12 *47.5*	7.95 *37.9*	9.99 *47.6*	9.48 *45.1*	11.75 *42.0*	11.60 *41.4*
$l = 3$	7.91 *65.9*	26.05 *43.4*	42.86 *40.8*	43.47 *41.4*	41.64 *39.7*	58.85 *35.0*	57.90 *34.5*
$l = 4$	5.65 *80.8*	46.30 *44.1*	115.22 *47.0*	116.45 *47.5*	107.03 *43.7*	199.31 *40.7*	194.81 *39.8*
$l = 5$		43.11 *47.9*	163.67 *52.0*	163.98 *52.1*	148.50 *47.1*	369.06 *43.9*	355.23 *42.3*
$l = 6$		16.54 *53.4*	114.87 *52.9*	116.77 *53.8*	102.41 *47.2*	394.25 *45.4*	369.19 *42.5*
$l = 7$			34.24 *54.3*	37.84 *60.1*	31.52 *50.0*	234.80 *46.6*	218.09 *43.3*
$l = 8$						62.17 *49.0*	57.27 *45.1*

Table 5: QuickECOC performance on BCH Codes. The maximal relative standard deviation for all values is 10.2% with mean 5.64%.

	vehicle	derm.	auto	glass	zoo	ecoli	machine
7	0.764	0.774	0.851	0.880	0.834	-	-
15	0.646	0.656	0.699	0.717	0.659	0.670	0.648
31	0.571	0.564	0.607	0.662	0.581	0.602	0.558
63	0.519	0.506	0.567	0.616	0.517	0.540	0.509
127	0.489	0.447	0.522	0.565	0.477	0.493	0.459
255	0.410	0.380	0.450	0.467	0.397	0.417	0.388



(a) in dependence of Sparsity (b) in dependence of Code Length

Fig. 3: QuickECOC performance on random codes

any zero entries. Even in this case, we see that there was no situation, where all classifiers were needed for multiclass classification. And again, we observe that for higher dimensions (increasing the length of the BCH bit code) higher reductions can be observed.

For random codes, we obtained qualitatively the same results. We do not show them here, but but some of them will appear in the following sections.

5.2.3 Sparsity of Coding Matrices

We define the sparsity of the ECOC matrix as the fraction of zero values it contains. Random codes provide a direct control over the matrix sparsity (as described in Section 2.4), and are thus suitable for analyzing the influence of the sparsity degree of the ECOC matrix for QuickECOC. Note, however, that the observed influences regarding sparsity and dimension of the matrix on the QuickECOC performance can also be seen in the evaluations of the other code types, but not as clearly as with the random codes presented in this section.

Figure 3a shows QuickECOC applied to random codes with varying matrix sparsity. A clear trend can be observed that the higher the sparsity of the coding matrix the better the reduction for all datasets. Keep in mind that the baseline performance (evaluating all binary classifiers) is a parallel to the x-axis with the y-value of 1.0. Note that the absolute reduction tends to be minimal over all considered datasets at datasets with higher class-counts i.e

machine at 80 % sparsity, and the lowest reduction can be seen for the dataset *vehicle* with the smallest number of classes $n = 4$ at 20 % sparsity.

The main effect of an increase of sparsity on the coding matrices is that for each class the number of incident classifiers decreases. For sparsity 0, all classes are involved in all classifiers, for sparsity 0.5, each class is (on average) involved in only half of the classifiers. This will clearly affect the performance of the QuickECOC algorithm. In particular, the second stopping criterion essentially specifies that the true class is found if all incident classifiers for the favorite class i_0 have been evaluated. Clearly, the algorithm will terminate faster for higher sparsity levels (ignoring, for the moment, the possibility that the first stopping criterion may lead to even faster termination).

5.2.4 Code Length

The second type of random codes, which were generated by randomly selecting a fixed number from the set of all possible binary classifiers can be seen in Figure 3b. All coding matrices for a k-class dataset have nearly the same sparsity, which relates to the average sparsity of (k, k) cumulative exhaustive codes and differ only in the length of the coding matrix (in percent of the total number of possible binary classifiers). This allows us to observe the effect of different numbers of classifiers on the QuickECOC performance. Here, we can also see an consistent relationship, that higher dimensions lead to better performance, but the differences are not as remarkable as for sparse matrices.

For a possible explanation, assume a coding matrix with fixed sparsity and we vary the dimension. For a higher dimension the ratio of number of classifiers per class increases. Thus, on average, the number of incident classifiers for each class also increases. If we now assume that this increase is uniform for all classes, this has the effect that the distance vector $\mathbf{l}$ is multiplied by a positive factor $x > 1$, i.e., $\mathbf{l}^+ = \mathbf{l} * x$. This alone would not change the QuickECOC performance, but if we consider that classifiers are not always perfect, we can expect that for higher number of classifiers, the variance of the overall prediction will be smaller. This smaller variance will lead to a more reliable voting vectors, which can, in turn, lead to earlier stopping. It also seems reasonable that this effect will not have such a strong impact as the sparsity of the coding matrix, which we discussed in the previous section.

5.2.5 Different Decoding Strategies

As previously stated, because of the large number of experiments, we can not give a complete account of all results. We evaluated all combinations of experiments, that includes also all mentioned decoding methods. All the previously shown results were based on Hamming decoding, since it is still one of the commonly used decoding strategies even for ternary ECOC matrices. However, we emphasize, that all observations on this small subset of results can also be found in the experiments on the other decoding strategies. As an exemplary data point, Table 6 shows an overview of the QuickECOC performance for all

Table 6: QuickECOC performance on the 8-class *ecoli* datasets with all decoding methods and Cumulative Exhaustive Ternary Codes. The first two columns show the number of non-zero code values for each class and the number of resulting classifiers. The maximal relative standard deviation for all values is 3.76% with mean 2.69%.

	$\lvert\mathcal{C}\rvert$	Hamming	Euclidean	A. Euclidean	LBL	LBE	Laplace	BDDP
$l = 2$	28	*0.420*	*0.420*	*0.420*	*0.399*	*0.398*	*0.406*	*0.426*
$l = 3$	196	*0.331*	*0.331*	*0.331*	*0.335*	*0.350*	*0.332*	*0.333*
$l = 4$	686	*0.377*	*0.377*	*0.377*	*0.383*	*0.402*	*0.374*	*0.375*
$l = 5$	1526	*0.400*	*0.400*	*0.400*	*0.414*	*0.439*	*0.399*	*0.401*
$l = 6$	2394	*0.421*	*0.421*	*0.421*	*0.437*	*0.466*	*0.419*	*0.418*
$l = 7$	2898	*0.427*	*0.427*	*0.427*	*0.444*	*0.475*	*0.426*	*0.425*
$l = 8$	3025	*0.428*	*0.428*	*0.428*	*0.446*	*0.477*	*0.427*	*0.426*

decoding strategies for the dataset *ecoli* using cumulative exhaustive ternary codes. It can be seen that the performance is quite comparable on all datasets. Even the optimal reduction for $l = 3$ can be found in the results of all decoding strategies.

5.2.6 Datasets with Larger Numbers of Classes

The previous sections evaluated and analyzed QuickECOC on a broad spectrum of various code types and decoding methods. This was only feasible for datasets with smaller number of classes. In this section, we will evaluate QuickECOC on datasets with larger number of classes. As the code length of most coding strategies is exponential in the number of classes k, we selected a few codes which generate a comparably low number of classifiers:

1. $(k, 3)$- and $(k, 4)$-exhaustive ternary codes
2. $(k, 4)$-cumulative exhaustive ternary codes
3. random codes of type 1 with fixed sparsity of 66%
4. random codes of type 2

For both random code types the code length was set to the equivalent of the number of $(k, 4)$-cumulative exhaustive codes, i.e. $n = \sum_{i=2}^{4} \binom{k}{i} \cdot (2^{i-1} - 1)$.

Table 7 shows the results for Hamming decoding. Each cell shows the average number of classifier evaluations of QuickECOC and, in italics, its corresponding ratio to the full number of classifiers. First, we can observe a considerably higher improvement than with the results on the datasets with lower number of classes. The best reduction can be found for the $(19, 3)$-exhaustive code for the soybean dataset, where QuickECOC only performs about 15% of the evaluation in order to determine the winning class.

Moreover, one can clearly see an increasing reduction for increasing number of classes k, especially for the first three columns respectively code types. For these code types, the sparsity increases with k, since the number of non-zero values per column stays fixed whereas the number of rows (the number of classes k) of the corresponding ECOC matrix is increased. This observation

Table 7: QuickECOC performance on datasets with high number of classes. The maximal relative standard deviation for all values is 2.37% with mean 1.65%.

	k	exh. $l=3$		exh. $l=4$		cum. exh. $l=4$		random1		random2	
yeast	10	105.92	*0.294*	524.13	*0.357*	631.02	*0.337*	844.02	*0.296*	1351.94	*0.474*
vowel	11	139.42	*0.282*	797.32	*0.345*	937.43	*0.328*	880.89	*0.306*	1388.74	*0.482*
soybean	19	443.89	*0.153*	5351.90	*0.197*	5804.73	*0.192*	8481.74	*0.282*	12964.83	*0.431*

Table 8: Average number $\hat{n}$ of comparisons for various number of classes and different error probabilities ϵ of ECOC classifiers using QuickECOC with exhaustive ternary codes of level 3, and for the full ensemble of classifiers. Below, we show the ratio to the full number of comparisons and an estimate of the growth ratio $\frac{\log(\hat{n}_2/\hat{n}_1)}{\log(k_2/k_1)}$ of successive values of $\hat{n}$.

	$k=5$	$k=10$	$k=25$	$k=50$	$k=100$
$\epsilon=0.0$	13.00	93.00	783.00	3,433.00	14,358.00
	0.433 | —	*0.258* | **2.839**	*0.113* | **2.325**	*0.058* | **2.132**	*0.030* | **2.064**
$\epsilon=0.05$	14.31	101.18	951.28	7,432.19	61,244.02
	0.477 | —	*0.281* | **2.822**	*0.138* | **2.446**	*0.126* | **2.966**	*0.126* | **3.043**
$\epsilon=0.1$	15.95	114.85	1,734.24	14,588.86	119,232.96
	0.532 | —	*0.319* | **2.848**	*0.251* | **2.963**	*0.248* | **3.072**	*0.246* | **3.031**
$\epsilon=0.2$	19.78	177.30	3,296.39	27,589.68	223,472.28
	0.659 | —	*0.492* | **3.164**	*0.478* | **3.190**	*0.469* | **3.065**	*0.461* | **3.018**
$\epsilon=0.3$	24.62	249.87	4,733.95	39,396.05	319,798.78
	0.821 | —	*0.694* | **3.343**	*0.686* | **3.210**	*0.670* | **3.057**	*0.659* | **3.021**
$\epsilon=0.4$	27.71	309.34	5,993.48	50,121.35	407,887.14
	0.924 | —	*0.859* | **3.481**	*0.869* | **3.235**	*0.852* | **3.064**	*0.841* | **3.025**
$\epsilon=0.5$	29.08	336.05	6,635.10	57,502.71	478,871.55
	0.969 | —	*0.933* | **3.530**	*0.962* | **3.255**	*0.978* | **3.115**	*0.987* | **3.058**
full	30	360	6900	58,800	485,100

confirms our results of Section 5.2.3, which showed that a high sparsity is beneficial for the performance of QuickECOC.

We can also confirm our results regarding the influence of the code length (Section 5.2.4). Both types of random codes, shown in the last two columns, have a fixed sparsity level. In both cases, although a small improvement can be observed (just as in Figure 3b), the improvement is small in comparison to the improvement resulting from increased sparseness. For example, on datasets *yeast* ($k=10$) and *soybean* ($k=19$), QuickECOC applies for the second type of random codes in average 47% and 43% classifier evaluations, which is a relative reduction/ratio of about 10 %, whereas for $(k,3)$-exhaustive codes a relative reduction of $0.153/0.294 \approx 48\%$ is gained.

5.2.7 Simulation Experiment

We also conducted a simulation experiment for QuickECOC, similar in spirit to the pairwise case (Section 5.1.2). Again, we consider classes $1 \ldots k$ and always assume that class 1 is the correct class and further assume that pseudo base classifiers f_i^ϵ return the desired prediction with probability ϵ, i.e., with probability ϵ, they predict the same sign in the ECOC matrix as the smallest

incident class of f_i^ϵ. We simulate the efficiency of QuickECOC using exhaustive ternary codes of level 3 for various class counts k and error probabilities ϵ. Table 8 shows the results.

In contrast to pairwise classification, we can observe that the base-classifier accuracy now has a stronger influence on the efficiency. In the case of random classifiers $\epsilon = 0.5$, we can observe almost no reduction, as was the case for pairwise classification (though for greater k, it might converge to the worst-case there too). But, for $\epsilon < 0.5$, focusing on the ratio values, one can see an increasing reduction trend for increasing k, which slowly looses its steepness. The growth values suggest that only in the near optimal case $\epsilon < 0.05$, a super-linear reduction with the number of classes can be expected.

However, in absolute terms, the reduction can be significant for predictors with high computational complexity. Furthermore, this analysis was based on one of many applicable code types which can be used with ECOC. Other code types, e.g. codes with beneficial error-correcting ability or codes which may not grow exponentially in k like the considered exhaustive ternary code, may perform differently.

5.2.8 Overall Complexity

Since the QuickECOC algorithm is more general than QWeighted, its overhead is significantly greater. The stopping criteria depicted in the pseudocode 2 can be implemented in an incremental manner such that the complexity is $O(k)$, by maintaining for each class a variable storing the potential worst-case hamming distance and updating only the relevant values after each comparison (prediction). All in all, the overhead is linear in k except for the *Next Classifier Selection* Scheme (cf. 4.2), which complexity is $O(nk)$. Therefore, the computational savings diminishes in this case far more quicker as in the case of QWeighted for pairwise classification.

A reduction of operations is still possible for problems up to about $k = 10$ and depending of the actual prediction complexity and code type. But for greater class counts the overhead starts to dominate the overall complexity such that the efficiency is worse than standard voting. In these cases, a reasonable choice is to work with alternative selection schemes, which check only a fixed number of classifiers incident of the current best class. Or selecting an unevaluated classifier randomly from the set of incident classifiers to the current best class is an alternative, with a slight decrease in comparisons efficiency but important increase in overall efficiency of the algorithm. Note, this passage holds for code types which grow exponentially in k, e.g. exhaustive ternary codes. In cases of more practicable code types, such as BCH codes, the overhead of the algorithm remains still in the tolerable range.

6 Conclusions

In this paper, we have proposed an algorithm that allows to speed up the prediction phase for binary decomposition methods such as pairwise classification and, more generally, ternary ECOC classifiers. Both variants only need to evaluate a fraction of the classifiers, but are guaranteed to make the same prediction as the original version using all classifiers. In general, this gain increases with the complexity of the problem, i.e., with the number of classes, with the sparsity of the coding matrix, and (somewhat less) with the length of the code words of the ECOC classifiers. But even for very hard problems, where the performance of the binary classifiers reduces to random guessing, practical gains can be expected.

For the general case of ternary ECOC matrices, which subsume nearly all possible binary decomposition schemes, we have demonstrated this gain for a wide variety of coding and decoding strategies. Regardless of the used code, QuickECOC improves the overall prediction efficiency, but, depending on the coding strategy, the amount of improvement is not always as striking as for the pairwise case, where we could observe a reduction from k^2 to $k \cdot \log k$. One must keep in mind that in ECOC codings, each class has a much larger number of incident classifiers, and thus a higher number of evaluations must be expected to determine the winning class. Moreover, for code types whose code length grows exponentially with the number of classes, the overhead of QuickECOC (in its presented form) can dominate the gained reduction of classifier evaluations, thus resulting in a worse performance than standard voting. However, we briefly described alternative more overhead-efficient approaches which allow to adjust QuickECOC to the problem at hand such that a beneficial reduction can still be expected. In general, we recommend the practitioner to carefully pre-assess the parameters of the present problem, such as the number of classes k, code type and prediction complexity in terms of base operations, integrate them into the overall complexity model and to adjust the selection scheme to maximize the efficiency performance.

One could argue that typically the training phase is more expensive than the classification phase, and that the gains obtained by QuickECOC are negligible in comparison to what can be gained by more efficient coding techniques. While this is true, we note that QuickECOC can obtain gains independent of the used coding technique, and can thus be combined with any coding technique. In particular in time-critical applications, where classifiers are trained once in batch and then need to classify on-line on a stream of in-coming examples, the obtained savings can be decisive.

Another point to consider is that in applications where the classification time is crucial, a parallel approach could be applied effectively because each classifier defined by a column of the ECOC matrix can be evaluated independently. QuickECOC looses this advantage because the choice of the next classifier to evaluate depends on the results of the previous evaluations. However, QuickECOC can still be parallelized on the instance level instead of the classifier level. Given n processors or n threads we want to utilize, we

select n incoming test instances and apply QuickECOC for each of them. Basically by paralleling the decoding process on the instance level, we avoid the problem that QuickECOC can not be directly parallelized on the classifier level for one instance. This method is still very efficient, since every CPU is constantly utilized. Considering that in total, the number of evaluations is decreased by using QuickECOC, a higher speed up can be expected as with a straight-forward parallelization of ECOC.

Recently, Hsu et al. (2009) presented an efficient ensemble approach for multilabel classification. They exploit the general label sparseness in target vectors of real-world multilabel problems to reduce the number of the labels to $O(\log k)$ using techniques from compressed sensing. Instead of learning k one-against-all regression predictors for generating a multiclass predictor, they only need to learn (and therefore to predict) about $\log k$ regression predictors. Since multiclass classification is a special case of multilabel classification (by limiting the label size to 1) and noting that it poses a maximal label-sparse multilabel problem, their results naturally also apply to multiclass classification. However, it is not entirely clear how their ensemble approach consisting of a one-against-all decomposition with regression base-learners performs in comparison to the commonly studied classification-based ensemble approaches with respect to predictive performance. Moreover, input or output data transformations yield often to a reduction of the comprehensibility of the learned models, which is disadvantageous for the acceptance in real-world applications, where a white-box property of the system is favoured. A direct comparison of this work to conventional classification-based decompositions is certainly interesting and overdue, but beyond the scope of this work, where our goal was to improve the classification time of well established ensemble techniques.

There might still be some potential for improving our results with better heuristics for the selection of the next classifier, we have not yet thoroughly explored this parameter. For example, one could try to adapt ideas from *active learning* for this process. Nevertheless, however, we do not expect a high gain. Furthermore we consider an in-depth analysis of existing fast decoding methods in coding theory, and the investigation of the transferability to the multiclass classification setting as promising directions for future work. Finally, we are also working on expanding these results to other learning problems where binary decomposition techniques are applied. To that end, we have obtained very positive results for the calibrated ranking approach to multilabel classification (Mencía et al. 2010), and have promising preliminary results for ranking problems (Park and Fürnkranz 2007b).

Acknowledgements This research was supported by the *German Science Foundation (DFG).*

References

Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:

113–141, 2000.

A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

R. C. Bose and D. K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79, March 1960.

Steven E. Brenner, Patrice Koehl, and Michael Levitt. The astral compendium for protein structure and sequence analysis. *Nucleic Acids Research*, 28(1):254–256, 2000.

Jaime S. Cardoso and Joaquim F. Pinto da Costa. Learning to classify ordinal data: The data replication method. *Journal of Machine Learning Research*, 8:1393–1429, 2007.

Koby Crammer and Yoram Singer. On the learnability and design of output codes for multiclass problems. *Machine Learning*, 47(2-3):201–233, 2002.

Florin Cutzu. How to do multi-way classification with two-way classifiers. In Okyay Kaynak, Ethem Alpaydin, Erkki Oja, and Lei Xu, editors, *Artificial Neural Networks and Neural Information Processing - ICANN/ICONIP 2003, Joint International Conference ICANN/ICONIP 2003*, volume 2714 of *Lecture Notes in Computer Science*, pages 375–384, Istanbul, Turkey, 2003a. Springer.

Florin Cutzu. Polychotomous classification with pairwise classifiers: A new voting principle. In Terry Windeatt and Fabio Roli, editors, *Multiple Classifier Systems, 4th International Workshop (MCS 2003)*, volume 2709 of *Lecture Notes in Computer Science*, pages 115–124, Guilford, UK, 2003b. Springer.

Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

Sergio Escalera, Oriol Pujol, and Petia Radeva. Decoding of ternary error correcting output codes. In José Francisco Martínez Trinidad, Jesús Ariel Carrasco-Ochoa, and Josef Kittler, editors, *Proceedings of the 11th Iberoamerican Congress in Pattern Recognition (CIARP-06)*, pages 753–763. Springer, 2006.

Johannes Fürnkranz. Round robin ensembles. *Intelligent Data Analysis*, 7(5):385–403, 2003.

Johannes Fürnkranz. Round robin classification. *Journal of Machine Learning Research*, 2:721–747, 2002.

Robert G. Gallager. *Information Theory and Reliable Communication*. Wiley, January 1968. ISBN 0471290483.

Trevor Hastie and Robert Tibshirani. Classification by pairwise coupling. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems 10 (NIPS 1997)*. The MIT Press, 1997.

Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.

Daniel Hsu, Sham Kakade, John Langford, and Tong Zhang. Multi-label prediction via compressed sensing. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 772–780. 2009.

Eyke Hüllermeier and Johannes Fürnkranz. Ranking by pairwise comparison: A note on risk minimization. In *Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE-04)*, Budapest, Hungary, 2004a.

Eyke Hüllermeier and Johannes Fürnkranz. Comparison of ranking procedures in pairwise preference learning. In *Proceedings of the 10th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU-04)*, Perugia, Italy, 2004b.

Eyke Hüllermeier, Johannes Fürnkranz, Weiwei Cheng, and Klaus Brinker. Label ranking by learning pairwise preferences. *Artif. Intell.*, 172(16-17):1897–1916, 2008.

Eun Bae Kong and Thomas G. Dietterich. Error-correcting output coding corrects bias and variance. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 313–321. Morgan Kaufmann, 1995.

David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

Ana Carolina Lorena, André Carlos Ponce Leon Ferreira de Carvalho, and João Gama. A review on the combination of binary classifiers in multiclass problems. *Artificial Intelligence Review*, 30(1-4):19–37, 2008.

F. Jessie MacWilliams and Neil J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland Mathematical Library. North Holland, January 1983.

Iain Melvin, Eugene Ie, Jason Weston, William Stafford Noble, and Christina Leslie. Multi-class protein classification using adaptive codes. *Journal of Machine Learning Research*, 8:1557–1581, 2007.

Eneldo Loza Mencía, Sang-Hyeun Park, and Johannes Fürnkranz. Efficient voting prediction for pairwise multilabel classification. *Neurocomputing*, 73(7-9):1164–1176, 2010.

Alexey G. Murzin, Steven E. Brenner, Tim Hubbard, and Cyrus Chothia. Scop: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247:536–540, 1995.

Sang-Hyeun Park and Johannes Fürnkranz. Efficient decoding of ternary error-correcting output codes for multiclass classification. In W. L. Buntine, M. Grobelnik, D. Mladenič, and J. Shawe-Taylor, editors, *Proceedings of 20th European Conference on Machine Learning (ECML-09)*, pages 189–204, Bled, Slovenia, 2009. Springer-Verlag.

Sang-Hyeun Park and Johannes Fürnkranz. Efficient pairwise classification. In J. N. Kok, J. Koronacki, R. Lopez de Mantaras, S. Matwin, D. Mladenič, and A. Skowron, editors, *Proceedings of 18th European Conference on Machine Learning (ECML-07)*, pages 658–665, Warsaw, Poland, 2007a. Springer-Verlag.

Sang-Hyeun Park and Johannes Fürnkranz. Efficient pairwise classification and ranking. Technical Report TUD-KE-2007-03, Knowledge Engineering Group, TU Darmstadt, 2007b.

Edgar Pimenta, João Gama, and André Carlos Ponce de Leon Ferreira de Carvalho. The dimension of ECOCs for multiclass classification problems. *International Journal on Artificial Intelligence Tools*, 17(3):433–447, 2008.

John C. Platt, Nello Cristianini, and John Shawe-Taylor. Large margin DAGs for multi-class classification. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *Advances in Neural Information Processing Systems 12 (NIPS 1999)*, pages 547–553, Denver, Colorado, USA, 1999. The MIT Press.

Oriol Pujol, Petia Radeva, and Jordi Vitrià. Discriminant ECOC: A heuristic method for application dependent design of error correcting output codes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(6), June 2006.

Terry Windeatt and Reza Ghaderi. Coding and decoding strategies for multi-class learning problems. *Information Fusion*, 4(1):11–21, 2003.

Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques, 2nd Edition*. Morgan Kaufmann, San Francisco, 2005.

Ting-Fan Wu, Chih-Jen Lin, and Ruby C. Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5:975–1005, 2004.

Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In Douglas H. Fisher, editor, *Proceedings of the Fourteenth International Conference on Machine Learning (ICML 1997)*, pages 412–420, Nashville, Tennessee, USA, 1997. Morgan Kaufmann.

Appendix 1 Analysis of (k,3)-Exhaustive Ternary Codes

Since we are interested in conditions under which QuickECOC performs well, this special case of exhaustive ternary codes was further investigated. In this regard, we examined the reduction effects of the two stopping criteria separately with varying levels on several datasets.

It is easy to see that the performance regarding the second stopping criterion is strongly dependent on the incidency of the ECOC matrix. Considering that the selection process of QuickECOC always selects incident classifiers of the current best class c_0, the number of classifier evaluations can be estimated as $\#I_0 + \#R_0$ whereas I_i is the set of incident and $R_i \subseteq N \setminus I_i$ is

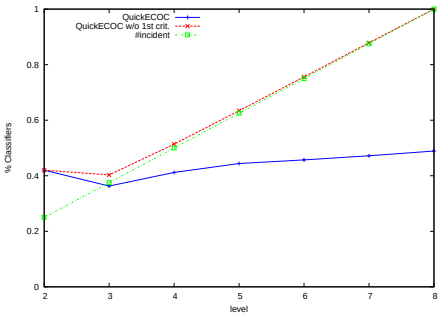


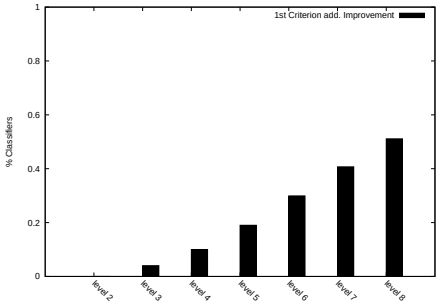
Fig. 4: Dependency of stopping criteria for different levels respectively incidencies (*ecoli*)

a subset of non-incident classifiers of c_i.[5] These remaining classifiers R can be caused by initial random pairings until a classifier involving the true class c_0 is evaluated. Even then, depending on the accuracy of the classifiers, still some non-incident classifiers can be falsely selected and evaluated in the next steps. In this context, the second stopping criterion can be seen as a reduction method which tries to minimize the non-incident classifier evaluations. Considering that for increasing level l the incidency of exhaustive ternary codes is constantly increasing, the reduction performance of the second criterion alone is decreasing percentagewise.

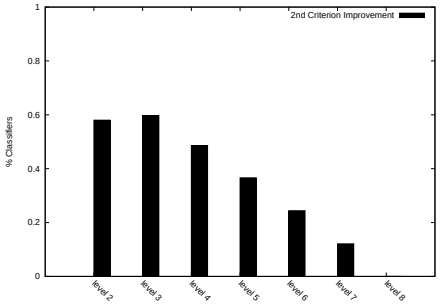
On the other hand, the first stopping criterion also tries to reduce the number of incident classifier evaluations. But this comes with a price: in general more evaluations of classes different from c_0 have to be evaluated to enable an earlier cut. But this case comes naturally by increasing the level, since each classifier involves an increasing number of classes, so that already with fewer evaluations, a reasonable amount of votes have been distributed. So in short, by increasing the level l, which increases the incidency, the reduction performance of QuickECOC is more and more due to the first stopping criterion whereas the impact of the second criterion decreases.

These considerations can be confirmed in Figure 4 and 5. Figure 4 shows the performances of QuickECOC with both stopping criteria, without the first criterion, and the incidency of the ECOC matrix for a given exhaustive ternary code level using the example of the dataset *ecoli* ($k = 8$). The differences between the red and blue lines on the left depict the additional improvement caused by the first stopping criterion, which can be seen also separately in Figure 5a. In line with the above considerations, one can see that the performance of QuickECOC without the first criterion is similar to the number of incident classifiers for a given level, it almost converges to it. Thus, the amount of non-incident classifiers R seems to decrease. We can observe that the first criterion begins to reduce the evaluations at $l = 3$ and the gain increases with increasing level (see also 5a). This additional improvement

[5] Actually, for exhaustive ternary codes, it holds $\#I_i = \#I_j$ and $\#R_i = \#R_j$ for arbitrary $c_i, c_j \in K$ and fixed level l. Thus, $\#I$ and $\#R$ are in this case only dependant on l and k.



(a) Additional First Criterion Improvement



(b) Second Criterion Improvement

Fig. 5: Impact of stopping criteria for reduction under varying level of exhaustive ternary codes (*ecoli*)

for $l = 3$ is nevertheless not the only reason for the best performance, since the improvement is too small. However, observing the figure, the right question seems rather why QuickECOC does perform so much worse for $l = 2$ rather than why $l = 3$ yields the best performance. For the sake of simplicity, it seems sufficient to consider only the second stopping criterion for this matter in the following.

For this case we describe a model which approximates the observed effect and therefore could yield a possible explanation. Assume that all classes form a linear order with respect to their votes, i.e. $v(c_1) > v(c_2) > \cdots > v(c_k)$ and that every classifier f returns a prediction in favour to the class with the highest (true) votes among the incident classes, i.e. the evaluation of f votes for the classes c_i, whose signs equal the one of class $c^* = \arg\max_{c_i \in IN(f)} v(c_i)$ and $IN(f)$ represents the set of incident classes of f. In addition, we reduce QuickECOC to a simple method which follows the only rule: Pick as the next classifier a remaining one which involves the classes with the lowest count of lost games. Now, we consider the worst case of classifier evaluation sequences, the maximal number of evaluations, until the true best class c_0 has been evaluated once as an estimate for R. It turns out that this count is $k - l$ for a given level l of exhaustive ternary codes. Then, because of our assumptions, the following holds: if a classifier involving the true class c_0 is evaluated, all

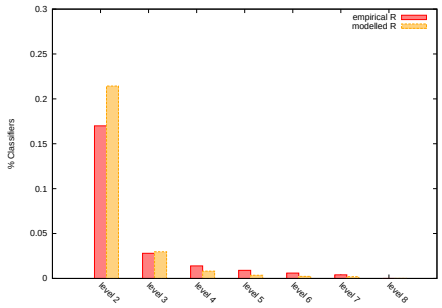


Fig. 6: Comparison of simplified QuickECOC versus actual performance ignoring the first stopping criterion (*ecoli*)

remaining classifiers will be an incident classifier of c_0. So, the worst case complexity of this setting is $k - l + \#I_0(l)$. As stated before, the cardinality of I_0 is given by l and k, more precisely $\#I_0(l) = \#I(l) = \frac{l}{k}n(k,l)$. This simplified model provides a surprisingly close fit to the empirical values, as one can see in Figure 6.

One can show that

$$2 = \arg\max_{l \in \{2 \ldots k\}} \frac{k-l}{n(k,l)} = \arg\max_{l \in \{2 \ldots k\}} \frac{k-l}{\binom{k}{l}(2^{l-1}-1)}$$

and in particular

$$\frac{k-2+\#I_0(2)}{n(k,2)} > \frac{k-3+\#I_0(3)}{n(k,3)} < \frac{k-l+\#I_0(l)}{n(k,l)}, \text{ where } k \geq l > 3$$

for $k > 4$.

So, the ratio of non-incident classifiers R has yet a significantly strong influence on the reduction for $l = 2$, in fact, it is the only case where R exceeds the constant increase of I_0 for the next level $l = 3$, i.e. $\frac{k-2}{n(k,2)} > \Delta\frac{\#I_0}{n} = \frac{1}{k}$. This yields a minimum of $(\#R + \#I)/n$ for $l = 3$, since R has an exponential decay, thus its influence for the overall reduction diminishes very fast (cf. Figure 6 or the differences of red and green values in Figure 4), whereas $\#I$ is constantly increasing. Here, the quantity of non-incident classifiers R was explained as the possible maximum amount of classifier evaluations avoiding the class c_0.