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. The code provides support to include your own coding, decoding, and base classifiers. A Demo file is included with the software simulating some state-of-the-art 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, "Error-Correcting Output Codes Library", *Journal of Machine Learning Research*, vol. 11, pp. 581-584, 2010.

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**

Webpage of the project:

http://ecoclib.svn.sourceforge.net/viewvc/ecoclib/

ECOC Library main training function

[Classifiers,Parameters]=ECOCTrain(data,labels,Parameters)

Input: data: $N \times M$ training data matrix, where N is the number of data samples and M is the number of features, requiring a minimum of two training samples.
Input: labels: vector of labels for the training samples.
Input: Parameters.coding: variable that defines the coding design. In this version of the ECOCs Library, the available coding designs are (the default value is Parameters.coding='OneVsOne'):
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].
using value of *coding='CUSTOM'* $\Longrightarrow$ Your own coding design[a].
Input: Parameters.decoding: variable that defines the decoding strategy. In this version of the ECOCs Library software, the decoding designs are those analyzed in [6] (the default value is Parameters.decoding='HD'):
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.
using value of *decoding='CUSTOM'* $\Longrightarrow$ Your own decoding design[a].
Input: Parameters.base: this variable defines the base classifier that learns the ECOC matrix dichotimizers:
using value of *base='CUSTOM'* $\Longrightarrow$ base classifier, substitute *CUSTOM* by the function name which will be called for training[a].
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).
Output: Classifiers: this variable contains the ECOC trained classifiers.
Output: Parameters: this variable contains the ECOC design parameters to use in the ECOCTest function.

ECOC Library main testing function

[Labels,Values,confusion]=ECOCTest(data,Classifiers,Parameters,labels)

Input: data: $N \times M$ testing data matrix, where N is the number of data samples and M is the number of features.
Input: Classifiers: trained classifiers obtained from the ECOCTrain function call.
Input: Parameters: this variable contains the parameters of the ECOC design obtained from the ECOCTrain function call (although its contain can be altered).
Input: Parameters.base_test: this variable defines the base classifier that lest the ECOC matrix dichotimizers:
using value of *base_test='CUSTOM'* $\Longrightarrow$ base classifier, substitute *CUSTOM* by the function name which will be called for testing[a].
Input: labels: vector of labels for the testing samples (if not given, no confusion matrix is computed).
Output: Labels: this variable contains the predicted labels for each test samples.
Output: Values: output variable that contains the decoding values for each test sample (row) and each class codeword (column).
Output: confusion: this variable contains the confusion matrix for the testing data.

[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.

using value of *ECOC_initial='CUSTOM'* $\Longrightarrow$ your own coding design.

other value $\Longrightarrow$ variable **Parameters.ECOC** 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:

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

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]. Note that in the original definition of the algorithm the splitting criteria is based on the mutual information instead of the SFFS methodology. 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).

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 parameters

In order to learn the data, the user must set the parameter **Parameters.base** with the name of the function of the base classifier for training. The prototype call made by the library is as follows:

```
Classifier = CustomFunction(matrix Data1, matrix Data2, structure
CustomParams)
```

where matrices Data1 and Data2 consist of examples corresponding to the elements of each meta-class. The output of the function must be a an structure where all information needed for evaluating the classifier is stored and that can be passed to the custom interface test function for its evaluation. An additional parameter must be defined: **Parameters.base_params** stores a structure with all needed parameters for training the custom classifier.

Observe that if the custom training function call does not follow the guidelines, an interface function is needed to convert input/output parameters to the needed ones. The following lines show an example of an hypothetical interface function:

```
function
c=interfaceMyClassifier_train(data1,data2,MyClassifparams)

data=[data1;data2];
labels=[ones(size(data1,1),1);-ones(size(data2,1),1)];
[c.out1,c.out2,c.alpha,c.params]=
train_MyClassifier(data,labels,MyClassifparams);
```

In this example the custom training function needs a data set containing all the data examples and its corresponding label set. The following lines show the code needed to call the custom interface:

```
ECOCparams.coding='OneVsOne'; ECOCparams.decoding='LLW';

% The following three lines define the parameters
% needed for the hypothetical custom classifier
params.weaktype='logistic'; params.logisticsmoothness=1;
params.lambda=50;

ECOCparams.base='interfaceMyClassifier_train';
ECOCparams.base_params=params;

[C,ECOCparams]=ECOCTrain(trainset,trainlabs,ECOCparams);
```

5.1 Custom base classifier testing

The testing using a custom base classifier is analogous to the custom training function definition. In this case **Parameters.base_test** defines the name of the function or interface call to the custom classifier test function. Additionally, if the test function requires some parameters they must be stored in a structure in **Parameters.base_test_params**. The expected prototype call in the ECOC library is as follows:

```
Label = CustomTest(matrix Data, structure Classifier, structure
CustomParameters)
```

where Data contain the data to be tested (a row per test sample), Classifier is a structure that contains the trained classifier, it is the output of the custom training interface function, and CustomParameters stores any additional parameter needed for testing. Label is a vector with the predicted memberships for the test samples. The following lines show an example of a hypothetical test interface:

```
function class=interfaceMyClassifier_test(data,c,MyClassifparams)

[class,arr]=test_MyClassifier(data,c.out1,c.out2,c.alpha,MyClassifparams);
```

and the lines needed to test this interface:

```
ECOCparams.coding='OneVsOne'; ECOCparams.decoding='LLW';

% The following line define a possible parameter
% needed for the hypothetical test function
params.weaktype='logistic';

ECOCparams.base='interfaceMyClassifier_test';
ECOCparams.base_test_params=params;

[Labels,Values,Confusion]=ECOCTest(testset,C,ECOCparams,testlabs);
```

5.2 Custom coding

In the coding step, the value **Parameters.coding** is set to *'CUSTOM'* in order to enable custom coding design to be used.

Additionally, one other parameters should be set: **Parameters.custom_coding_params** variable stores a free structure with all the parameters needed in the coding process. The expected structure of the call to the interface to the custom function is the following:

ECOCMatrix = *Parameters.custom_coding*(classes labels, *Parameters.custom_coding_params*)

where "classes labels" are the different possible labels of the multi-class problem and ECOCMatrix is the ECOC coding matrix.

5.3 Custom decoding

In the decoding step, the value **Parameters.decoding** is set to *'CUSTOM'* in order to enable custom decoding strategy to be used.

Additionally, one other parameters should be set: **Parameters.custom_decoding_params** variable stores a free structure with all the parameters needed in the decoding process. The expected structure of the call to the interface to the custom function is the following:

Measure = *Parameters.custom_decoding*(test codeword, class codeword,*Parameters.custom_coding_params*)

where "test codeword" and "class codeword" are codewords for the testing sample and one class from the ECOC matrix, respectively. Measure contains the decoding value.

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 214×9, corresponding to 214 samples for six different classes (labels 1 to 6), and a 9-dimensional space. In order to use the coding and decoding designs, the folders Codings, and Decodings have to be included to the Matlab/Octave path. Additionally, we included a folder Classifiers which includes an implementation of Nearest Mean Classifier and Adaboost classifier as examples of base classifiers for training and testing.

In this case, first we train the classifiers for this data using an one-versus-one ECOC design with Hamming decoding and Adaboost base classifier:

```
load data_g.mat % load data and labels of the data set
Parameters.coding='OneVsOne';
Parameters.decoding='HD';
Parameters.base='ADA'; % ADA is the name of the function for training using Adaboost
Parameters.base_params.iterations=50; % Required parameter for training using Adaboost. In this case it corresponds to the number of decision stumps
[Classifiers,Parameters]=ECOCTrain(data,labels,Parameters)
No base classifier defined, using Discrete Adaboost Learning coding matrix :
   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
Classifiers =

Columns 1 through 7

[1x1 struct] [1x1 struct] [1x1 struct] [1x1 struct] [1x1 struct] [1x1 struct] [1x1 struct]

Columns 8 through 14

[1x1 struct] [1x1 struct] [1x1 struct] [1x1 struct] [1x1 struct] [1x1 struct] [1x1 struct]

Column 15

[1x1 struct]

Parameters = '..'
```

Table 1
Training example.

The system is tested using the same training data:

Parameters.base_test='ADAtest'; % ADAtest is the name of the function for testing using Adaboost. No additional parameters are required in this case, all is stored in the Classifiers structure
[Labels,Values,confusion]=ECOCTest(data,Classifiers,Parameters,labels)
Testing ECOC design
Labels =

1	1	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	1

...
Values =

5	6	7	9	9	9
5	6	7	9	10	8
5	6	7	9	10	8
5	6	7	9	10	8
5	6	7	9	10	8
6	5	7	9	10	8
5	6	7	9	10	8
5	6	7	9	10	8
5	6	7	8	10	9

...
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

Table 2
Testing example. Predicted labels, matrix of Hamming decoding values per class and confusion matrix are obtained.

In the file *Demo.m* provided with the code you can look for more learning and testing ECOC examples over different data sets, coding, decoding, and base classifiers. In particular, the following experiments are performed:

• 1) Calling the main ECOC function to train Glass data set with the following configuration: one-versus-one coding, Hamming decoding, and Adaboost base classifier.

• 2) Calling ECOCTest function to classify the previous design over the same data.

• 3) Perform the same steps using Sparse Random design with Euclidean Decoding and Nearest Mean Classifier.

• 4) Calling ECOCTest function to classify the previous design over the same data. No test labels are passed as parameters.

• 5) Calling the ECOCTrain function with Ecoli data set, DECOC coding, Linear Loss-Weighted decoding, and Adaboost base classifier. 90% of the data is used to train.

• 6) Calling ECOCTest function to classify the previous design over remaining 10% of data.

• 7) Calling the ECOCTrain function with Iris data set, ECOC-ONE coding, Exponential Loss-Weighted decoding, and Adaboost base classifier. 90% of the data is used to train.

• 8) Calling ECOCTest function to classify the previous design over remaining 10% of data.

References

[1] S. Escalera, O. Pujol, P. Radeva, Separability of ternary codes for sparse designs of error-correcting output codes, in: Pattern Recognition Letters, Vol. 30, 2009, pp. 285–297.

[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, Vol. 99, 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`