Efficient Decoding of Ternary Error-Correcting Output Codes for Multiclass Classification

Technical Report TUD-KE-2009-01

Sang-Hyeun Park, Johannes Fürnkranz
Knowledge Engineering Group, Technische Universität Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Knowledge
Engineering

Abstract

We present an adaptive decoding algorithm for ternary ECOC matrices which reduces the number of needed classifier evaluations for multiclass classification. The resulting predictions are guaranteed to be equivalent with the original decoding strategy except for ambiguous final predictions. The technique works for Hamming Decoding and several commonly used alternative decoding strategies. We show its effectiveness in an extensive empirical evaluation considering various code design types: Nearly in all cases, a considerable reduction is possible. We also show that the performance gain depends on the sparsity and the dimension of the ECOC coding matrix.

Contents

1 Introduction

Error-correcting output codes (ECOCs) (Dietterich and Bakiri, 1995) are a well-known technique for handling multiclass classification problems, i.e., for problems where the target attribute is a categorical variable with $k > 2$ values. Their key idea is to reduce the k-class classification problem to a series of n binary problems, which can be handled by a 2-class classification algorithm, such as a support vector machine or a rule learning algorithm.

This conversion may be viewed as *encoding* 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. The $k \times n$ matrix that consists of these entries is called the *coding matrix*. 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.

Ternary ECOCs (Allwein et al., 2000) are a generalization of the basic idea, which allows for three-valued code words and coding matrices, where the third value indicates that an example is ignored in the training set of the corresponding binary classifier. For example, pairwise classification (Friedman, 1996; Fürnkranz, 2002), which trains a classifier for each pair of classifiers, is a special case of this framework, where the coding matrix has $n = k(k-1)/2$ columns, each consisting of exactly one positive value $(+1)$, exactly one negative value (-1), and $k-2$ ignore values (0).

Although not strictly necessary, the number of the generated binary classification problems typically exceeds the number of class values, i.e., $n > k$. This allows for greater distances between the 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.

For many common general encoding techniques, the number of binary classifiers may exceed the number of classes 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 previous work (Park and Fürnkranz, 2007), we recently proposed the QWeighted algorithm, a fast decoding method for pairwise classifiers using a voting aggregation. Our experimental results showed that the quadratic complexity of the decoding step could be reduced to $O(k \log k)$ in practice. In this paper, we generalize the above-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, Euclidian distance, their attenuated counter-parts, loss-based decoding, and the Laplace decoding strategy.

In section 2, we will briefly recapitulate ECOC with an overview of typical code designs and decoding methods. In section 3, the previously mentioned QWeighted algorithm is presented and subsequently generalized for use with general ternary ECOCs. Its performance is evaluated and discussed in section 4 and 5. Finally, we will conclude and elaborate on possible future directions.

2 ECOC

Error-Correcting Codes (ECC) are a well-known topic in the field of Coding and Information Theory (Macwilliams and Sloane, 1983). Their main purpose was to detect and correct errors in noisy physical communication channels. Messages, represented as binary sequences, can be distorted during the communication process. Here, an error is regarded as receiving falsely 0 instead of 1 (or vice versa) at a given position. In principle, the task is tackled by enhancing the message sequence with redundant information bits, which are used for verification and correction on the receiver side.

Dietterich and Bakiri (1995) adapted this concept for multiclass classification and named it in this context as *Error Correcting Output Codes (ECOC)*. They consider classifier predictions as information signals which ideally describe the correct class for a given instance. Due to external influences (such as, e.g., a too small sample size) these signals are sometimes wrong, and such errors have to be detected and corrected. 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. 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.

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 $d(.)$ (such as the Hamming distance) is returned as the overall prediction.

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

Later, 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 say a classifier f_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 $m_{i,j} \neq 0$. This extension increases the expressive power of ECOCs, so that now nearly all common multiclass binarization methods can be modelled. For example, pairwise classification, 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.

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).

2.1 Code Design

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, many 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.

Since the introduction of ECOC, a considerable amount of research has been devoted to code design (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, which we will discuss in the following four sections.

2.1.1 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.

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

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.

2.1.2 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: $p(\{1\}) = p(\{-1\}) = \frac{1-p}{2}$. 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 the QuickECOC algorithm.

The second random code generation method selects randomly a subset from the set of all possible classifiers N_{all}. This set of classifiers N_{all} 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.

2.1.3 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, contrary to QuickECOC, 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 the number of classes is k.

2.1.4 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 these types of codes, because they need to be fit to each individual dataset. Besides, for some datasets, we simply lacked the necessary domain-specific knowledge.

2.2 Decoding

The traditional ECOC framework is accompanied with the Hamming Distance. After receiving an ensemble of base predictions, the class with the shortest Hamming Distance is selected as the output. In the meantime, several decoding strategies have been proposed. Along with the generalisation 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, and summarize them below.

In the following, let $\mathbf{p} = (p_1, \ldots, p_n)$ be the prediction vector and $\mathbf{cw_i} = (m_{i,1}, \ldots, m_{i,n})$ a code word from a ternary ECOC matrix.

- **Hamming Distance:**
  It is used in the traditional decoding method and describes the number of bit positions in which $\mathbf{cw_i}$ and $\mathbf{p}$ differ. Zero symbols ($m_{i,j} = 0$) increase the distance by $1/2$. Note that the prediction vector is considered as a set of binary predictions which can only predict either -1 or 1.

$$d_H(\mathbf{cw_i}, \mathbf{p}) = \sum_{j=1}^{n} \frac{|m_{i,j} - p_j|}{2} \tag{2.1}$$

- **Euclidian Distance:**
  computes the distance of the two n-dimensional vectors in Euclidian space.

$$d_E(\mathbf{cw_i}, \mathbf{p}) = ||\mathbf{cw_i} - \mathbf{p}||_2 = \sqrt{\sum_{j=1}^{n} (m_{i,j} - p_j)^2} \tag{2.2}$$

- **Attenuated Euclidian/Hamming Distance:**
  These measures simply ignore the zero symbols for computing the distance.

$$d_{AE}(\mathbf{cw_i}, \mathbf{p}) = \sqrt{\sum_{j=1}^{n} |m_{i,j}|(m_{i,j} - p_j)^2} \tag{2.3}$$

  The analogue version for the Hamming distance is:

$$d_{AH}(\mathbf{cw_i}, \mathbf{p}) = \sum_{j=1}^{n} \left| m_{i,j} \right| \frac{|m_{i,j} - p_j|}{2} \tag{2.4}$$

- **Loss based:**
  In loss based decoding we assume that we have given a score-based classifier $f(.)$.

$$d_L(\mathbf{cw_i}, \mathbf{p}) = \sum_{j=1}^{n} l(m_{i,j} \cdot f_j) \tag{2.5}$$

  where $l(.)$ is the loss function. Typical functions are $l(s) = -s$ and $l(s) = e^{-s}$.

- **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.

  $$d_{LA}(\mathbf{cw_i}, \mathbf{p}) = \frac{E+1}{E+C+T} = \frac{d_{AH}(\mathbf{cw_i}, \mathbf{p}) + 1}{\sum_{j=1}^{n} |m_{i,j}| + T} \tag{2.6}$$

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

- **Beta Density Distribution Pessimistic Strategy:**
  This measure assumes that the distance is a Beta-distributed random variable parametrized by C and E of two code words. It can be seen as a probabilistic version of the Laplace strategy, because its expected value equals the one from the Laplace strategy. 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 $Z_i := \text{argmax}_{z \in [0,1]}(\psi_i(z))$ and $a_i \in [0,1]$ such that

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

  then we define

  $$d_{BD}(x, y) = Z_i + a_i \tag{2.7}$$

  The value a_i is herefore regarded as a pessimistic value, which incorporates, all in all, the uncertainty of Z_i into the distance measure, which is provided by the probabilistic approach.

3 Efficient Decoding for ECOC

In this section, we will introduce the QuickECOC algorithm for efficiently determining the predicted class without the need to evaluate all binary classifiers. It builds upon the QWeighted algorithm (Park and Fürnkranz, 2007), which is tailored to the special case of pairwise classification with voting aggregation as a decoding technique. We will first briefly recapitulate this algorithm in Section 3.1, and then discuss the three key modifications that have to be made: first, Hamming decoding has to be reduced to a voting process (Section 3.2), 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 3.3), and finally the stopping criterion can be improved to take multiple incidences into account (Section 3.4). We will then present the generalized QuickECOC algorithm for Hamming decoding in Section 3.5. Finally, we will discuss how QuickECOC can be adapted to different decoding techniques (Section 3.6).

3.1 QWeighted for Pairwise Classification

Pairwise classification (Friedman, 1996) tackles the problem of multiclass classification by decomposing the main problem into a set of binary problems, one problem for each pair of classes. At prediction time, all binary classifiers are queried, and each classifier emits a vote for one of its two classes. The class which receives the maximum amount of votes is eventually predicted.
Though it can be shown that the training time of pairwise classification is smaller than in the one-against-all case (Fürnkranz, 2002), a quadratic number of classifiers still has to be evaluated at classification time. The QWeighted algorithm (Park and Fürnkranz, 2007) addresses this problem by exploiting the fact that usually not all evaluations are necessary to compute the class with the maximum votes. If one class has received more votes than every other class can possibly achieve in their remaining evaluations, this class can be safely predicted. The QWeighted algorithm tries to enforce this situation by always focusing on the class that has lost the least amount of voting mass. Experiments showed that QWeighted uses an average runtime of $O(k \log k)$ instead of the $O(k^2)$ that would be required for computing the same prediction with all evaluations.

3.2 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 (+1), one negative (−1), and $k-2$ ignore values (0). 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.[1] In voting aggregation, the class that receives the most votes from the binary classifiers is predicted, i.e.,

$$\tilde{c} := \operatorname*{argmax}_{i \in K} \sum_{j \neq i, j \in K} f_{i,j}$$

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 $\mathbf{p} = (p_1, \ldots, p_n)$. The following lemma allows to reduce minimization of Hamming distances to voting aggregation:

Lemma 1. *Let $v_{i,j} := \left(1 - \frac{|m_{i,j} - p_j|}{2}\right)$ be a voting procedure for classifier f_j for class c_i then*

$$\operatorname*{argmin}_{i=1\ldots n} d_H(\mathbf{cw_i}, \mathbf{p}) = \operatorname*{argmax}_{i=1\ldots n} \sum_{j \in N} v_{i,j}$$

[1] Other choices for decoding pairwise classifiers are possible (cf., e.g., (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).

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 $b_{i,a} \in \{0, 0.5, 1\}$ and

$$\min_{i \in 1 \dots k} \sum_{a=1}^{n} b_{i,a} \rightarrow \max_{i \in 1 \dots k} \sum_{a=1}^{n} 1 - b_{i,a} = \max_{i \in 1 \dots k} \sum_{a=1}^{n} v_{i,a}$$

holds, we obtain the statement. □

To be clear, the above used definition of $v_{i,j}$ formalizes a voting procedure, for which class c_i receives one vote $(+1)$, if the prediction p_j of classifier j equals the corresponding encoding bit $m_{i,j}$ and an half vote $(+0.5)$ for the case $m_{i,j} = 0$ where the classifier was not trained with instances from c_i.

This voting schemes differs slightly from the commonly known voting aggregation. The exact voting aggregation procedure described within the ECOC framework would be

$$v_{i,j} = |m_{i,j}| \cdot \left(1 - \frac{|m_{i,j} - p_j|}{2}\right)$$

which ignores the zero symbols and is not equivalent with Hamming decoding for arbitrary ternary coding matrices (but for e.g. pairwise codes w.r.t final prediction). Nevertheless, it is easy to see, that voting aggregation is equivalent to ECOC decoding using the Attenuated Hamming distance.

3.3 Next Classifier Selection

The QWeighted algorithm always pairs the class with the least amount of voting loss 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. The rationale behind this approach is that the current favorite (the class with the least amount of voting loss) should quickly be paired against its strongest competitors, so that it can emerge as a winner as quickly as possible. In pairwise classification, the choice of a classifier for a given pair of classes 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 involve more than two classes, and, conversely, 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, assume a 4-class problem (A, B, C, D) using 3-level ternary exhaustive codes, and classes A and B have currently the greatest vote amount, we could select one of four different classifiers that discriminate the classes A and B, namely $A|BC$, $A|BD$, $AC|B$ and $AD|B$.[2]

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 N_{i_0}. Let K_j denote the set of classes, which are involved in the binary classifier f_j, but with a different sign than c_{i_0}. In other words, it contains all rows i of column j in the coding matrix M, for which holds: $m_{i,j} \neq m_{i_0,j} \wedge m_{i,j} \neq 0$. We then compute a score

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

for every classifier $c_j \in N_{i_0}$, where $r(i)$ is a function which returns the position of class c_i in a ranking, where all classes are increasingly ordered by their current votings respectively ordered decreasingly by distances. Finally, we select the classifier f_{j_0} with the maximal score $s(j_0)$. Roughly speaking, this relates 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.

2 We use the following notation: The classifier is trained using the instances from the classes left of the |-symbol as positive examples and the instances from the classes of the right side as negative examples.

Algorithm 1 QuickECOC

```
Require: ECOC Matrix $\mathbf{M} = (m_{i,j}) \in \{-1,0,1\}^{k \times n}$, binary classifiers $f_1, \ldots, f_n$, testing instance $\mathbf{x} \in X$
 1: $\mathbf{l} \in \mathbb{R}^k \Leftarrow 0$                                   # Hamming distance vector
 2: $c^* \leftarrow$ NULL
 3: $N \leftarrow \{1, \ldots, n\}$
 4: while $c^* =$ NULL do
 5:     $j \leftarrow$ SelectNextClassifier($\mathbf{M}, \mathbf{l}$)
 6:     $p \leftarrow f_j(\mathbf{x})$                                        # Evaluate classifier
 7:     for each $i \in K$ do
 8:         $l_i \leftarrow l_i + \frac{|m_{i,j} - p|}{2}$
 9:     $M \leftarrow M \backslash M_j$
10:     $N \leftarrow N \backslash \{j\}$
11:     $i_0 = \operatorname{argmin}_{i \in K} l_i$
12:                                                                  # First stop Criterion
13:     abort $\leftarrow$ true
14:     for each $i \in K \backslash \{i_0\}$ do
15:         $e_{Full} \leftarrow |\{j \in N | m_{i,j} \times m_{i_0,j} = -1\}|$
16:         $e_{Half} \leftarrow |\{j \in N | m_{i,j} \neq 0$ and $m_{i_0,j} = 0\}|$
17:         if $l_{i_0} + e_{Full} + \frac{1}{2} e_{Half} > l_i$ then
18:             abort $\leftarrow$ false
19:                                                                  # Second stop Criterion
20:     if abort or $\forall j \in N . m_{i_0,j} = 0$ then
21:         $c^* \leftarrow c_{i_0}$
22: return $c^*$
```

3.4 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 $c_i \in K \backslash \{c_{i_0}\}$. 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. Thus, we simply add the number of remaining incident classifiers of c_{i_0} 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 increase the Hamming distance of *all* (or none) of the incident classifiers that have the same sign (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}, but only those where c_i and c_{i_0} are on different sides of the training set. For the cases where c_i was ignored in the training phase, $\frac{1}{2}$ is added to the distance, according to the definition of the Hamming distance for ternary code words. 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.

3.5 Quick ECOC Algorithm

Algorithm 1 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 indicates the current accumulated Hamming distance of the associated code word $\mathbf{cw}_i$ of class c_i to the currently evaluated prediction bits $\mathbf{p}$. The l_i can be seen as lower bounds of the distances $d_H(\mathbf{cw}_i, \mathbf{p})$, 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 $i_0 = \text{argmin}_i l_i$ with another class that is selected as described in Section 3.3. 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 3.2) and f_j is removed from the set of possible classifiers.

- **(3) First Stopping Criterion:**
  In 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 3.4.

- **(4) Second Stopping Criterion:**
  At line 19, the algorithm stops when all incident classifiers of c_{i_0} have been evaluated.[3] 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. Self evidently, this criterion covers also the situation when all classifiers have been evaluted (which would be the worst case regarding the runtime because QUICKECOC would not have been able to reduce the number of classifier evaluations).

3.6 Decoding Adaptions

All decoding methods that we discussed in section 2.2 are compatible with QUICKECOC by applying small modifications. 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. We will briefly describe the modifications for all considered decoding strategies:

- **Euclidian Distance:**
  For minimizing the Euclidian distance we can ignore the root operation and simply substitute the update statement of the pseudocode (line 7) with: $l_i \leftarrow l_i + (m_{i,j} - p)^2$. The factor for e_{Half} is changed to 1 and the one for e_{Full} to 4.

- **Att. Euclidian Distance:**
  Similar to the above modifications we change line 7 with: $l_i \leftarrow l_i + |m_{i,j}|(m_{i,j} - p)^2$ and set the factor of e_{Full} to 4 and remove the occurrences of e_{Half}.

- **Loss based linear:**
  For both loss based versions, we assume that we have given a normalizing function $w(.)$ which projects $f_i(x)$ from $[-\infty : \infty]$ to $[-1, 1]$, e.g.,

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

  We substitute line 6 with: $p \leftarrow w(f_j(x))$ and the update procedure with: $l_i \leftarrow l_i + \frac{1 - p \cdot m_{i,j}}{2}$ and remove the occurrences of e_{Half}.[4]

- **Loss based exponential:**
  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 factor of e_{Full} is set to e^1 and e_{Half} to e^{-1}.

[3] The second stopping criterion is actually a special case of the first stopping criterion and could be removed. However, since this distinction is useful later in the appendix for analysis purposes, we let it unchanged.

[4] Note that we did not use such a normalizing function in our actual evaluation since we used a decision tree learner as our base learner. 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).

- **Laplace Strategy:**
  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** amounts $\frac{1}{b+T}$ towards 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 has to be changed to $l_i \leftarrow l_i + I_i$ and the factor of e_{Full} changes to I_i. Besides, e_{Half} can be removed.

- **Beta Density Distribution Pessimistic Strategy:**
  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 to $l_i \leftarrow l_i + \min(1, (I_i + \sigma_i))$. The factor for e_{Full} has to be changed to I_i and e_{Half} has to be removed.[5]

  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] Note that this approximation yielded in all our evaluations the same prediction as the original strategy.

Table 4.1: QuickECOC Performance using Hamming Decoding and Exhaustive Ternary Codes

	vehicle	dermatology	auto	glass	zoo	ecoli	machine
$l = 2$	3.82 *0.637*	7.12 *0.475*	7.95 *0.379*	9.99 *0.476*	9.48 *0.451*	11.75 *0.420*	11.60 *0.414*
$l = 3$	7.91 *0.659*	26.05 *0.434*	42.86 *0.408*	43.47 *0.414*	41.64 *0.397*	58.85 *0.350*	57.90 *0.345*
$l = 4$	5.65 *0.808*	46.30 *0.441*	115.22 *0.470*	116.45 *0.475*	107.03 *0.437*	199.31 *0.407*	194.81 *0.398*
$l = 5$		43.11 *0.479*	163.67 *0.520*	163.98 *0.521*	148.50 *0.471*	369.06 *0.439*	355.23 *0.423*
$l = 6$		16.54 *0.534*	114.87 *0.529*	116.77 *0.538*	102.41 *0.472*	394.25 *0.454*	369.19 *0.425*
$l = 7$			34.24 *0.543*	37.84 *0.601*	31.52 *0.500*	234.80 *0.466*	218.09 *0.433*
$l = 8$						62.17 *0.490*	57.27 *0.451*

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.

4.1 Experimental Setup

All experiments were performed within the Weka (Witten and Frank, 2005) framework using the decision tree learner J48 with default parameters as a base learner. All evaluations were performed using 10-fold stratified cross-validation. Our setup consisted of

- **5 encoding strategies**: BCH Codes and two versions each of exhaustive and random codes.
- **7 decoding methods**: Hamming, Euclidian, Att. Euclidian, linear loss-based, exponential loss-based, Laplacian Strategy and Beta Density Probabilistic Pessimistic
- **7 multiclass datasets** selected from the UCI Machine Learning Repository (Asuncion and Newman, 2007)

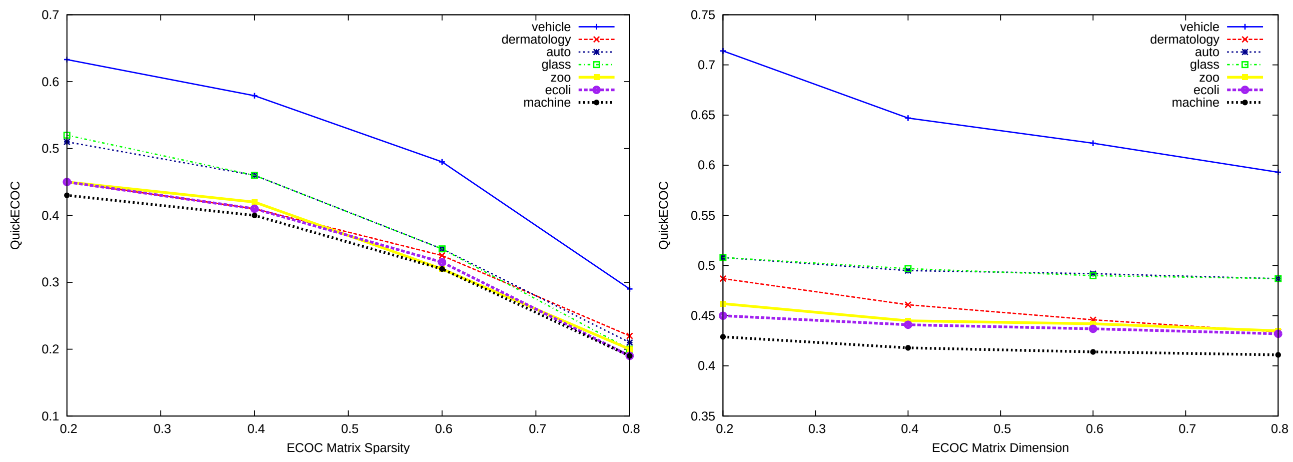
For the encoding strategies, we also tried several different parameters. Regarding the exhaustive codes, we evaluated all (k, l) codes ranging from $l = 2$ to $l = k$ per dataset and analogously for the cumulative version. For the generation of the first type of random codes the zero symbol probability 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.

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 want to stress that our technique is applicable to larger number of classes (with reasonable codes), and, as our results will show, the expected gain increases with the number of classes.

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.

Table 4.2: QuickECOC Performance on BCH Codes

	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

Figure 4.1: QuickECOC Performance of random codes

4.2 Reduction in Number of Evaluations

Table 4.1 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*, $l = 3$). 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" was acquired, which is further described in the appendix.

For BCH Codes, we can report also that in all cases a reduction was possible, as one can see in Table 4.2. Note that all coding matrices in this case are dense, i.e., no coding matrix contains a 0. 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 BCH bit code) higher reductions can be observed.

We do not show a detailed table of results for random codes, but they will be used in the following sections.

4.3 Sparsity of Coding Matrices

We define the sparsity of the ECOC matrix as the fraction of (0)-values it contains. Random codes provide a direct control over the matrix sparsity (as described in section 2.1.2), 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

Table 4.3: QuickECOC Performance on *ecoli* with all Decoding Methods and Cumulative Exhaustive Ternary Codes

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

dimension of the matrix on the QuickECOC performance can also be seen in the evaluations of the other code types, but not as clearly or structured as here.

Figure 4.1a 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).

4.4 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 4.1b. 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.

4.5 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 4.3 shows an overview of the QuickECOC performance for all 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 Discussion

At first glance, the results may not be striking, because a saving of a little less than 40% 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 (cf. Table 4.1). A $(4,3)$-exhaustive code has 12 classifiers, and each individual class is involved in 75% of these classifiers (cf. the example in section 2.1.1). Thus, on average, QuickECOC did not even evaluate all the classifiers that involve the winning class before this class was predicted. It would be interesting to derive a lower bound on the possible optimal performance, and to relate these empirical results to such a bound.

One could also argue that in applications where the classification time is crucial, a parallel approach could be applied much more effectively. Since each classifier defined by a column of the ECOC matrix can be evaluated independently, the implementation could be done very easily. QuickECOC loses 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.

Another point is 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.

6 Conclusions

We have shown a general algorithm for reducing the number of classifier evaluations for ternary ECOC matrices without compromising the overall prediction. It is based on a similar algorithm that was tailored to pairwise classification. Since ternary ECOCs subsume nearly all possible binary decomposition schemes, the reduction applies now to a broader spectrum of applications. For example, data-specific optimal codes can now also take advantage of reduced classifier evaluations. Regardless of the used code, QUICKECOC improves the overall prediction efficiency. At first sight, the amount of improvement may not seem to be as striking as for the pairwise case, where we could report a reduction from k^2 to $k \log k$ (Park and Fürnkranz, 2007), but 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. We observed that the performance gain increases with higher sparsity of the coding matrix, again putting pairwise classification at the more efficient end of the spectrum. We also noted an increase in the performance gain with increasing code lengths of the chosen code.

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, because they seem to share some similarities.

Acknowledgments

We would like to thank Eyke Hüllermeier and Lorenz Weizsäcker for helpful suggestions and discussions. This work was supported by the *German Science Foundation (DFG)*.

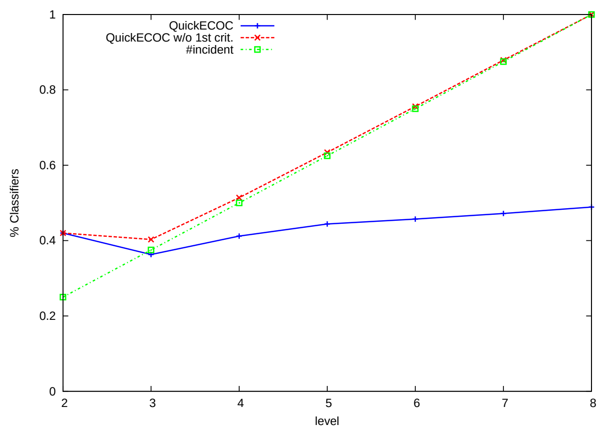


Figure A.1: Dependency of stopping criteria for different levels respectively incidencies (*ecoli*)

A Appendix

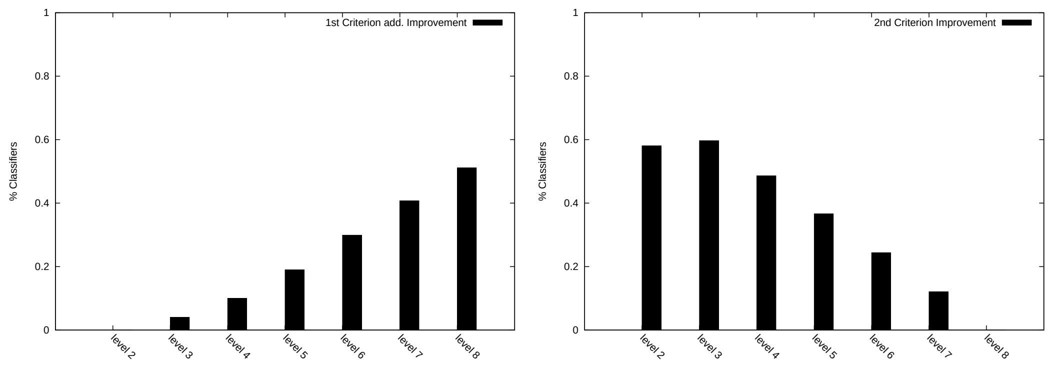
A.1 Analysis of (k,3)-Exhaustive Ternary Codes

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 a subset of non-incident classifiers of c_i.[1] 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 A.1 and A.2. Figure A.1 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 A.2a. 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 A.2a). This additional improvement 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.

[1] 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

Figure A.2: Impact of stopping criteria for reduction under varying level of exhaustive ternary codes (*ecoli*)

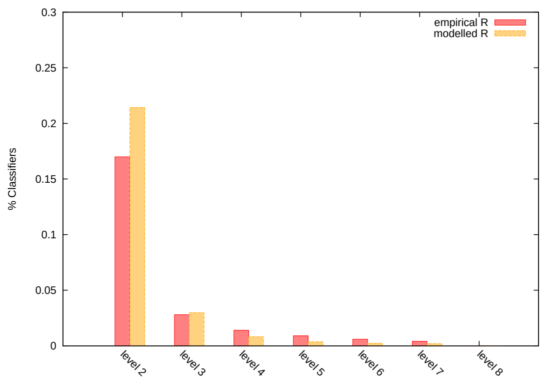


Figure A.3: Comparison of simplified QuickECOC versus actual performance ignoring the first stopping criterion (*ecoli*)

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 $c^* = \text{argmax}_{c_i \in IN(f)} v(c_i)$ 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 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 A.3.

One can show that

$$2 = \underset{l \in \{2 \dots k\}}{\text{argmax}} \frac{k-l}{n(k,l)} = \underset{l \in \{2 \dots k\}}{\text{argmax}} \frac{k-l}{\binom{k}{l}(2^{l-1}-1)}$$

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 A.3 or the differences of red and green values in Figure A.1), 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.

Bibliography

E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000. 3, 4, 6

A. Asuncion and D. Newman. UCI machine learning repository, 2007. 13

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. 5, 6

J. S. Cardoso and J. F. P. da Costa. Learning to classify ordinal data: The data replication method. *Journal of Machine Learning Research*, 8:1393–1429, 2007. 6

K. Crammer and Y. Singer. On the learnability and design of output codes for multiclass problems. *Machine Learning*, 47 (2-3):201–233, 2002. 5

T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995. 3, 4, 5, 6

S. Escalera, O. Pujol, and P. Radeva. Decoding of ternary error correcting output codes. In J. F. M. Trinidad, J. A. Carrasco-Ochoa, and J. Kittler, editors, *CIARP*, volume 4225 of *Lecture Notes in Computer Science*, pages 753–763. Springer, 2006. 6

J. H. Friedman. Another approach to polychotomous classification. Technical report, Department of Statistics, Stanford University, Stanford, CA, 1996. 3, 8

J. Fürnkranz. Round robin classification. *Journal of Machine Learning Research*, 2:721–747, 2002. 3, 8

E. Hüllermeier, J. Fürnkranz, W. Cheng, and K. Brinker. Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172:1897–1916, 2008. 8

E. B. Kong and T. 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. 4

F. J. Macwilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland Mathematical Library. North Holland, January 1983. 4, 6

I. Melvin, E. Ie, J. Weston, W. S. Noble, and C. Leslie. Multi-class protein classification using adaptive codes. *Journal of Machine Learning Research*, 8:1557–1581, 2007. 6

S.-H. Park and J. 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, 2007. Springer-Verlag. 3, 8, 17

E. Pimenta, J. Gama, and A. C. P. de Leon Ferreira de Carvalho. The dimension of ECOCs for multiclass classification problems. *International Journal on Artificial Intelligence Tools*, 17(3):433–447, 2008. 5

O. Pujol, P. Radeva, and J. Vitria. 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. 6

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

I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques, 2nd Edition*. Morgan Kaufmann, San Francisco, 2005. 13

T.-F. Wu, C.-J. Lin, and R. C. Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5:975–1005, August 2004. 8