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 imbalance 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.

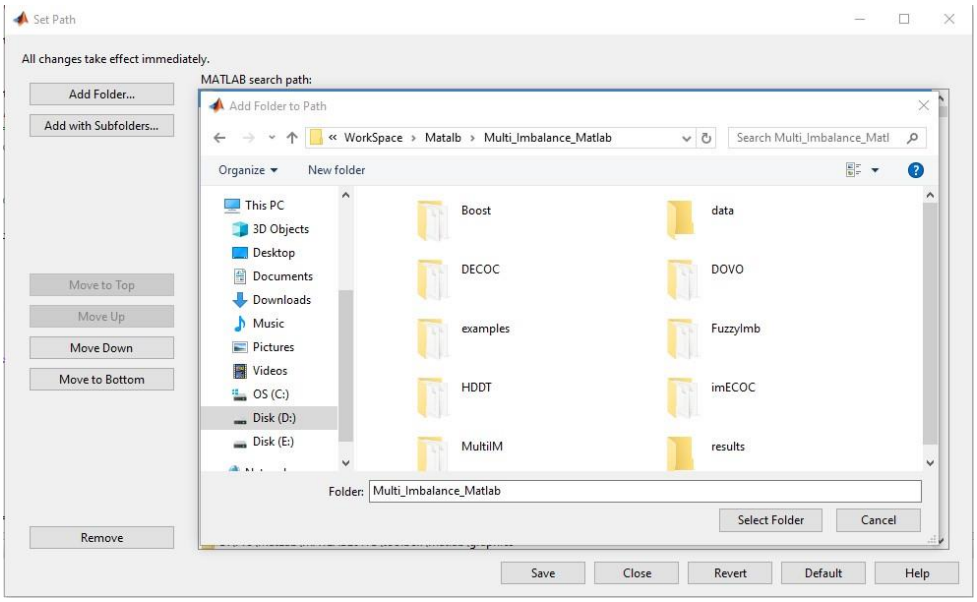

Figure 2. Adding the Multi-Imbalance package to the MATLAB path

Similarly, users should also add 'weka.jar' to the MATLAB path.

3. API Reference

3.1 AdaBoost.M1

[trainTime,testTime, predictedResults] = **adaBoostCartM1**(traindata, trainlabel, testdata, Max_Iter)

AdaBoost.M1 extends AdaBoost to the imbalance learning scenario, in the update of the samples' weights and the combination strategy of the base classifiers.

Table 1 : AdaBoost.M1

Parameters	**traindata** The data matrix in the training dataset.
	trainlabel The corresponding labels of each instance in the training dataset.
	testdata The test data matrix to be used in the testing phase (without label information).
	Max_Iter The maximum number of base classifiers to be built, which will be combined by the ensemble strategy.
Returns	**trainTime** The training time cost.
	testTime

	The prediction time cost.
	predictedResults The prediction results for testdata.

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

[trainTime,testTime, predictedResults] = **sammeCart**(traindata ,trainlabel ,testdata ,Max_Iter)

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.

Table 2 : SAMME

Parameters	**traindata** The data matrix in the training dataset.
	trainlabel The corresponding labels of each instance in the training dataset.
	testdata The test data matrix to be used in the testing phase (without label information).
	Max_Iter The maximum number of base classifiers to be built, which will be combined by the ensemble strategy.
Returns	**trainTime** The training time cost.
	testTime The prediction time cost.
	predictedResults The prediction results for testdata.

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

[trainTime,testTime , predictedResults] = **adaC2CartM1**(traindata ,trainlabel ,testdata ,Max_Iter ,C)

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.

Table 3 : AdaC2.M1

Parameters	**traindata** The data matrix in the training dataset.
	trainlabel The corresponding labels of each instance in the training dataset.
	testdata The test data matrix to be used in the testing phase (without label information).
	Max_Iter The maximum number of base classifiers to be built, which will be combined by the ensemble strategy.
	C the optimum cost setup of each class
Returns	**trainTime** The training time cost.
	testTime The prediction time cost.
	predictedResults The prediction results for testdata.

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

[trainTime,testTime,predictedResults]=**adaBoostCartNC**(traindata,trainlabel,testdata,Max_Iter,lama)

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.

Table 4 : AdaBoost.NC

Parameters	**traindata** The data matrix in the training dataset.
	trainlabel The corresponding labels of each instance in the training dataset.
	testdata The test data matrix to be used in the testing phase (without label information).
	Max_Iter The maximum number of base classifiers to be built, which will be combined by the ensemble strategy.
Returns	**trainTime** The training time cost.
	testTime The prediction time cost.
	predictedResults The prediction results for testdata.

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

[trainTime,testTime, predictedResults] = **PIBoostCart**(traindata,trainlabel,testdata,Max_Iter)

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.

Table 5 : PIBoost

	traindata The data matrix in the training dataset.
	trainlabel The corresponding labels of each instance in the training dataset.
	testdata

Parameters	The test data matrix to be used in the testing phase (without label information).
	Max_Iter The maximum number of base classifiers to be built, which will be combined by the ensemble strategy.
Returns	**trainTime** The training time cost.
	testTime The prediction time cost.
	predictedResults The prediction results for testdata.

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

[trainTime,testTime,prelabel] = **DECOC**(traindata, trainlabel, testdata, type, withw)

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.

Table 6 : DECOC

Parameters	**traindata** The data matrix in the training dataset.
	trainlabel The corresponding labels of each instance in the training dataset.
	testdata The test data matrix to be used in the testing phase (without label information).
	type The coding type
Returns	**trainTime** The training time cost.
	testTime The prediction time cost.
	prelabel

	The prediction results for testdata.

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. Knowledgebased Systems, 2018, Vol.158, pp. 81-93.

3.7 DOVO

[trainTime,testTime,prelabel,C] = **DOVO**(train, testdata, testlabel, kfold)

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.

Table 7 : DOVO

Parameters	**train** The data matrix in the training dataset and its corresponding labels.
	testdata The test data matrix to be used in the testing phase (without label information).
	testlabel The corresponding labels of each instance in the test dataset.
	kfold the parameter for cross validation.
Returns	**trainTime** The training time cost.
	testTime The prediction time cost.
	prelabel The prediction results for testdata.
	C Which contains the id of the classification algorithm chosen and the specific parameter needed by the KNN classifier.

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

prelabel = **fuzzyImbECOC**(traindata, trainlabel, testdata, testlabel, weightStrategy, gamma)

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.

Table 8 : fuzzyImbECOC

Parameters	**traindata** The data matrix in the training dataset.
	trainlabel The corresponding labels of each instance in the training dataset.
	testdata The test data matrix to be used in the testing phase (without label information).
	testlabel The corresponding labels of each instance in the test dataset.
	weightStrategy The configuration of different weight vectors assigned to all the classes.
	gamma A parameter to limit the number of instances which receive positive weights, its range is [0, 1]. For details, please see the reference.
Returns	**prelabel** The prediction results for testdata.

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

[trainTime,testTime,prelabel] = **HDDTOVA**(traindata,trainlabel,testdata,testlabel)

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.

Table 9 : HDDTOVA

Parameters	**traindata** The data matrix in the training dataset.
	trainlabel The corresponding labels of each instance in the training dataset.
	testdata The test data matrix to be used in the testing phase (without label information).
	testlabel The corresponding labels of each instance in the test dataset.
Returns	**trainTime** The training time cost.
	testTime The prediction time cost.
	prelabel The prediction results for testdata.

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

[trainTime,testTime,prelabel] = **HDDTECOC**(traindata,trainlabel,testdata,testlabel)

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.

Table 10 : HDDTECOC

Parameters	**traindata** The data matrix in the training dataset.
	trainlabel The corresponding labels of each instance in the training dataset.
	testdata The test data matrix to be used in the testing phase (without label information).
	testlabel

	The corresponding labels of each instance in the test dataset.
Returns	**trainTime** The training time cost.
	testTime The prediction time cost.
	prelabel The prediction results for testdata.

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

[trainTime,testTime,predictions] = **MCHDDT**(traindata,trainlabel,testdata,testlabel)

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.

Table 11 : MCHDDT

Parameters	**traindata** The data matrix in the training dataset.
	trainlabel The corresponding labels of each instance in the training dataset.
	testdata The test data matrix to be used in the testing phase (without label information).
	testlabel The corresponding labels of each instance in the test dataset.
Returns	**trainTime** The training time cost.
	testTime The prediction time cost.
	predictions

	The prediction results for testdata.

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

[trainTime,testTime,prelabel] = **imECOC**(traindata,trainlabel,testdata,type,withw)

The imECOC method is based upon ECOC, it considers the between-class and within-class imbalances, and assigns different weights to dichotomies according to their accuracy performance. It uses weighted distance for decoding, where the optimal dichotomy weights are obtained by minimizing a weighted loss in favor of the minority classes.

Table 12 : imECOC

Parameters	**traindata** The data matrix in the training dataset.
	trainlabel The corresponding labels of each instance in the training dataset.
	testdata The test data matrix to be used in the testing phase (without label information).
	type The coding type.
Returns	**trainTime** The training time cost.
	testTime The prediction time cost.
	prelabel The prediction results for testdata.

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 Multi-IM

prelabel = **multiIMCart**(traindata,trainlabel,testdata)

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.

Table 13 Multi-IM

Parameters	**traindata** The data matrix in the training dataset.
	trainlabel The corresponding labels of each instance in the training dataset.
	testdata The test data matrix to be used in the testing phase (without label information).
Returns	**prelabel** The prediction results for testdata.

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. Usage Examples

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.

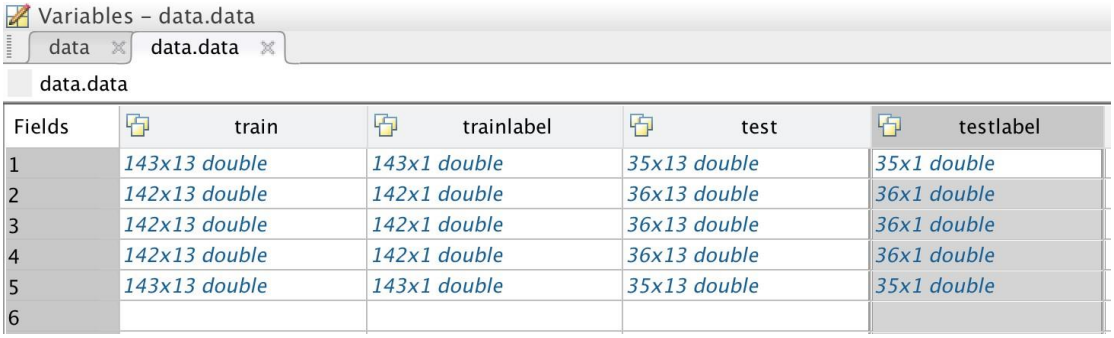

Figure 3. Elements of "Wine_data_set_index_fixed" data after loading into Matlab

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).

4.1 AdaBoost.M1

Usage example:

```
% function runAdaBoostM1.m
% javaaddpath('weka.jar');
p = genpath(pwd);
addpath(p, '-begin');

% note that each of our data has been split into 5 different training/test sets and fixed, as can be seen figure 3.
% as a ruuning example, after loading a dataset(e.g.,'Wine_data_set_indx_fixed.mat'), we will only use its first split hereafter.

load('Wine_data_set_indx_fixed.mat');
trainData=data(1).train;
trainLabel=data(1).trainlabel;
testData=data(1).test;

% the final predicted results for testData will be kept in predictedResults
% the meanings of the rest parameters can be found in the corresponding API reference
[trainTime,testTime,predictedResults] = adaBoostCartM1(traindata, trainlabel, testdata, 20)
```

4.2 SAMME

Usage example:

```
% function runSAMME.m
% javaaddpath('weka.jar');
p = genpath(pwd);
addpath(p, '-begin');

load('Wine_data_set_indx_fixed.mat');
trainData=data(1).train;
trainLabel=data(1).trainlabel;
testData=data(1).test;

% the final predicted results for testData will be kept in predictedResults
% the meanings of the rest parameters can be found in the corresponding API reference
[trainTime, testTime, predictedResults] = sammeCart(trainData, trainLabel, testData, 20);
```

4.3 AdaC2.M1

Usage example:

```
% function runAdaC2M1.m
% javaaddpath('weka.jar');
p = genpath(pwd);
addpath(p, '-begin');

load('Wine_data_set_indx_fixed.mat');
trainData=data(1).train;
trainLabel=data(1).trainlabel;
testData=data(1).test;

% the final predicted results for testData will be kept in predictedResults
% the meanings of the rest parameters can be found in the corresponding API reference
C0=testGA(traindata, trainlabel);
[trainTime ,testTime , predictedResults] = adaC2CartM1(traindata ,trainlabel ,testdata ,20 ,C0);
```

4.4 AdaBoost.NC

Usage example:

```
% function runAdaBoostNC.m
% javaaddpath('weka.jar');
p = genpath(pwd);
addpath(p, '-begin');

load('Wine_data_set_indx_fixed.mat');
trainData=data(1).train;
trainLabel=data(1).trainlabel;
testData=data(1).test;

% the final predicted results for testData will be kept in predictedResults
% the meanings of the rest parameters can be found in the corresponding API reference
[trainTime, testTime, predictedResults] = adaBoostCartNC (trainData, trainLabel, testData, 20, 2);
```

4.5 PIBoost

Usage example:

```
% function runPIBoost.m
% javaaddpath('weka.jar');
p = genpath(pwd);
addpath(p, '-begin');

load('Wine_data_set_indx_fixed.mat');
trainData=data(1).train;
trainLabel=data(1).trainlabel;
testData=data(1).test;

% the final predicted results for testData will be kept in predictedResults
% the meanings of the rest parameters can be found in the corresponding API reference
[trainTime, testTime, predictedResults] = PIBoost(trainData, trainLabel, testData, 20);
```

4.6 DECOC

Usage Example

```
% function runDECOC.m
% javaaddpath('weka.jar');
p = genpath(pwd);
addpath(p, '-begin');

load('Wine_data_set_indx_fixed.mat');
trainData=data(1).train;
trainLabel=data(1).trainlabel;
testData=data(1).test;

% the final predicted results for testData will be kept in predictedResults
% the meanings of the rest parameters can be found in the corresponding API reference
[trainTime, testTime, predictedResults] = DECOC(trainData, trainLabel, testData, 'sparse',1);
```

4.7 DOVO

Usage example:

```
% function runDOVO.m
% javaaddpath('weka.jar');
p = genpath(pwd);
addpath(p, '-begin');

load('Wine_data_set_indx_fixed.mat');
trainData=data(1).train;
trainLabel=data(1).trainlabel;
testData=data(1).test;
testLabel = data(1).testlabel;

% the final predicted results for testData will be kept in predictedResults
% the meanings of the rest parameters can be found in the corresponding API reference
[trainTime, testTime, predictedResults,C] = DOVO([trainData, trainLabel], testData, testLabel,
5);
```

4.8 FuzzyImbECOC

Usage example:

```
% function runFuzzyImbECOC.m
% javaaddpath('weka.jar');
p = genpath(pwd);
addpath(p, '-begin');

load('Wine_data_set_indx_fixed.mat');
trainData=data(1).train;
trainLabel=data(1).trainlabel;
testData=data(1).test;

% the final predicted results for testData will be kept in predictedResults
% the meanings of the rest parameters can be found in the corresponding API reference
[predictedResults] = fuzzyImbECOC(trainData, trainLabel, testData, testLabel, 'w6', 0.1);
```

4.9 HDDTOVA

Usage example:

```
% function runHDDTOVA.m
% javaaddpath('weka.jar');
p = genpath(pwd);
addpath(p, '-begin');

load('Wine_data_set_indx_fixed.mat');
trainData=data(1).train;
trainLabel=data(1).trainlabel;
testData=data(1).test;

% the final predicted results for testData will be kept in predictedResults
% the meanings of the rest parameters can be found in the corresponding API reference
[trainTime, testTime, predictedResults] = HDDTOVA(trainData,trainLabel,testData,testLabel);
```

4.10 HDDTECOC

Usage example:

```
% function runHDDTECOC.m
% javaaddpath('weka.jar');
p = genpath(pwd);
addpath(p, '-begin');

load('Wine_data_set_indx_fixed.mat');
trainData=data(1).train;
trainLabel=data(1).trainlabel;
testData=data(1).test;

% the final predicted results for testData will be kept in predictedResults
% the meanings of the rest parameters can be found in the corresponding API reference
[trainTime,testTime,predictedResults]=HDDTECOC(trainData,trainLabel, testData,testLabel);
```

4.11 MCHDDT

Usage example:

```
% function runMCHDDT.m
% javaaddpath('weka.jar');
p = genpath(pwd);
addpath(p, '-begin');

load('Wine_data_set_indx_fixed.mat');
trainData=data(1).train;
trainLabel=data(1).trainlabel;
testData=data(1).test;

% the final predicted results for testData will be kept in predictedResults
% the meanings of the rest parameters can be found in the corresponding API reference
[trainTime, testTime, predictedResults] = MCHDDT(trainData, trainLabel, testData, testLabel);
```

4.12 imECOC + sparse

Usage example:

```
% function runImECOCSparse.m
% javaaddpath('weka.jar');
p = genpath(pwd);
addpath(p, '-begin');

load('Wine_data_set_indx_fixed.mat');
trainData=data(1).train;
trainLabel=data(1).trainlabel;
testData=data(1).test;

% the final predicted results for testData will be kept in predictedResults
% the meanings of the rest parameters can be found in the corresponding API reference
[trainTime, testTime, predictedResults] = imECOC(trainData, trainLabel, testData, 'sparse' ,1);
```

4.13 imECOC + OVA

Usage example:

```
% function runImECOCOVA.m
% javaaddpath('weka.jar');
p = genpath(pwd);
addpath(p, '-begin');

load('Wine_data_set_indx_fixed.mat');
trainData=data(1).train;
trainLabel=data(1).trainlabel;
testData=data(1).test;

% the final predicted results for testData will be kept in predictedResults
% the meanings of the rest parameters can be found in the corresponding API reference
[trainTime, testTime, predictedResults] = imECOC(trainData, trainLabel, testData, 'OVA' ,1);
```

4.14 imECOC + dense

Usage example:

```
% function runImECOCDense.m
% javaaddpath('weka.jar');
p = genpath(pwd);
addpath(p, '-begin');

load('Wine_data_set_indx_fixed.mat');
trainData=data(1).train;
trainLabel=data(1).trainlabel;
testData=data(1).test;

% the final predicted results for testData will be kept in predictedResults
% the meanings of the rest parameters can be found in the corresponding API reference
[trainTime, testTime, predictedResults] = imECOC(trainData, trainLabel, testData, 'dense' ,1);
```

4.15 Multi-IM + OVA

Usage example:

```
% function runMultiImOVA.m
% javaaddpath('weka.jar');
p = genpath(pwd);
addpath(p, '-begin');

load('Wine_data_set_indx_fixed.mat');
trainData=data(1).train;
trainLabel=data(1).trainlabel;
testData=data(1).test;

% the final predicted results for testData will be kept in predictedResults
% the meanings of the rest parameters can be found in the corresponding API reference
[trainTime, testTime, predictedResults] = classOVA(trainData, trainLabel, testData);
```

4.16 Multi-IM + OVO

Usage example:

```
% function runMultiImOVO.m
% javaaddpath('weka.jar');
p = genpath(pwd);
addpath(p, '-begin');

load('Wine_data_set_indx_fixed.mat');
trainData=data(1).train;
trainLabel=data(1).trainlabel;
testData=data(1).test;

% the final predicted results for testData will be kept in predictedResults
% the meanings of the rest parameters can be found in the corresponding API reference
[trainTime, testTime, predictedResults] = classOAO(trainData, trainLabel, testData);
```

4.17 Multi-IM + OAHO

Usage example:

```
% function runMultiImOAHO.m
% javaaddpath('weka.jar');
p = genpath(pwd);
addpath(p, '-begin');

load('Wine_data_set_indx_fixed.mat');
trainData=data(1).train;
trainLabel=data(1).trainlabel;
testData=data(1).test;

% the final predicted results for testData will be kept in predictedResults
% the meanings of the rest parameters can be found in the corresponding API reference
[trainTime, testTime, predictedResults] = classOAHO(trainData, trainLabel, testData);
```

4.18 Multi-IM + A&O

Usage example:

```
% function runMultiImAandO.m
% javaaddpath('weka.jar');
p = genpath(pwd);
addpath(p, '-begin');

load('Wine_data_set_indx_fixed.mat');
trainData=data(1).train;
trainLabel=data(1).trainlabel;
testData=data(1).test;

% the final predicted results for testData will be kept in predictedResults
% the meanings of the rest parameters can be found in the corresponding API reference
[trainTime, testTime, predictedResults] = classAandO(trainData, trainLabel, testData);
```

5. 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. Knowledgebased Systems, 2018, Vol.158, pp. 81-93.

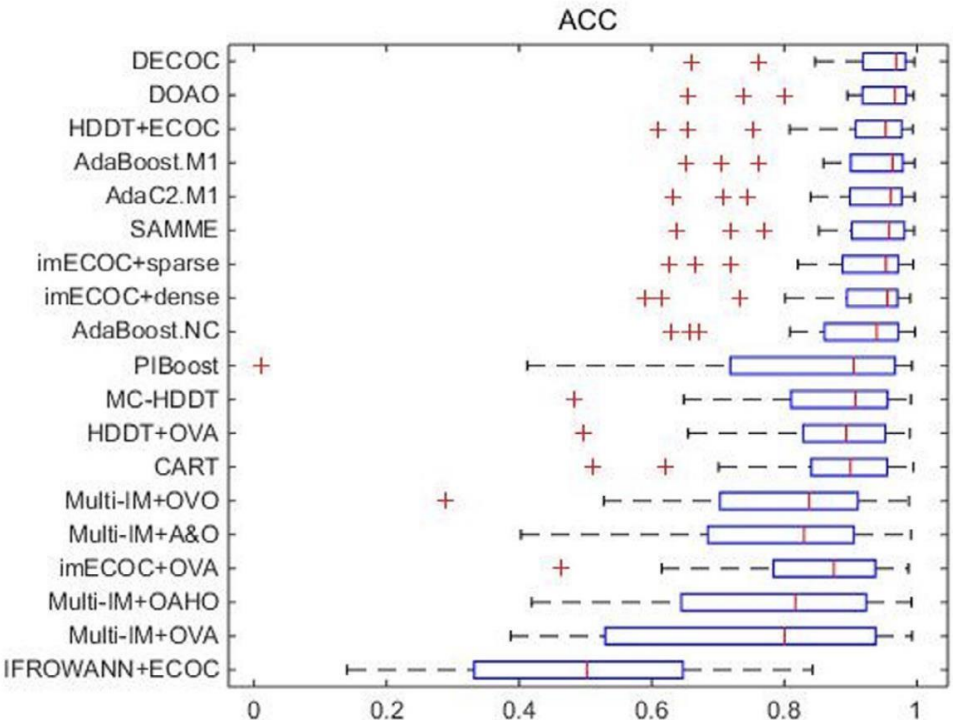

Figure 4. The overall accuracy performance of different algorithms on 19 datasets

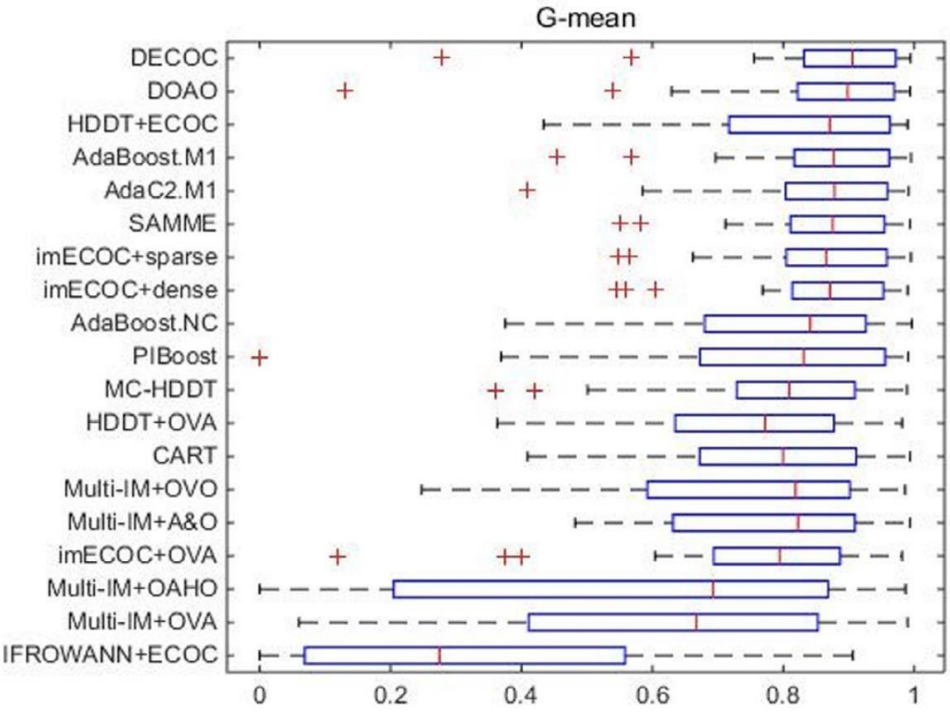

Figure 5. The G-mean performance of different algorithms on 19 datasets

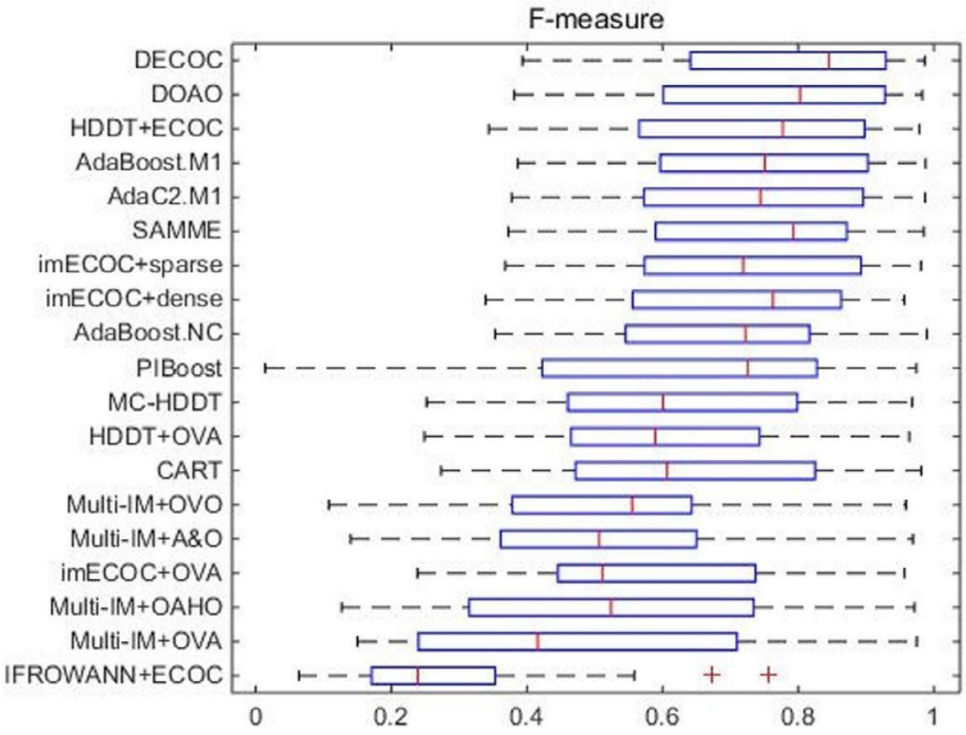

Figure 6. The F-measure performance of different algorithms on 19 datasets

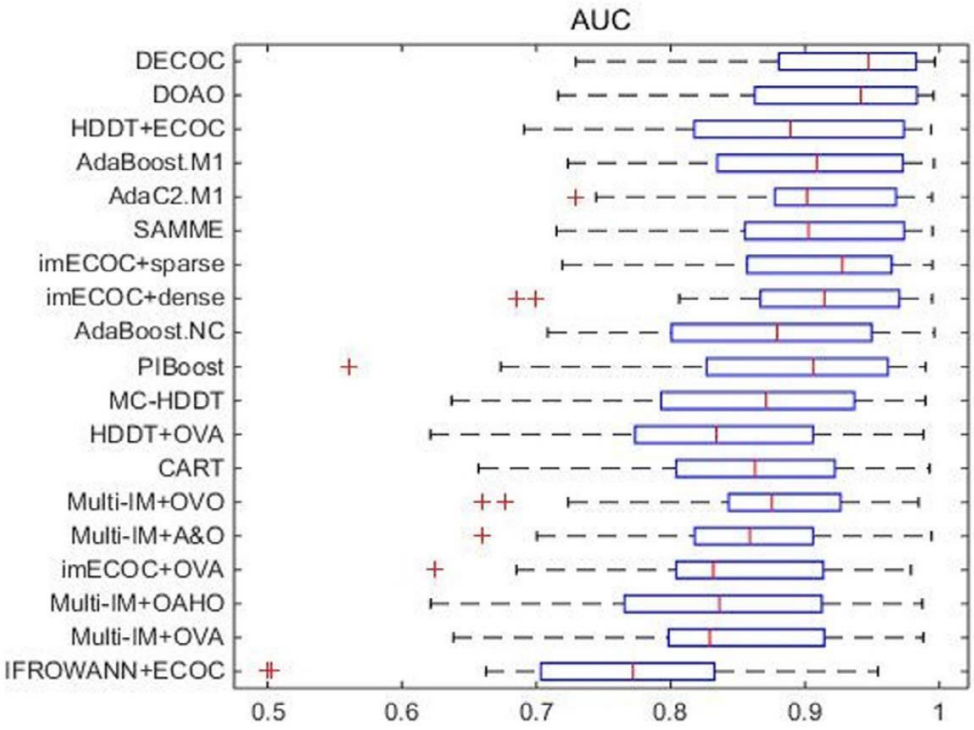

Figure 7. The AUC performance of different algorithms on 19 datasets