ECOC Toolkit

This toolkit is based on **Python 3.6.3**, also based on **numpy** and **scikit-learn**. If you want to use this toolkit, please install these 2 packages first.

Provide by zhong.t.y@qq.com & fengkaijie1995@foxmail.com, authorized by lkhqz@xmu.edu.cn.

NOTE: Welcome to use our toolkit. If you find any faults in this toolkit, please contact with zhong.t.y@qq.com as soon as possible. We will be very very very glad to hear from you.

This toolkit provide:

Encoding: OVA, OVO, Dense Random, Sparse Random, Discriminant ECOC, ECOC-ONE, Agglomerative ECOC, Centroid loss ECOC.
Decoding: 'HD' - Hamming Decoder. 'IHD' - Inverse Hamming Decoder. 'LD' - Laplacian Decoder. 'ED' - Euclidean Decoder. 'AED' - Attenuated Euclidean Decoder. 'RED' - Ratio Euclidean Decoder. 'EuD' - Euler Decoder. 'LLB' - Linear Loss Based Decoder. 'ELB' - Exponential Loss Based Decoder. 'LLW' - Linear Loss Weighted Decoder. 'ELW' - Exponential Loss Weighted Decoder. 'PD' - Probabilistic Decoder (Coming soon).
Data Complexity: 'F1' - Maximum Fisher's discriminant ratio (noted as F1) 'F2' - Volume of overlap region (noted as F2) 'F3' - Maximal (individual) feature efficiency (noted as F3) 'N1' - Fraction of points on class boundary (noted as N1) 'N2' - Ratio of average intra/inter class nearest neighbor distance (noted as N2) 'N3' - Error rate of 1 nearest neighbor classifier (noted as N3)
Feature Select: BSSWSS, Variance, F1, F2, F3.

Encoding:

CodeMatrix.py
Functions:
ova: Get a One-vs-All matrix.
ovo: Get a One-vs-One matrix.
dense_rand: Get a Dense Random matrix.
sparse_rand: Get a Sparse Random matrix.
decoc: Get a Discriminant ECOC matrix.
agg_ecoc: Get an Agglomerative matrix.
cl_ecoc: Get a Centroid loss ECOC, which use regressors as base estimators.
ecoc_one: Get an Optimal node embedded ECOC.

Descriptions:
ova (X, y): Get a One-vs-All matrix.
Parameters:
X: {array-like, sparse matrix}, shape = [n_samples, n_features]
Training vector, where n_samples in the number of samples and n_features is the number of features.
y: array-like, shape = [n_samples]
Target vector relative to X
Returns:
M: 2-d array, shape = [n_classes, n_dichotomies]
The coding matrix.
Functions ovo, dense_rand, sparse_rand, decoc, agg_ecoc, cl_ecoc have the same structure with ova.
ecoc_one(train_X, train_y, valid_X, valid_y, estimator): Get an Optimal node embedded ECOC.
Parameters:
train_X: {array-like, sparse matrix}, shape = [n_samples, n_features]
Training vector, where n_samples in the number of samples and n_features is the number of features.
train_y: array-like, shape = [n_samples]
Target vector relative to train_X
valid_X: {array-like, sparse matrix}, shape = [n_samples, n_features]
Validate vector, where n_samples in the number of samples and n_features is the number of features.
valid_y: array-like, shape = [n_samples]
Target vector relative to valid_X
estimator: classifier object.
Classifier used in the validating process.
Returns:
M: 2-d array, shape = [n_classes, n_dichotomies]
The coding matrix.

Decoding:

Decoder.py
Functions:
get_decoder: Get a decoder object.

Descriptions:
get_decoder(dec):
Get a decoder object.
Parameters:
dec: str
Indicates a kind of decoder. Cognitive dec list below.
'HD' - Hamming Decoder.
'IHD' - Inverse Hamming Decoder.
'LD' - Laplacian Decoder.
'ED' - Euclidean Decoder.
'AED' - Attenuated Euclidean Decoder.
'RED' - Ratio Euclidean Decoder.
'EuD' - Euler Decoder.
'LLB' - Linear Loss Based Decoder.
'ELB' - Exponential Loss Based Decoder.
'LLW' - Linear Loss Weighted Decoder.
'ELW' - Exponential Loss Weighted Decoder.
'PD' - Probabilistic Decoder (Coming soon).

Returns:
o: object
A decoder object.

Classes:
Decoding.Decoder. **Decoder**
ECOC Decoder
Methods:
decode(Y, M): decode Y (predict matrix), by M (code matrix), into distance matrix.

Description:
decode(Y, M): decode Y (predict matrix), by M (code matrix), into distance matrix.
Parameters:
Y: 2-d array, shape = [n_samples, n_dichotomies]
M: 2-d array, shape = [n_classes, n_dichotomies]
Returns:
D: 2-d array, shape = [n_samples, n_classes]

Classifiers:

BaseClassifier.py:
Functions:
get_base_clf: Get classifiers from scikit-learn [1].

Descriptions:
get_base_clf(base, adaboost=False):
    Get classifiers from scikit-learn.

    Parameters:
        base: str, indicates classifier, alternative str list below.
            'KNN' - K Nearest Neighbors (sklearn.neighbors.KNeighborsClassifier)
            'DTree' - Decision Tree (sklearn.tree.DecisionTreeClassifier)
            'SVM' - Support Vector Machine (sklearn.svm.SVC)
            'Bayes' - Naive Bayes (sklearn.naive_bayes.GaussianNB)
            'Logi' - Logistic Regression (sklearn.linear_model.LogisticRegression)
            'NN' - Neural Network (sklearn.neural_network.MLPClassifier)
        adaboost: bool, Whether to use adaboost to promote the classifier
            True - use.
            False - don't use

    Return:
        model: object, A classifier object.

ECOCClassifier.py:
Classes:
Classifiers.ECOCClassifier. **SimpleECOCClassifier**
A simple ECOC classifier

Parameters:
    estimator: object
        unfitted base classifier object.
    code_matrix: 2-d array
        code matrix (Classes×Dichotomies).
    decoder: str
        indicates the type of decoder, get a decoder object immediately when initialization.
        For more details, check Decoding.Decoder.get_decoder.
    soft: bool, default True.
        Whether to use soft distance to decode.

Attributes:

estimator_type: str, {'decision_function','predict_proba'}
which type the estimator belongs to.
'decision_function' - predict value range (-∞,+∞)
'predict_proba' - predict value range [0,1]
classes_: set
the set of labels.
estimators_: 1-d array
trained classifers.

Methods:
fit(X, y): Fit the model according to the given training data.
predict(X): Predict class labels for samples in X.
fit_predict(X, y, test_X): fit(X, y) then predict(X_test).

Descriptions:
fit(X, y): Fit the model according to the given training data.
Parameters:
X: {array-like, sparse matrix}, shape = [n_samples, n_features]
Training vector, where n_samples in the number of samples and n_features is the number of features.
y: array-like, shape = [n_samples]
Target vector relative to X
Returns:
self: object
Returns self.

predict(X): Predict class labels for samples in X.
Parameters:
X: {array-like, sparse matrix}, shape = [n_samples, n_features]
Samples.
Returns:
C: array, shape = [n_samples]
Predicted class label per sample.

fit_predict(X, y, test_X): fit(X, y) then predict(X_test).
Parameters:
X: {array-like, sparse matrix}, shape = [n_samples, n_features]
Training vector, where n_samples in the number of samples and n_features is the number of features.
y: array-like, shape = [n_samples]
Target vector relative to
X_test: {array-like, sparse matrix}, shape = [n_samples, n_features]
Samples.
Returns:

C: array, shape = [n_samples]

Predicted class label per sample.

Notes: This is a combination of two methods fit & predict, with X, y for fit and X_test for predict.

Run fit first and then run predict

DataComplexity:

datacomplexity.py
Functions:
get_data_complexity:

Descriptions:
get_data_complexity(dc_code):
Get a data complexity object by dc_code.
Parameters:
dc_code: str
indicate the type of data complexity to be returned.
Recognizable de_code list below:
'F1' - Maximum Fisher's discriminant ratio (noted as F1)
'F2' - Volume of overlap region (noted as F2)
'F3' - Maximal (individual) feature efficiency (noted as F3)
'N1' - Fraction of points on class boundary (noted as N1)
'N2' - Ratio of average intra/inter class nearest neighbor distance (noted as N2)
'N3' - Error rate of 1 nearest neighbor classifier (noted as N3)

Returns:
o: object
data complexity object.

Classes:
DataComplexity.datacomplexity. **DataComplexity:**
The data complexity measure complexities of binary-class-data.
Methods:
score(X, y): Return the complexity score of X after checking X & y.
SMALL SCORES MEAN LOW DATA COMPLEXITIES.
_score(X, y): Return the complexity score of X, should be implemented by subclasses.
_check_Xy(X, y): Check if X and y meet demands.

Description:
score(X, y): Return the complexity score of X after checking X & y.
Parameters:
X: {array-like, sparse matrix}, shape = [n_samples, n_features]
Training vector, where n_samples in the number of samples and n_features is the number of features.
y: array-like, shape = [n_samples]
Target vector relative to X
Notes:

The public method is socre(X, y), of which the returning SMALL SCORES MEAN LOW DATA COMPLEXITIES.

FeatureSelection:

FeatureSelector.py
Classes:
FeatureSelection.FeatureSelector. **FeatureSelector**
Select k best features.
Parameters:
k: int or 'all'
Number of top features to select. The “all” option bypasses selection, for use in a parameter search.

Attributes:
feature_length: int
Number of features.
scores_: array-like, shape=(n_features,)
Scores of features.
ranking_indices: array-like, shape=(n_features,)
Ranking feature indices by scores.
support: array-like, shape=(k,)
Indices of k best (high scores) features, ranking by indices.
mask: array-like of bool, shape=(n_features,)
Indicates whether features are selected. An array full of bool, total k Trues.

Methods:
fit(X, y): Run score function on (X, y) and get the appropriate features.
transform(X): Reduce X to the selected features.
fit_transform(X, y): Fit to data, then transform it.
get_support(indices=False): Get a mask, or integer index, of the features selected
get_ranking_indices(): Get ranked feature indices by scores.

Descriptions:
fit(X, y): Run score function on (X, y) and get the appropriate features.
Parameters:
X : array-like, shape = [n_samples, n_features]
The training input samples.
y : array-like, shape = [n_samples]
The target values (class labels in classification, real numbers in regression).
Returns:
self : object
Returns self.

transform(X): Reduce X to the selected features.
Parameters:

X : array of shape [n_samples, n_features]
The input samples.

Returns:

X_r : array of shape [n_samples, n_selected_features]
The input samples with only the selected features.

fit_transform(X, y): Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters:

X : numpy array of shape [n_samples, n_features]
Training set.

y : numpy array of shape [n_samples]
Target values.

Returns:

X_new : numpy array of shape [n_samples, n_features_new]
Transformed array.

get_support(indices=False): Get a mask, or integer index, of the features selected.

Parameters:

indices : bool (default False)
If True, the return value will be an array of integers, rather than a boolean mask.

Returns:

support : array
An index that selects the retained features from a feature vector.
If indices is False, this is a boolean array of shape [# input features], in which an element is True iff its corresponding feature is selected for
retention. If indices is True, this is an integer array of shape
[# output features] whose values are indices into the input feature vector.

get_ranking_indices(): Get ranked feature indices by scores.

Returns:

ranking_indices: array-like
Feature indices ranked by feature scores in a descending order.
Top elements of the array got high scores.

FeatureSelection.FeatureSelector. **BSSWSS**
FeatureSelection.FeatureSelector. **VarianceSelector**
FeatureSelection.FeatureSelector. **DataComplexitySelector**

BSSWSS, a features select algorithm from:

AC Lorena, IG Costa, N Spolaôr, MCPD Souto. Analysis of complexity indices for classification problems: Cancer gene expression data [J]. Neurocomputing, 2012, 75(1): 33-42