Optimizing Linear Discriminant Error Correcting Output Codes Using Particle Swarm Optimization

Dimitrios Bouzas, Nikolaos Arvanitopoulos, and Anastasios Tefas

Department of Informatics, Aristotle University of Thessaloniki
Box 451, 54124 Thessaloniki, Greece
{bouzas,niarvani}@ieee.org
tefas@aiia.csd.auth.gr

Abstract. *Error Correcting Output Codes reveal an efficient strategy in dealing with multi-class classification problems. According to this technique, a multi-class problem is decomposed into several binary ones. On these created sub-problems we apply binary classifiers and then, by combining the acquired solutions, we are able to solve the initial multi-class problem. In this paper we consider the optimization of the Linear Discriminant Error Correcting Output Codes framework using Particle Swarm Optimization. In particular, we apply the Particle Swarm Optimization algorithm in order to optimally select the free parameters that control the split of the initial problem's classes into sub-classes. Moreover, by using the Support Vector Machine as classifier we can additionally apply the Particle Swarm Optimization algorithm to tune its free parameters. Our experimental results show that by applying Particle Swarm Optimization on the Sub-class Linear Discriminant Error Correcting Output Codes framework we get a significant improvement in the classification performance.*

Keywords: Error Correcting Output Codes, Sub-classes, Particle Swarm Optimization, Support Vector Machines

1 Introduction

Many real life classification problems are usually multi-class. However, many powerful classifiers like *Support Vector Machines (SVM)* [4] or *AdaBoost* [14] are capable of solving only binary classification problems. To deal with this problem the *Error Correcting Output Codes (ECOC)* emerged [1]. Due to its ability to correct the bias and variance errors of the base classifiers [5][11], the ECOC procedure has been applied to a wide range of applications, such as face recognition [16], face verification [10] and handwritten digit recognition [17].

On the ECOC framework Pujol et al. proposed that we can use a ternary problem dependent design of ECOC, the so called *Discriminant Error Correcting Output Codes (DECOC)* where, given a number of N_c classes, we can achieve a high classification performance by training only $N_c - 1$ binary classifiers [13].

Escalera et al. extended the DECOC algorithm with the use of *sub-classes* [7]. 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. Recently, in [2] the use of the *Fisher's Linear Discriminant Ratio (FLDR)* was proposed as a problem decomposition criterion in the DECOC framework, resulting in a new algorithm, the so called *Linear Discriminant Error Correcting Output Codes (Linear DECOC)*. The new framework not only improves the classification performance, but it also reduces significantly the running time of the algorithm. This fact enables the promising DECOC with sub-classes approach to be applied efficiently on large scale datasets.

In the DECOC with sub-classes and consequently in the linear DECOC frameworks the split of the initial multi-class problem's classes into sub-classes is controlled by a triplet of thresholds. However, these thresholds are chosen in an arbitrary manner and thus, the risk of overtraining in the resulting classification is high. In this work, we propose the use of the *Particle Swarm Optimization (PSO)* algorithm to optimally select the values of these thresholds. Furthermore, by using as classifier the SVM we can simultaneously apply the PSO algorithm to select the values of the SVM's free parameters. Our experimental results show that by using PSO for tuning the values of the split control thresholds and of the free parameters of the SVM classifier the generalization capabilities of the linear DECOC classification algorithm are significantly increased.

2 Linear Discriminant Error Correcting Output Codes Framework

In this section we give a brief description of the previously mentioned Linear DECOC framework.

As already mentioned, the idea that lies behind the ECOC is to decompose the multi-class problem into several binary ones, and combine the resulting solutions in order to solve the initial problem. We assume that the training dataset contains N data samples $\{\mathbf{x}_i\}_{i=1}^{N} \in \mathbb{R}^d$ that belong to N_c different classes with labels $\{y_i\}_{i=1}^{N} \in \{1, \ldots, N_c\}$. The ECOC procedure consists of the *coding* and the *decoding* step. In the coding step, the goal is to construct L binary classifiers according to which, each class attains a distinct codeword comprised of the outputs of these classifiers on the trained classes. A classifier contributes a zero value to the codeword of a specific class if this class is not considered in the training of the given classifier. Otherwise, a -1 or $+1$ value is contributed to the codeword according to the classifier's training partition this class belongs to. By arranging these codewords as rows of a matrix we obtain the so called *coding matrix* $\mathbf{M} \in \{-1, 0, +1\}^{N_c \times L}$. The number of classifiers L depends strongly on the coding strategy used, that is, the method which controls how to decompose the problem and create the binary sub-problems. In the decoding step, each test sample $\mathbf{x}$ is feeded into the L previously mentioned classifiers and a codeword for this sample is obtained. The resulting codeword is then compared with all

the codewords in the coding matrix **M** and the test sample is assigned to the class that corresponds to the closest codeword according to a decoding measure.

Additionally to the ECOC framework, Pujol et al. proposed that we can use a binary tree design of ECOC, the so called *discriminant ECOC (DECOC)* where, given a number of N_c classes, we can achieve a high classification performance by training only $N_c - 1$ binary classifiers [13]. On the resulting DECOC algorithm 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 $|\mathcal{C}'| > |\mathcal{C}|$ [7]. The new set of binary problems that will be created will improve the created classifiers' training performance.

In both the DECOC and DECOC with sub-classes procedures, the discriminant binary tree structure is built by choosing the partitioning that maximizes the *mutual information (MI)* between the data 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 the *Sequential Forward Floating Search (SFFS)* algorithm [12]. On each step of the DECOC with sub-classes tree creation procedure, we can split the bi-partitions that consist the current sub-problem. The split can be achieved using K-means or some other clustering method. After the split, two new problems are formed that can be examined separately. The assignment of tree nodes to these sub-problems, depends on the satisfaction of three user defined thresholds. If these thresholds are satisfied, two new tree nodes are added to the overall tree structure along with their respective binary classifiers. Otherwise, the split is rejected and a tree node with its respective classifier is created assigned to solve the previously unsplitted problem. The three thresholds are defined as follows:

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

The computation of the MI in each step of the SFFS algorithm has complexity proportional to $\mathcal{O}(d^2N^2)$ where d is the number of dataset's attributes. Hence, the DECOC algorithm can't be applied on large datesets in reasonable running time. In dealing with this problem The use of the FLDR as an alternative optimization criterion in the SFFS algorithm resulting in the so called linear DECOC algorithm has been proposed in [2]. The resulting algorithm not only reduces the computational complexity of the overall procedure but also displays an improvement in the classification performance.

3 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 [9]. In this context,

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 $\mathbf{p}_i(t)$ 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

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

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

In our approach, we implement the *global best* PSO algorithm, or *gbest* PSO. The gbest PSO is summarized in Algorithm 1.

Algorithm 1 gbest PSO

```
Create and initialize an n_p-dimensional swarm;
repeat
    for each particle i = 1, ..., n_s do
        {set the personal best position}
        if f(p_i) < f(y_i) then
            y_i = p_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 (2)
        update the position using (1)
    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

$$\mathbf{v}_i(t+1) = \mathbf{v}_i(t) + c_1\mathbf{r}_1(t)[\mathbf{y}_i(t) - \mathbf{p}_i(t)] + c_2\mathbf{r}_2(t)[\hat{\mathbf{y}}(t) - \mathbf{p}_i(t)] \tag{2}$$

where $\mathbf{v}_i(t)$ is the velocity of particle i at time step t, $\mathbf{p}_i(t)$ is the position of particle i at time step t, c_1 and c_2 are positive *acceleration coefficients*, 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.

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

$$\mathbf{y}_i(t+1) = \begin{cases} \mathbf{y}_i(t) & \text{if } f(\mathbf{p}_i(t+1)) \geq f(\mathbf{y}_i(t)) \\ \mathbf{p}_i(t+1) & \text{if } f(\mathbf{p}_i(t+1)) < f(\mathbf{y}_i(t)) \end{cases} \tag{3}$$

where $f : \mathbb{R}^{n_p} \to \mathbb{R}$ 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

$$\hat{\mathbf{y}}(t) \in \{\mathbf{y}_0(t), \ldots, \mathbf{y}_{n_s}(t)\} | f(\hat{\mathbf{y}}(t)) = \min f(\mathbf{y}_0(t)), \ldots, f(\mathbf{y}_{n_s}(t)) \tag{4}$$

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

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 [6]. That is, we used the classical gbest PSO algorithm that integrates an inertia weight w [15]. 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

$$\mathbf{v}_i(t+1) = w\mathbf{v}_i(t) + c_1\mathbf{r}_1(t)[\mathbf{y}_i(t) - \mathbf{p}_i(t)] + c_2\mathbf{r}_2(t)[\hat{\mathbf{y}}(t) - \mathbf{p}_i(t)] \tag{5}$$

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.

The previously mentioned, user defined thresholds of the linear DECOC with sub-classes approach 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 their values. Therefore, we purpose the use of the PSO algorithm in order to tune the thresholds $\{\theta_{perf}, \theta_{size}, \theta_{impr}\}$. Additionally, if we choose to use SVM as classifier in the linear DECOC technique, we proposed the use of the PSO for selecting simultaneously optimal values for the SVM classifier's free parameters (i.e., the cost parameter C and, in the case of an RBF kernel SVM, the kernel's σ). In this case, the PSO algorithm searches in a 5-dimensional swarm in order to find the optimal solution.

4 Experimental Results

For our experiments we used eight datasets of the UCI Machine Learning Repository [8]. The features of each dataset were scaled to the interval $[-1, +1]$. The characteristics of each dataset can be seen in Table 1.

Table 1. 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

In our experimental configuration we proceeded as follows: We split randomly the datasets into two sets, a training and a test set. The training set contained 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.

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:

$$\frac{1}{p}\sum_{i=1}^{n_s} \|\hat{\mathbf{y}}(t) - \mathbf{x}_i(t)\| < tol \tag{6}$$

  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 obtained as an output three optimized thresholds $\{\theta_{perf}, \theta_{size}, \theta_{impr}\}$ and also the optimized parameters of the SVM, C and in the case of RBF SVM also the width σ of the kernel function. Finally, we evaluated 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 was used during the parameters optimization. That is, the 10-fold cross validation inside the training set provided 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.

The PSO resulting splitting parameters were compared with the set of default parameters $\theta = \{\theta_{perf}, \theta_{size}, \theta_{impr}\}$ which were fixed for each dataset to the values as proposed in [7]:

- $\theta_{perf} = 0\%$, i.e., 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. [7], 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.

As a standard classifier for our experiments we used the LIBSVM [3] implementation of the Support Vector Machine with linear and RBF kernel. We

compared our optimized classifier against the default classifier used in the LIB-SVM 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.

The resulting classification accuracies obtained are shown in Table 2. we also give the (number of rows × number of columns) of the encoding matrices formed as a measure of the split.

Table 2. 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	**84.14**% (13 × 15)	54.48% (15 × 17)	**84.83**% (14 × 17)	22.76% (13 × 17)
Wine	**95.71**% (3 × 4)	94.29% (3 × 4)	**98.57**% (3 × 4)	97.14% (3 × 4)
Glass	**55.81**% (8 × 10)	52.53% (6 × 5)	**61.63**% (6 × 5)	54.65% (6 × 5)
Thyroid	**95.24**% (3 × 2)	92.06% (3 × 2)	**93.65**% (3 × 2)	84.13% (4 × 4)
Vowel	**50.43**% (27 × 31)	46.75% (37 × 46)	**57.14**% (11 × 10)	51.08% (15 × 15)
Balance	**92.4**% (21 × 26)	49.2% (59 × 67)	**98.8**% (3 × 2)	46.8% (3 × 2)
Yeast	**53.20**% (11 × 10)	38.38% (11 × 10)	**57.07**% (10 × 9)	34.68% (17 × 21)

From the results, it is obvious that the optimized linear DECOC 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 sub-classes. 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 the creation of any sub-classes. In this case, PSO reveals that, in the specific dataset, it is highly probable that the use of sub-classes 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.

5 Conclusion

In this paper we used PSO in order to optimize the linear DECOC framework. As was shown, PSO can be a reliable method for choosing optimal values for the free parameters that control the split of the initial multi-class problem's classes into sub-classes resulting in improved classification performance.

References

1. Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing Multi-class to Binary: A Unifying Approach for Margin Classifiers. *Journal of Machine Learning Research*, 1:113–141, 2002.
2. N. Arvanitopoulos, D. Bouzas, and A. Tefas. Subclass Error Correcting Output Codes Using Fisher's Linear Discriminant Ratio. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 2953–2956, 2010.
3. Chih Chung Chang and Chih Jen Lin. LIBSVM: a library for support vector machines, 2001.
4. Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995. 10.1007/BF00994018.
5. Thomas G. Dietterich and Ghulum Bakiri. Solving Multi-class Learning Problems via Error-Correcting Output Codes. *Journal of Machine Learning Research*, 2:263–282, 1995.
6. Russell C. Eberhart and Yuhui Shi. Computational intelligence, 2007.
7. 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*, 30(6):1041–1054, June 2008.
8. A. Frank and A. Asuncion. UCI Machine Learning Repository, 2010.
9. J. Kennedy and R.C. Eberhart. Particle swarm optimization. In *Proceedings of IEEE international conference on neural networks*, volume 4, pages 1942–1948. Perth, Australia, 1995.
10. J. Kittler, R. Ghaderi, T. Windeatt, and J. Matas. Face verification using error correcting output codes. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, 2001.
11. E.B. Kong and T.G. Dietterich. Error-Correcting Output Coding Corrects Bias and Variance. In *Proceedings of the 12th International Conference on Machine Learning*, pages 313–321, 1995.
12. P. Pudil, F.J. Ferri, J. Novovicova, and J. Kittler. Floating Search Methods for Feature Selection with Non-monotonic Criterion Functions. In *Proceedings of the International Conference on Pattern Recognition*, volume 3, pages 279–283, March 1994.
13. 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*, 6:1001–1007, June 2006.
14. Cynthia Rudin, Ingrid Daubechies, and Robert E. Schapire. The Dynamics of Adaboost: Cyclic Behavior and Convergence of Margins. *J. Mach. Learn. Res.*, 5:1557–1595, December 2004.
15. Y. Shi and R.C. Eberhart. Empirical study of particle swarm optimization. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, 1999.
16. T. Windeatt and G. Ardeshir. Boosted ECOC ensembles for face recognition. In *Visual Information Engineering, 2003. VIE 2003. International Conference on*, pages 165–168, 2003.
17. Jie Zhou and Ching Y. Suen. Unconstrained Numeral Pair Recognition Using Enhanced Error Correcting Output Coding: A Holistic Approach. *Document Analysis and Recognition, International Conference on*, 0:484–488, 2005.