Hyperparameter settings of the baseline methods

All the hyperparameter settings of the baseline methods used to compare with our MSDT algorithm are described as follows. Some of the baseline methods, including *CART-ML*, ExtraTree-ML, RandomForest-ML, ExtraTrees-ML, ML-KNN and *DNN-MLP-ML*, are implemented in Python with the Scikit-learn package; iSOUP-Tree is implemented in Python with the Scikit-multiflow package(Montiel et al. 2018); MLTSVM, RAkEL and MVoting are implemented in Python with the Scikit-multilearn package (Szymanski & Kajdanowicz 2019); and CDT is implemented in Python with the Meka package. To get the optimal result for each experiment, we describe their hyperparameter settings in order as follows.

First, in *CART-ML*, we acquire the optimal hyperparameters from those that the measures for the best splitting attribute = {gini, entropy}, the minimum number of samples required to split an internal node same as *minqty* of MSDT is increased from 2 to 18 in increments of 4, and the other good default settings from the DecisionTreeClassifier function of Scikit-learn. Second, in ExtraTree-ML, we acquire the optimal hyperparameters from those that the measures for the best splitting attribute and the minimum number of samples required to split are set as *CART-ML* did; the maximum number of features to consider when looking for the best split is all the features if the strategy used to choose the split at each node is set as 'best', otherwise the functions used to calculate the maximum number of features = {auto, sqrt, log2} if the strategy is set as 'random'; and the other good default settings from the ExtraTreeClassifier function of Scikit-learn. Third, in both of RandomForest-ML and ExtraTrees-ML, we acquire the optimal hyperparameters from those that the measures for the best splitting attribute and the minimum number of samples required to split are set as *CART-ML* did, whether bootstrap samples are used when building trees = {true, false}, whether to use out-of-bag samples to estimate the generalization accuracy = {true, false} if the bootstrap samples were used, the number of trees in the forest = 100, and the other good default settings from the RandomForestClassifier function and the ExtraTreesClassifier function respectively in Scikit-learn. Fourth, in ML-KNN, we acquire the optimal hyperparameters from those that the number of neighbors is increased from 1 to 10 in increments of 1, the weight function used in prediction = {*uniform*, *distance*}, and the other good default settings from the KNeighborsClassifier function of Scikit-learn.

Fifth, *DNN-MLP-ML*, the Deep Multi-layer Perceptron classifier, optimizes the log-loss function using three well-known optimization algorithms, *adam* (namely *DNN-MLP-ML-adam*) (Kingma & Ba 2014), stochastic gradient descent (namely *DNN-MLP-ML-sgd*) and LBFGS (namely *DNN-MLP-ML-lbfgs*). As for the common *DNN-MLP* part,

Bengio (2009) deemed that the number of hidden layers of deep architectures in the neural network is 3 or more. Schmidhuber (2014) mentioned that no rule about at which depth of CAP (Credit Assignment Path, i.e. the number of hidden layers plus 1) for Shallow Learning to end and for Deep Learning to begin. But, he proposed that the depth > 10 is a problem of very Deep Learning definitely. Therefore, we have initially pretrained at the depth of CAP from 4 to 6 (the deep model), each hidden layer of which has the number of hidden neurons contained in {13, 16, 32, 64, 128}, to acquire the optimal hyperparameters for each of the three classification lifecycles. And then, each optimal hyperparameters are used to train in the deeper CAP from the depth 11 to 16 (the deeper model). Under each configuration of the optimal hyperparameters, *DNN-MLP-ML* has produced the best accuracy under the following architecture of the hidden layers. We denote the architecture as the pair, (n, m), where n: the number of CAP layers and m: the number of hidden neurons. Those pairs for adam in the three classification lifecycles in order are (3, 128), (3, 32) and (3, 16); those for sgd are (4, 128), (4, 128) and (5, 128); and those for lbfgs are (4, 128), (5, 32) and (3, 16). We will choose the best accuracy from the accuracies predicted by both of the deep model and the deeper model in each lifecycle. As for the *adam* part, learning rate is increased from 1.e-05 to 1.e-01 in increments of 10, α is increased from 1.e-05 to 1.e+03 in increments of 100, and some setting suggestion from Kingma and Ba (2014), including β1= {0, 0.9}, β2 = {0, 0.999}, ε = {0, 1e-08}, the maximum number of iterations = 250, early stopping = true, and the other good default settings from the MLPClassifier function for *adam* in the Scikit-learn package. As for the *sgd* part, learning rate mode = {constant, inverse-scaling, adaptive}, the initial learning rate = 0.001, α is increased from 1.e-05 to 1.e+03 in increments of 100, whether to use Nesterov's momentum = {true, false}, and momentum = 0.9; we set the same maximum number of iterations and early stopping as *DNN-MLP-ML-adam* did, and the other good default settings from the MLPClassifier function for *sgd* in Scikit-learn. As for the *lbfgs* part, α is increased from 1.e-05 to 1.e+03 in increments of 100; we set the same maximum number of iterations as *DNN-MLP-ML-adam* did, and the other good default settings from the MLPClassifier function for *lbfgs* in Scikit-learn.

Sixth, in iSOUP-Tree, under that learning_ratio_const is set as 'true', we acquire the optimal hyperparameters from those that learning_ratio_perceptron is increased from 1.e-04 to 1.e-01 in increments of 10, learning_ratio_decay is increased from 1.e-04 to 1.e-02 in increments of 10, and leaf_prediction = {'perceptron', 'adaptive', 'mean'}. Otherwise, under that learning_ratio_const is set as 'false', we acquire the optimal hyperparameters from those that learning_ratio_perceptron is increased from 1.e-04 to 1.e-01 in increments of 10, and leaf_prediction = {'perceptron', 'adaptive', 'mean'}. Additionally, the other good default settings above are both set in the

MultiTargetRegressionHoeffdingTree function in Scikit-multiflow. Seventh, in MLTSVM, under threshold_sel = 1e-06 and the other good default settings from the MLTSVM function in Scikit-multilearn, we acquire the optimal hyperparameters from those that threshold = {2^m * threshold_sel| m is an integer, m = -4..4}, c_k = {2^i| i is an integer, i = -1..1} and sor_omega = {j / 2| j is an integer, j = 1.. 3}. Eighth, in RAkEL, under base_classifier_require_dense = {True, True}, model_count = 10 and the other good default settings from the RakelO function in Scikit-multilearn, we acquire the optimal hyperparameters from those that base_classifier = {KNN, CART, NaiveBayes, SVM}, and labelset_size is increased from 2 to 6 in increments of 1. Nineth, in MVoting, under the settings: the classifier is the ClassifierChain algorithm and the label space clustering method is the FixedLabelSpaceClusterer algorithm, and **require_dense =** {True, True}, we acquire the optimal hyperparameters from that ClassifierChain's base_classifiers = {KNN, CART, NaiveBayes, SVM}. Finally, in CDT, under the total number of iterations = 1,000 and the other good default settings from the CDT function in Meka, we acquire the optimal hyperparameters from that the base classifiers = {CART, NaiveBayes, SVM}.

References

Bengio Y (2009) Learning Deep Architectures for AI. Found Trends Mach Learn 2(1):1--127

Kingma DP, Ba JL (2014) Adam: A method for stochastic optimization. arXiv preprint 1412.6980

Montiel J, Read J, Bifet A, Abdessalem T (2018) Scikit-Multiflow: A Multioutput Streaming Framework. J Mach Learn Res 19(72):1—5

Schmidhuber J (2015) Deep learning in neural networks: An overview. Neural Netw 61: 85–-117

Szymanski P, Kajdanowicz T (2019) Scikit-multilearn: a scikit-based Python environment for performing multi-label classification. J Mach Learn Res 20(1):209--230