Exploiting Coding Theory for Classification: an LDPC-based Strategy for Multiclass-to-Binary Decomposition

C. Marrocco[a,*], F. Tortorella[a]

[a]*Department of Electrical and Information Engineering, University of Cassino and L.M., Via G. Di Biasio 43, 03043 Cassino (FR), Italy*

Abstract

A powerful strategy to face the classification of multiple classes is to create a classifier ensemble that decomposes the polychotomy into several dichotomies. The central issue in designing a multiclass-to-binary decomposition scheme is the definition of both the coding matrix and the decoding algorithm. In this paper, we propose a new classification system based on the Low-Density Parity-Check codes, a very effective class of binary block codes. The main idea is to exploit the algebraic properties of such codes both to generate the codewords of the coding matrix and to define two decoding approaches that allow us to detect and recover possible errors or rejects produced by the dichotomizers. Experiments on benchmark datasets have shown that the proposed approach provides a statistically significant improvement in classification performance over state-of-the-art decomposition strategies.

Keywords: Multiple Classifier Systems, Multiclass-to-binary decomposition, Coding Theory, Low-Density Parity-Check (LDPC), Reject, Error-Correcting Output Coding (ECOC)

1. Introduction

Many real-world applications involve multiclass classification problems. Face recognition, image categorization, biometric identification are only some of the challenging tasks in Pattern Recognition dealing with multiple classes. A practical way to face these problems is to use a monolithic classifier that works by modeling the probability distribution functions or by building the decision regions for each class. An alternative approach is to split the original polychotomy

*Corresponding author. Address: Department of Electrical and Information Engineering, University of Cassino and L.M., Via G. Di Biasio 43, 03043 Cassino (FR), Italy. Tel.: +39 07762993381

Email addresses: c.marrocco@unicas.it (C. Marrocco), tortorella@unicas.it (F. Tortorella)

https://doi.org/10.1016/j.ins.2016.02.039

into a series of dichotomies that can be faced through an ensemble of two-class classifiers (a.k.a. dichotomizers) each facing a particular dichotomy. The outputs of the dichotomizers are then combined to infer the multiclass prediction.

There are several approaches to design a multiclass-to-binary decomposition scheme. Two well-known strategies are *one-vs-all* (OVA) [35] and *one-vs-one* (OVO) [21]. The former determines a dichotomy for each class by separating it from the remaining classes, whereas the latter considers all pairs of different classes and defines each subproblem by discriminating one class from another. Another technique is the *Error Correcting Output Coding* (ECOC) [10]. The rationale is to assign each class a unique binary string (referred to as *codeword*). Codewords can be arranged in the rows of a discrete decomposition matrix, named *coding matrix*, where each column defines a binary partition that groups the original classes into two superclasses. In the decision stage, for each unknown sample, the outputs of the dichotomizers are collected in a word that is used to predict a class according to a suitable decoding technique.

The performance of a decomposition scheme is highly dependent on the coding and decoding strategies. As for the coding, there are two groups of techniques: data-independent and data-dependent. The first one exploits the correction capabilities of predefined codes. In this context, several research studies were conducted to improve OVA [17, 42] and OVO [18, 25] or to introduce linear codes such as the exhaustive codes [10] and the random codes [1]. The second group has recently drawn great attention and focuses on coding strategies designed for the multiclass problem at hand. Several approaches were proposed to design efficient codes depending on the set of dichotomizers [8], the data distribution [2, 40, 54] or the binary subproblems [15]. A method to extend the coding matrix with new dichotomizers was proposed in [39], whereas in [22] the coding matrix was shrunk by eliminating "useless" subproblems. The feature space of an ECOC system was studied in [53], while in [4] an ECOC-based feature extraction was proposed. Genetic programming was considered in [5, 20] to build the coding matrix, and also the reject rule was introduced in an ECOC system [46]. Other methods were proposed to introduce ternary codes [1, 14, 52], to embed optimal classifiers in the ECOC approach [11, 49] or to reduce the number of employed dichotomizers [6, 43]. In the decoding stage, the decision is typically based on the Hamming distance [35] between the codewords and the output word. Other decoding strategies [51] were however proposed including Euclidean distance [21], probabilistic rules [36], loss function [1], weighted loss-based distance [14] or reject rules [45].

All the described approaches took inspiration from the seminal paper by Dietterich and Bakiri [10] which states that the learning task of a decomposition scheme can be seen "as a kind of communications problem in which the identity of the correct output class for a new example is being transmitted over a channel". However, all the strategies usually employed in the literature have drifted away from this statement and, generally speaking, do not exploit the capabilities provided by the robust theoretical foundations of Coding Theory. In such framework, Error Correcting Codes are usually employed to introduce redundancy, i.e., to increase the length of the codewords, with the purpose of

https://doi.org/10.1016/j.ins.2016.02.039

recovering the original information from the output of a channel through sets of suitably distinct codewords. In this paper, we exploit the features of Coding Theory to build a multiclass-to-binary learning system. Since we are interested in solving a classification problem, the criteria that guided the design of our coding and decoding system are the improvement of the classification performance and, at the same time, an affordable computational complexity. For this purpose, the coding matrix of the proposed decomposition scheme is designed by exploiting the algebraic properties of a well-known family of binary block codes, the *Low-Density Parity-Check* (LDPC) codes [19].

A classification system employing LDPC codes for problem decomposition was firstly introduced in [49]. In that paper, the emphasis is placed on maximizing the diversity among the used dichotomizers. To this end, a two-stage method is proposed to choose the best coding matrix from a large number (e.g. 10000) of LDPC codes previously generated. The training is realized as in the traditional ECOC system, whereas an iterative rule based on the sum-product algorithm [27] is applied in the decoding procedure.

In this paper, we propose a new classification system where the characteristics of the LDPC codes are fully exploited both in coding and decoding. Different from [49], the coding procedure of our method does not require any selection of the coding matrix. At the same time, we are able to limit the number of dichotomizers to be trained by assigning the same dichotomizer to those columns in the coding matrix facing the same binary problem. In the decoding stage, the sparsity of the *parity-check matrix*, that characterizes the LDPC codes, allows the use of very efficient algorithms, namely the *bit-flipping* and the *recovery algorithm.* We have already used such algorithms in [32, 33], where the aim was to embed LDPC codes in a traditional ECOC framework. Here we considerably extend the approach presented in our previous papers and introduce two improved variants of the decoding rules: the *block bit-flipping* and the *block recovery* algorithms. The first one exploits the redundancy of the code to algebraically recover the errors made by dichotomizers. The second decoding rule is able to manage rejects, i.e., events where the dichotomizer abstains from deciding when it is likely to be in error. To this end, a decomposition scheme is presented where all the dichotomizers are designed with a reject option that allows us to significantly increase the reliability of their outcomes. The proposed framework is particularly suitable to design a strategy that strongly relies on trustworthy dichotomizers and algebraically recover the outputs of erroneous or unreliable classifiers so improving the performance of the whole classification system.

In this paper, we also present an extensive evaluation of the proposed decomposition scheme. Three experiments were performed on several benchmark datasets with different aims: (i) to study how the LDPC code parameters influence the performance of the whole classification system; (ii) to analyze how the use of a reject rule on the two-class classifiers influences the behavior of the decoding procedure; (iii) to show that the proposed classification system provides a statistically significant improvement in performance over state-of-the-art decomposition schemes with a feasible computational complexity.

https://doi.org/10.1016/j.ins.2016.02.039

The paper is organized as follows: Sect. 2 reports a brief survey of state-of-the-art methods. Sect. 3 gives an overview of Coding Theory concepts, and in Sect. 4 the LDPC codes are introduced. Sect. 5 presents the proposed LDPC-based classification system. The experimental results have been reported and discussed in Sect. 6. Finally, Sect. 7 concludes the paper.

2. An overview of decomposition schemes

Generally speaking, the decomposition of a classification problem with M classes generates L dichotomies (corresponding to L different two-class aggregations of the original classes) that can be faced through L dichotomizers. A *coding matrix* $\mathbf{C} = \{c_{ij}\}_{i=0,\dots,M-1;j=0,\dots,L-1}$ of dimensions $M \times L$ is usually employed to represent the decomposition and to connect each class label $\omega_i, \forall i = 1, \dots, M$ to a unique bit string of length L, named *codeword*. Each row of $\mathbf{C}$ defines a codeword, whereas each column represents the two-class problem on which a dichotomizer has to be trained. The relation between the classes and the dichotomizers can be binary if $\mathbf{C} \in \{-1, +1\}^{M\times L}$ or ternary if $\mathbf{C} \in \{-1, 0, +1\}^{M\times L}$. The j-th classifier f_j is trained according to the dichotomy in the j-th column by building the two-class training set as follows: samples of the i-th class belong to the positive class if $c_{ij} = +1$, to the negative class if $c_{ij} = -1$, and do not participate in the training of f_j if $c_{ij} = 0$.

In the decision stage an unknown sample $\mathbf{x}$ is classified by the L trained dichotomizers. The L outputs are collected in an *output word* $\mathbf{o} = (o_0(\mathbf{x}), \dots, o_{L-1}(\mathbf{x}))$ that is used to determine the class of the unlabeled pattern $\mathbf{x}$. To this end, $\mathbf{o}$ is compared with the codewords of $\mathbf{C}$ using a proper measure of distance, and $\mathbf{x}$ is assigned to the class ω associated with the "closest" codeword:

$$\omega = \arg \min_{0\le h\le M-1} Dist(\mathbf{c}_h, \mathbf{o}). \tag{1}$$

Different measures can be adopted [1, 14, 51], but the most common is the Hamming Distance.

Definition 2.1 (**Hamming Distance**)**.** The *Hamming distance* D_H between two words is given by the number of positions where the bit patterns of the two words differ.

Following this definition, the Hamming distance D_H between the i-th codeword $\mathbf{c}_i$ and the output word $\mathbf{o}$ is given by:

$$D_H(\mathbf{c}_i, \mathbf{o}) = |\{h : c_{ih} \neq o_h\}|, \tag{2}$$

where the notation $|\cdot|$ denotes the cardinality of a set.

To reduce multiclass to binary problems, the literature reports several approaches that can be grouped in two great families: data-independent and data-dependent strategies. In the first case, the coding matrices are designed independently of the learning algorithm and the training data. In the second group

https://doi.org/10.1016/j.ins.2016.02.039

the characteristics of the data are considered in the design of the codewords and in the number of dichotomizers to be employed.

2.1. Data-independent strategies

The most popular approaches in this group of strategies are OVA [35] and OVO [21]. In OVA M two-class problems, one for each class, are defined to separate a class from the remaining ones, whereas OVO splits the M multiclass problem into a set of $M(M-1)/2$ two-class problems including all combinations of pairs of classes. Another common technique is to employ an ECOC to find suitable codewords to be assigned to different classes. In this framework, coding matrices are usually designed to increase the Hamming distance between both rows and columns, with the aim of reducing both the confusion among classes and the correlation among dichotomizers. To this end, in [10] exhaustive codes are used by considering $2^{M-1} - 1$ possible dichotomies. When M increases, randomized hill climbing and Bose-Chaudhuri-Hocquenghem (BCH) codes are used to reduce the number of employed dichotomizers. Even two families of random codes have been proposed in [1]: *dense random codes* where the codewords are binary with $\lceil 10 \log_2 M \rceil$ bits and *sparse random codes* consisting of ternary codewords with length $\lceil 15 \log_2 M \rceil$. Other methods employ genetic programming [6, 20] or diversity measures [28] to build data-independent coding matrices.

2.2. Data-dependent strategies

The first decomposition method focusing on a data-dependent coding matrix has been proposed in [2] where multi-layer perceptrons are used as dichotomizers and the backpropagation algorithm is employed to find the codewords. Thereafter, a suboptimal decomposition scheme based on the Expectation-Maximization algorithm [50] has been proposed, whereas an approach to find optimal codewords by designing continuous codes has been introduced in [8]. More recently, several relevant strategies have been designed. In particular, Data-Driven ECOC [54] explored data-per-class distributions to optimize the coding matrix and the number of base classifiers by measuring the confidence degree of each two-class subproblem. Pujol et al. [40] proposed Discriminant ECOC, a heuristic method for building ECOC matrices of $M - 1$ columns through a hierarchical partition of the class space. This method has also been extended in [11] where different trees are combined in a forest to ensure the required classification performance. To improve the performance of an initial coding matrix, ECOC Optimizing Node Embedding [39] has been proposed, which iteratively adds dichotomizers by discriminating the most confusing subproblems. Even ternary codes have been studied: Escalera et al. [14] proposed an approach where the code dependence from subclass problems is analyzed by splitting the most confusing class to several subsets, whereas in [12] a new sparse random coding matrix with ternary distance maximization has been proposed.

https://doi.org/10.1016/j.ins.2016.02.039

3. Basics of Coding Theory

To highlight some useful properties of linear codes, we first recall some basic concepts of Coding Theory [34, 41]. Let us consider a particular case of Galois fields, the binary field $GF(2)$, defined on a set containing only two elements, that usually are $\{0, 1\}$[1]. To describe a linear block code, we have to refer to $GF^L(2)$, the vector space over the field $GF(2)$, that contains 2^L ordered sequences of L components belonging to $GF(2)$. Over $GF^L(2)$ two operations are defined: the $\bmod 2$ addition between two vectors of $GF^L(2)$ and the $\bmod 2$ multiplication between an element of the field $GF(2)$ and a vector of $GF^L(2)$.

Definition 3.1 (**Linear Block Code**). A *linear block code* $\mathcal{C}(L, K)$ is a K-dimensional vector subspace of $GF^L(2)$: the vectors of the subspace are the codewords of $\mathcal{C}$, the sum of any two codewords is a codeword and the product of a codeword with 0 or 1 is still a codeword.

Let us denote with $\mathbf{u} = (u_0, u_1, ..., u_{K-1}))$ a K-bit source message and with $\mathbf{c} = (c_0, c_1, ..., c_{L-1})$ an L-bit codeword; to encode a source message means to take one of the 2^K source vectors $\mathbf{u}$ and employ a bijective function to associate it to one of the 2^L vectors of L bits.

Definition 3.2 (**Redundancy**). The *redundancy* of the code $\mathcal{C}(L, K)$ is the difference $L - K$.

Definition 3.3 (**Code Rate**). The *code rate*, i.e., the *transmission rate* of the code $\mathcal{C}(L, K)$, is the ratio $R_{\mathcal{C}} = K/L$ of message symbols to coded symbols.

Definition 3.4 (**Minimum Hamming Distance**). The *minimum Hamming distance* d_{min} of a code $\mathcal{C}(L, K)$ is the minimum Hamming distance between any pair of codewords in the code: $d_{min} = \min_{i,j} D_H(\mathbf{c}_i, \mathbf{c}_j)$

Since $K < L$, the selection of the 2^K codewords among the 2^L possible vectors has to be done using the lowest level of redundancy while maximizing the distance among the codewords. d_{min} is, therefore, a measure of the quality of the code since it is related to both the redundancy and the error correction capability of the code. Basically, it is possible to show that an upper bound between the redundancy and d_{min} is defined by $d_{min} \leq L - K + 1$ and that a code can correct an erroneous word if there are no more than $\lfloor (d_{min} - 1)/2 \rfloor$ erroneous bits. Therefore, L and d_{min} are strictly related, and we can say that increasing the redundancy even the error correction capability of the code increases. Note that d_{min} is also related to the code rate: in particular, the smaller the code rate, the larger the minimum distance.

Since $\mathcal{C}$ is a K-dimensional vector subspace of $GF^L(2)$, there will be K linearly independent vectors $\mathbf{g_0}, \ldots, \mathbf{g_{K-1}}$ that form a basis for $GF^L(2)$. The

[1]Hereafter, without loss of generality and consistently with Coding Theory, we will consider 0 and 1 as bit values for the code and the coding matrix instead of -1 and $+1$ as usually employed in the ECOC literature.

https://doi.org/10.1016/j.ins.2016.02.039

codeword **c** corresponding to the source message **u** can be determined as the linear combination of the basis vectors:

$$\mathbf{c} = u_0\mathbf{g_0} + \ldots + u_{K-1}\mathbf{g_{K-1}}. \tag{3}$$

The linearly independent vectors $\mathbf{g_i}$ can be arranged in a $K \times L$ matrix $\mathbf{G} = \begin{pmatrix} \mathbf{g_0} & \ldots & \mathbf{g_{K-1}} \end{pmatrix}^T$ so that:

$$\mathbf{c} = \mathbf{uG} \tag{4}$$

Definition 3.5 (**Generator Matrix**)**.** A $K \times L$ matrix **G** whose rows form a basis for a linear block code $\mathcal{C}(L, K)$ is called a *generator matrix* of the code $\mathcal{C}$.

It is worth noting that this approach is different from the decomposition methods presented in the previous section where the set of codewords does not necessarily form a vector space and the correspondence between the source message (and thus the class label) and the associated codeword is not based on an algebraic relation.

The structure provided by the linear block code $\mathcal{C}$ can be usefully exploited in the decoding procedure to take a decision on the output word of the system. For this purpose, let us consider the dual vector subspace $\mathcal{C}^*$ associated with the same vector space $GF_L(2)$ and its basis $\mathbf{h_0}, \ldots, \mathbf{h_{L-K-1}}$. $\mathcal{C}^*$ contains the set of vectors belonging to $GF^L(2)$ which are orthogonal to the codewords of $\mathcal{C}$. Thus, collecting the vectors $\mathbf{h_i}$ in an $(L-K) \times L$ matrix $\mathbf{H} = \begin{pmatrix} \mathbf{h_0} & \ldots & \mathbf{h_{L-K-1}} \end{pmatrix}^T$, the following relation holds:

$$\mathbf{HG}^T = \mathbf{0}. \tag{5}$$

Definition 3.6 (**Parity-Check Matrix**)**.** An $(L-K) \times L$ matrix **H** such that a codeword $\mathbf{c} \in \mathcal{C}(L, K)$ if and only if $\mathbf{Hc}^T = \mathbf{0}$ is termed *parity-check matrix* of the code $\mathcal{C}$.

Note that given a generator matrix **G** we can evaluate the associated parity-check matrix **H**, and, conversely, given a parity-check matrix **H**, we can evaluate the associated generator matrix **G**.

The parity-check matrix can also be used to detect and correct errors. Basically, in the decoding stage, when a word **o** is received, it can be seen as a codeword containing some possible errors, i.e., as the sum between a codeword **c** and an error pattern **e**: $\mathbf{o} = \mathbf{c} + \mathbf{e}$. An error can be detected by studying the following condition:

$$\mathbf{s} = \mathbf{Ho}^T = \mathbf{Hc}^T + \mathbf{He}^T = \mathbf{He}^T \neq \mathbf{0} \tag{6}$$

where **s** is an $L-K$-vector called the *syndrome* of **o**. Eq. 6 represents a parity-check condition: if the error pattern is the all-zero vector, then the syndrome is also an all-zero vector, and thus **o** is assumed as a valid codeword. When **s** contains at least one non-zero component, one or more erroneous bits are present in **o** [34]. There is, however, the possibility that **s** is the all-zero vector also in presence of errors. This happens when the error pattern **e** is such that

https://doi.org/10.1016/j.ins.2016.02.039

the vector $\mathbf{c} + \mathbf{e}$ corresponds to another codeword, different from the true one. We will refer to this situation as an *undetectable error*.

4. Low-Density Parity-Check Codes

Introduced by Gallager [19] in 1963, LDPC codes are linear block codes that in Coding Theory provide very high performance by strongly increasing the redundancy. The name "low-density parity-check" indicates that they are characterized by a sparse pseudo-random parity-check matrix $\mathbf{H}$, containing relatively few ones in comparison to the number of zeros. In this way, each parity-check condition involves few bits of the output vector and each bit is contained in few parity-check equations.

Definition 4.1 (**LDPC Code**)**.** A binary LDPC code is a linear code $\mathcal{C}(L, K)$ whose $(L-K) \times L$ parity-check matrix $\mathbf{H}$ is sparse, i.e., the number of ones in $\mathbf{H}$ is much lower than $L(L-K)$.

Aside from the sparsity of the $\mathbf{H}$ matrix, there are two main differences between LDPC and classical block codes. First, LDPC codes are designed by constructing the parity-check matrix and then by evaluating the corresponding generator matrix through eq. 5. Second, classical codes are usually decoded through Maximum Likelihood-based algorithms, and thus they are short and algebraically designed to make this task less complex. LDPC codes, instead, are iteratively decoded taking advantage of the sparsity of $\mathbf{H}$, and thus they are designed by making particular attention to the properties of the parity-check matrix.

To describe an LDPC code two values can be defined: the *weight per row* w_r and the *weight per column* w_c, respectively given by the number of ones in each row and each column of $\mathbf{H}$.

Definition 4.2 (**Regular and Irregular LDPC Code**)**.** An LDPC code is called *regular* if $\mathbf{H}$ contains exactly w_c ones in each column and w_r ones in each row. Otherwise, it is an *irregular* LDPC code.

In the construction of the parity-check matrix of a (w_c, w_r)-regular LDPC code, the following properties have to be satisfied:

1. $w_c > 2$, it has been shown in [19] that this condition ensures a d_{min} linearly increasing with the code length;
2. As the number of ones on the rows must equal the number of ones on the columns, the parameters L, w_c and w_r must satisfy the condition $(L-K)w_r = Lw_c$;
3. Rows and columns should have at most one overlapping position with non-zero values. This is necessary to reduce the possible correlation between different parity-check conditions that could affect the validity of the iterative decoding rules [19];

https://doi.org/10.1016/j.ins.2016.02.039

4. $w_c << L$ and $w_r << L$, i.e., the parameters w_c and w_r should be small compared with the code length to respect the requirement of sparsity of the parity-check matrix.

For an irregular code the weights per row and/or per column are not constant, and thus we can not refer directly to them. We can however consider the numbers c_i and r_i of, respectively, columns and rows with weights w_{c_i} and w_{r_i}. In this way, a similar condition to property 2 can be expressed as:

$$(L-K)\sum_{i=0}^{L-K} w_{r_i} r_i = L \sum_{i=0}^{L} w_{c_i} c_i. \tag{7}$$

This means that, even if the weights are not fixed, it is still possible to relate the average number of ones per row with those per column (often called *degree distributions* of the code).

The original LDPC codes presented in [19] are regular and are constructed by randomly determining the positions of ones in $\mathbf{H}$. Later, several algorithms have been proposed to construct suitable LDPC codes, both regular and irregular, using finite field geometries [3, 26, 47]. In this paper, however, we will refer to the *pseudo-random approach* proposed by MacKay and Neal in [31], where it is proved that randomly-built LDPC codes are very effective with high probability.

A useful graphical translation of the parity-check matrix of an LDPC code is the *Tanner graph* [48], a bipartite graph commonly used to show the connections between the bits of the output word and the parity-check constraints. The graph is bipartite, i.e., it connects two types of nodes: *variable nodes* and *check nodes*. There are L variable nodes, each corresponding to a bit of the output word, and $L-K$ *check nodes* for each of the parity-check constraints in $\mathbf{H}$. An edge joins a variable node to a check node if that bit is included in the corresponding parity-check equation, i.e., a check node i is connected to a variable node j if and only if the entry (i, j) of $\mathbf{H}$ is equal to 1; the number of edges in the Tanner graph is equal to the number of ones in $\mathbf{H}$.

Definition 4.3 (**Cycle**)**.** A *cycle* in a Tanner graph is a sequence of connected vertices which begins and ends at the same vertex in the graph without passing more than once on the same edge.

Definition 4.4 (**Length of a Cycle**)**.** The *length of a cycle* in a Tanner graph is the number of edges it contains.

Definition 4.5 (**Girth**)**.** The *girth* of a Tanner graph is the length of the smallest cycle in the graph.

An example of a Tanner graph for a regular LDPC code and a cycle of length 6 is shown in Fig. 1. The presence of a cycle in a bipartite graph of an LDPC code violates property 3 since the non-zero entries at overlapping positions in $\mathbf{H}$ will be more than one. Ideally, an efficient LDPC code should not contain any cycle in its bipartite graph. The presence of cycles of relatively short lengths is however unavoidable, even if it is often possible to remove the shortest ones (of

https://doi.org/10.1016/j.ins.2016.02.039

$$\mathbf{H} = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

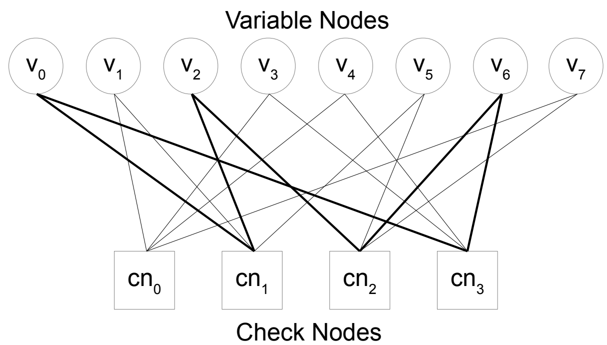


Figure 1: An example of parity-check matrix $\mathbf{H}$ and its corresponding Tanner graph for a regular LDPC code with $w_c = 2$, $w_r = 4$ and $L = 8$. The connections in bold show a cycle of length 6.

length 4 or 6) to ensure a large girth. Cycles, especially short cycles, influence the error correction capability of the LDPC codes and degrade the performance of the iterative decoding rules. This is essentially due to some inconsistencies that can appear during the information exchange between variable nodes and check nodes. For example, the presence of a 4-cycle is equivalent to having two variable nodes connected to the same two check nodes; in this case, in presence of errors, it can happen that the corresponding parity-check conditions are satisfied for different values of the same variable node. Nevertheless, it has been shown that the degrading effect of short cycles diminishes as the code length increases [34].

5. LDPC-based classification system

In this section we show how to build a classification system that takes advantage of the properties of LDPC codes. This goal is accomplished by integrating LDPC codes into a multiclass-to-binary decomposition scheme. To this end, first, we analyze how to find a suitable coding matrix, and second, we propose two decoding techniques that can suitably manage both errors and erasures in the output word.

5.1. Coding

To correctly exploit the properties of LDPC codes, the crucial point during the coding phase is the definition of the code parameters relative to the dimen-

https://doi.org/10.1016/j.ins.2016.02.039

sions (i.e., L and K) and the sparsity of the parity-check matrix (i.e., w_c and w_r).

Ideally, we need to increase the redundancy, and thus the minimum Hamming distance between codewords. Therefore, L has to be the highest possible value, while K the lowest. L can be considered as a free parameter that can be empirically determined striking a balance between the redundancy and the complexity of the system. K is the length of source messages to be encoded, and, in our classification problem, it is lower bounded by the number of classes M: $K \geq \lceil \log_2 M \rceil$. In Coding Theory, high values of K corresponds to high code rates, i.e., to a more efficient and quick transmission of the symbols, even though at the cost of a lower d_{min} and a lower error correction capability. In a classification system, where we are only interested in decreasing the errors, it is convenient, for a fixed L, to keep K as low as possible so as to decrease the code rate and consequently to increase the d_{min} among the codewords. For this reason, we use the lowest possible value for K, that is: $K = \lceil \log_2 M \rceil$.

As for the sparsity of $\mathbf{H}$, it can be managed with only one parameter: for regular codes w_c (or equivalently w_r) can be considered as a free parameter, whereas w_r (w_c) can be evaluated through property 2; for irregular codes, the weights per column w_{c_i} (or equivalently the weights per row w_{r_i}) can be fixed by keeping their average constant, and the weights per row w_{r_i} (the weights per column w_{c_i}) can be evaluated by respecting the constraint in eq. 7.

Once the parameters have been fixed, it is possible to generate the matrices $\mathbf{H}$ and $\mathbf{G}$ as described in Sect. 3. Then, through eq. 4, the codewords $\mathbf{c_i}, \forall i = 0, \ldots, M-1$, can be determined and arranged in the coding matrix $\mathbf{C} = (\mathbf{c_0} \ldots \mathbf{c_{M-1}})^T$. Each column of $\mathbf{C}$ defines a two-class subproblem on which a dichotomizer has to be trained. Note that a code $\mathcal{C}(L, K)$ converts 2^K messages of K bits into 2^K codewords of L bits. To build $\mathbf{C}$, among the 2^K possible codewords we choose the M that maximizes d_{min}.

Another important remark is that, for an LDPC-based system, $\mathbf{C}$ can contain equal columns as well as all-zeros or all-ones columns, whereas this is avoided in a traditional decomposition scheme where the columns define distinct and feasible subproblems, for which distinct dichotomizers are built. Actually, the all-zeros/all-ones columns do not define a dichotomy, and thus they are neglected during the training phase. In the decoding stage, it is however necessary to consider these bits to ensure all the algebraic properties of the code. In particular, such bits must be considered when verifying the parity-check conditions, and therefore, they must be reinserted in the right position of the output vector before the decoding procedure begins. We name these bits as *safe bits* since their values are known a priori in the output word and are intrinsically correct. As for the equal columns, they are assigned the same dichotomizer that provides a *block of bits* in the output word. However, such correlated bits are likely to be forwarded to different parity-check conditions because of the sparsity of the parity-check matrix. In this way, the following decoding procedures are not affected by the correlation among the bits of the same block, and we can both consider high values for L and maintain reasonably low the number D of dichotomizers.

https://doi.org/10.1016/j.ins.2016.02.039

D is equal to the number of different columns in $\mathbf{C}$ that can be evaluated with eq. 4. In particular, by arranging all the source messages in an $M \times K$ matrix $\mathbf{U} = (\mathbf{u_0} \dots \mathbf{u_{M-1}})^T$, we have $\mathbf{C} = \mathbf{UG}$. The columns of $\mathbf{C}$ can be seen as the linear combination of the K M-dimensional columns of $\mathbf{U}$ through every column of $\mathbf{G}$; therefore, the number of M-dimensional different columns in $\mathbf{C}$ is not 2^M, but it is at most equal to 2^K. Excluding the all-zeros column that is always present in $\mathbf{C}$ for the sparsity of $\mathbf{H}$, the number of dichotomizers employed in the proposed approach is at most equal to $2^K - 1$, i.e., $D \leq 2^K - 1 = 2^{\lceil \log_2 M \rceil} - 1 << L$.

As an example, let us consider a multiclass problem with 6 classes. In this case we have $K = \lceil \log_2 6 \rceil = 3$, and thus we can represent our 6 classes with 6 messages of 3 bits $(001, 010, 011, 100, 101, 110)$. If we consider a code $\mathcal{C}(100, 3)$, i.e., $L = 100$, with a redundancy of 97 bits and a code rate equal to 0.03, the matrices $\mathbf{G}$ and $\mathbf{H}$ will have dimensions respectively equal to 3×100 and 97×100. Each codeword will thus be made of 100 bits, $\mathbf{C}$ will be a 6×100 matrix and the number of dichotomizers to be trained will be $D = 2^3 - 1 = 7$.

5.2. Decoding

In a decomposition scheme the decoding algorithm is run on an output word $\mathbf{o} = (o_0, \dots, o_{L-1})$ formed by the set of the L predictions of the dichotomizers on an unknown sample $\mathbf{x}$. To this end, we can employ two models of classifiers: *hard classifiers* where the output is a binary-valued prediction (that can be correct or wrong), and *abstaining classifiers* where a decision can be rejected if its reliability is not sufficient.

This is very similar to what happens in a communication system, where an output word is the result of the transmission of unknown codewords over a channel. The transmitted bits can be contaminated so that the received message can contain *errors* or *erasures*; in the first case, the received bit is wrong, whereas, in the second case, the transmitted bit gets scrambled so that the receiver has no idea what it was. In the same way, we can have errors or rejects depending on the classifier model, hard or abstaining, we are dealing with.

Our objective here is to deal with these situations through two decoding procedures of Coding Theory specifically defined to exploit the features of the LDPC codes described in Sect. 4. These decoding rules are termed *message-passing* algorithms because they are based on iterative procedures where the bits pass forward and backward between the nodes of a Tanner graph, iteratively until a result is achieved. It has been shown that these algorithms guarantee good performance when employed with codes of length less than 10^4 [38].

5.2.1. Decoding algorithm with hard dichotomizers

Let us assume that each dichotomizer $d_j, \forall j = 1, \dots, D$ outputs a real value. In a *hard-decision decoding rule* the output word $\mathbf{o}$ is composed by L binary values, i.e., 0 and 1, coming from the dichotomizers. For this purpose, we

https://doi.org/10.1016/j.ins.2016.02.039

consider that a hard decision on an unknown sample $\mathbf{x}$ is taken by comparing the value $d_j(\mathbf{x})$ with a threshold τ_j, i.e.:

$$o_j(\mathbf{x}) = \begin{cases} 1 & \text{if } d_j(\mathbf{x}) \geq \tau_j \\ 0 & \text{if } d_j(\mathbf{x}) < \tau_j \end{cases}. \quad (8)$$

In a decomposition scheme, we can usually take a decision for $\mathbf{x}$ by choosing for the class corresponding to the minimum Hamming distance between $\mathbf{o}$ and the codewords of $\mathbf{C}$. In our framework, we can apply a more refined and beneficial approach: the *bit-flipping algorithm* [19]. This is an iterative message-passing algorithm based on the assumption that a bit of an output word involved in many incorrect parity-check equations is likely to be incorrect itself. To find the erroneous bits let us focus on the Tanner graph representation. The bit-flipping decoding works by passing bit values between the nodes of the Tanner graph: a variable node sends its bit value to each of the connected check nodes, and each check node replies by determining if its parity-check equation is satisfied or not. As shown in Eq. 6, when the syndrome is the all-zero vector, a valid codeword has been found, and thus the decoding procedure can terminate. When some conditions are not satisfied, we are sure that some errors are present in $\mathbf{o}$, and thus that some dichotomizers took a wrong decision. To correct such errors, for each variable node, we can evaluate the number ε_i as the erroneous checks received in the i-th variable node and flip the bit value of the node involved in the maximum number of erroneous checks, i.e., flip those bits belonging to the set $\{o_i|\varepsilon_i = \max(\varepsilon_0, \ldots, \varepsilon_{L-1})\}$.

In our classification system, the codeword is composed of L bits, but only $D << L$ dichotomizers are used. This means that different variable nodes can contain a bit value coming from the same dichotomizer. For this reason, we heuristically modified the bit-flipping rule, to propose the *block bit-flipping*. In this algorithm, we do not flip only one bit per time, but we flip the value of all the variable nodes hosting the same dichotomizer output. It is worth remembering that the variable nodes contain not only bits coming from dichotomizers but also safe bits (see Sect. 5.1) that are intrinsically correct. Also these bits contribute to the parity-check conditions, but obviously they are not involved in the flipping.

To identify which dichotomizer should be flipped, we do not refer to each variable node independently, but we evaluate the average number of erroneous checks per dichotomizer:

$$E_j = \frac{\sum_{i=1}^{|\mathcal{V}_j|} \varepsilon_{v_i}}{|\mathcal{V}_j|} \quad \forall j = 1, \ldots, D \quad (9)$$

where $\mathcal{V}_j$ is the set of the variable nodes fed by the j-th dichotomizer, $v_i \in \mathcal{V}_j$ is the i-th element of $\mathcal{V}_j$ and the notation $|\cdot|$ indicates the cardinality of a set. The output of the dichotomizer d_j for which E_j is maximum is chosen, and thus all the bits of the corresponding block are flipped. One could argue that the correlation among bits of the same block can affect the capability of the code of detecting and correcting the errors. Actually, this does not threaten

https://doi.org/10.1016/j.ins.2016.02.039

the effectiveness of the proposed decoding rule in recovering the erroneous bits. We have to consider, indeed, that the dichotomizer, whose output has to be flipped, is identified by examining all the check nodes in which the dichotomizer is involved. As a consequence, the bit flipping is made on all the bits coming from the same dichotomizer, but the dichotomizer is chosen on the basis of an accumulation of evidence coming from several check nodes, each individually reporting an error. In other words, the dichotomizer to be flipped is the one with highest evidence to be in error, and this choice makes the decoding rule considerably effective.

A simple step-by-step example of the block bit-flipping algorithm that terminates in just one round is shown in Fig. 2, where three dichotomizers are used to hand out their outputs to the variable nodes of the Tanner graph of Fig. 1.

The algorithm has two stopping conditions: a maximum number of iterations is reached or all the parity-check equations (and thus the syndrome) are equal to zero. The first condition is needed to avoid additional iterations when a solution can not be reached and, moreover, allows us to detect when the algorithm fails to converge to a codeword. In the second case, the algorithm terminates with a solution, but this does not ensure that the output word is a codeword of $\mathbf{C}$. This can happen since we only consider M codewords among all the possible 2^K, and thus the algorithm can fall into an undetectable error (see Sect. 3). In both cases, to find a suitable solution, we can refer to the distance rule (see Sect. 2), and we choose for the codeword that has the minimum Hamming distance to the output word[2].

In Algorithm 1, we describe the block bit-flipping decoding algorithm. The complexity of the decoding procedure for LDPC codes has been deeply analyzed in the literature [56, 48, 7], where it has been shown that the bit-flipping algorithm has a complexity of $O(L \log L)$.

5.2.2. *Decoding algorithm with abstaining dichotomizers*

Let us now consider dichotomizers that abstain from deciding, i.e., dichotomizers that reject a sample instead of risking a wrong decision. When using such classifiers in our decomposition scheme, it is possible to keep out of the output word the unreliable bits (as if they are "erased") and apply an appropriate iterative decoding rule that recovers the rejected bits using the information carried by the reliable dichotomizers.

To this end, let us introduce a reject option for the dichotomizer d_j described by eq. 8. According to such rule, d_j outputs a real value that is compared with a threshold τ_j to assign an unknown sample to one of the two classes. The choice of τ_j is quite critical since, ideally, it has to completely separate the distributions of the dichotomizers scores for the two classes. However, such distributions are usually overlapping, and values close to the threshold are difficult to assign a class, so generating unreliable decisions. In this context, for each dichotomizer,

[2] It is worth noting that the minimum Hamming distance is zero in case the bit-flipping algorithm converges to a codeword belonging to the coding matrix.

https://doi.org/10.1016/j.ins.2016.02.039

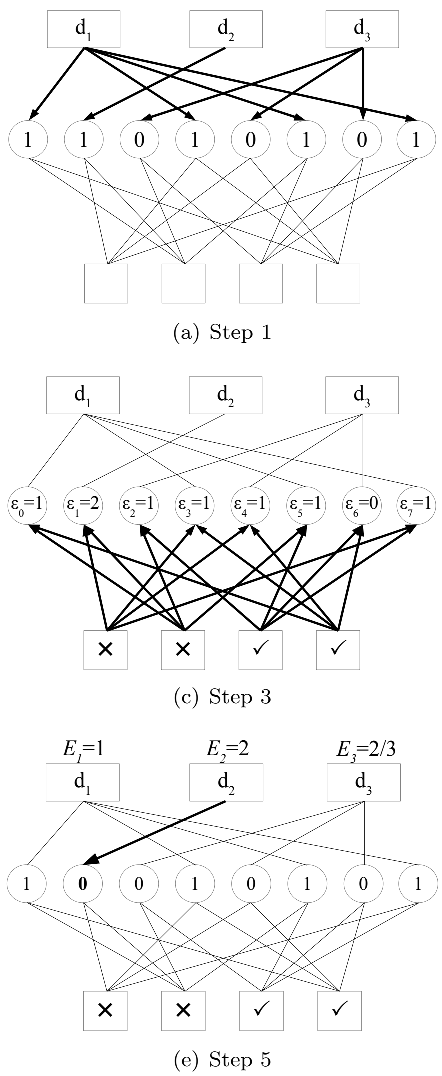


Figure 2: A step-by-step example of the block bit-flipping algorithm: (a) Three dichotomizers d_1, d_2 and d_3 output respectively the bits 1, 1 and 0, and, as $\mathcal{V}_1 = \{0, 3, 5, 7\}$, $\mathcal{V}_2 = \{1\}$, $\mathcal{V}_3 = \{2, 4, 6\}$, we obtain the word $\mathbf{o} = (11010101)$. (b) The values received by the variable nodes are transmitted to the check nodes where the parity-check constraints are verified. (c) The check nodes pass its values back to the variable nodes. (d) The quantities E_j are evaluated according to eq. 9. (e) The dichotomizer d_2 with the maximum E_j flips its decision. (f) The parity-check equations are now verified.

https://doi.org/10.1016/j.ins.2016.02.039

Algorithm 1 Block Bit-Flipping Decoding

Require: The coding matrix $\mathbf{C} = \{\mathbf{c_h}\}$; the parity-check matrix $\mathbf{H}$; the dichotomizers outputs $d_1, \ldots, d_D$ with $d_i \in \{0, 1\}$; a maximum number of iterations N_{max}

for $j = 1, \ldots, D$ **do**
  Evaluate the set $\mathcal{V}_j$ of the variable nodes fed by the j-th dichotomizer
  for each variable node $v_i \in \mathcal{V}_j$ **do**
    Initialize the output word $\mathbf{o}$ with blocks of bits: $o_{v_i} \leftarrow d_j$
  end for
end for
Stop $\leftarrow$ **false** ▷ Initialize a boolean variable
$N_{it} \leftarrow 0$ ▷ Initialize the number of iterations
do
  Evaluate the syndrome vector: $\mathbf{s} \leftarrow \mathbf{H}\mathbf{o}^T$
  if $\mathbf{s} == \mathbf{0}$ **then**
    Stop $\leftarrow$ **true**
  else
    $N_{it} \leftarrow N_{it} + 1$
    for $j = 1, \ldots, D$ **do**
      for each variable node $v_i \in \mathcal{V}_j$ **do**
        Evaluate the erroneous checks ε_{v_i}
      end for
      Evaluate E_j according to Eq. 9
    end for
    Flip the dichotomizer output corresponding to the maximum E_j
    Update the output word $\mathbf{o}$
  end if
while not *Stop* **and** $N_{it} < N_{max}$
Choose for the class $\omega = \arg \min_{0 \le h \le M-1} D_H(\mathbf{c}_h, \mathbf{o})$

it is reasonable to introduce a *safety interval* around τ_j. All the samples corresponding to outcomes falling in this region are rejected. A simple way to find this interval is to employ a decision rule with two thresholds, $\tau_{1,j}$ and $\tau_{2,j}$ with $\tau_{1,j} \le \tau_{2,j}$, such that the j-th bit in the output vector is:

$$o_j(\mathbf{x}) = \begin{cases} 1 & \text{if } d_j(\mathbf{x}) > \tau_{2,j} \\ 0 & \text{if } d_j(\mathbf{x}) < \tau_{1,j} \\ reject & \text{if } \tau_{1,j} \le d_j(\mathbf{x}) \le \tau_{2,j} \end{cases} . \tag{10}$$

The safety interval $[\tau_{1,j}, \tau_{2,j}]$ aims at reducing the number of errors due to the class overlap by turning them into rejects, and its size has to be accurately chosen to meet two contrasting requirements: to be wide enough to eliminate as many errors as possible and to be narrow enough to preserve as many correct

https://doi.org/10.1016/j.ins.2016.02.039

classifications as possible. In our framework, however, we deal with several dichotomizers that usually generate different score distributions, and thus the values of the pair of thresholds can not be equal for all the classifiers. Instead of using equal thresholds, we decided to make all the dichotomizers work at the same level of reliability by imposing a fixed rejection rate ρ to all of them through the method presented in [37]. In such approach, the *Receiver Operating Characteristics* (ROC) curve of each dichotomizer is used to evaluate the pair of thresholds $(\tau_{1,j}, \tau_{2,j})$, so that d_j abstains for no more than ρ samples at the lowest possible error rate.

When the reject rule is turned on, the output word can contain rejected bits, and the decoding procedure presented in the previous section is not directly applicable as the parity-check equations are not completely defined. To solve this problem, supposing that no errors are present among the non-rejected bits, the parity-check condition $\mathbf{H}\mathbf{o}^T = \mathbf{0}$ is a linear system where the rejected bits are the unknown variables. Assuming R as the index set of the rejected bits and R^* as its dual set (i.e., the index set of non-rejected bits) with $|R \cup R^*| = L$, we have:

$$\mathbf{H}\mathbf{o}^T = \mathbf{H}_R\mathbf{o}_R^T + \mathbf{H}_{R^*}\mathbf{o}_{R^*}^T = \mathbf{0} \quad \Rightarrow \quad \mathbf{H}_R\mathbf{o}_R^T = \mathbf{H}_{R^*}\mathbf{o}_{R^*}^T \ . \tag{11}$$

The quantity $\mathbf{H}_{R^*}\mathbf{o}_{R^*}^T$ is a known term, and, when $\mathbf{H}$ has a subset of $|R|$ independent rows (i.e., $rank(\mathbf{H}_R) = |R|$), the system has a unique solution that can be evaluated through Gaussian elimination and back substitution. It is therefore possible to determine the correct values of the unknowns assuming that the received bits are always correct. However, the dichotomizers can introduce errors even when the reject rule is applied. In such a case, to solve Eq. 11 means to individuate the best "possible values" for the rejected bits based on the current information available.

To improve the performance of the decoding system, we should guarantee the correctness of the non-rejected bits. This is ensured when the parity-check conditions, that do not involve any abstaining dichotomizer, are satisfied. Since we know the positions of the rejected bits in the output word, we can easily individuate such equations. Thus, to make our system more robust, when some parity-check conditions are violated, we apply the block bit-flipping algorithm to possibly correct the errors on the non-rejected bits. The block bit-flipping proceeds as described in the previous section, but in Eq. 9 the number of involved dichotomizers is given by the non-abstaining classifiers. Such a number is not a fixed quantity but varies according to the output obtained by each dichotomizer on the sample to be decoded.

When the bit-flipping algorithm has been applied, we can presume that all the non-rejected bits are correct, and the goal of the decoding procedure becomes to determine the value of the unknown bits. To this end, Eq. 11 can be solved by means of an iterative message-passing procedure (referred to as *direct recovery algorithm* [44]) borrowed again from Coding Theory, where it is usually applied when some erasures are present in the output word. This approach is based on the concept that the correct value for a rejected bit can be found by

https://doi.org/10.1016/j.ins.2016.02.039

satisfying the even parity constraint in a parity-check equation that includes all but one known bits. Focusing on the Tanner graph, we can easily analyze how this message-passing algorithm works. A variable node sends its value, i.e., $(0, 1$ or $reject)$, to each of its connected check nodes. If there is only one rejected bit received by a check node, we can evaluate the missing bit by choosing the value that satisfies the parity, i.e., by setting the unknown variable node to the $\bmod 2$ sum of the other variable nodes connected to the same check node. The procedure proceeds iteratively until there are no more check nodes connected to only one rejected bit. This means that either all the rejects have been recovered or there are check nodes connected with two or more variable nodes with rejects which cannot be recovered. The second situation occurs when $rank(\mathbf{H}_R) < |R|$, and thus, the sparser the parity-check matrix (as in LDPC codes), the higher the probability of recovering the rejected bits [41].

When dealing with blocks of bits per dichotomizer we do not recover only one bit per time, but in a single iteration we recover the values of all the variable nodes hosting the same classifier output. In this case, different bits coming from the same dichotomizer can be involved in different parity-check equations with all but one known variables. If there are some errors in the non-rejected bits, the values satisfying the parity constraints can be different, and thus we can have no information about the right value to be back substituted in the variable nodes. To this end, we can evaluate the bit value q_h that should be used to verify the parity-check condition in the h-th check node, and we introduce the quantity Q_j that measures how many zeros and ones are needed in the variable nodes connected to the j-th dichotomizer to verify the parity-check conditions. Assuming $\mathcal{N}_j^r$ as the set of the check nodes involved in parity-check equations with r rejected bits for the j-th dichotomizer and $cn_i \in \mathcal{N}_j^r$ as the i-th element of $\mathcal{N}_j^r$, we have:

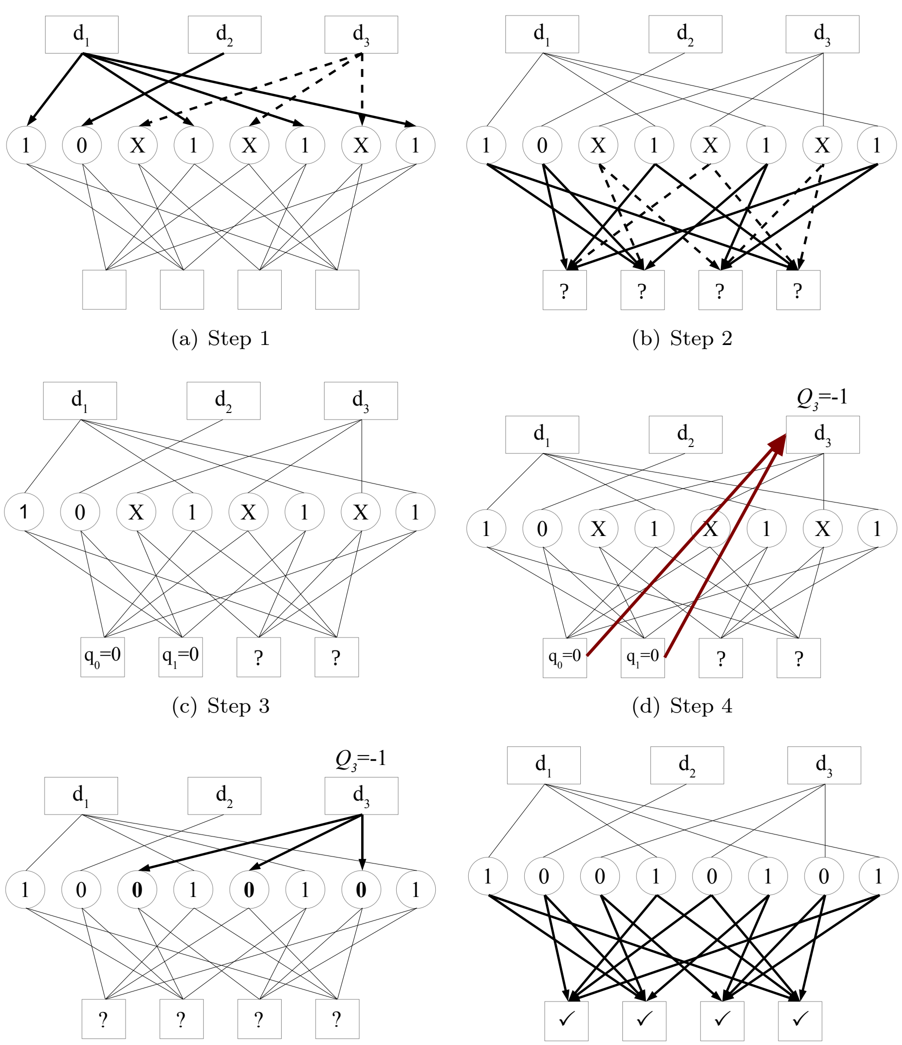
$$Q_j = \frac{\sum_{i=1}^{\left|\mathcal{N}_j^1\right|}\left[q_{cn_i}=1\right] - \sum_{i=1}^{\left|\mathcal{N}_j^1\right|}\left[q_{cn_i}=0\right]}{\left|\mathcal{N}_j^1\right|} \quad \forall j = 1, \ldots, D_{Abst} \tag{12}$$

where D_{Abst} is the number of abstaining dichotomizers and the notation $[\mathcal{P}]$ is the Iverson bracket defined by $[\mathcal{P}] = 1$ if the proposition $\mathcal{P}$ is true, and $[\mathcal{P}] = 0$ if $\mathcal{P}$ is false. The dichotomizer for which the absolute value $|Q_j|$ is maximum is assigned the value 1 if $Q_j > 0$ or the value 0 if $Q_j < 0$. We will refer to this heuristic approach as *block recovery algorithm*. An example of how this method works is shown in Fig. 3 where three dichotomizers (one of which abstains from deciding) are used to hand out their outputs to the variable nodes of the Tanner graph of Fig. 1.

The block recovery algorithm has two stopping conditions:

1. no more rejects are present in the variable nodes, and thus we have recovered the outputs of all the abstaining dichotomizers;
2. $\mathcal{N}^1 = \emptyset$ for all the dichotomizers but $\mathcal{N}^r \neq \emptyset$ (with $r > 1$) for at least one dichotomizer, and thus there are some outputs that can not be recovered (see Fig. 4). Note that, using blocks of bits, this occurs when $rank(\mathbf{H}_R) <$

https://doi.org/10.1016/j.ins.2016.02.039



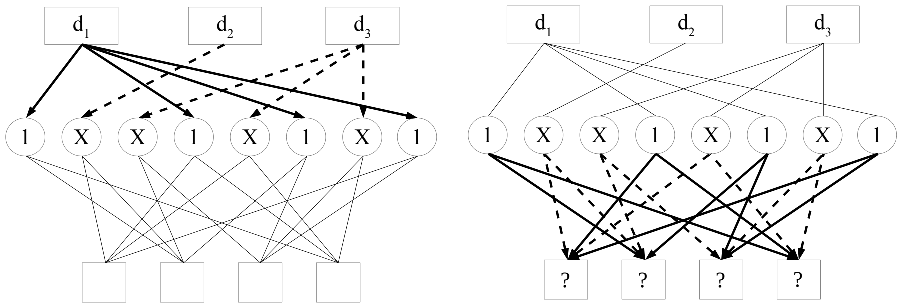
(a) Step 1 (b) Step 2

(c) Step 3 (d) Step 4

(e) Step 5 (f) Step 6

Figure 3: A step-by-step example of the block recovery algorithm: (a) Two dichotomizers d_1 and d_2 output respectively the bits 1 and 0, whereas the dichotomizer d_3 rejects the sample (the reject is here denoted with the symbol X). As $\mathcal{V}_1 = \{0, 3, 5, 7\}, \mathcal{V}_2 = \{1\}, \mathcal{V}_3 = \{2, 4, 6\}$, we obtain the word $\mathbf{o} = (10X1X1X1)$. (b) The values received by the variable nodes are transmitted to the check nodes. (c) Being $\mathcal{N}_3^1 = \{0, 1\}$ and $\mathcal{N}_3^2 = \{2, 3\}$, the quantities q_{cn_i} are evaluated for the check nodes $cn_i \in \mathcal{N}_3^1$. (d) The quantities Q_j are evaluated according to eq. 12. (e) The dichotomizer d_3 with the maximum $|Q_j|$ is assigned the value 0 since $Q_3 < 0$. (f) The parity-check equations are verified.

https://doi.org/10.1016/j.ins.2016.02.039



(a) Step 1 (b) Step 2

Figure 4: A stopping condition for the block recovery algorithm: (a) The dichotomizer d_1 outputs the bit 1, whereas the dichotomizers d_2 and d_3 reject the sample. As $\mathcal{V}_1 = \{0,3,5,7\}, \mathcal{V}_2 = \{1\}, \mathcal{V}_3 = \{2,4,6\}$, we obtain the word $\mathbf{o} = (1XX1X1X1)$. (b) The values received by the variable nodes are transmitted to the check nodes, but the rejects can not be solved through the block recovery algorithm since $\mathcal{N}_2^1 = \mathcal{N}_3^1 = \emptyset$ but $\mathcal{N}_2^2 = \{0,1\}$ and $\mathcal{N}_3^2 = \{0,1,2,3\}$.

D_{Abst}. Since $D_{Abst} \leq |R|$, the block recovery algorithm can solve the rejects more probably than the direct recovery algorithm.

When the second condition is met, we can extend the previous approach with the *guess algorithm* [38] that breaks the stopping condition through several "guesses" of the unsolved rejected bits. In particular, when a dichotomizer feeds several variable nodes, "to guess a bit" means to guess the output of a single classifier. The algorithm is more efficient when the dichotomizer to be guessed is carefully chosen. Basically, our goal is to break the majority of the stopping conditions through a single guess. To this end, we can consider the "crucial" parity-check conditions defined as those equations including only two unknown bits (i.e., those check nodes belonging to $\mathcal{N}^2$). Therefore, if we choose the dichotomizer d_j with $j = \arg\max_{i=1,\dots,D_{Abst}} \left|\mathcal{N}_i^2\right|$ we are solving the highest number of crucial equations, so increasing the probability of recovering all the rejects. When a dichotomizer output is guessed, the block recovery algorithm can be applied again to find the output word. If a new stopping condition is met a second guess is made and so on until all the bits are recovered.

To guess an output means that both values (0 and 1) are considered in two separated decoding processes. Therefore, after g guesses we have a list of 2^g solutions from which we pick up the output word as the $\mathbf{o}_k$ with $k \in \{1,2,\dots,2^g\}$ satisfying all the parity-check conditions $\mathbf{H}\mathbf{o}_k^T = \mathbf{0}$. The complexity of the guess algorithm increases with the number of guesses g that is at most equal to (but usually lower than) D_{Abst}. When using LDPC codes, the complexity is mitigated by the sparsity of $\mathbf{H}$, and, moreover, in [38] it has been demonstrated that the guess algorithm can improve the performance of the decoder when $g \leq 3$, a condition usually respected in our approach.

The algorithm terminates when all the rejects have been recovered. In such a case, however, it is not sure that the decoding algorithm outputs a word

https://doi.org/10.1016/j.ins.2016.02.039

belonging to the coding matrix **C**. Nevertheless, if some erroneous bits are present, it can happen either that the decoding algorithm does not output a real codeword or that we are in the case of an undetectable error. In both cases, an effective rule is to decide for the codeword of **C** that has the minimum Hamming distance from the output word.

A pseudo-code describing the whole decoding procedure (block recovery and guess algorithms) is reported in Algorithm 2. To evaluate the computational complexity of this decoding procedure, we can refer to the analysis made in [55, 30] that estimates in $O(L \log L)$ the complexity of the recovery algorithm. Since we have to consider the extension with the guess algorithm, we obtain a total complexity of $O(2^g L \log L)$.

6. Experiments

To evaluate the performance of the proposed LDPC-based classification system, three different experiments were performed on several datasets publicly available at the UCI Machine Learning Repository [29]. All the employed datasets have numerical input features and a variable number of classes (for more details see Table 1). For each data set, 10 runs of a multiple hold-out procedure were performed to avoid any bias in the comparison. In each run, the data set was split in three subsets: a training, a tuning and a test set containing respectively the 50%, the 30% and the 20% of the samples of each class. The training set was used to train the base classifiers, the tuning set to optimize the dichotomizer parameters and the test set to evaluate the performance of the multiclass classification system.

As base dichotomizer we employed SVM with RBF kernel [24]. The training of SVM-RBF required the tuning of the kernel parameter γ and the regularization parameter C. Such parameters were carefully tuned through an exhaustive grid search in order to find the best pair (γ, C) over a discretization of the parameter space.

6.1. Analyzing the parity-check matrix structure

The goal of the first experiment was to verify how the structure of the parity-check matrix affected the performance of the proposed approach. For this purpose, the characteristics of the **H** matrix of an LDPC coding architecture were analyzed. For each dataset, 50 different **H** matrices were defined and employed in the decoding rule with hard dichotomizers considering both regular and irregular codes and varying the parameters w_c, w_r and L. It is worth noting that the value of K depends only on the number of classes (see Sect. 5.1); its value for each dataset is reported in Table 1. Five values for L were chosen in the range $[50, 250]$ to have a good compromise between the computational complexity of the decoding rule and the redundancy of the code. Five pairs of values were considered for the parameters w_c and w_r to ensure a good sparsity of the parity-check matrix. We varied w_c and w_r in percentage of the length of the code L. For regular codes we considered w_c between the 10% and the

https://doi.org/10.1016/j.ins.2016.02.039

Algorithm 2 Block Recovery and Guess Decoding

Require: The coding matrix $\mathbf{C} = \{\mathbf{c_h}\}$; the parity-check matrix $\mathbf{H}$; the dichotomizers outputs $d_1, \ldots, d_D$ with $d_i \in \{0, 1, reject\}$.

1: **for** $j = 1, \ldots, D$ **do**
2: Evaluate the set $\mathcal{V}_j$ of the variable nodes fed by the j-th dichotomizer
3: **for each** variable node $v_i \in \mathcal{V}_j$ **do**
4: Initialize the output word $\mathbf{o}$ with blocks of bits: $o_{v_i} \leftarrow d_j$
5: **end for**
6: **end for**
7: Apply the Bit-Flipping algorithm to the non-rejected bits
8: $Stop \leftarrow$ **false** ▷ Initialize a boolean variable
9: $g \leftarrow 0$ ▷ Initialize the number of guesses
10: **do**
11: **for each** check node cn_i **do** ▷ Evaluate the syndrome vector $\mathbf{s}$
12: **if** all the variable nodes connected to cn_i are known **then**
13: $s_i \leftarrow \mathbf{H}_i \mathbf{o}^T$
14: **else**
15: $s_i \leftarrow reject$
16: **end if**
17: **end for**
18: **if** no rejects are present in the syndrome vector $\mathbf{s}$ **then**
19: $Stop \leftarrow$ **true**
20: **else**
21: **for each** abstaining dichotomizer d_j^{Abst}, $j = 1, \ldots, D_{Abst}$ **do**
22: Evaluate the sets $\mathcal{N}_j^r$ of check nodes involved with r rejected bits
23: **end for**
24: **if** $\mathcal{N}_j^1 \neq \emptyset$ for at least one d_j^{Abst} **then**
25: **for** $j = 1, \ldots, D_{Abst}$ **do**
26: **for each** $cn_i \in \mathcal{N}_j^1$ **do**
27: Evaluate the bit values q_{cn_i}
28: **end for**
29: Evaluate Q_j according to Eq. 12
30: **end for**
31: $j^* \leftarrow \arg \max_{1 \leq h \leq D_{Abst}} |Q_h|$
32: **if** $Q_{j^*} < 0$ **then** $d_{j^*} \leftarrow 0$ **else** $d_{j^*} \leftarrow 1$ **end if**
33: Update the output word $\mathbf{o}$ with the new output of d_{j^*}
34: **else** ▷ Guess Algorithm
35: $g \leftarrow g + 1$
36: $j^* \leftarrow \arg \max_{1 \leq h \leq D_{Abst}} |\mathcal{N}_h^r|$ for the minimum $r > 1$
37: Generate two words by updating $\mathbf{o}$ with $d_{j^*} \leftarrow 0$ and $d_{j^*} \leftarrow 1$
38: **for each** output word $\mathbf{o}_k$, $k = 1, \ldots, 2^g$ **do**
39: Repeat recursively lines $9 - 42$
40: **end for**
41: **end if**
42: **end if**
43: **while not** $Stop$
44: Choose for the class $\omega = \arg \min_{0 \leq h \leq M-1} D_H(\mathbf{c}_h, \mathbf{o}_k)$ with $k = 1, \ldots, 2^g$

https://doi.org/10.1016/j.ins.2016.02.039

Table 1: Datasets and code parameters used in the experiments.

Datasets	**Classes**	**Features**	**Samples**	**K**
Iris	3	4	150	2
Thyroid	3	5	215	2
Wine	3	13	178	2
Vehicle	4	18	846	2
Dermatology	6	33	366	3
Satimage	6	36	6435	3
Glass	7	9	214	3
Segmentation	7	18	2310	3
Ecoli	8	7	341	3
Optdigits	10	62	5620	4
Pendigits	10	16	10992	4
Yeast	10	8	1484	4
Vowel	11	10	990	4
Letter	26	16	20000	5
Abalone	29	8	4177	5

50% of L whereas w_r was evaluated through the property 2 reported in Sect. 4. For irregular codes, where the weights w_c and w_r are not constant, we used a random number of ones on each column (see Sect. 4), always ensuring that the average $\sum_{i=0}^{L} w_{c_i} c_i$ was between the 10% and the 50% of L. Even the values of w_{r_i} were randomly chosen but always respecting eq. 7.

In each experiment (i.e., for each coding matrix and for each dataset), we evaluated the mean classification error by averaging the error rates obtained on the test set in the 10 runs of the multiple hold-out procedure, for a total of 50 mean error rates for each dataset. Since such results were obtained on different datasets, they were not commensurable, and thus we used a rank-based comparison. Separately for each dataset, we evaluated the *rank* of each coding matrix: the best performing matrix got rank 1 while the worst got the maximum rank (i.e., 50, since we considered 50 different LDPC coding matrices). In case of ties, average ranks were assigned. In this way, if r_k^h was the rank obtained by the h-th coding matrix on the k-th dataset, the average performance of the h-th coding matrix on all the datasets was $R^h = \frac{1}{T}\sum_{k=1}^{T} r_k^h$, where T is the number of datasets considered.

Table 2 reports the results obtained on all the datasets. Generally speaking, we can observe that the best results were attained for relatively high sparsity of the parity-check matrix, i.e., low number of ones per row and column. We can also note that the performance dropped for high values of L. In this case, the bit-flipping procedure did not converge to the right codeword since it had to deal with too many erroneous bits.

https://doi.org/10.1016/j.ins.2016.02.039

Table 2: Results in terms of mean ranks among the various datasets. The lower the value, the better performs the corresponding coding matrix.

LDPC type	**L**	% $\mathbf{w_c/L}$				
		10	**20**	**30**	**40**	**50**
Regular	50	15.47	14.67	11.93	22.07	23.60
	75	12.33	8.53	10.33	19.47	20.93
	100	9.13	5.67	9.20	13.47	16.33
	150	32.53	30.60	30.47	35.47	38.33
	250	40.53	39.33	38.07	41.07	45.67
		% $\mathbf{\sum_{i=0}^{L} w_{c_i} c_i / L}$				
		10	**20**	**30**	**40**	**50**
Irregular	50	12.00	10.07	17.93	16.40	20.27
	75	9.53	12.73	13.27	19.33	19.53
	100	7.93	15.20	12.73	20.60	14.80
	150	29.33	29.20	33.47	35.00	34.00
	250	37.33	36.87	40.53	42.27	40.87

6.2. Evaluating the decoding rules

The second experiment was intended to evaluate how the performance of the LDPC-based decoding rules varies when the dichotomizers are provided with a reject option. For this purpose, the performance obtained with our approach was evaluated in terms of curves reporting the error rate when varying the parameter ρ of the reject rate for the two-class classifiers. It is worth noting that, unlike the well-known error-reject curve, this curve shows the error rate of the whole system with respect to an "internal" reject rate applied on each dichotomizer. As explained in Sect. 5.2, our approach recovers all the outputs coming from dichotomizers that rejected the sample. In this way, the output word is always assigned to a class.

Fig. 5 shows the results of our experiments for the employed datasets. Each plot represents the trend of the test error rate when fixing the percentage ρ of internally rejected samples. We show the error rate averaged on the 10 runs of the multiple hold-out procedure together with symmetric error bars of two standard deviation units in length. The value of ρ was varied in the interval $[0.00, 0.30]$ with a step of 0.025. To have a fair comparison, the parity-check matrix was selected through a *leave-one-dataset-out* approach. As in the previous section, we considered 50 different parity-check matrices, but we evaluated the average ranks on $T - 1 = 14$ datasets. The parameters corresponding to the best performance were then chosen for the experiments on the remaining dataset. From the obtained results (available as supplementary material), we can see that a regular LDPC code with $L = 100$, $w_c = 0.2L$ and $w_r = 0.2\frac{L^2}{L-K}$ was the best choice for all the datasets.

It is worth noting that the curves in Fig. 5 also show the performance obtained with hard dichotomizers, that are represented by the values for $\rho = 0$. We can note that, when ρ grew and the reject option for the dichotomizers was

https://doi.org/10.1016/j.ins.2016.02.039

acting, the performance generally improved. In particular, this happened for 11 datasets. For 3 datasets (i.e., Iris, Thyroid, PenDigits) we obtained at least one point of the curve with the same error rate than the value for $\rho = 0$, and only in one case, i.e., OptDigits, the reject option did not improve the performance of hard dichotomizers. This means that the decoding rules were generally able to turn in correct classifications a significant number of errors previously made by the dichotomizers, and this is even more noticeable when using the reject option.

Looking at the trends of the curves in Fig. 5, we can observe that, in the majority of cases, the lowest error rate corresponds to low values of ρ (around 0.025, 0.05), whereas the error rate is slightly higher (around 0.15) for higher number of classes, e.g., for Letter and Abalone datasets. For these datasets, the score distributions of the dichotomizers were such that we had to enlarge the safety interval in order to include more errors to be possibly corrected by the decoding procedure. After the minimum is reached, the error rate increases with ρ. This behavior can be easily explained: when ρ (and thus the rejects) increases, the bit-flipping applied only to the non-rejected bits (see Sect. 5.2) is less meaningful because it works on fewer parity-check equations; moreover, the number of guesses may also increase and become greater than 3 (see Sect. 5.2), so producing a similar-to-random decision on the rejected bits.

6.3. Comparisons with other decomposition schemes

The last experiment was addressed to a comparison of our method with several approaches in the literature of multiclass-to-binary decomposition.

Nine state-of-the-art approaches were considered:

- *One-vs-All* (OVA) that discriminates one class against the others.
- *One-vs-One* (OVO) that defines as many binary problems as the possible pairs of different classes.
- *Standard ECOC* codes as reported in the seminal paper of Dietterich and Bakiri [10].
- *Dense Random* codes and *Sparse Random* codes, respectively binary and ternary random codes, as presented in [1]. In both cases, we generated $5,000$ different coding matrices, and, through an exhaustive search, we chose the coding matrix that maximized the minimum Hamming distance between both rows and columns.
- *Discriminant ECOC* (DECOC) [40] that constructs the coding matrix through a hierarchical partition of the class space performed with a binary tree.
- *Forest ECOC* [11] where a forest of decision trees is embedded in the ECOC framework. As suggested in [11], a set of 3 trees was considered in these experiments.

https://doi.org/10.1016/j.ins.2016.02.039

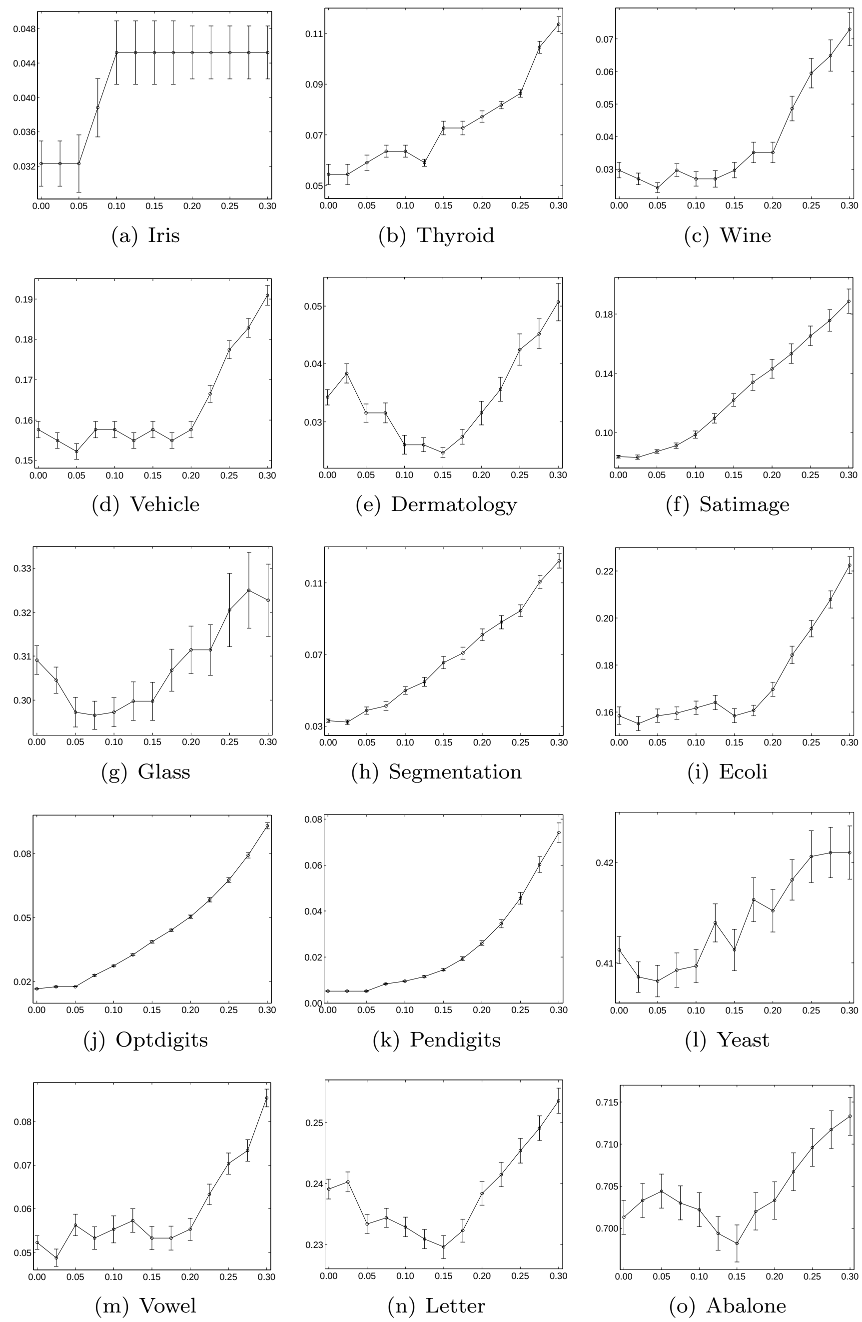


Figure 5: The curves plotting the mean error rate (and the symmetric error bars of two standard deviation units in length) at the output stage towards the rate of rejects in the base classifiers.

https://doi.org/10.1016/j.ins.2016.02.039

- *ECOC Optimizing Node Embedding* (ECOC-ONE) [39], a ternary code where a base coding matrix is built and then incremented by adding dichotomies corresponding to different spatial partitions of classes subsets. To reduce the number of employed dichotomizers, in our experiments we used OVA as base coding matrix.

- *Recursive ECOC* (RECOC) [49], this approach is based on the selection of a "potential good" LDPC code among $10,000$ LDPC codes to be used with an iterative decoding based on the sum-product algorithm [27]. In our experiments, we optimized the number of dichotomizers in the interval $[1, \lceil 3 \cdot log_2 M \rceil]$, and we used 150 iterations in the decoding procedure as suggested in [49].

All the compared approaches were implemented through the ECOCs Library presented in [13] except for RECOC that was re-implemented following [49]. To have a fair comparison, for each decomposition scheme, we employed SVM-RBF as base classifiers with parameters optimized on the tuning set with the previously described technique.

The evaluation of the error rate for the decoding rule with abstaining dichotomizers (hereafter referred to as *LDPC-AD*) required the estimation of the reject parameter ρ. In these experiments, the value of ρ was chosen for each run of the hold-out procedure as the one minimizing the error rate of the corresponding run on the tuning set. As a consequence, the average values of the error rate obtained in these experiments were lower than those shown in Fig. 5. As for the parity-check matrix, we referred to the leave-one-dataset-out technique performed in the previous section.

The performance of the ten considered systems are reported in Table 3. In such table, we also show the performance obtained by the LDPC-based approach when the reject rule is not applied (referred to as *LDPC-HD*). Each cell of the table contains a value relative to the performance of each method on each dataset and corresponding to the error rate averaged on the 10 runs of the multiple hold-out procedure. In the last row of the table, we also report the mean rank of each method averaged on the 15 datasets. The rankings r_k^j for the j-th method were obtained for the k-th dataset by assigning 1 to the best approach and 11 to the worst one. In case of ties, average ranks were assigned. The mean ranking R was computed as $R = \frac{1}{T} \sum_{k=1}^{T} r_k$, where T was the number of considered datasets.

To have a statistical validation of the obtained results, we employed the Friedman test and the Holm step-down test [9]. The Friedman statistic [16] is used as a general test to check if all the compared approaches are equivalent or not. In our case, the null hypothesis referred to a not statistically significant difference among the error rates of the employed methods. When the null hypothesis was rejected, we applied the Holm step-down procedure [23], a post-hoc test that was executed to find out which methods had statistically significant better performance [9]. Both the statistical tests (Friedman and Holm) were performed with a significance level equal to 0.05.

https://doi.org/10.1016/j.ins.2016.02.039

Table 3: Mean error rates per coding design. Statistically significant performance are marked in bold.

Datasets	LDPC-AD	LDPC-HD	OVA	OVO	Standard ECOC	Dense Random	Sparse Random	DECOC	Forest ECOC	ECOC ONE	RECOC
Iris	**0.0194**	0.0323	0.0323	0.0323	0.0487	0.0451	0.0451	0.0258	0.0310	0.0310	0.0261
Thyroid	**0.0364**	0.0545	0.0545	0.0545	0.0425	0.0568	0.0614	0.0500	**0.0340**	0.0425	0.0435
Wine	**0.0135**	0.0297	0.0432	0.0297	0.0297	0.0486	0.0432	0.0270	0.0260	0.0297	0.0250
Vehicle	**0.1501**	0.1576	0.1945	0.1745	0.1728	0.1984	0.2034	0.1566	0.1725	0.1575	0.1591
Dermatology	**0.0205**	0.0342	0.0456	0.0562	0.0356	0.0654	0.0612	0.0297	0.0412	0.0515	0.0301
Satimage	**0.0826**	**0.0836**	0.1074	**0.0846**	0.0903	0.0956	0.0991	0.0971	0.0983	0.0972	0.0943
Glass	**0.2958**	0.3091	0.3977	0.3409	0.3287	0.4342	0.4235	0.3591	0.3621	0.3545	0.3550
Segmentation	**0.0406**	0.0432	0.0487	0.0436	0.0440	0.0542	0.0560	**0.0421**	**0.0412**	0.0435	**0.0396**
Ecoli	**0.1528**	0.1584	0.2067	0.1652	0.1876	0.2225	0.2065	0.2056	0.2035	0.2056	**0.1458**
OptDigits	**0.0144**	**0.0166**	0.0313	0.0261	0.0214	0.0295	0.0315	0.0238	0.0254	0.0245	0.0233
Pendigits	**0.0051**	**0.0052**	**0.0089**	0.0109	**0.0086**	0.0157	0.0202	0.0111	**0.0086**	**0.0057**	0.0175
Yeast	**0.4054**	0.4113	0.4805	0.4335	0.4240	0.4442	0.4456	0.4366	0.4397	0.4429	0.4105
Vowel	**0.0291**	0.0523	0.0658	0.0754	0.0548	0.0614	0.0654	0.0784	0.0682	0.0578	0.0624
Letter	**0.2275**	0.2391	0.2691	0.2355	0.2634	0.2413	0.2467	0.2446	0.2617	0.2400	0.2455
Abalone	**0.6952**	0.7013	0.7912	0.7446	0.8232	0.8452	0.8313	0.7412	0.7453	0.7274	0.7598
Mean Rank	1.20	3.80	8.90	6.20	5.73	9.17	9.87	5.43	5.93	5.30	4.47

Since the null hypothesis of the Friedman test was rejected for all the considered datasets, we show in Table 3 only the results of the Holm test. A bold value in this table indicates that the corresponding method on that dataset had statistically significant higher performance than all the other approaches according to the Holm test. If more than one value on a row is marked in bold, it means that the corresponding methods were equivalent on that dataset, but also that they had statistically significant higher performance than all the other not marked approaches.

The results in Table 3 show a clear superiority of the LDPC-based approaches especially when the reject rule was employed. LDPC-AD had the lowest mean rank (1.20) and was always in the group of the best classifiers. Even LDPC-HD obtained very good performance and had the second lowest mean rank (3.80). More in detail, on 9 datasets LDPC-AD was statistically significantly better than all the other methods, whereas on Optdigits it was the best one together with LDPC-HD. On Satimage also OVO was equivalent to the two proposed rules, while on Segmentation and Pendigits several approaches were equivalent to the LDPC-based rules. In three datasets, the mean error rate of Forest ECOC (on Thyroid) and RECOC (on Segmentation and Ecoli) was lower than the LDPC-based rules, but the differences with LDPC-AD were not statistically significant, whereas they were with LDPC-HD.

Another important remark is about the computational feasibility of the proposed approach. To this end, two measures were considered. The first one was the number of dichotomizers since their training represented the most time-consuming part of the whole training phase; the second measure was the processing time of the decoding procedure, i.e., the time needed to classify an unknown sample. To estimate the decoding time, all the considered methods were implemented in Matlab® and run on a laptop equipped with a CPU Intel Core I7-5500U 2.4 GHz and 16.0 GB of RAM. For the LDPC-based techniques, it is worth remembering that equal columns in the coding matrix were assigned

https://doi.org/10.1016/j.ins.2016.02.039

to the same dichotomizer; in this way, the number of dichotomizers was maintained low ($D \leq 2^K - 1$, see Sect. 5.1) even though the number of total columns, and thus the redundancy of the code, was much larger (i.e., $L = 100$). This was computationally beneficial because less classifiers were processed. At the same time, the sparsity of the parity-check matrix guaranteed an error correction capability linearly increasing with the code length.

The results obtained for the LDPC-based approaches and all the compared strategies are shown in Table 4, where, for each dataset and each coding design, we report the number of dichotomizers in the upper rows and the decoding time in the lower rows. From Table 4 we can see that LDPC-based techniques were competitive in terms of both employed dichotomizers and decoding time, even if they were not the fastest ones. For a low number of classes, OVA and DECOC used a slightly lower number of dichotomizers than LDPC. When M increased (more than 10) even RECOC employed less dichotomizers than our approach. Actually, DECOC and OVA use a number of classifiers linearly proportional to the classes, whereas this number is logarithmically proportional for RECOC. In other words, our approach was still comparable with these three methods in terms of number of dichotomizers and much less demanding than all the other considered approaches. As for the decoding time, DECOC and OVA were again the best approaches. The proposed decoding rules were comparable to RECOC and definitely faster than the other approaches, especially when the number of classes increased. In summary, we can note that LDPC-AD was not much worse than its competitors in terms of number of dichotomizers and decoding time but with a much significantly higher classification performance.

One could have expected a more time-consuming operating phases for the proposed decoding rules; actually, their complexity was kept low by the use of block of bits, as explained in Sect. 5. In practice, the block bit-flipping decoding procedure usually ended in one or two iterations, and, in the worst case, its complexity was limited by the maximum number of iterations N_{max} that depended on the employed dichotomizers. N_{max} was experimentally chosen equal to the half of the involved dichotomizers, i.e, $D/2$ for the bit-flipping rule and $(D - D_{Abst})/2$ for the bit-flipping on the non-rejected bits. As for the block recovery and guess algorithm, its computational load depends on the number of abstaining dichotomizers and thus on the reject rate. However, the value of ρ was estimated on the tuning set by minimizing the corresponding error rate, and in these experiments it was never higher than 0.15. As a consequence, only few dichotomizers abstained from deciding and the number of guesses was always lower than 3 (see Sect. 5.2), thus providing good performance and ensuring a reasonable computational complexity. We can therefore conclude that the proposed method obtained the best recognition performance with an affordable complexity even if, in some cases, it was slightly higher than some other schemes.

7. Conclusions

In this paper, we proposed a new multiclass-to-binary decomposition system founded on Coding Theory and, in particular, on LPDC codes that allowed us

https://doi.org/10.1016/j.ins.2016.02.039

Table 4: Number of dichotomizers (upper rows) and average decoding time in seconds (lower rows) per coding design for each dataset.

Datasets	LDPC-AD	LDPC-HD	OVA	OVO	Standard ECOC	Dense Random	Sparse Random	DECOC	Forest ECOC	ECOC ONE	RECOC
Iris	3	3	3	3	3	16	24	2	6	4	3
	0.0828	0.0786	0.0693	0.0714	0.0713	0.3121	0.4218	0.0437	0.0985	0.0814	0.0771
Thyroid	3	3	3	3	3	16	24	2	6	4	3
	0.0650	0.0668	0.0669	0.0618	0.0662	0.03070	0.5413	0.0509	0.1214	0.1012	0.0713
Wine	3	3	3	3	3	16	24	2	6	4	3
	0.0720	0.0681	0.0664	0.0760	0.0651	0.2861	0.3612	0.0490	0.1416	0.1621	0.0764
Vehicle	3	3	4	6	7	20	30	3	9	6	3
	0.1236	0.1144	0.1268	0.1548	0.1881	0.5961	0.6848	0.1095	0.2715	0.3019	0.1202
Dermatology	7	7	6	15	31	26	39	5	15	9	5
	0.2257	0.1836	0.1343	0.3351	0.6012	0.5817	0.7992	0.1174	0.2941	0.1943	0.1742
Satimage	7	7	6	15	31	26	39	5	15	10	8
	0.2178	0.1692	0.1394	0.3523	0.6314	0.6821	0.8914	0.1198	0.3512	0.2743	0.1867
Glass	7	7	7	21	63	29	43	6	18	9	6
	0.2028	0.1711	0.1435	0.3866	0.6711	0.8064	0.9133	0.1305	0.3943	0.2509	0.1691
Segmentation	7	7	7	21	63	29	43	6	18	11	5
	0.1645	0.1700	0.1720	0.4559	1.1621	0.6169	0.8743	0.1359	0.3716	0.2003	0.1381
Ecoli	7	7	8	28	127	30	45	7	21	15	8
	0.1497	0.1493	0.1852	0.5055	1.8282	0.7413	0.9112	0.1507	0.4013	0.2051	0.1881
Optdigits	14	14	10	45	31	34	50	9	27	22	10
	0.3519	0.3661	0.2359	0.9558	0.6914	0.7251	1.0541	0.2122	0.5814	0.4213	0.3415
Pendigits	14	14	10	45	31	34	50	9	27	23	9
	0.4019	0.3275	0.2003	0.8994	0.5609	0.6079	0.9234	0.1863	0.5545	0.2983	0.2763
Yeast	14	14	10	45	31	34	50	9	27	20	10
	0.3977	0.3550	0.2334	1.4106	0.6616	0.8115	1.6204	0.2603	0.7418	0.4063	0.3272
Vowel	15	15	11	55	63	35	52	10	30	23	10
	0.3316	0.3422	0.2150	1.1763	1.2251	0.6925	1.0816	0.1847	0.5421	0.3267	0.2819
Letter	30	30	26	325	255	48	71	25	75	40	15
	1.2332	1.1345	1.0982	11.1254	9.2435	1.8630	2.4351	1.0603	2.8832	1.4544	1.0921
Abalone	31	31	29	406	511	49	73	28	84	55	15
	0.9293	0.9021	0.8882	9.9102	11.3049	1.5463	1.9921	0.8723	2.2313	1.0912	0.8992

to design both the coding and the decoding procedures through robust theoretical foundations. Based on the algebraic properties of Galois fields theory, LDPC codes are characterized by a sparse parity-check matrix able to generate codewords separated by high Hamming distance. The decoding procedure was also studied and two iterative rules were proposed, namely the block bit-flipping and the block recovery (and guess) algorithms. They exploited the redundancy of the code to algebraically recover both erroneous and unreliable outputs. Our approach provided many advantages over traditional strategies such as OVA, OVO and ECOC solutions. With a limited number of dichotomizers to be trained it ensured a high error correction capability and a feasible computational complexity, as proved by the extensive experiments performed on several benchmark datasets.

Acknowledgments

The authors would like to thank the anonymous reviewers for their very helpful comments and suggestions that improved the quality of the paper.

https://doi.org/10.1016/j.ins.2016.02.039

References

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

[2] E. Alpaydin, E. Mayoraz, Learning error-correcting output codes from data, in: Artificial Neural Networks, 1999. ICANN 99. Ninth International Conference on (Conf. Publ. No. 470), vol. 2, 1999, pp. 743 –748 vol.2.

[3] B. Ammar, B. Honary, Y. Kou, J. Xu, S. Lin, Construction of low-density parity-check codes based on balanced incomplete block designs, Information Theory, IEEE Transactions on 50 (6) (2004) 1257–1269.

[4] M. A. Bagheri, Q. Gao, S. Escalera, A genetic-based subspace analysis method for improving error-correcting output coding, Pattern Recognition 46 (10) (2013) 2830–2839.

[5] M. Á. Bautista, S. Escalera, X. Baró, O. Pujol, On the design of an ecoc-compliant genetic algorithm, Pattern Recognition 47 (2) (2014) 865–884.

[6] M. Á. Bautista, S. Escalera, X. Baró, P. Radeva, J. Vitrià, O. Pujol, Minimal design of error-correcting output codes, Pattern Recognition Letters 33 (6) (2012) 693–702.

[7] D. Burshtein, On the error correction of regular LDPC codes using the flipping algorithm, IEEE Transactions on Information Theory 54 (2) (2008) 517–530.

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

[9] J. Demšar, Statistical comparisons of classifiers over multiple data sets., Journal of Machine Learning Research (7) (2006) 1–30.

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

[11] S. Escalera, O. Pujol, P. Radeva, Boosted landmarks of contextual descriptors and forest-ecoc: A novel framework to detect and classify objects in cluttered scenes, Pattern Recognition Letters 28 (13) (2007) 1759 – 1768.

[12] S. Escalera, O. Pujol, P. Radeva, Separability of ternary codes for sparse designs of error-correcting output codes, Pattern Recognition Letters 30 (3) (2009) 285–297.

[13] S. Escalera, O. Pujol, P. Radeva, Error-correcting ouput codes library, Journal of Machine Learning Research 11 (2010) 661–664.

https://doi.org/10.1016/j.ins.2016.02.039

[14] S. Escalera, O. Pujol, P. Radeva, On the decoding process in ternary error-correcting output codes, IEEE Trans. Pattern Anal. Mach. Intell. 32 (1) (2010) 120–134.

[15] S. Escalera, D. M. J. Tax, O. Pujol, P. Radeva, R. P. W. Duin, Subclass problem-dependent design for error-correcting output codes, IEEE Trans. Pattern Anal. Mach. Intell. 30 (6) (2008) 1041–1054.

[16] M. Friedman, A comparison of alternative tests of significance for the problem of m rankings., Annals of Mathematical Statistics 11 (1940) 86–92.

[17] M. Galar, A. Fernández, E. B. Tartas, H. B. Sola, F. Herrera, An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes, Pattern Recognition 44 (8) (2011) 1761–1776.

[18] M. Galar, A. Fernández, E. B. Tartas, H. B. Sola, F. Herrera, Dynamic classifier selection for one-vs-one strategy: Avoiding non-competent classifiers, Pattern Recognition 46 (12) (2013) 3412–3424.

[19] R. G. Gallager, Low density parity-check codes, MIT press, 1963.

[20] N. García-Pedrajas, C. Fyfe, Evolving output codes for multiclass problems, IEEE Trans. Evolutionary Computation 12 (1) (2008) 93–106.

[21] T. Hastie, R. Tibshirani, Classification by pairwise coupling, in: M. I. Jordan, M. J. Kearns, S. A. Solla (eds.), Advances in Neural Information Processing Systems, vol. 10, The MIT Press, 1998.

[22] N. Hatami, Thinned-ecoc ensemble based on sequential code shrinking, Expert Syst. Appl. 39 (1) (2012) 936–947.

[23] S. Holm, A simple sequentially rejective multiple test procedure., Scandinavian Journal of Statistics 6 (1979) 65–70.

[24] T. Joachims, Making large-scale SVM learning practical, in: B. Schlkopf, C. Burges, A. Smola (eds.), Advances in Kernel Methods - Support Vector Learning, chap. 11, MIT Press, Cambridge, MA, 1999.

[25] A. Klautau, N. Jevtic, A. Orlitsky, On nearest-neighbor error-correcting output codes with application to all-pairs multiclass support vector machines, Journal of Machine Learning Research 4 (2003) 1–15.

[26] Y. Kou, S. Lin, M. Fossorier, Low-density parity-check codes based on finite geometries: a rediscovery and new results, Information Theory, IEEE Transactions on 47 (7) (2001) 2711–2736.

[27] F. R. Kschischang, B. J. Frey, H. A. Loeliger, Factor graphs and the sum-product algorithm, IEEE Trans. Inf. Theor. 47 (2) (2001) 498–519.

https://doi.org/10.1016/j.ins.2016.02.039

[28] L. I. Kuncheva, Using diversity measures for generating error-correcting output codes in classifier ensembles, Pattern Recognition Letters 26 (1) (2005) 83–90.

[29] M. Lichman, UCI Machine Learning Repository (2013).
URL `http://archive.ics.uci.edu/ml`

[30] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, Efficient erasure correcting codes, IEEE Transactions on Information Theory 47 (2) (2001) 569–584.

[31] D. J. C. MacKay, Good error-correcting codes based on very sparse matrices, IEEE Transactions on Information Theory 45 (2) (1999) 399–431.

[32] C. Marrocco, P. Simeone, F. Tortorella, Coding theory tools for improving recognition performance in ECOC systems, in: Z. Zhou, F. Roli, J. Kittler (eds.), Multiple Classifier Systems, vol. 7872 of Lecture Notes in Computer Science, Springer, 2013, pp. 201–211.

[33] C. Marrocco, F. Tortorella, Bit error recovery in ECOC systems through LDPC codes, in: 22nd International Conference on Pattern Recognition, ICPR 2014, Stockholm, Sweden, August 24-28, 2014, 2014, pp. 1454–1459.

[34] J. C. Moreira, P. G. Farrell, Essentials of Error-Control Coding, John Wiley & Sons, Chichester, 2006.

[35] N. J. Nilsson, Learning machines: Foundations of Trainable Pattern Classifying Systems, McGraw-Hill, New York, 1965.

[36] A. Passerini, M. Pontil, P. Frasconi, New results on error correcting output codes of kernel machines, IEEE Transactions on Neural Networks 15 (1) (2004) 45–54.

[37] T. Pietraszek, On the use of ROC analysis for the optimization of abstaining classifiers., Machine Learning 68 (2) (2007) 137–169.

[38] H. Pishro-Nik, F. Fekri, On decoding of low-density parity-check codes over the binary erasure channel, IEEE Trans. Inf. Theor. 50 (3) (2006) 439–454.

[39] O. Pujol, S. Escalera, P. Radeva, An incremental node embedding technique for error correcting output codes, Pattern Recognition 41 (2) (2008) 713 – 725.

[40] O. Pujol, P. Radeva, J. Vitrià, Discriminant ECOC: A heuristic method for application dependent design of error correcting output codes., IEEE Trans. Pattern Anal. Mach. Intell. 28 (6) (2006) 1007–1012.

[41] T. J. Richardson, R. Urbanke, Modern Coding Theory, Cambridge University Press, 2008.

https://doi.org/10.1016/j.ins.2016.02.039

[42] R. Rifkin, A. Klautau, In defense of one-vs-all classification, J. Mach. Learn. Res. 5 (2004) 101–141.

[43] A. Rocha, S. K. Goldenstein, Multiclass from binary: Expanding one-versus-all, one-versus-one and ecoc-based approaches, IEEE Trans. Neural Netw. Learning Syst. 25 (2) (2014) 289–302.

[44] A. Shokrollahi, Theoretical aspects of computer science, chap. An Introduction to Low-density Parity-check Codes, Springer-Verlag New York, Inc., New York, NY, USA, 2002, pp. 175–197.

[45] P. Simeone, C. Marrocco, F. Tortorella, Exploiting system knowledge to improve ECOC reject rules, in: ICPR, IEEE, 2010, pp. 4340–4343.

[46] P. Simeone, C. Marrocco, F. Tortorella, Design of reject rules for ECOC classification systems, Pattern Recognition 45 (2) (2012) 863–875.

[47] H. Tang, J. Xu, Y. Kou, S. Lin, K. Abdel-Ghaffar, On algebraic construction of gallager and circulant low-density parity-check codes, Information Theory, IEEE Transactions on 50 (6) (2004) 1269–1279.

[48] R. M. Tanner, A Recursive Approach to Low Complexity Codes, IEEE Transactions on Information Theory 27 (5) (1981) 533–547.

[49] E. Tapia, P. Bulacio, L. Angelone, Recursive ECOC classification, Pattern Recognition Letters 31 (3) (2010) 210–215.

[50] W. Utschick, W. Weichselberger, Stochastic organization of output codes in multiclass learning problems, Neural Computation 13 (5) (2001) 1065–1102.

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

[52] X. Zhang, Heuristic ternary error-correcting output codes via weight optimization and layered clustering-based approach, IEEE T. Cybernetics 45 (2) (2015) 289–301.

[53] G. Zhong, C. Liu, Error-correcting output codes based ensemble feature extraction, Pattern Recognition 46 (4) (2013) 1091–1100.

[54] J. Zhou, H. Peng, C. Y. Suen, Data-driven decomposition for multi-class classification, Pattern Recogn. 41 (2008) 67–76.

[55] V. V. Zyablov, M. S. Pinsker, Decoding complexity of low-density codes for transmission in a channel with erasures, Problems of Information Transmission 10 (1) (1974) 10–21.

[56] V. V. Zyablov, M. S. Pinsker, Estimation of error-correction complexity of Gallager low-density codes, Problems of Information Transmission 11 (1) (1975) 18–28.

https://doi.org/10.1016/j.ins.2016.02.039