Optimizing subclass Discriminant Error Correcting Output Codes using Particle Swarm Optimization

Dimitrios Bouzas, Nikolaos Arvanitopoulos and Anastasios Tefas

***Abstract*— *Error-Correcting Output Codes (ECOC)* reveal a common way to model multi-class classification problems. According to this state of the art technique, a multi-class problem is decomposed into several binary ones. Additionally, on the ECOC framework we can apply the *subclass technique (sub-ECOC)*, where by splitting the initial classes of the problem we create larger but easier to solve ECOC configurations. The multi-class problem's decomposition is achieved via a discriminant tree creation procedure. This discriminant tree's creation is controlled by a triplet of thresholds that define a set of user defined splitting standards. The selection of the thresholds plays a major role in the classification performance. In our work we show that by optimizing these thresholds via particle swarm optimization we improve significantly the classification performance. Moreover, using Support Vector Machines (SVMs) as classifiers we can optimize in the same time both the thresholds of sub-ECOC and the parameters C and σ of the SVMs, resulting in even better classification performance. Extensive experiments in both real and artificial data illustrate the superiority of the proposed approach in terms of performance.**

I. Introduction

In the literature one can find various binary classification techniques. However, in the real world the problems to be addressed are usually multi-class. In dealing with multi-class problems we must use the binary techniques as a leverage. This can be achieved by defining a method that decomposes the multi-class problem into several binary ones, and combines their solutions to solve the initial multi-class problem [6]. In this context, the *Error-Correcting Output Codes (ECOC)* emerged. Based on the error correcting principles [5] and on its ability to correct the bias and variance errors of the base classifiers [10], this state of the art technique has been proved valuable in solving multi-class classification problems over a number of fields and applications.

As proposed by Escalera et al., on the ECOC framework we can apply the subclass technique [1]. According to this technique, we use a guided problem dependent procedure to group the classes and split them into subsets with respect to the improvement we obtain in the training performance. This splitting is controlled via a set of thresholds. The selection of these thresholds has a major impact in the classification performance of the technique. In this paper we show that by optimizing these set of thresholds with *particle swarm optimization (PSO)* we can boost the classification performance.

In our experiments we used the support vector machine with linear and RBF kernels as a standard classifier [4] and applied it on 8 multi-class learning problems provided by the UCI machine learning repository [22] and 4 artificially created datasets. In parallel with the optimization of the above mentioned thresholds we also optimized with PSO the SVM's parameters C and σ. Thus, we propose the use of PSO as an efficient and fast optimization technique of complex classifiers.

The paper is organized as follows: A brief introduction to the ECOC framework is given in Section II. The sub-ECOC technique is described in Section III. The *loss weighted decoding* technique is illustrated in Section IV. A description of the *sequential forward floating search (SFFS)* algorithm is given in Section V. The *fast quadratic mutual information (FQMI)* procedure used to model the mutual information between classes is analyzed in Section VI. A description of the PSO algorithm and its variations is given in Section VII. The configuration we used in our experiments is described in Section VIII. Our experimental results are illustrated in Section IX. Finally, in Section X we conclude our work.

II. Error Correcting Output Codes

ECOC is a general framework to solve multi-class problems by decomposing them into several binary ones. This technique consists of two separate steps: a) the *encoding* and b) the *decoding* step [2].

a) In the encoding step, given a set of N classes, we assign a unique binary string called *codeword* (i.e., a sequence of bits of a code representing each class, where each bit identifies the membership of the class for a given binary classifier) to each class. The length n of each codeword represents the number of *bi-partitions* (i.e., groups of classes) that are formed and, consequently, the number of binary problems to be trained. Each bit of the codeword represents the response of the corresponding binary classifier and it is coded by +1 or -1, according to its class membership. The next step is to arrange all these codewords as rows of a matrix obtaining the so-called *coding matrix* $\mathbf{M}$, where $\mathbf{M} \in \{-1, +1\}^{N \times n}$. Each column of this matrix defines a partition of classes, while each row defines the membership of the corresponding class in the specific binary problem.
An extension of this standard ECOC approach was proposed by Allwein et al. by adding a third symbol in the coding process [6]. The new coding matrix $\mathbf{M}$ is now $\mathbf{M} \in \{-1, 0, +1\}^{N \times n}$. In this approach, the zero symbol means that a certain class is not considered by a specific binary classifier. As a result, this symbol

The authors are with the Department of Informatics, Aristotle University of Thessaloniki, Box 451, 54124, Thessaloniki, Greece, (phone: +30 2310 991932; email: {dmpouzas, niarvani}@csd.auth.gr, tefas@aiia.csd.auth.gr).

978-1-4244-8126-2/10/$26.00 ©2010 IEEE

increases the number of bi-partitions to be created in the ternary ECOC framework.

b) The decoding step of the ECOC approach consists of applying the n different binary classifiers to each data sample in the test set, in order to obtain a code for this sample. This code is then compared to all the codewords of the classes defined in the coding matrix $\mathbf{M}$ and the sample is assigned to the class with the closest codeword. The most frequently used decoding methods are the *Hamming* and the *Euclidean* decoding distances.

III. SUB-ECOC

Escalera et al. proposed that from an initial set of classes $\mathcal{C}$ of a given multi-class problem, we can define a new set of classes $\mathcal{C}'$, where the cardinality of $\mathcal{C}'$ is greater than that of $\mathcal{C}$, that is $|\mathcal{C}'| > |\mathcal{C}|$ [1]. The new set of binary problems that will be created will improve the created classifiers' training performance. Additionally to the ECOC framework, Pujol proposed that we can use a ternary problem dependent design of ECOC, called *discriminant ECOC (DECOC)* where, given a number of N classes, we can achieve a high classification performance by training only $N-1$ binary classifiers [3]. The combination of the above mentioned methods results in a new classification procedure called *sub-ECOC*. The procedure is based on the creation of discriminant tree structures which depend on the problem domain.

These binary trees are built by choosing the problem partitioning that maximizes the MI between the samples and their respective class labels. The structure as a whole describes the decomposition of the initial multi-class problem into an assembly of smaller binary sub-problems. Each node of the tree represents a pair that consists of a specific binary sub-problem with its respective classifier. The construction of the tree's nodes is achieved through an evaluation procedure [1]. According to this procedure, we can split the bi-partitions that consist the current sub-problem examined. Splitting can be achieved using K-means or some other clustering method. After splitting, we form two new problems that can be examined separately. On each one of the new problems created, we repeat the SFFS procedure independently in order to form two new separate sub-problem domains that are easier to solve. Next, we evaluate the two new problem configurations against three user defined thresholds $\{\theta_{perf}, \theta_{size}, \theta_{impr}\}$ described below. If the thresholds are satisfied, the new created pair of sub-problems is accepted along with their new created binary classifiers, otherwise they are rejected and we keep the initial configuration with its respective binary classifier.

- θ_{perf}: Split the classes if the created classifier attains greater than $\theta_{perf}\%$ training error.
- θ_{size}: Minimum cluster's size of an arbitrary created subclass.
- θ_{impr}: Improvement in the training error attained by classifiers for the newly created problems against previous classifier (before splitting).

IV. LOSS-WEIGHTED DECODING

In the decoding process of the sub-ECOC approach we use the *Loss Weighted Decoding* algorithm [2]. As already mentioned, the 0 symbol in the decoding matrix allows to increase the number of binary problems created and as a result the number of different binary classifiers to be trained. Standard decoding techniques, such as the Euclidean or the Hamming distance do not consider this third symbol and often produce non-robust results. So, in order to solve the problems produced by the standard decoding algorithms, the loss weighted decoding was proposed.

The loss weighted decoding algorithm is summarized in Algorithm 1.

Algorithm 1 Loss Weighted Decoding Algorithm

Calculate Hypothesis matrix H
$H(i,j) = \frac{1}{|J_i|}\sum_{k=1}^{|J_i|}\gamma(h_j(J_i^k), i, j)$
based on

$$\gamma(x_j, i, j) = \begin{cases} 1 & \text{if } x_j = M(i,j) \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Normalize H so that $\sum_{j=1}^{n} M_W(i,j) = 1,\ \forall i = 1, \ldots, N$:

$$M_W(i,j) = \frac{H(i,j)}{\sum_{j=1}^{n} H(i,j)},\ \forall i \in [1, \ldots, N],\ \forall j \in [1, \ldots, n] \quad (2)$$

Given a test input $\wp$, decode based on:

$$d(x,i) = \sum_{j=1}^{n} -M(i,j) \cdot h_j(x) \cdot M_W(i,j) \quad (3)$$

Obtain the final class c_x for sample x by:

$$C_{i^*} : (i^*, j^*) = \arg\min_{(i'j')} d(x, (i'j')), \quad C_{(i'j')} \in C' \quad (4)$$

The main objective is to define a *weighting matrix* $\mathbf{M}_W$ that weights a *loss function* to adjust the decision of the classifiers. In order to obtain the matrix $\mathbf{M}_W$, a *hypothesis matrix* $\mathbf{H}$ is constructed first. The elements $H(i,j)$ of this matrix are continuous values that correspond to the accuracy of the binary classifier h_j classifying the samples of class i. The matrix $\mathbf{H}$ has zero values in the positions which correspond to unconsidered classes, since these positions do not contain any representative information. The next step is the normalization of the rows of matrix $\mathbf{H}$. By this normalization, the matrix $\mathbf{M}_W$ can be considered as a discrete probability density function. This is very important, since we assume that the probability of considering each class for the final classification is the same. Finally, we decode by computing the weighted sum of our coding matrix $\mathbf{M}$ and our binary classifier with the weighting matrix $\mathbf{M}_W$ and assign our test sample to the class that attains the minimum *decoding value*.

V. SEQUENTIAL FORWARD FLOATING SEARCH

The *Floating search methods* are a family of suboptimal sequential search methods that were developed as an alternative counterpart to the more computational costly exhaustive

search methods. These methods allow the search criterion to be *non-monotonic*. They are also able to counteract the nesting effect by considering conditional inclusion and exclusion of features controlled by the value of the criterion itself. In our approach we use a variation of the *Sequential Forward Floating Search (SFFS)* algorithm [8]. The SFFS method is described in Algorithm 2.

Algorithm 2 SFFS for Classes

Input:
$Y = \{y_j | j = 1, \ldots, N_c\}$ // available classes
Output: // disjoint subsets with maximum MI between the features and their class labels
$X_k = \{x_j | j = 1, \ldots, k,\ x_j \in Y\},\ k = 0, 1, \ldots, N_c$
$X'_{k'} = \{x_j | j = 1, \ldots, k',\ x_j \in Y\},\ k' = 0, 1, \ldots, N_c$
Initialization:
$X_0 := \emptyset, X'_{N_c} := Y; \quad k := 0, k' := N_c$ // k and k' denote the number of classes in each subset
Termination:
Stop when $k = N_c$ and $k' = 0$
Step 1 (*Inclusion*)
// x^+ is the most significant class with respect to the group $\{X_k, X'_{k'}\}$
$x^+ := \arg \max_{x \in Y - X_k} J(X_k + x, X'_{k'} - x)$
$X_{k+1} := X_k + x^+; X'_{k'-1} := X'_{k'} - x^+; k := k+1, k' := k'-1$
Step 2 (*Conditional exclusion*)
// x^- is the least significant class with respect to the group $\{X_k, X'_{k'}\}$
$x^- := \arg \max_{x \in X_k} J(X_k - x, X'_{k'} + x)$
if $J(X_k - x^-, X'_{k'} + x^-) > J(X_{k-1}, X'_{k'+1})$ **then**
    $X_{k-1} := X_k - x^-;\ X'_{k'+1} := X'_{k'} + x^-;\ k := k - 1, k' := k' + 1$
    go to **Step 2**
else
    go to **Step 1**
end if

We modified the algorithm so that it can handle criterion functions evaluated using subsets of classes. We apply a number of backward steps after each forward step, as long as the resulting subsets are better than the previously evaluated ones at that level. Consequently, there are no backward steps at all if the performance cannot be improved. Thus, backtracking in this algorithm is controlled dynamically and, as a consequence, no parameter setting is needed.

VI. Fast Quadratic Mutual Information

Consider two random vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ and let $p(\mathbf{x}_1)$ and $p(\mathbf{x}_2)$ be their probability density functions respectively. Then the MI of $\mathbf{x}_1$ and $\mathbf{x}_2$ can be regarded as a measure of the dependence between them and is defined as follows:

$$\mathcal{I}(\mathbf{x}_1, \mathbf{x}_2) = \int\int p(\mathbf{x}_1, \mathbf{x}_2) \log \frac{p(\mathbf{x}_1, \mathbf{x}_2)}{p(\mathbf{x}_1)p(\mathbf{x}_2)} d\mathbf{x}_1 d\mathbf{x}_2 \quad (5)$$

It is of great importance to mention that (5) can be interpreted as a Kullback-Leibler divergence, defined as follows:

$$\mathcal{K}(f_1, f_2) = \int f_1(\mathbf{x}) \log \frac{f_1(\mathbf{x})}{f_2(\mathbf{x})} dx \quad (6)$$

where $f_1(\mathbf{x}) = p(\mathbf{x}_1, \mathbf{x}_2)$ and $f_2(\mathbf{x}) = p(\mathbf{x}_1)p(\mathbf{x}_2)$.

According to Kapur and Kesavan [9], if we seek to find the distribution that maximizes or alternatively minimizes the divergence, several axioms could be relaxed and it can be proven that $\mathcal{K}(f_1, f_2)$ is analogically related to $D(f_1, f_2) = \int (f_1(\mathbf{x}) - f_2(\mathbf{x}))^2 dx$. Consequently, maximization of $\mathcal{K}(f_1, f_2)$ leads to maximization of $D(f_1, f_2)$ and vice versa. Considering the above we can define the *quadratic mutual information* as

$$\mathcal{I}_Q(\mathbf{x}_1, \mathbf{x}_2) = \int\int (p(\mathbf{x}_1, \mathbf{x}_2) - p(\mathbf{x}_1)p(\mathbf{x}_2))^2 d\mathbf{x}_1 d\mathbf{x}_2 \quad (7)$$

The practical implementation of the FQMI computation is defined as follows: Let N be the number of pattern samples in the entire data set, J_i the number of samples of class i, N_c the number of classes in the entire data set, $\mathbf{x}_i$ the ith feature vector of the data set, and $\mathbf{x}_{ij}$ the jth feature vector of the set in class i. Consequently, $p(\mathbf{x})$, $p(y = y_p)$ and $p(\mathbf{x}|y = y_p)$, where $1 \leq p \leq N_c$ can be written as:

$$\begin{aligned} p(\mathbf{x}) &= \frac{1}{N} \sum_{j=1}^{J_p} \mathcal{N}(\mathbf{x} - \mathbf{x}_j, \sigma^2 I), \\ p(y = y_p) &= \frac{J_p}{N}, \\ p(\mathbf{x}|y = y_p) &= \frac{1}{J_p} \sum_{j=1}^{J_p} \mathcal{N}(\mathbf{x} - \mathbf{x}_{pj}, \sigma^2 I). \end{aligned}$$

By the expansion of (7) while using a Parzen estimator with *symmetrical kernel* of width σ, we get the following equation:

$$\mathcal{I}_Q(\mathbf{x}, y) = V_{IN} + V_{ALL} - 2V_{BTW} \quad (8)$$

where

$$\begin{aligned} V_{IN} &= \sum_y \int_{\mathbf{x}} p(\mathbf{x}, y)^2 d\mathbf{x} \\ &= \frac{1}{N^2} \sum_{p=1}^{N_c} \sum_{l=1}^{J_p} \sum_{k=1}^{J_p} \mathcal{N}(\mathbf{x}_{pl} - \mathbf{x}_{pk}, 2\sigma^2 I) \quad (9) \\ V_{ALL} &= \sum_y \int_{\mathbf{x}} p(\mathbf{x})^2 p(y)^2 d\mathbf{x} \\ &= \frac{1}{N^2} \sum_{p=1}^{N_c} \left(\frac{J_p}{N}\right)^2 \sum_{l=1}^{N} \sum_{k=1}^{N} \mathcal{N}(\mathbf{x}_l - \mathbf{x}_k, 2\sigma^2 I) \quad (10) \\ V_{BTW} &= \sum_y \int_{\mathbf{x}} p(\mathbf{x}, y) p(\mathbf{x}) p(y) d\mathbf{x} \\ &= \frac{1}{N^2} \sum_{p=1}^{N_c} \frac{J_p}{N} \sum_{l=1}^{N} \sum_{k=1}^{J_p} \mathcal{N}(\mathbf{x}_l - \mathbf{x}_{pk}, 2\sigma^2 I) \quad (11) \end{aligned}$$

It is known that the FQMI requires many samples to be accurately computed by Parzen window estimation [11]. Thus, we can assume that when the number of samples N is much greater than their respective dimensionality d (i.e. $N >> d$), the complexity of V_{ALL}, which is $O(N_c N^2 d^2)$, is dominant for the equation (8).

VII. Particle Swarm Optimization

The PSO algorithm is a population-based search algorithm whose initial intent was to simulate the unpredictable behavior of a bird flock [12]. From this concept, a simple and efficient optimization algorithm emerged. Individuals in a particle population called *swarm* emulate the success of neighboring individuals and their own successes. A PSO algorithm maintains a swarm of these individuals called *particles*, where each particle represents a potential solution to the optimization problem. The position of each particle is adjusted according to its own experience and that of its neighbors. Let $\mathbf{x}_i(t)$ be the position of particle i in the search space at time step t. The position of the particle is changed by adding a velocity $\mathbf{v}_i(t)$ to the current position. This update can be written as

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \tag{12}$$

with $\mathbf{x}_i(0) \sim U(\mathbf{x}_{min}, \mathbf{x}_{max})$, where U denotes the uniform distribution.

A. Global Best PSO

In our approach, we implement the *global best* PSO algorithm, or *gbest* PSO. The gbest PSO is summarized in Algorithm 3.

Algorithm 3 gbest PSO

```
Create and initialize an n_x-dimensional swarm;
repeat
    for each particle i = 1, ..., n_s do
        // set the personal best position
        if f(x_i) < f(y_i) then
            y_i = x_i;
        end if
        // set the global best position
        if f(y_i) < f(ŷ) then
            ŷ = y_i;
        end if
    end for
    for each particle i = 1, ..., n_s do
        update the velocity using (13)
        update the position using (12)
    end for
until stopping criterion is true
```

In this algorithm, the neighborhood for each particle is the entire swarm. For gbest PSO, the velocity of particle i is calculated as

$$\mathbf{v}_i(t+1) = \mathbf{v}_i(t) + c_1\mathbf{r}_1(t)[\mathbf{y}_i(t) - \mathbf{x}_i(t)] + c_2\mathbf{r}_2(t)[\hat{\mathbf{y}}(t) - \mathbf{x}_i(t)] \tag{13}$$

where $\mathbf{v}_i(t)$ is the velocity of particle i at time step t, $\mathbf{x}_i(t)$ is the position of particle i at time step t, c_1 and c_2 are positive acceleration constants, and $\mathbf{r}_1(t)$, $\mathbf{r}_2(t) \sim U(0,1)$ are random vectors with elements in the range $[0,1]$, sampled from a uniform distribution. These vectors introduce a stochastic element to the algorithm.

From the above equation we can see that the velocity calculation consists of three terms:

- The **previous velocity** $\mathbf{v}_i(t)$ which serves as a memory of the previous flight direction. This memory term can be seen as a momentum that prevents the particle from drastically changing direction.
- The **cognitive component** $c_1\mathbf{r}_1(\mathbf{y}_i - \mathbf{x}_i)$ which quantifies the performance of particle i relative to past performances. The effect of this term is that particles are drawn back to their own best positions.
- The **social component** $c_2\mathbf{r}_2(\hat{\mathbf{y}} - \mathbf{x}_i)$ which quantifies the performance of particle relative to a neighborhood of particles. The effect of the social component is that each particle is also drawn towards the best position found by the particle's neighborhood.

The personal best position $\mathbf{y}_i$ associated with particle i is the best position the particle has visited since the first time step. Considering minimization problems, the personal best position at the next time step $t+1$ is calculated as

$$\mathbf{y}_i(t+1) = \begin{cases} \mathbf{y}_i(t) & \text{if } f(\mathbf{x}_i(t+1)) \geq f(\mathbf{y}_i(t)) \\ \mathbf{x}_i(t+1) & \text{if } f(\mathbf{x}_i(t+1)) < f(\mathbf{y}_i(t)) \end{cases} \tag{14}$$

where $f : \mathbb{R}^{n_x} \rightarrow \mathbb{R}$ is the fitness function. This function measures how close the corresponding solution is to the optimum.

The global best position $\hat{\mathbf{y}}(t)$ at time step t is defined as

$$\hat{\mathbf{y}}(t) \in \{\mathbf{y}_0(t), \ldots, \mathbf{y}_{n_s}(t)\} | f(\hat{\mathbf{y}}(t)) = \min f(\mathbf{y}_0(t)), \ldots, f(\mathbf{y}_{n_s}(t)) \tag{15}$$

where n_s is the total number of particles in the swarm.

B. Velocity Clamping

Using the standard gbest PSO algorithm, we observe that the velocity of the particles quickly explodes to very large values and as a result the swarm diverges from the optimal solution. In order to control this phenomenon we use the so-called *Velocity clamping* in our approach [13]. If a particle's velocity exceeds a specified maximum velocity, this particle's velocity is set to this maximum velocity. Let $V_{max,j}$ denote the maximum velocity allowed in dimension j. Particle velocity is then adjusted before the position update as

$$v_{ij}(t+1) = \begin{cases} v'_{ij}(t+1) & \text{if } v'_{ij}(t+1) < V_{max,j} \\ V_{max,j} & \text{if } v'_{ij}(t+1) \geq V_{max,j} \end{cases} \tag{16}$$

The value of $V_{max,j}$ is very important, since it controls the granularity of the search by clamping escalating velocities. If $V_{max,j}$ is too small, the swarm may not explore sufficiently beyond locally good regions. Furthermore, the optimization process needs more time steps to reach an optimum. It can also be trapped in a local optimum with no means of escape. On the other hand, if $V_{max,j}$ is too large, the swarm may not explore a good region at all. The particles may jump over good optima.

C. Inertia Weight

In our approach we use a modified version of the classical gbest PSO algorithm that integrates an inertia weight [14]. This weight w controls the momentum of the particle by weighing the contribution of the previous velocity. So, the equation of the gbest PSO becomes

$$\mathbf{v}_i(t+1) = w\mathbf{v}_i(t) + c_1\mathbf{r}_1(t)[\mathbf{y}_i(t) - \mathbf{x}_i(t)] + c_2\mathbf{r}_2(t)[\hat{\mathbf{y}}(t) - \mathbf{x}_i(t)] \tag{17}$$

The value of w is extremely important to ensure convergent behavior of the algorithm. For $w \geq 1$, velocities increase over time, accelerating towards the maximum velocity and the swarm diverges. For $w < 1$, particles decelerate until their velocities become zero. It is important to note here the strong relationship between the inertia and the acceleration constants. Van den Bergh and Engelbrecht showed that

$$w > \frac{1}{2}(c_1 + c_2) - 1 \tag{18}$$

guarantees convergent particle trajectories [15], [16]. Many approaches have been proposed for dynamically varying the inertia weight. In our paper we use the following linear decreasing method [17], [18], [19], [20]:

$$w(t) = (w(0) - w(n_t))\frac{n_t - t}{n_t} + w(n_t) \tag{19}$$

where n_t is the maximum number of time steps for which the algorithm is executed, $w(0)$ the initial inertia weight, $w(n_t)$ the final inertia weight, and $w(t)$ the inertia at time step t. Note that $w(0) > w(n_t)$.

D. Acceleration Coefficients

The acceleration coefficients c_1 and c_2, together with the random vectors $\mathbf{r}_1$ and $\mathbf{r}_2$ control the stochastic influence of the cognitive and social components on the overall velocity of the particle. If $c_1 = c_2 = 0$, particles keep flying at their current speed until they hit a boundary of the search space (assuming no inertia). If $c_1 > 0$ and $c_2 = 0$, all particles are independent hill-climbers which perform a local search. On the other hand, if $c_1 = 0$ and $c_2 > 0$, the entire swarm is attracted to a single point $\hat{\mathbf{y}}$. The swarm turns into one stochastic hill-climber.

Usually, c_1 and c_2 are static with their optimized values being found empirically. Wrong initialization of c_1 and c_2 may result in divergent or cyclic behavior [15] [16]. However, in the literature one can find many schemes for adaptive acceleration coefficients. In our paper we use a scheme proposed by Ratnaweera who suggested that c_1 decreases linearly over time, while c_2 increases linearly [18]. This strategy focuses on exploration in the early stages of the optimization, while encouraging convergence near a good optimum near the end of the optimization process by attracting particles towards the neighborhood best position. The values of $c_1(t)$ and $c_2(t)$ are calculated as follows

$$c_1(t) = (c_{1,min} - c_{1,max})\frac{t}{n_t} + c_{1,max} \tag{20}$$

$$c_2(t) = (c_{2,max} - c_{2,min})\frac{t}{n_t} + c_{2,min} \tag{21}$$

where $c_{1,max}, c_{2,max}$ and $c_{1,min}, c_{2,min}$ are user defined.

VIII. Using PSO to optimize the Sub-ECOC technique

As mentioned above, the user defined thresholds of the Sub-ECOC approach and the parameters of the respective SVM classifier are clearly problem-dependent. As a result, due to the fact that we have no a priori knowledge about the structure of the data, we are in no position to efficiently select our parameters. Therefore, the purpose of our paper is to use the PSO algorithm to optimize both these user defined thresholds $\{\theta_{perf}, \theta_{size}, \theta_{impr}\}$ of the sub-ECOC technique and the parameters of the SVM, that is the cost parameter C and, in the case of an RBF SVM, the parameter σ. In this case, the PSO algorithm searches in a 5-dimensional swarm in order to find the optimal solution. In our approach we proceed as follows: We split randomly the datasets into two sets, a training and a test set. The training set contains approximately 60% of the whole data samples, and the test set the remaining 40%. As an objective function f in our PSO algorithm we used the 10-fold cross validation error of our classifier in the training set. That is,

$$f = \frac{1}{10}\sum_{k=1}^{10} err_k \tag{22}$$

where err_k is the error of our classifier in the k-th fold. The respective PSO parameters used were the following:

- Inertia weight: $w(0) = 0.9$, $w(n_t) = 0.4$
- Number of particles: 20
- Number of iterations: 100
- Additional stopping criterion:

$$\frac{1}{p}\sum_{i=1}^{n_s} \|\hat{\mathbf{y}}(t) - \mathbf{x}_i(t)\| < tol \tag{23}$$

  where $tol = 10^{-3}$
- Acceleration coefficients:

$$c_{1,max} = c_{2,max} = 2.5$$

  and

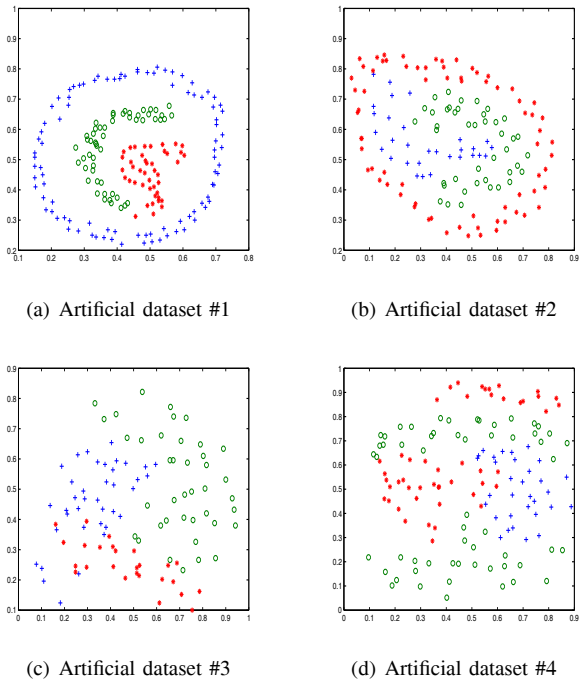
$$c_{1,min} = c_{2,min} = 0.5$$

After the termination of the optimization procedure, we obtain as an output three optimized thresholds $\{\theta_{perf}, \theta_{size}, \theta_{impr}\}$ and also the optimized parameters of the SVM C, and σ in the case of RBF SVM. Finally, we evaluate the optimized classifier in the test set by computing the resulting test error in each dataset. It is worth noting that no information about the test data is used during the parameters optimization. That is, the 10-fold cross validation inside the training set provides all the information needed for optimizing the parameters. We note this because it is common practice for many researchers to report optimized parameters in the test set without proposing a procedure on how someone can find these optimal parameters.

IX. Experimental Results

A. Datasets

We evaluated our experiments using eight datasets of the UCI Machine Learning Repository [22] and four 2D datasets which were artificially created with the svm-toy application of the libsvm package [21]. The features of each of the UCI datasets were scaled to the interval $[-1, +1]$ whereas those of the artificial datasets to $[0, 1]$. The characteristics of each



(a) Artificial dataset #1 (b) Artificial dataset #2

(c) Artificial dataset #3 (d) Artificial dataset #4

Fig. 1. Artificial Datasets.

dataset of the UCI repository can be seen in Table I and the four artificially created datasets are illustrated in Figure 1.

TABLE I
UCI MACHINE LEARNING REPOSITORY DATA SETS CHARACTERISTICS

Database	Samples	Attributes	Classes
Iris	150	4	3
Ecoli	336	8	8
Wine	178	13	3
Glass	214	9	7
Thyroid	215	5	3
Vowel	990	10	11
Balance	625	4	3
Yeast	1484	8	10

B. Default Sub-class ECOC configuration

The PSO resulting splitting parameters were compared with the set of default parameters $\theta = \{\theta_{perf}, \theta_{size}, \theta_{impr}\}$ which were fixed in each dataset to the following values [1]:

- $\theta_{perf} = 0\%$, split the classes if the classifier does not attain zero training error.
- $\theta_{size} = \frac{|J|}{50}$, minimum number of samples in each constructed cluster, where $|J|$ is the number of features in each dataset.
- $\theta_{impr} = 5\%$, the improvement of the newly constructed binary problems after splitting.

Furthermore, as a clustering method we used the K-means algorithm with the number of clusters $K = 2$. As stated by Escalera et al. [1], the K-means algorithm obtains similar results with other more sophisticated clustering algorithms, such as hierarchical and graph cut clustering, but with much less computational cost.

C. Default SVM configuration

As a standard classifier for our experiments we used the libsvm implementation of the Support Vector Machine with linear and RBF kernel. We compared our optimized classifier against the default classifier used in the libsvm package, that is a linear SVM with $C = 1$ and an RBF SVM with $C = 1$ and $\sigma = 1/attributes_{nr}$, where $attributes_{nr}$ is the number of features in each dataset.

D. Results

The resulting classification performances attained in our experiments are shown in Tables II and III. In the case of the sub-ECOC method we also give the (number of rows $\times$ number of columns) of the encoding matrices formed.

TABLE II
UCI REPOSITORY EXPERIMENTS FOR LINEAR AND RBF SVM.

	Linear SVM		**RBF SVM**	
Database	**PSO**	**Default**	**PSO**	**Default**
Iris	**97**% (3 × 4)	96.67% (3 × 4)	**98.3**% (3 × 4)	96.67 % (3 × 4)
Ecoli	**82.07**% (13 × 15)	66.82% (15 × 17)	**82.76**% (14 × 17)	58.62% (13 × 17)
Wine	**98.2**% (3 × 4)	97.14% (3 × 4)	**98.57**% (3 × 4)	97.14% (3 × 4)
Glass	**59.31**% (8 × 10)	58.14% (6 × 5)	**58.14**% (6 × 5)	55.81% (6 × 5)
Thyroid	**95.23**% (3 × 2)	93.65% (3 × 2)	**93.65**% (3 × 2)	84.127% (4 × 4)
Vowel	**49.14**% (27 × 31)	47.1861% (37 × 46)	**59.96**% (11 × 10)	57.58% (15 × 15)
Balance	**93.2**% (21 × 26)	90.7% (59 × 67)	**96.7**% (3 × 2)	94% (3 × 2)
Yeast	**50.3367**% (11 × 10)	40.7407% (11 × 10)	**59.43**% (10 × 9)	37.71% (17 × 21)

From the results, it is obvious that the optimized sub-ECOC using PSO always outperforms the default classifiers in all of the experiments conducted. The improvement is attributed to the fact that PSO finds the optimum values for the thresholds that control the resulting number of subclasses. Furthermore, by finding via PSO optimum values for the SVM parameters (i.e., C and σ), the classification performance is further improved. In certain datasets the thresholds returned by PSO do not result in any subclasses. In this case, PSO reveals that, in the specific dataset, it is highly probable that

TABLE III

ARTIFICIAL DATASETS EXPERIMENTS FOR LINEAR AND RBF SVM.

	Linear SVM		RBF SVM	
Database	**PSO**	**Default**	**PSO**	**Default**
Set # 1	**98.68**%	94.74%	**100**%	59.21 %
	(14×16)	(19×23)	(4×3)	(3×2)
Set # 2	**87.72**%	59.65%	**82.76**%	58.62%
	(50×58)	(48×56)	(14×17)	(13×17)
Set # 3	**70.56**%	64.56%	**82.36**%	77.89%
	(5×4)	(3×2)	(4×3)	(3×2)
Set # 4	**52.49**%	43.89%	**79.86**%	40.28%
	(113×151)	(113×151)	(21×25)	(114×153)

the use of subclasses will lead to over-fitting. We can also see that in the RBF SVM the performance improvement is more significant than in the Linear SVM. This can be associated to the major role the σ parameter plays in the classification performance of the RBF SVM.

X. CONCLUSION

ECOC with subclasses is a power classification technique that takes advantage of the fact that by splitting the classes we can create more complex discriminant surfaces. As mentioned, the splitting is parameter dependent and there's no standard way to choose the right values for these parameters. If the values we choose are very strict, we will create very complex surfaces that will improve the training performance of the classifier, but will probably result in over fitting in the test domain. On the other hand, if we choose very loose values we will not take full advantage of the subclass technique. From the above, it is clear that the splitting parameters need some kind of optimization. As we showed here by applying PSO optimization we can find optimum values for the splitting parameters resulting in much better classification performance.

REFERENCES

[1] Sergio Escalera, David M.J. Tax, Oriol Pujol, Petia Radeva and Robert P.W. Duin, "Subclass Problem-Dependent Design for Error-Correcting Output Codes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, pp. 1041–1054, June 2008.

[2] Sergio Escalera, Oriol Pujol and Petia Radeva, "Loss-Weighted Decoding for Error-Correcting Output Coding" *Proc. Int'l Conf. Computer Vision Theory and Applications*, vol. 2, pp. 117–122, June 2008.

[3] Oriol Pujol, Petia Radeva and Jordi Vitria, "Discriminant ECOC A Heuristic Method for Application Dependent Design of Error Correcting Output Codes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, pp. 1001–1007, June 2006.

[4] Vladimir Vapnik, *Statistical Learning Theory*, Wiley & Sons, 1998.

[5] Thomas G. Dietterich and Ghulum Bakiri, "Solving Multi-class Learning Problems via Error-Correcting Output Codes," *Journal of Machine Learning Research*, vol. 2, pp. 263–282, 1995.

[6] Erin L. Allwein, Robert E. Schapire and Yoram Singer, "Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers," *Journal of Machine Learning Research*, vol. 1, pp. 113–141, 2002.

[7] Kari Torkkola, "Feature Extraction by Non-Parametric Mutual Information Maximization," *Journal of Machine Learning Research*, vol. 3, pp. 1415–1438, March 2003.

[8] P. Pudil, F.J. Ferri, J. Novovicova and J. Kittler, "Floating Search Methods for Feature Selection with Non-monotonic Criterion Functions," *Proc. Int'l Conf. Pattern Recognition*, vol. 3, pp. 279–283, March 1994.

[9] J. Kapur and H. Kesavan, *Entropy Optimization principles with Applications*, Academic Press, 1992.

[10] E.B. Kong and T.G. Dietterich, "Error-Correcting Output Coding Corrects Bias and Variance," *Proc. 12th Intl Conf. Machine Learning* pp. 313–321, 1995.

[11] Richard O. Duda and Peter E. Hart and David G. Stork, *Pattern Classification,* Wiley-Interscience, 2nd Edition, 2000.

[12] J. Kennedy and R.C. Eberhart, "Particle Swarm Optimization," *Proc. IEEE Int'l Joint Conf. Neural Networks*, vol. 3, pp. 279–283, March 1994.

[13] R.C. Eberhart, P.K Simpson and R.W. Dobbins, *Computational Intelligence Tools*, Academic Press Professional, first edition, 1996.

[14] Y.Shi and R.C. Eberhart, "A modified Particle Swarm Optimizer," *Proc. IEEE Congress on Evolutionary Computation*, vol. 3, pp. 279–283, March 1994.

[15] F. van den Bergh, *An analysis of Particle Swarm Optimizers*, PhD thesis, Department of Computer Science, University of Pretoria, South Africa, 2002

[16] F. van den Bergh and A.P. Engelbrecht, "A Study of Particle Swarm Optimization Swarm Trajectories," *Information Sciences*, vol. 3, pp. 937–971, March 1994.

[17] S. Naka, T. Genji, T. Yura and Y. Fukuyama, "Practical Distribution State Estimation using Hybrid Particle Swarm Optimization," *Proc. IEEE PowerEngineering Society Winter Meeting*, vol. 2, pp. 815–820, 2001.

[18] A. Ratnaweera, S. Halgamuge and H. Watson, "Particle Swarm Optimization with Self-Adaptive Acceleration Coefficients," *Proc. First Int'l Conf. Fuzzy Systems and Knowledge Discovery*, pp. 264–268, 2003.

[19] P.N. Suganthan, "Particle Swarm Optimizer with Neighborhood Operator," *Proc. IEEE Congress on Evolutionary Computation*, pp. 1958–1962, 1999.

[20] H. Yoshida, Y. Fukuyama, S. Takayama and Y. Nakanishi, "A Particle Swarm Optimization for reactive Power and Voltage Control in Electric Power Systems Considering Voltage Security Assessment," *Proc. IEEE Int'l Conf. Systems, Man, and Cybernetics*, vol. 6, pp. 497–502, Oct. 1999.

[21] Chih-Chung Chang and Chih-Jen Lin, *LIBSVM: a library for support vector machines*, http://www.csie.ntu.edu.tw/ cjlin/libsvm, 2001.

[22] A. Asuncion and D.J. Newman, *UCI Machine Learning Repository*, University of California, Irvine, School of Information and Computer Sciences, http://www.ics.uci.edu/ mlearn/MLRepository.html, 2007.