ECOCs Library

Sergio Escalera, Oriol Pujol, and Petia Radeva

Computer Vision Center, Campus UAB, Edifici O, 08193, Bellaterra, Barcelona, Spain.
Dept. Matemàtica Aplicada i Anàlisi, UB, Gran Via 585, 08007, Barcelona, Spain.
{*sergio, oriol, petia*}*@maia.ub.edu*

ECOCs Library

This software contains the Matlab code with the implementation of coding and decoding designs of Error-Correcting Output Codes to deal with multi-class categorization problems. Two base classifiers are included with the source files, and a custom function to include your own base classifier is available. A Demo file is included with the software with some learning and testing examples.

How to Reference this Library

We would appreciate if you cite our work when using the ECOCs Library:

S. Escalera, O. Pujol, and Petia Radeva, "ECOCs Library", *Journal of Machine Learning Research*, 2009.

About the software

Version 0.1 corresponds to the first version of the software. Please, contact us for any comments, bugs, or suggestions: **sergio@maia.ub.es**

ECOC function

The main code function is as follows (see execution examples at the end of the document):

[result,confusion,ECOC]=ECOCs(data,TestData,coding,decoding,base,Parameters)

Input: data: $N \times M$ training data matrix, where N is the number of data samples, $M-1$ is the number of features, and the Mth feature corresponds to the label of the samples, requiring at least two different labels, and thus, a minimum of two training examples. If the **data** structure is empty, then, only testing is performed.
Input: TestData: test data with the same structure than **data** and size $N' \times M$, where N' is the number of test samples. If this structure is empty, then, only training is performed.
Input: coding: variable that defines the coding design. In this version of the ECOCs Library, the available coding designs are:

using value of *coding='OneVsOne'* $\Longrightarrow$ one-versus-one.
using value of *coding='OneVsAll'* $\Longrightarrow$ one-versus-all.
using value of *coding='Random'* $\Longrightarrow$ Sparse/Dense random [1] [2][a].
using value of *coding='ECOCONE'* $\Longrightarrow$ ECOC-ONE [3][a].
using value of *coding='DECOC'* $\Longrightarrow$ DECOC [4][a].
using value of *coding='Forest'* $\Longrightarrow$ Forest-ECOC [5][a].

Input: decoding: variable that defines the decoding strategy. In this version of the ECOCs Library software, the decoding designs are those analyzed in [6]:

using value of *decoding='HD'* $\Longrightarrow$ Hamming decoding.
using value of *decoding='ED'* $\Longrightarrow$ Euclidean decoding.
using value of *decoding='LAP'* $\Longrightarrow$ Laplacian decoding.
using value of *decoding='BDEN'* $\Longrightarrow$ β-Density decoding.
using value of *decoding='AED'* $\Longrightarrow$ Attenuated Euclidean decoding.
using value of *decoding='IHD'* $\Longrightarrow$ Inverse Hamming decoding.
using value of *decoding='LLB'* $\Longrightarrow$ Linear Loss-based decoding.
using value of *decoding='ELB'* $\Longrightarrow$ Exponential Loss-based decoding.
using value of *decoding='PD'* $\Longrightarrow$ Probabilistic-based decoding.
using value of *decoding='LLW'* $\Longrightarrow$ Linear Loss-Weighted decoding.
using value of *decoding='ELW'* $\Longrightarrow$ Exponential Loss-Weighted decoding.

Input: base: define the base classifier that learns the ECOC matrix dichotimizers. The classifiers included in this version of the ECOCs Library are:

using value of *base='NMC'* $\Longrightarrow$ Nearest Mean classifier (NMC folder must be included to the Matlab/Octave path).
using value of *base='ADA'* $\Longrightarrow$ Discrete Adaboost classifier (AdaBoostROC folder must be included to the Matlab/Octave path). In this case, we have the parameter **Parameters.ada_iterations** at the beginning of the *ECOCs* function, which defines the number of decision stumps of the boosting procedure (the default value is Parameters.ada_iterations=50).
using value of *base='CUSTOM'* $\Longrightarrow$ custom base classifier[a].

Input: Parameters: structure that contains the parameters that tune each technique. The parameters regarding each design are explained in the next sections. General parameters are:

Input: Parameters.show_info: if this parameter is set to 1, information about the learning and testing process is shown. Other value does not show information (the default value is Parameters.show_info=1).
Input: Parameters.ECOC: this variable contains the initial coding matrix to test a design or to extend the design of the matrix using the ECOC-ONE coding algorithm (the default value is Parameters.ECOC=[]).
Input: Parameters.folder_classifiers: this variable contains the folder where the learnt classifiers are saved. By default, the learnt classifiers are not removed, and they can be found in this folder (the default value is Parameters.folder_classifiers='classifiers').

Output: result: this variable contains the accuracy $\in [0, .., 1]$ of classifying the TestData given the learnt dichotomizers.
Output: confusion: this variable contains the confusion matrix for the TestData samples, where at each position (i, j) it contains the number of samples of class c_i classified as class c_j.
Output: ECOC: this variable contains the ECOC matrix designed in the execution of the main *ECOCs* function.
Output: labels: this variable contains the predicted labels for the test samples.

[a] **See next sections for method details.**

1 Sparse Random designs

The random designs define the coding by selecting the matrix, from a set of randomly-generated matrices, so that it maximizes a row separability measure. This strategy can define binary (dense) or ternary (sparse) random ECOC matrices [1] [2]. In this case the parameters are the followings:

Parameters.columns $\Longrightarrow$ this variable defines the number of columns of the sparse design (the default value is Parameters.columns=10).

Parameters.iterations $\Longrightarrow$ this variable defines the number of random matrices from which the sparse one is selected (the default value is Parameters.iterations=3000).

Parameters.zero_prob $\Longrightarrow$ this variable defines the probability of the symbol zero to appear $\Theta \in [0, .., 1]$, defining matrices with different sparseness degree. To define a "Dense random strategy, this value must be set to zero (the default value is Parameters.zero_prob=0.5).

2 ECOC-ONE

Problem-dependent ECOC coding design. A validation sub-set is used to extend any initial coding matrix and to increase its generalization by including new dichotomizers that focus on difficult to split classes [3]. In this case the parameters are the followings:

Parameters.validation $\Longrightarrow$ this variable defines the percentage of data used as a validation subset $\Theta \in [0, .., 1]$ (the default value is Parameters.validation=0.15).

Parameters.w_validation $\Longrightarrow$ this variable defines the weight of the validation subset $\Theta \in [0, .., 1]$. The weight of the training subset corresponds to (**1-w_validation**) (the default value is Parameters.w_validation=0.5).

Parameters.epsilon $\Longrightarrow$ this variable defines the minimum accuracy of the ensemble to stop the ECOC-ONE coding procedure $\Theta \in [0, .., 1]$ (the default value is Parameters.epsilon=0.05).

Parameters.iterations_one $\Longrightarrow$ this variable defines the maximum number of iterations (classifiers to be embed in the ECOC-ONE coding matrix) (the default value is Parameters.iterations_one=10).

Parameters.ECOC_initial $\Longrightarrow$ this variable defines the input coding matrix to the ECOC-ONE extension algorithm (the default value is Parameters.ECOC_initial='OneVsOne'). In this version of the code, the alternatives are:

using value of *ECOC_initial='OneVsOne'* $\Longrightarrow$ coding "one-versus-one".

using value of *ECOC_initial='OneVsAll'* $\Longrightarrow$ coding "one-versus-all".

using value of *ECOC_initial='Random'* $\Longrightarrow$ coding dense/sparse random designs.

other value $\Longrightarrow$ variable **Parameters.ECOC** (last input variable of *ECOCs* main function) is selected as the input of the ECOC-ONE extension algorithm.

Parameters.one_mode $\Longrightarrow$ given two classes c_1 and c_2 to be considered by a new classifier, this variable defines the procedure to determine the new bi-partition of classes that discriminates c_1 from c_2. In this version of the code, the different alternatives are (the default value is Parameters.one_mode=2):

using value of $Parameters.one\_mode = 1 \Longrightarrow$ if the classes to be discriminated are c_1 and c_2, the new classifier is just c_1 versus c_2 omitting the rest of classes.

using value of $Parameters.one\_mode = 2 \Longrightarrow$ if the classes to be split are c_1 and c_2, an $SFFS$ procedure is applied in order to minimize the distance d, where d is computed as follows:

$$d = \sqrt{\sum_{i \in [1,..,m]} \left(h_i(p_1) - h_i(p_2)\right)^2} \quad (1)$$

being m the number of data features and $h_i(p_1)$ the histogram of the ith feature considering the data of the first partition of classes. Note: by default, the output of h is a 10-vector Probability Density Function. The parameters that governs the $SFFS$ procedure are:

Parameters.iterations_sffs $\Longrightarrow$ number of iterations of the $SFFS$ procedure. The procedure starts with the including step and changes to the removing step at the next iteration as many times as defined by this variable (the default value is Parameters.iterations_sffs=5).

Parameters.steps_sffs $\Longrightarrow$ number of iterations at each including and removing step of the $SFFS$ procedure (the default value is Parameters.steps_sffs=5).

Parameters.criterion_sffs $\Longrightarrow$ measure to be minimized in the $SFFS$ procedure. The criterion included in this version of the code corresponds to eq. 1.

3 DECOC

Problem-dependent ECOC coding design. The classifiers are learnt in the form of binary tree structures using a $SFFS$ criterion and embedded as columns in the coding matrix [4]. The parameters for the $SFFS$ procedure are the followings:

Parameters.iterations_sffs $\Longrightarrow$ number of iterations of the $SFFS$ procedure. The procedure starts with the including step and changes to the removing step at the next iteration as many times as defined by this variable (the default value is Parameters.iterations_sffs=5).

Parameters.steps_sffs $\Longrightarrow$ number of iterations at each including and removing step of the $SFFS$ procedure (the default value is Parameters.steps_sffs=5).

Parameters.criterion_sffs $\Longrightarrow$ measure to be minimized in the $SFFS$ procedure. The criterion included in this version of the code corresponds to eq. 1.

4 Forest-ECOC

Problem-dependent ECOC coding design, where T binary tree structures are embedded in the ECOC matrix. This approach extends the variability of the classifiers of the DECOC design by including extra dichotomizers and avoiding to repeat previously learnt classifiers [5]. In this case the parameters are the followings:

Parameters.number_trees $\Longrightarrow$ Number of binary tree structures to be embedded in the ECOC coding matrix (avoiding the repetition of previous dichotomizers) (the default value is Parameters.number_trees=3).

And the parameters for the $SFFS$ procedure are:

Parameters.iterations_sffs $\Longrightarrow$ number of iterations of the $SFFS$ procedure. The procedure starts with the including step and changes to the removing step at the next iteration as many times as defined by this variable (the default value is Parameters.iterations_sffs=5).

Parameters.steps_sffs $\Longrightarrow$ number of iterations at each including and removing step of the $SFFS$ procedure (the default value is Parameters.steps_sffs=5).

Parameters.criterion_sffs $\Longrightarrow$ measure to be minimized in the $SFFS$ procedure. The criterion included in this version of the code corresponds to eq. 1.

5 Custom base classifier

Using value of *base='CUSTOM'* $\Longrightarrow$ custom base classifier:

Two functions are available in order to include your own base classifier in the ECOC designs:

function classifier=Custom_base(FirstSet,SecondSet)

You have to include your training code so that you learn the problem FirstSet-versus-SecondSet, where FirstSet and SecondSet are in the form of **data** without including the labels. The output **classifier** $\{-1, +1\}$ will be saved to be used at the decoding step.

function label=Custom_test(data,classifier)

You have to include your testing code so that you predict a label for an input data (in the form of **data** without the labels) given the input classifier learnt at the previous Custom_base function. The prediction of the classifier may be a confidence value. In the case that the decoding strategies work with margin, the continuous value will be used, otherwise, the sign of the margin is selected as the label +1 or -1.

6 Examples

To show a separately training and testing example, we selected the Glass data set from the UCI repository data set [7]. This data set contains six classes. The samples are combined in a **data** structure of size 214×10, corresponding to 214 samples for six different classes (labels 1 to 6), and a 9-dimensional space (10th feature corresponds to the label). To include the classifiers and coding designs, the folders Codings, AdaboostROC, NMC, etc. must be included to the Matlab/Octave path.

In this case, first we train the classifiers for this data using a one-versus-one ECOC design with Hamming decoding:

[result,confusion,ECOC]=ECOCs(data,[],'OneVsOne','HD','ADA')

The test data is empty in order to only train the ECOC design, 2nd and 3rd parameters correspond to the one-versus-one coding and Hamming decoding, and the next parameter corresponds to the Discrete Adaboost base classifier. The classifiers are saved in the 'classifiers' folder (defined by default in **Parameters.folder_classifiers**). The output is shown in Table 1.

The system learns the classifiers and returns empty variables since we did not test any data. Now, we can test the system by setting the **Parameters.ECOC** input variable to the previous one-versus-one ECOC matrix, and call with the testing parameters (in order to simplify the example, we tested with the same training data). The result is shown in Table 2.

In the previous case the training data is empty, and we obtain the classification results in terms of performance, the confusion matrix, the ECOC matrix, and the predicted labels.

Now, we train and test in the same run with the Glass data and the previous parameters but using a DECOC codification. In this case, we include both training and test data (to simplify we use the training data to test), the coding design is set to 'DECOC', and we fixed the following parameters: **Parameters.iterations_sffs**=5 and **Parameters.criterion_sffs**=1. The execution results are shown in Tables 3 and 4, respectively.

Run the file *Demo.m* provided with the code to look for more examples.

```
>> [result,confusion,ECOC,labels]=ECOCs(data,[],'OneVsOne','HD','ADA')
Learning coding matrix :
ECOC =
  1   1   1   1   1   0   0   0   0   0   0   0   0   0   0
 -1   0   0   0   0   1   1   1   1   0   0   0   0   0   0
  0  -1   0   0   0  -1   0   0   0   1   1   1   0   0   0
  0   0  -1   0   0   0  -1   0   0  -1   0   0   1   1   0
  0   0   0  -1   0   0   0  -1   0   0  -1   0  -1   0   1
  0   0   0   0  -1   0   0   0  -1   0   0  -1   0  -1  -1
result = []
confusion = []
ECOC =
  1   1   1   1   1   0   0   0   0   0   0   0   0   0   0
 -1   0   0   0   0   1   1   1   1   0   0   0   0   0   0
  0  -1   0   0   0  -1   0   0   0   1   1   1   0   0   0
  0   0  -1   0   0   0  -1   0   0  -1   0   0   1   1   0
  0   0   0  -1   0   0   0  -1   0   0  -1   0  -1   0   1
  0   0   0   0  -1   0   0   0  -1   0   0  -1   0  -1  -1
labels =[]
```

Table 1
Example 1.

References

[1] S. Escalera, O. Pujol, P. Radeva, Separability of ternary codes for sparse designs of error-correcting output codes, in: Pattern Recognition Letters, 2008.

[2] E. Allwein, R. Schapire, Y. Singer, Reducing multiclass to binary: A unifying approach for margin classifiers, in: JMLR, Vol. 1, 2002, pp. 113–141.

[3] O. Pujol, S. Escalera, P. Radeva, An incremental node embedding technique for error-correcting output codes, in: Pattern Recognition, Vol. 4, 2008, pp. 713–725.

[4] O. Pujol, P. Radeva, , J. Vitrià, Discriminant ECOC: A heuristic method for application dependent design of error correcting output codes, in: PAMI, Vol. 28, 2006, pp. 1001–1007.

[5] S. Escalera, O. Pujol, P. Radeva, Boosted landmarks of contextual descriptors and Forest-ECOC: A novel framework to detect and classify objects in clutter scenes, in: Pattern Recognition Letters, Vol. 28(13), 2007, pp. 1759–1768.

[6] S. Escalera, O. Pujol, P. Radeva, On the decoding process in ternary error-correcting output codes, in: IEEE Transactions in Pattern Analysis and Machine Intelligence, 2008.

[7] A. Asuncion, D. Newman, UCI machine learning repository, in: University of California, Irvine, School of Information and Computer Sciences, 2007.
URL `http://mlearn.ics.uci.edu/MLRepository.html`

```
>> Parameters.ECOC=ECOC;
>> [result,confusion,ECOC]=ECOCs([],data,'OneVsOne','HD','ADA',Parameters)
Testing ECOC design
result =
0.9860
confusion =
    68     2     0     0     0     0
     1    75     0     0     0     0
     0     0    17     0     0     0
     0     0     0    13     0     0
     0     0     0     0     9     0
     0     0     0     0     0    29

ECOC =

     1     1     1     1     1     0     0     0     0     0     0     0     0     0     0
    -1     0     0     0     0     1     1     1     1     0     0     0     0     0     0
     0    -1     0     0     0    -1     0     0     0     1     1     1     0     0     0
     0     0    -1     0     0     0    -1     0     0    -1     0     0     1     1     0
     0     0     0    -1     0     0     0    -1     0     0    -1     0    -1     0     1
     0     0     0     0    -1     0     0     0    -1     0     0    -1     0    -1    -1

labels =
Columns 1 through 22
1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Columns 23 through 44
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Columns 45 through 66
1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1
Columns 67 through 88
1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
Columns 89 through 110
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
Columns 111 through 132
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2
Columns 133 through 154
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3
Columns 155 through 176
3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4
Columns 177 through 198
5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6
Columns 199 through 214
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
```

Table 2
Example 2.

```
>> [result,confusion,ECOC,labels]=ECOCs(data,data,'DECOC','HD','ADA')
Computing DECOC coding matrix
4 iterations left for DECOC SFFS.
SFFS moving class 2
3 iterations left for DECOC SFFS.
SFFS moving class 5
2 iterations left for DECOC SFFS.
SFFS moving class 5
1 iterations left for DECOC SFFS.
SFFS moving class 6
0 iterations left for DECOC SFFS.
SFFS moving class 1
4 iterations left for DECOC SFFS.
SFFS moving class 4
3 iterations left for DECOC SFFS.
2 iterations left for DECOC SFFS.
1 iterations left for DECOC SFFS.
SFFS moving class 5
0 iterations left for DECOC SFFS.
Learning coding matrix :
ECOC =
  -1     0    -1     0     1
  -1     0     1     1     0
   1     1     0     0     0
  -1     0    -1     0    -1
  -1     0     1    -1     0
   1    -1     0     0     0
Testing ECOC design
result =
0.9159
confusion =
  65     4     1     0     0     0
   4    71     0     1     0     0
   4     0    13     0     0     0
   0     4     0     9     0     0
   0     0     0     0     9     0
   0     0     0     0     0    29
ECOC =
  -1     0    -1     0     1
  -1     0     1     1     0
   1     1     0     0     0
  -1     0    -1     0    -1
  -1     0     1    -1     0
   1    -1     0     0     0
```

Table 3
Example 3.

labels =
Columns 1 through 22
1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1
Columns 23 through 44
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Columns 45 through 66
1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1
Columns 67 through 88
1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
Columns 89 through 110
2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2
Columns 111 through 132
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 4 2 2
Columns 133 through 154
2 2 2 1 1 2 2 2 2 2 2 2 2 2 2 1 3 3 3 3 3 3 3 3
Columns 155 through 176
3 1 3 1 3 3 3 1 3 4 2 2 4 4 4 4 2 4 4 4 4 2
Columns 177 through 198
5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6
Columns 199 through 214
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6

Table 4
Example 4.