Multi-Imbalance: an open-source software for multi-class imbalance learning

User Manual (MATLAB Version)

Chongsheng Zhang[a], Jingjun Bi[a], Shixin Xu[a], Enislay Ramentol[b], Gaojuan Fan[a], Baojun Qiao[a], Hamido Fujita[c]

[a]The Big Data Research Center, Henan University, 475001 KaiFeng, China
[b]SICS Swedish ICT, Isafjordsgatan 22, Box 1263, SE-164 29 Kista, Sweden
[c]Faculty of Software and Information Science, Iwate Prefectural University, Iwate, Japan

---

This user manual presents "Multi-Imbalance", which is an open source software for multi-class imbalanced learning field. It contains 18 algorithms for multi-class imbalanced data classification.

CONTENTS

1. Overview of Multi-Imbalance

In recent years, although many researchers have proposed different algorithms and techniques to address the multi-class imbalanced data classification issue, there is still no open-source software for this specific field. To address this issue, we develop the "Multi-Imbalance" (Multi-class Imbalanced data classification) software package and share it with the community, to boost research in this field.

The developed Multi-Imbalance software contains 18 different algorithms for multi-class imbalance learning, which are depicted in Figure 1, many of them were proposed in recent years. We divide these algorithms into 7 modules (categories). We will introduce the framework and functionalities of this software in the next sections.

Figure 1. The major modules in Multi-Imbalance

Using Multi-Imbalance, researchers can directly re-use our implementations on multi-class imbalanced data classification, thus avoid implementing them from scratch. Hence, Multi-Imbalance will be helpful and indispensable for researchers in the multi-class imbalance learning field.

2. Software Installation/Deployment with MATLAB

In order to use the MATLAB version Multi-Imbalance, users only need to add the Multi-Imbalance software package to the MATLAB search path.

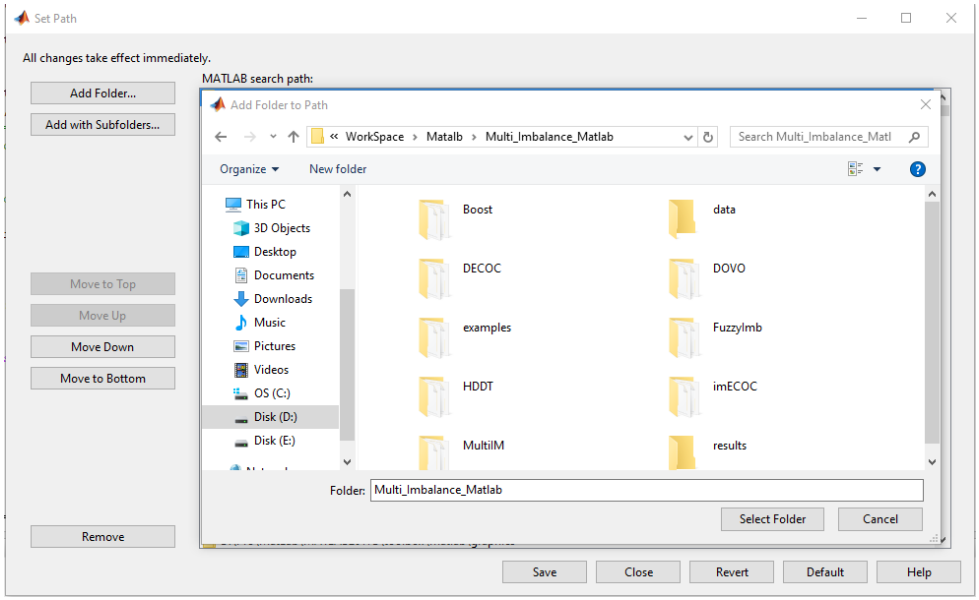

Figure 2. Adding the Multi-Imbalance package to the MATLAB path

3. Usage Example for Each Algorithm

There are 7 classes (categories) of algorithms for multi-class imbalance learning, each class consisting of one or more algorithms. In total, there are 18 major algorithms for multi-class imbalance learning. In the following, we give the usage examples of these 18 major algorithms for multi-class imbalance learning.

If users need to test a new dataset, they only need to replace the current "Wine_data_set_index_fixed" data with the new dataset.

Variables – data.data

data | data.data

data.data

Fields	train	trainlabel	test	testlabel
1	*143x13 double*	*143x1 double*	*35x13 double*	*35x1 double*
2	*142x13 double*	*142x1 double*	*36x13 double*	*36x1 double*
3	*142x13 double*	*142x1 double*	*36x13 double*	*36x1 double*
4	*142x13 double*	*142x1 double*	*36x13 double*	*36x1 double*
5	*143x13 double*	*143x1 double*	*35x13 double*	*35x1 double*
6				

Figure 3. Elements of "Wine_data_set_index_fixed" data after loading into Matlab

It should be noted that, when we load a dataset, e.g., Wine_data_set_index_fixed.mat, the corresponding data contains five rows, while each row has 4 structures, which are the training data and the corresponding lables, train and trainlabel; the test data and the corresponding labels, test and testlabel. The reason that it contains 5 row is for 5-fold cross-validation purpose: we sequentially split the dataset into 5 parts, then successively switch the training and test data (with a ratio of 4:1).

3.1 AdaBoost.M1

Principles: AdaBoost.M1 extends AdaBoost in both the update of samples' weights and the classifier combination strategy.

The main steps of this algorithm are as follows:

Step 1: Initialize the weight Vector with uniform distribution
Step 2: for t=1 to Max_Iter do
Step 3: Fit a classifier nb to the training data using weights
Step 4: Compute weighted error
Step 5: Compute AlphaT=0.5*log((CorrectRate+eps)/(errorRate+eps))
Step 6: Update weights
Step 7: Re-normalize the weights
Step 8: end for
Step 9: Output the final Classifier

Input: the imbalanced dataset "Wine_data_set_index_fixed"
Output: the prediction results on the dataset using the AdaBoost.M1 algorithm, where _p.mat is the prediction results, and _c.mat is the ground truth.

Usage example:

```
function runAdaBoostM1
javaaddpath('weka.jar');

p = genpath(pwd);
addpath(p, '-begin');
% record = 'testall.txt';
% save record record

dataset_list = {'Wine_data_set_indx_fixed'};

for p = 1:length(dataset_list)
    load(['data\', dataset_list{p},'.mat']);
    disp([dataset_list{p}, ' - numero dataset: ',num2str(p), ]);

    %AdaBoost.M1
    %input: traindata, trainlabel, testdata, testlabel,
            Max_Iter (which is the number of CART decision trees to be built for AdaBoost)
    %output: trainCostTime,predictCostTime,predictResult
    for d=1:5

        [Cost(d).adaboostcartM1tr,Cost(d).adaboostcartM1te,Pre(d).adaboostcartM1] =
        adaBoostCartM1(data(d).train,data(d).trainlabel,data(d).test,20);
```

```
    end

    save (['results/', dataset_list{p},'_',    'p', '.mat'], 'Pre');
    save (['results/', dataset_list{p},'_', 'c', '.mat'],    'Cost');

    clear Cost Pre Indx;

end
```

Reference:
Freund, Y. & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, August 1997, 55(1).

3.2 SAMME

Principles: SAMME (Stagewise Additive Modeling using a Multi-class Exponential loss function) also extends AdaBoost in both the update of samples' weights and the classifier combination strategy. The main difference between SAMME and AdaBoost.M1 is the method for updating the weights of the samples.

The main procedure (steps) of the SAMME algorithm are as follows:

Step 1: Initialize the weight Vector with uniform distribution
Step 2: for t=1 to Max_Iter do
Step 3: Fit a classifier nb to the training data using weights
Step 4: Compute weighted error: errorRate=sum(weight(find(predicted~=trainlabel)));
Step 5: Compute AlphaT=log((1-errorRate)/(errorRate+eps))+log(length(labels)-1)
Step 6: Update weights weight(i)=weight(i)* exp(AlphaT(t));(trainlabel(i)~=predicted(i))
Step 7: Re-normalize weight
Step 8: end for
Step 9: Output the final Classifier

Input: the imbalanced dataset
Output: the prediction results on the dataset using the SAMME algorithm,
where _p.mat is the prediction results, and _c.mat is the ground truth.

Usage example:

```
function runSAMME
javaaddpath('weka.jar');

p = genpath(pwd);
addpath(p, '-begin');
% record = 'testall.txt';
```

```
% save record record

dataset_list = {'Wine_data_set_indx_fixed'};

for p = 1:length(dataset_list)
    load(['data\', dataset_list{p},'.mat']);
    disp([dataset_list{p}, ' - numero dataset: ',num2str(p), ]);

    %SAMME
    %input: traindata,trainlabel,testdata,testlabel
            Max_Iter (which is the number of CART decision trees to be built for AdaBoost)
    %output: trainCostTime,predictCostTime,predictResult
    for d=1:5

        [Cost(d).SAMMEcarttr,Cost(d).SAMMEcartte,Pre(d).SAMMEcart] =
sammeCart(data(d).train,data(d).trainlabel,data(d).test,20);

    end

    save (['results/', dataset_list{p},'_',    'p', '.mat'], 'Pre');
    save (['results/', dataset_list{p},'_', 'c', '.mat'],    'Cost');

    clear Cost Pre Indx;

end
```

Reference:

Zhu, J., Zou, H., Rosset, S., et al. (2006). Multi-class AdaBoost. Statistics & Its Interface, 2006, 2(3), 349-360.

3.3 AdaC2.M1

Principles: AdaC2.M1 is also a variant of AdaBoost. It derives the best cost setting through the genetic algorithm (GA) method, then takes this cost setting into consideration in the subsequent boosting.

Input: the imbalanced dataset

Output: the prediction results on the dataset using the AdaC2.M1 algorithm, where _p.mat is the prediction results, and _c.mat is the ground truth.

Usage example:

```
function runAdaC2M1
javaaddpath('weka.jar');
```

```
p = genpath(pwd);
addpath(p, '-begin');
% record = 'testall.txt';
% save record record

dataset_list = {'Wine_data_set_indx_fixed'};

for p = 1:length(dataset_list)
    load(['data\', dataset_list{p},'.mat']);
    disp([dataset_list{p}, ' - numero dataset: ',num2str(p), ]);

    %AdaC2.M1
    %input: traindata, trainlabel, testdata, testlabel,
            Max_Iter (which is the number of CART decision trees to be built for AdaBoost)
            C0 is the optimum cost setup (array) of each class
    %output: trainCostTime,predictCostTime,predictResult
    for d=1:5
        tic;
        C0=testGA(data(d).train,data(d).trainlabel);
        Cost(d).GA=toc;
        Indx(d).GA=C0;

        [Cost(d).adaC2cartM1GAtr,Cost(d).adaC2cartM1GAte,Pre(d).adaC2cartM1GA]    =
adaC2CartM1(data(d).train,data(d).trainlabel,data(d).test,20,C0);

    end

    save (['results/', dataset_list{p},'_',    'p', '.mat'], 'Pre');
    save (['results/', dataset_list{p},'_', 'c', '.mat'],    'Cost');

    clear Cost Pre Indx;

end
```

Reference:

Sun, Y., Kamel, M. S. & Wang, Y. (2006). Boosting for learning multiple classes with imbalanced class distribution. Proceedings of the 6th International Conference on Data Mining, 2006 (PP. 592-602).

3.4 AdaBoost.NC

Principles: AdaC2.M1 derives the best cost setting through the genetic algorithm (GA) method, then takes this cost setting into consideration in the subsequent boosting.

AdaBoost.NC is another variant of AdaBoost. Since GA is very time consuming, AdaBoost.NC deprecates the GA algorithm, but emphasizes ensemble diversity during training and exploits its good generalization performance to facilitate class imbalance learning.

Input: the imbalanced dataset

Output: the prediction results on the dataset using the AdaBoost.NC algorithm, where _p.mat is the prediction results, and _c.mat is the ground truth.

Usage example:

```
function runAdaBoostNC
javaaddpath('weka.jar');

p = genpath(pwd);
addpath(p, '-begin');
% record = 'testall.txt';
% save record record

dataset_list = {'Wine_data_set_indx_fixed'};

for p = 1:length(dataset_list)
    load(['data\', dataset_list{p},'.mat']);
    disp([dataset_list{p}, ' - numero dataset: ',num2str(p), ]);

    %AdaBoost.NC
    %input: traindata, trainlabel, testdata, testlabel,
            Max_Iter (which is the number of CART decision trees to be built for AdaBoost)
            lama is the weight update parameter
    %output: trainCostTime,predictCostTime,predictResult
    for d=1:5

        [Cost(d).adaboostcartNCtr,Cost(d).adaboostcartNCte,Pre(d).adaboostcartNC]  =
adaBoostCartNC(data(d).train,data(d).trainlabel,data(d).test,20,2);

    end

    save (['results/', dataset_list{p},'_',    'p', '.mat'], 'Pre');
    save (['results/', dataset_list{p},'_', 'c', '.mat'],    'Cost');

    clear Cost Pre Indx;

end
```

Reference:

Wang, S., Chen, H. & Yao, X. Negative correlation learning for classification ensembles. Proc. Int. Joint Conf. Neural Netw., 2010 (PP. 2893-2900).

3.5 PIBoost

Principles: PIBoost is also a variant of AdaBoost for multi-class imbalance learning. It combines binary weak-learners to separate groups of classes and uses a margin-based exponential loss function to classify multi-class imbalanced data.

Input: the imbalanced dataset
Output: the prediction results on the dataset using the PIBoost algorithm,
where _p.mat is the prediction results, and _c.mat is the ground truth.

Usage example:

```
function runPIBoost
javaaddpath('weka.jar');

p = genpath(pwd);
addpath(p, '-begin');
% record = 'testall.txt';
% save record record

dataset_list = {'Wine_data_set_indx_fixed'};

for p = 1:length(dataset_list)
    load(['data\', dataset_list{p},'.mat']);
    disp([dataset_list{p}, ' - numero dataset: ',num2str(p), ]);

    %PIBoost
    %input: traindata, trainlabel, testdata, testlabel,
            Max_Iter (which is the number of CART decision trees to be built for AdaBoost)
    %output: trainCostTime,predictCostTime,predictResult
    for d=1:5

        [Cost(d).PIBoostcarttr,Cost(d).PIBoostcartte,Pre(d).PIBoostcart]                    =
PIBoostCart(data(d).train,data(d).trainlabel,data(d).test,20);

    end

    save (['results/', dataset_list{p},'_',    'p', '.mat'], 'Pre');
    save (['results/', dataset_list{p},'_', 'c', '.mat'],    'Cost');

    clear Cost Pre Indx;
end
```

Reference:

Fernndez, B. A. & Baumela. L. (2014). Multi-class boosting with asymmetric binary weak-learners.

Pattern Recognition, 2014, 47(5), PP. 2080-2090.

3.6 DECOC

Principles: DECOC algorithm is an ensemble of hetergenuous classifiers for multi-class imbalanced dataset, which is decomposed into binary ones using ECOC method. It adopts imECOC algorithm for the ECOC decoding step.

This algorithm contains the following steps:

Step 1: It uses ECOC strategy to tranform the multi-class data into multiple binary data, then finds the best classifier for each specific binaried data, which will be kept by ft.

Step 2: Using funcwEDOVO, it next builds the best classifier for each binarized data (by ECOC) and the predictions are in allpre.

Step 3: With funcpretestEDOVO, it makes the predictions on the test data, using the imECOC algorithm.

Input: the imbalanced dataset
Output: the prediction results on the dataset using the DECOC algorithm,
where _p.mat is the prediction results, and _c.mat is the ground truth.

Usage Example

```
function runDECOC
javaaddpath('weka.jar');

p = genpath(pwd);
addpath(p, '-begin');
% record = 'testall.txt';
% save record record

dataset_list = {'Wine_data_set_indx_fixed'};

for p = 1:length(dataset_list)
    load(['data\', dataset_list{p},'.mat']);
    disp([dataset_list{p}, ' - numero dataset: ',num2str(p), ]);

     %DECOC
    %input: traindata, trainlabel, testdata, testlabel,
            'sparse' is the ECOC coding type
     %output: trainCostTime,predictCostTime,predictResult
    for d=1:5
```

```
            [Cost(d).imECOCDOVOs1tr,Cost(d).imECOCDOVOs1te,Pre(d).imECOCDOVOs1]
= DECOC(data(d).train,data(d).trainlabel,data(d).test, 'sparse',1);

        end

        save (['results/', dataset_list{p},'_',    'p', '.mat'], 'Pre');
        save (['results/', dataset_list{p},'_', 'c', '.mat'],    'Cost');

        clear Cost Pre Indx;

end
```

Reference:

Jingjun Bi, Chongsheng Zhang*. (2018). An Empirical Comparison on State-of-the-art Multi-class Imbalance Learning Algorithms and A New Diversified Ensemble Learning Scheme. Knowledge-based Systems, 2018, Vol.158, pp. 81-93.

3.7 DOVO

Principles: DOVO is an ensemble based approach for multi-class imbalance learning. It uses OVO to decompose the multi-class data and majority voting to ensemble the outputs of different binary classifiers.

The main steps are as follows:

Step 1. for the multi-class imbalanced data, let nc be the number of classes.

Step 2. using the One-Versus-One (OVO) decomposition strategy, it splits the original data into nc*(nc-1)/2 sub-datasets, each sub-dataset only contains two classes.

Step 3. for each sub-dataset, it exhausitively try all the different classification algorithms, at the end, pick the classification algorithm that achieves the best accuracy (in terms of ACC, or G-mean, or F-measure, or AUC).

Step 4. finally, each sub-dataset, it chooses the best classification algorithm (and classification model) that achieves the best accuracy on this sub-dataset.

Step 5. in the prediction phase, all the nc*(nc-1)/2 classification models will be used predict the labels of the test data instances, it uses majority voting to make the final prediction for every instance in the test data.

Input: the imbalanced dataset
Output: the prediction results on the dataset using the DOVO algorithm,

where _p.mat is the prediction results, and _c.mat is the ground truth.

Usage example:

```
function runDOVO
javaaddpath('weka.jar');

p = genpath(pwd);
addpath(p, '-begin');
% record = 'testall.txt';
% save record record

dataset_list = {'Wine_data_set_indx_fixed'};

for p = 1:length(dataset_list)
    load(['data\', dataset_list{p},'.mat']);
    disp([dataset_list{p}, ' - numero dataset: ',num2str(p), ]);

    %DOVO
    %input: traindata, trainlabel, testdata, testlabel, kfold
    %output: trainCostTime,predictCostTime,predictResult,bestChosen
    for d=1:5

        [Cost(d).DOAOtr,Cost(d).DOAOte,Pre(d).DOAO,Indx(d).C]                    =
DOVO([data(d).train,data(d).trainlabel],data(d).test,data(d).testlabel,5);

    end

    save (['results/', dataset_list{p},'_',    'p', '.mat'], 'Pre');
    save (['results/', dataset_list{p},'_', 'c', '.mat'],    'Cost');

    clear Cost Pre Indx;

end
```

Reference:

Kang, S., Cho, S. & Kang P. (2015) Constructing a multi-class classifier using one-against-one approach with different binary classifiers. Neurocomputing, 2015, Vol. 149, pp. 677-682.

3.8 FuzzyImbECOC

Principles: IFROWANN (FuzzyImb) was originally designed for binary imbalanced data. In fuzzyImbECOC, we extend it with the ECOC encoding strategy to handle multi-class imbalanced data.

The main steps of fuzzyImbECOC are as follows:

Step 1. It first generates the ECOC matrix (with each codeword for a specific class), each class will be represented by an array of codes such as 1 1 -1 -1 1 -1.

Step 2. It next extracts the instances (and the corresponding labels) of each original class, and keeps them in train{i};

Step 3. the ECOC matrix for all the classes is a matrix, each row represents the codeword of one class.

a) for each column of the ECOC matrix, it first retrieve the corresponding bit value of each class, then assign this bit value as the label for all the instances of the current class. This is for handling the multi-class data.

b) for each two-class data, it uses fuzzyImb to train the binary classifier (see the above reference). At the end, we will train a few binary classifiers.

c) for each test instance from testdata, it uses all the classifiers obtained from b) to make predictions; their predictions will be combined as an array, then use the ECOC decoding method to find the nearest ECOC codeword, and output the corresponding class label as the final prediction.

Input: the imbalanced dataset
Output: the prediction results on the dataset using the FuzzyImbECOC algorithm,
where _p.mat is the prediction results, and _c.mat is the ground truth.

Usage example:

```
function runFuzzyImbECOC
javaaddpath('weka.jar');

p = genpath(pwd);
addpath(p, '-begin');
% record = 'testall.txt';
% save record record

dataset_list = {'Wine_data_set_indx_fixed'};

for p = 1:length(dataset_list)
    load(['data\', dataset_list{p},'.mat']);
    disp([dataset_list{p}, ' - numero dataset: ',num2str(p), ]);

    %FuzzyImb+ECOC
    %input: traindata, trainlabel, testdata, testlabel, weightStrategy, gamma
```

```
    %output: trainCostTime,predictCostTime,predictResult
     for d=1:5
         tic;
         [Pre(d).fuzzyw6]                                                    =
fuzzyImbECOC(data(d).train,data(d).trainlabel,data(d).test,data(d).testlabel, 'w6',0.1);
         Cost(d).fuzzyw6=toc;
     end

    save (['results/', dataset_list{p},'_',    'p', '.mat'], 'Pre');
    save (['results/', dataset_list{p},'_', 'c', '.mat'],    'Cost');

    clear Cost Pre Indx;

end
```

Refrence:

E. Ramentol, S. Vluymans, N. Verbiest, et al. , IFROWANN: Imbalanced Fuzzy-Rough Ordered Weighted Average Nearest Neighbor Classification, IEEE Transactions on Fuzzy Systems 23 (5) (2015) 1622-1637.

3.9 HDDTOVA

Principles: HDDTOVA is HDDT plus the decomposition technique OVA for multi-class imbalanced data. It is our extension of HDDT to multi-class imbalanced data. It builds numberc of binary HDDT classifiers by combining the OVA strategy and HDDT, then combines the outputs of different binary HDDT classifiers generated using the OVA strategy. Here, the decoding strategy for OVA is the same as the imECOC decoding.

Input: the imbalanced dataset
Output: the prediction results on the dataset using the HDDTova algorithm,
where _p.mat is the prediction results, and _c.mat is the ground truth.

Usage example:

```
function runHDDTOVA
javaaddpath('weka.jar');

p = genpath(pwd);
addpath(p, '-begin');
% record = 'testall.txt';
% save record record

dataset_list = {'Wine_data_set_indx_fixed'};
```

```
for p = 1:length(dataset_list)
    load(['data\', dataset_list{p},'.mat']);
    disp([dataset_list{p}, ' - numero dataset: ',num2str(p), ]);

    %HDDT+OVA
    %input: traindata, trainlabel, testdata, testlabel,
    %output: trainCostTime,predictCostTime,predictResult
    for d=1:5

        [Cost(d).HDDTovatr,Cost(d).HDDTovate,Pre(d).HDDTova] =
HDDTOVA(data(d).train,data(d).trainlabel,data(d).test,data(d).testlabel);

    end

    save (['results/', dataset_list{p},'_',    'p', '.mat'], 'Pre');
    save (['results/', dataset_list{p},'_', 'c', '.mat'],    'Cost');

    clear Cost Pre Indx;

end
```

Reference:

Hoens, T. R., Qian, Q., Chawla, N. V., et al. (2012). Building decision trees for the multi-class imbalance problem. Advances in Knowledge Discovery and Data Mining. Springer Berlin Heidelberg, 2012 (PP. 122-134).

3.10 HDDTECOC

Principles: HDDTECOC is HDDT plus the decomposition technique ECOC for multi-class imbalanced data. It builds a few binary HDDT classifiers by combining the ECOC strategy and HDDT. It next combines the outputs of different binary HDDT classifiers generated using the ECOC strategy to make predicitions.

Input: the imbalanced dataset

Output: the prediction results on the dataset using the HDDTecoc algorithm, where _p.mat is the prediction results, and _c.mat is the ground truth.

Usage example:

```
function runHDDTECOC
javaaddpath('weka.jar');

p = genpath(pwd);
addpath(p, '-begin');
```

```
% record = 'testall.txt';
% save record record

dataset_list = {'Wine_data_set_indx_fixed'};

for p = 1:length(dataset_list)
    load(['data\', dataset_list{p},'.mat']);
    disp([dataset_list{p}, ' - numero dataset: ',num2str(p), ]);

    %HDDT+ECOC
    %input: traindata, trainlabel, testdata, testlabel
    %output: trainCostTime,predictCostTime,predictResult
    for d=1:5

        [Cost(d).HDDTecoctr,Cost(d).HDDTecocte,Pre(d).HDDTecoc]                    =
HDDTECOC(data(d).train,data(d).trainlabel,data(d).test,data(d).testlabel);

    end

    save (['results/', dataset_list{p},'_',    'p', '.mat'], 'Pre');
    save (['results/', dataset_list{p},'_', 'c', '.mat'],    'Cost');

    clear Cost Pre Indx;

end
```

Reference：

Hoens, T. R., Qian, Q., Chawla, N. V., et al. (2012). Building decision trees for the multi-class imbalance problem. Advances in Knowledge Discovery and Data Mining. Springer Berlin Heidelberg, 2012 (PP. 122-134).

3.11 MCHDDT

Principles: MCHDDT, the multi-Class HDDT method, successively takes one or a pair of classes as the positive class and the rest as negative class, when calculating the Hellinger distance for each feature. It selects the maximum Hellinger value for each feature. The feature with the maximum Hellinger distance will be used to split the node. Then, after determining the best split feature, it recursively build the subtrees.

Input: the imbalanced dataset

Output: the prediction results on the dataset using the MCHDDT algorithm,
where _p.mat is the prediction results, and _c.mat is the ground truth.

Usage example:

```
function runMCHDDT
javaaddpath('weka.jar');

p = genpath(pwd);
addpath(p, '-begin');
% record = 'testall.txt';
% save record record

dataset_list = {'Wine_data_set_indx_fixed'};

for p = 1:length(dataset_list)
    load(['data\', dataset_list{p},'.mat']);
    disp([dataset_list{p}, ' - numero dataset: ',num2str(p), ]);

    %MC-HDDT
    %input: traindata, trainlabel, testdata, testlabel
    %output: trainCostTime,predictCostTime,predictResult
    for d=1:5

        [Cost(d).MCHDDTtr,Cost(d).MCHDDTte,Pre(d).MCHDDT] =
MCHDDT(data(d).train,data(d).trainlabel,data(d).test,data(d).testlabel);

    end

    save (['results/', dataset_list{p},'_',  'p', '.mat'], 'Pre');
    save (['results/', dataset_list{p},'_', 'c', '.mat'],  'Cost');

    clear Cost Pre Indx;

end
```

Reference:

Hoens, T. R., Qian, Q., Chawla, N. V., et al. (2012). Building decision trees for the multi-class imbalance problem. Advances in Knowledge Discovery and Data Mining. Springer Berlin Heidelberg, 2012 (PP. 122-134).

3.12 imECOC + sparse

Principles: The imECOC algorithm contains the following steps:

Step 1. in each binary classifier, it simultaneously considers the between-class and the within-class imbalance;

Step 2. in the training/prediction phase, it assigns different weights to different binary classifiers;

Step 3. in the prediction phase, it decodes it with weighted distance to obtain the optimal weight of the classifier by minimizing the weighted loss.

In our implementations, we use Sparse, Dense, OVA as 3 different coding methods for ECOC. imECOC + sparse is one of such methods.

Input: the imbalanced dataset

Output: the prediction results on the dataset using the ImECOC sparse algorithm, where _p.mat is the prediction results, and _c.mat is the ground truth.

Usage example:

```
function runImECOCsparse
javaaddpath('weka.jar');

p = genpath(pwd);
addpath(p, '-begin');
% record = 'testall.txt';
% save record record

dataset_list = {'Wine_data_set_indx_fixed'};

for p = 1:length(dataset_list)
    load(['data\', dataset_list{p},'.mat']);
    disp([dataset_list{p}, ' - numero dataset: ',num2str(p), ]);

    %imECOC+sparse
    %input: traindata, trainlabel, testdata, testlabel, ECOC coding type, withw
    %output: trainCostTime,predictCostTime,predictResult
    for d=1:5

        [Cost(d).imECOCs1tr,Cost(d).imECOCs1te,Pre(d).imECOCs1] =
imECOC(data(d).train,data(d).trainlabel,data(d).test, 'sparse',1);

    end

    save (['results/', dataset_list{p},'_',  'p', '.mat'], 'Pre');
    save (['results/', dataset_list{p},'_', 'c', '.mat'],  'Cost');

    clear Cost Pre Indx;

end
```

Reference:

Liu, X. Y., Li, Q. Q. & Zhou Z H. (2013). Learning imbalanced multi-class data with optimal dichotomy weights. IEEE 13th International Conference on Data Mining (IEEE ICDM), 2013 (PP. 478-487).

3.13 imECOC + OVA

Principles: The imECOC algorithm contains the following steps:

Step 1. in each binary classifier, it simultaneously considers the between-class and the within-class imbalance;

Step 2. in the training/prediction phase, it assigns different weights to different binary classifiers;

Step 3. in the prediction phase, it decodes it with weighted distance to obtain the optimal weight of the classifier by minimizing the weighted loss.

In our implementations, we use Sparse, Dense, OVA as 3 different coding methods for ECOC. imECOC + OVA is one of such methods.

Input: the imbalanced dataset

Output: the prediction results on the dataset using the ImECOC OVA algorithm, where _p.mat is the prediction results, and _c.mat is the ground truth.

Usage example:

```
function runImECOCOVA
javaaddpath('weka.jar');

p = genpath(pwd);
addpath(p, '-begin');
% record = 'testall.txt';
% save record record

dataset_list = {'Wine_data_set_indx_fixed'};

for p = 1:length(dataset_list)
    load(['data\', dataset_list{p},'.mat']);
    disp([dataset_list{p}, ' - numero dataset: ',num2str(p), ]);

    %imECOC+OVA
    %input: traindata, trainlabel, testdata, testlabel, ECOC coding type, withw
    %output: trainCostTime,predictCostTime,predictResult
    for d=1:5
```

```
        [Cost(d).imECOCo1tr,Cost(d).imECOCo1te,Pre(d).imECOCo1] =
imECOC(data(d).train,data(d).trainlabel,data(d).test, 'OVA',1);

    end

    save (['results/', dataset_list{p},'_',    'p', '.mat'], 'Pre');
    save (['results/', dataset_list{p},'_', 'c', '.mat'],    'Cost');

    clear Cost Pre Indx;

end
```

Reference:

Liu, X. Y., Li, Q. Q. & Zhou Z H. (2013). Learning imbalanced multi-class data with optimal dichotomy weights. IEEE 13th International Conference on Data Mining (IEEE ICDM), 2013 (PP. 478-487).

3.14 imECOC + dense

Principles: The imECOC algorithm contains the following steps:

Step 1. in each binary classifier, it simultaneously considers the between-class and the within-class imbalance;

Step 2. in the training/prediction phase, it assigns different weights to different binary classifiers;

Step 3. in the prediction phase, it decodes it with weighted distance to obtain the optimal weight of the classifier by minimizing the weighted loss.

In our implementations, we use Sparse, Dense, OVA as 3 different coding methods for ECOC. imECOC + dense is one of such methods.

Input: the imbalanced dataset
Output: the prediction results on the dataset using the ImECOC dense algorithm, where _p.mat is the prediction results, and _c.mat is the ground truth.

Usage example:

```
function runImECOCdense
javaaddpath('weka.jar');

p = genpath(pwd);
addpath(p, '-begin');
% record = 'testall.txt';
% save record record
```

```
dataset_list = {'Wine_data_set_indx_fixed'};

for p = 1:length(dataset_list)
    load(['data\', dataset_list{p},'.mat']);
    disp([dataset_list{p}, ' - numero dataset: ',num2str(p), ]);

    %imECOC+dense
    %input: traindata, trainlabel, testdata, testlabel, ECOC coding type, withw
    %output: trainCostTime,predictCostTime,predictResult
    for d=1:5

        [Cost(d).imECOCd1tr,Cost(d).imECOCd1te,Pre(d).imECOCd1] =
imECOC(data(d).train,data(d).trainlabel,data(d).test, 'dense',1);

    end

    save (['results/', dataset_list{p},'_',  'p', '.mat'], 'Pre');
    save (['results/', dataset_list{p},'_', 'c', '.mat'],  'Cost');

    clear Cost Pre Indx;

end
```

Reference:

Liu, X. Y., Li, Q. Q. & Zhou Z H. (2013). Learning imbalanced multi-class data with optimal dichotomy weights. IEEE 13th International Conference on Data Mining (IEEE ICDM), 2013 (PP. 478-487).

3.15 Multi-IM + OVA

Principles: PRMs-IM is a classification algorithm (originally designed) for binary imbalanced data. Let m be the ratio between the number of majority samples and that of the minority samples. PRMs-IM randomly divides the majority samples into m parts, next combines each part with all the minority instances, then trains a corresponding binary classifier. In the prediction phase, it uses weighted voting to ensemble the outputs of the m classifiers and makes the final prediction.

Multi-IM algorithm combines A&O and PRMs-IM, where PRMs-IM is adopted to train the classifier for A&O. Besides A&O, in our work, we also combine the OVA, OVO and OAHO decomposition methods with PRMs-IM to further investigate the performance of PRMs-IM for multi-class imbalance learning. Multi-IM+OVA is one of such methods.

Input: the imbalanced dataset

Output: the prediction results on the dataset using the Multi-IM OVA algorithm,

where _p.mat is the prediction results, and _c.mat is the ground truth.

Usage example:

```
function runMultiImOVA
javaaddpath('weka.jar');

p = genpath(pwd);
addpath(p, '-begin');
% record = 'testall.txt';
% save record record

dataset_list = {'Wine_data_set_indx_fixed'};

for p = 1:length(dataset_list)
    load(['data\', dataset_list{p},'.mat']);
    disp([dataset_list{p}, ' - numero dataset: ',num2str(p), ]);

    %Multi-IM+OVA
    %input: traindata, trainlabel, testdata, testlabel
    %output: trainCostTime,predictCostTime,predictResult
    for d=1:5

        [Cost(d).classOVAtr,Cost(d).classOVAte,Pre(d).classOVA]                    =
classOVA(data(d).train,data(d).trainlabel,data(d).test);

    end

    save (['results/', dataset_list{p},'_',    'p', '.mat'], 'Pre');
    save (['results/', dataset_list{p},'_', 'c', '.mat'],    'Cost');

    clear Cost Pre Indx;

end
```

Reference:

Ghanem, A. S., Venkatesh, S. & West, G. (2010). Multi-class pattern classification in imbalanced data. International Conference on Pattern Recognition (ICPR), 2010 (PP. 2881-2884).

3.16 Multi-IM + OVO

Principles: PRMs-IM is a classification algorithm (originally designed) for binary imbalanced data. Let m be the ratio between the number of majority samples and that of the minority samples. PRMs-IM randomly divides the majority samples into m parts, next combines each part with all the minority instances, then trains a corresponding binary classifier. In the prediction phase, it uses

weighted voting to ensemble the outputs of the m classifiers and makes the final prediction.

Multi-IM algorithm combines A&O and PRMs-IM, where PRMs-IM is adopted to train the classifier for A&O. Besides A&O, in our work, we also combine the OVA, OVO and OAHO decomposition methods with PRMs-IM to further investigate the performance of PRMs-IM for multi-class imbalance learning. Multi-IM+OVO is one of such methods.

Input: the imbalanced dataset
Output: the prediction results on the dataset using the Multi-IM OVO algorithm, where _p.mat is the prediction results, and _c.mat is the ground truth.

Usage example:

```
function runMultiImOVO
javaaddpath('weka.jar');

p = genpath(pwd);
addpath(p, '-begin');
% record = 'testall.txt';
% save record record

dataset_list = {'Wine_data_set_indx_fixed'};

for p = 1:length(dataset_list)
    load(['data\', dataset_list{p},'.mat']);
    disp([dataset_list{p}, ' - numero dataset: ',num2str(p), ]);

    %Multi-IM+OVO
    %input: traindata, trainlabel, testdata, testlabel
    %output: trainCostTime,predictCostTime,predictResult
    for d=1:5

        [Cost(d).classOAOtr,Cost(d).classOAOte,Pre(d).classOAO] =
classOAO([data(d).train,data(d).trainlabel],data(d).test);

    end

    save (['results/', dataset_list{p},'_',    'p', '.mat'], 'Pre');
    save (['results/', dataset_list{p},'_', 'c', '.mat'],    'Cost');

    clear Cost Pre Indx;

end
```

Reference:

Ghanem, A. S., Venkatesh, S. & West, G. (2010). Multi-class pattern classification in imbalanced data. International Conference on Pattern Recognition (ICPR), 2010 (PP. 2881-2884).

3.17 Multi-IM + OAHO

Principles: PRMs-IM is a classification algorithm (originally designed) for binary imbalanced data. Let m be the ratio between the number of majority samples and that of the minority samples. PRMs-IM randomly divides the majority samples into m parts, next combines each part with all the minority instances, then trains a corresponding binary classifier. In the prediction phase, it uses weighted voting to ensemble the outputs of the m classifiers and makes the final prediction.

Multi-IM algorithm combines A&O and PRMs-IM, where PRMs-IM is adopted to train the classifier for A&O. Besides A&O, in our work, we also combine the OVA, OVO and OAHO decomposition methods with PRMs-IM to further investigate the performance of PRMs-IM for multi-class imbalance learning. Multi-IM+OAHO is one of such methods.

Input: the imbalanced dataset

Output: the prediction results on the dataset using the Multi-IM OAHO algorithm, where _p.mat is the prediction results, and _c.mat is the ground truth.

Usage example:

```
function runMultiImOAHO
javaaddpath('weka.jar');

p = genpath(pwd);
addpath(p, '-begin');
% record = 'testall.txt';
% save record record

dataset_list = {'Wine_data_set_indx_fixed'};

for p = 1:length(dataset_list)
    load(['data\', dataset_list{p},'.mat']);
    disp([dataset_list{p}, ' - numero dataset: ',num2str(p), ]);

    %Multi-IM+OAHO
    %input: traindata, trainlabel, testdata, testlabel
    %output: trainCostTime,predictCostTime,predictResult
    for d=1:5

        [Cost(d).classOAHOtr,Cost(d).classOAHOte,Pre(d).classOAHO] =
classOAHO([data(d).train,data(d).trainlabel],data(d).test);
```

```
    end

    save (['results/', dataset_list{p},'_',    'p', '.mat'], 'Pre');
    save (['results/', dataset_list{p},'_', 'c', '.mat'],    'Cost');

    clear Cost Pre Indx;

end
```

Reference:

Ghanem, A. S., Venkatesh, S. & West, G. (2010). Multi-class pattern classification in imbalanced data. International Conference on Pattern Recognition (ICPR), 2010 (PP. 2881-2884).

3.18 Multi-IM + A&O

Principles: PRMs-IM is a classification algorithm (originally designed) for binary imbalanced data. Let m be the ratio between the number of majority samples and that of the minority samples. PRMs-IM randomly divides the majority samples into m parts, next combines each part with all the minority instances, then trains a corresponding binary classifier. In the prediction phase, it uses weighted voting to ensemble the outputs of the m classifiers and makes the final prediction.

Multi-IM algorithm combines A&O and PRMs-IM, where PRMs-IM is adopted to train the classifier for A&O. Besides A&O, in our work, we also combine the OVA, OVO and OAHO decomposition methods with PRMs-IM to further investigate the performance of PRMs-IM for multi-class imbalance learning. Multi-IM+A&O is one of such methods.

Input: the imbalanced dataset

Output: the prediction results on the dataset using the Multi-IM A&O algorithm, where _p.mat is the prediction results, and _c.mat is the ground truth.

Usage example:

```
function runMultiImAO
javaaddpath('weka.jar');

p = genpath(pwd);
addpath(p, '-begin');
% record = 'testall.txt';
% save record record

dataset_list = {'Wine_data_set_indx_fixed'};

for p = 1:length(dataset_list)
    load(['data\', dataset_list{p},'.mat']);
    disp([dataset_list{p}, ' - numero dataset: ',num2str(p), ]);
```

```
    %Multi-IM+A&O
    %input: traindata, trainlabel, testdata, testlabel
    %output: trainCostTime,predictCostTime,predictResult
    for d=1:5

        [Cost(d).classAandOtr,Cost(d).classAandOte,Pre(d).classAandO]                    =
classAandO(data(d).train,data(d).trainlabel,data(d).test);

    end

    save (['results/', dataset_list{p},'_',    'p', '.mat'], 'Pre');
    save (['results/', dataset_list{p},'_', 'c', '.mat'],    'Cost');

    clear Cost Pre Indx;

end
```

Reference:

Ghanem, A. S., Venkatesh, S. & West, G. (2010). Multi-class pattern classification in imbalanced data. International Conference on Pattern Recognition (ICPR), 2010 (PP. 2881-2884).

4. Performance of Different Algorithms

In our DECOC paper, we have reported and analyzed the accuracy and efficiency performance of different algorithms. In Figures 4-7, we respectively report their accuracy performance using different evaluation metrics. The horizontal axis represents the accuracy value.

We emphasize that, there is no constant winner out of these algorithms. Depending on the specific datasets, the best imbalance classification algorithm varies. But in general, DECOC and DOVO achieve the best accuracy on more datasets than the others. But for efficiency considerations, they are among the slowest algorithms. We suggest users to read the following paper for more details.

Reference:

Jingjun Bi, Chongsheng Zhang*. (2018). An Empirical Comparison on State-of-the-art Multi-class Imbalance Learning Algorithms and A New Diversified Ensemble Learning Scheme. Knowledge-based Systems, 2018, Vol.158, pp. 81-93.

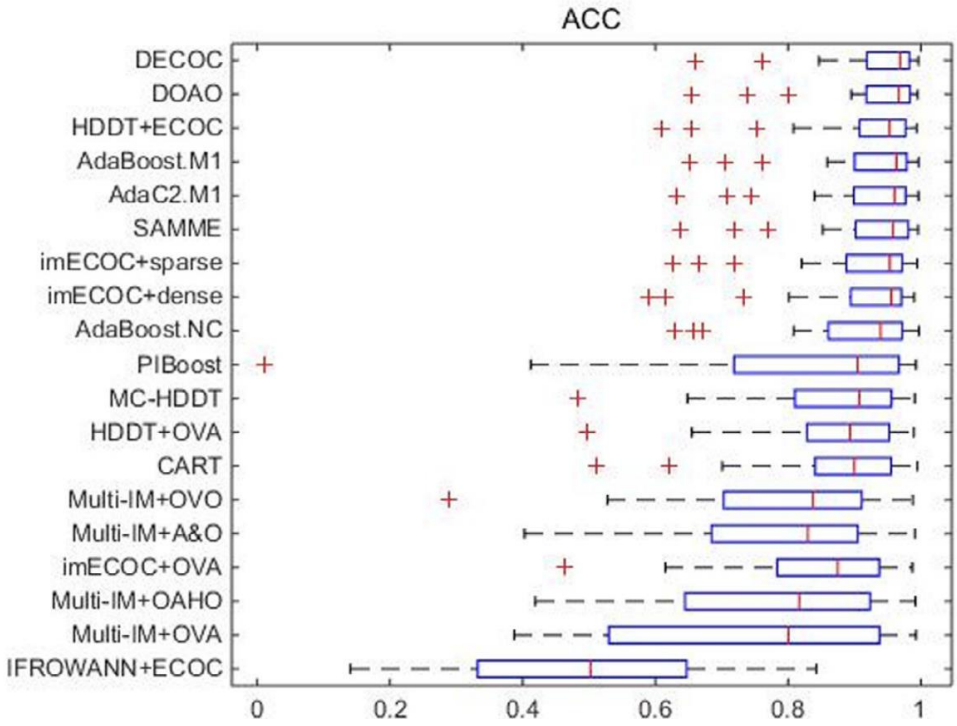

Figure 4. The overall accuracy performance of different algorithms on 19 datasets

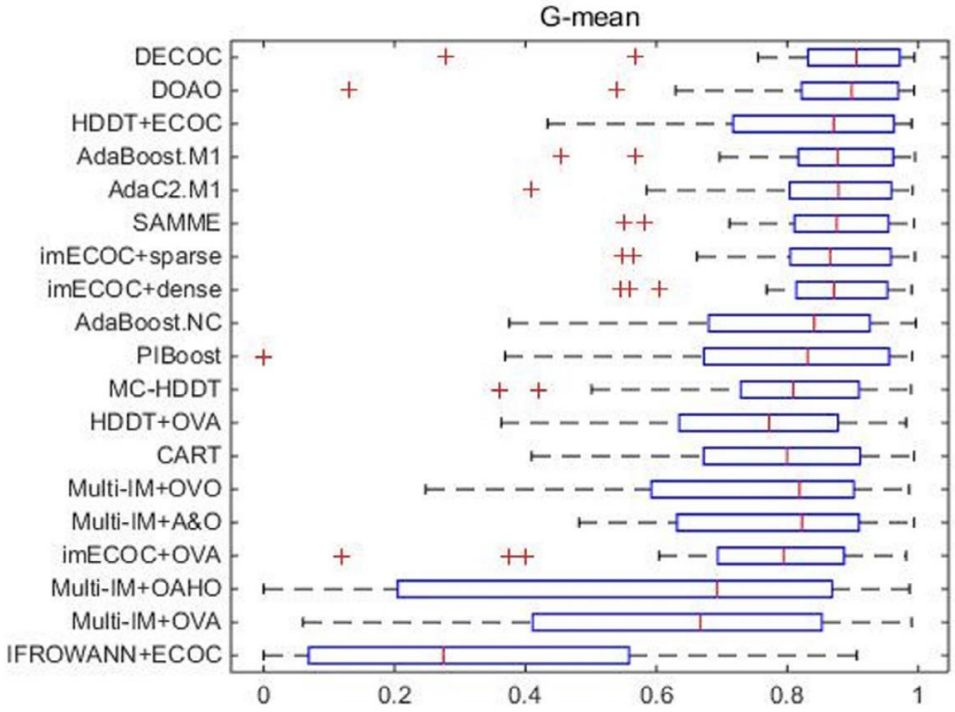

Figure 5. The G-mean performance of different algorithms on 19 datasets

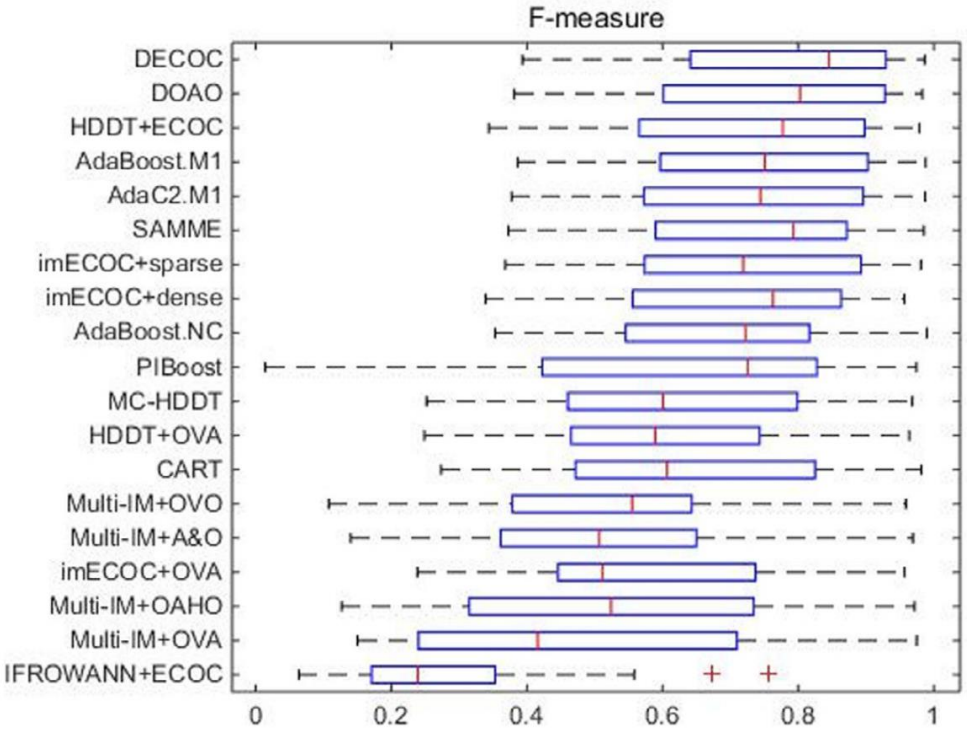

Figure 6. The F-measure performance of different algorithms on 19 datasets

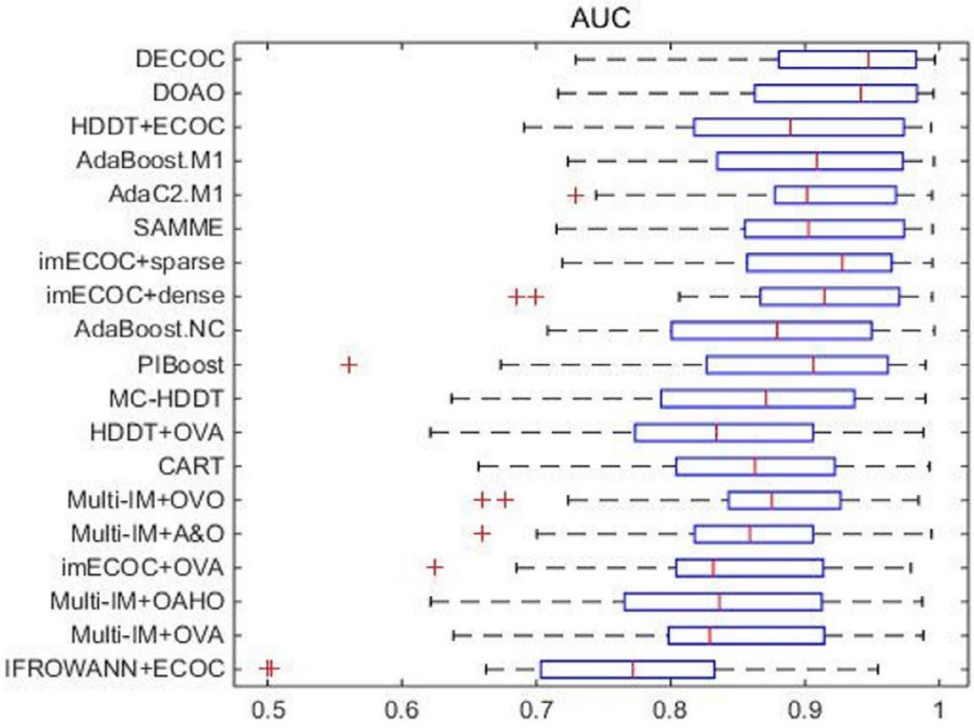

Figure 7. The AUC performance of different algorithms on 19 datasets