IntelELM

Release 1.1.1

Thieu

Mar 27, 2024

QUICK START:

IntelELM: A Python Framework for Intelligent Metaheuristic-based Extreme Learning Machine

IntelELM (Intelligent Metaheuristic-based Extreme Learning Machine) is a Python library that implements a framework for training Extreme Learning Machine (ELM) networks using Metaheuristic Algorithms. It provides a comparable alternative to the traditional ELM network and is compatible with the Scikit-Learn library. With IntelELM, you can perform searches and hyperparameter tuning using the functionalities provided by the Scikit-Learn library.

- **Free software:** GNU General Public License (GPL) V3 license
- **Provided Estimator**: ElmRegressor, ElmClassifier, MhaElmRegressor, MhaElmClassifier
- **Total Optimization-based ELM Regression**: > 200 Models
- **Total Optimization-based ELM Classification**: > 200 Models
- **Supported datasets**: 54 (47 classifications and 7 regressions)
- **Supported performance metrics**: >= 67 (47 regressions and 20 classifications)
- **Supported objective functions (as fitness functions or loss functions)**: >= 67 (47 regressions and 20 classifications)
- **Documentation:** https://intelelm.readthedocs.io/en/latest/
- **Python versions:** >= 3.7.x
- **Dependencies:** numpy, scipy, scikit-learn, pandas, mealpy, permetrics

CHAPTER

ONE

INSTALLATION

There are so many ways to install our library. For example:

- Install from the PyPI release:

```
$ pip install intelelm==1.1.1
```

- Install directly from source code:

```
$ git clone https://github.com/thieu1995/intelelm.git
$ cd intelelm
$ python setup.py install
```

- In case, you want to install the development version from Github:

```
$ pip install git+https://github.com/thieu1995/intelelm
```

After installation, you can check the version of IntelELM:

```
$ python
>>> import intelelm
>>> intelelm.__version__
```

CHAPTER

TWO

TUTORIALS

2.1 1) Getting started in 30s

In the example below, we will apply the traditional ELM model to the diabetes prediction problem. This dataset is already available in our library. The process consists of the following steps:

- Import libraries
- Load and split dataset
- Scale dataset
- Define the model
- Train the model
- Test the model
- Evaluate the model

```
## Import libraries
import numpy as np
from intelelm import get_dataset, ElmRegressor

## Load dataset
data = get_dataset("diabetes")
data.split_train_test(test_size=0.2, random_state=2)
print(data.X_train.shape, data.X_test.shape)

## Scale dataset
data.X_train, scaler_X = data.scale(data.X_train, scaling_methods=('minmax', ))
data.X_test = scaler_X.transform(data.X_test)

data.y_train, scaler_y = data.scale(data.y_train, scaling_methods=('minmax', ))
data.y_test = scaler_y.transform(np.reshape(data.y_test, (-1, 1)))

## Define the model
model = ElmRegressor(hidden_size=10, act_name="elu", seed=42)

## Test the model
model.fit(data.X_train, data.y_train)

## Test the model
y_pred = model.predict(data.X_test)
```

(continues on next page)

(continued from previous page)

```
## Evaluate the model
print(model.score(data.X_test, data.y_test, method="RMSE"))
print(model.scores(data.X_test, data.y_test, list_methods=("RMSE", "MAPE")))
print(model.evaluate(y_true=data.y_test, y_pred=y_pred, list_metrics=("MAPE", "R2", "NSE
↪")))
```

As you can see, it is very similar to any other Estimator model in the Scikit-Learn library. They only differ in the model definition part. In the provided example, we used the ElmRegressor from the library, which is specifically designed for Extreme Learning Machines. However, the overall workflow follows the familiar pattern of loading data, preprocessing, training, and evaluating the model.

2.2 2) Model Definition

Metaheuristic Optimization-based ELM model If you want to use the Whale Optimization-based ELM (WO-ELM) model, you can change the model definition like this:

```
from intelelm import MhaElmRegressor

opt_paras = {"name": "WOA", "epoch": 100, "pop_size": 30}
model = MhaElmRegressor(hidden_size=10, act_name="elu", obj_name="MSE",
                optimizer="OriginalWOA", optimizer_paras=opt_paras, verbose=False,␣
↪seed=42)
```

In the example above, I had to import the MhaElmRegressor class. This is the class that contains all Metaheuristics-based ELM models for regression problems. Then, I defined parameters for the Whale Optimization algorithm. And I defined parameters for the Whale Optimization-based ELM model.

What about hybrid model for Classification problem

In case you want to use the model for a classification problem, you need to import either the ElmClassifier class (this is the traditional ELM model) or the MhaElmClassifier class (these are hybrid models combining metaheuristics algorithms and ELM networks).

```
from intelelm import ElmClassifier

model = ElmClassifier(hidden_size=10, act_name="elu", seed=42)
```

```
from intelelm import ElmClassifier

opt_paras = {"name": "GA", "epoch": 100, "pop_size": 30}
model = MhaElmClassifier(hidden_size=10, act_name="elu", obj_name="BSL",
                optimizer="BaseGA", optimizer_paras=opt_paras, verbose=False, seed=42)
```

2.3 3) Data Preparation

If you want to use your own data, it's straightforward. You won't need to load the data into our Data class. However, you'll need to use the Scikit-Learn library to split and scale the data.

```
### Step 1: Importing the libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from intelelm import MhaElmClassifier

#### Step 2: Reading the dataset
dataset = pd.read_csv('Position_Salaries.csv')
X = dataset.iloc[:, 1:2].values         # This is features
y = dataset.iloc[:, 2].values           # This is output

#### Step 3: Next, split dataset into train and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True,␣
↪random_state=100)

#### Step 4: Feature Scaling
scaler_X = MinMaxScaler()
scaler_X.fit(X_train)
X_train = scaler_X.transform(X_train)
X_test = scaler_X.transform(X_test)

le_y = LabelEncoder()       # This is for classification problem only
le_y.fit(y)
y_train = le_y.transform(y_train)
y_test = le_y.transform(y_test)

#### Step 5: Fitting ELM-based model to the dataset

##### 5.1: Use standard ELM model for classification problem
classifer = ElmClassifier(hidden_size=10, act_name="tanh")

##### 5.2: Use Metaheuristic-based ELM model for classification problem
print(MhaElmClassifier.SUPPORTED_OPTIMIZERS)
print(MhaElmClassifier.SUPPORTED_CLS_OBJECTIVES)
opt_paras = {"name": "GA", "epoch": 10, "pop_size": 30}
classifier = MhaElmClassifier(hidden_size=10, act_name="elu", obj_name="KLDL", optimizer=
↪"BaseGA", optimizer_paras=opt_paras, seed=42)

#### Step 6: Traint the model
classifer.fit(X_train, y_train)

#### Step 7: Predicting a new result
y_pred = regressor.predict(X_test)

y_pred_cls = classifier.predict(X_test)
y_pred_label = le_y.inverse_transform(y_pred_cls)

#### Step 8: Calculate metrics using score or scores functions.
```

(continues on next page)

(continued from previous page)

```
print("Try my AS metric with score function")
print(regressor.score(data.X_test, data.y_test, method="AS"))

print("Try my multiple metrics with scores function")
print(classifier.scores(data.X_test, data.y_test, list_methods=["AS", "PS", "F1S", "CEL",
↪ "BSL"]))
```

A real-world dataset contains features that vary in magnitudes, units, and range. We would suggest performing normalization when the scale of a feature is irrelevant or misleading. Feature Scaling basically helps to normalize the data within a particular range.

2.4 4) Scikit-Learn Integration

There's no need to delve further into this issue. The classes in the IntelELM library inherit from the BaseEstimator class from the Scikit-Learn library. Therefore, the features provided by the Scikit-Learn library can be utilized by the classes in the IntelELM library.

In the example below, we use the Whale Optimization-based ELM model as the base model for the recursive feature selection method for feature selection problem.

```
# import necessary class, modules, and functions
from intelelm import Data, MhaElmRegressor
from sklearn.feature_selection import RFE

# load X features and y label from file
X, y = load_my_data() # Assumption that this is loading data function

# create data object
data = Data(X, y)

# create model and selector
opt_paras = {"name": "GA", "epoch": 100, "pop_size": 30}
model = MhaElmRegressor(hidden_size=10, act_name="relu", obj_name="MSE",
                optimizer="BaseGA", optimizer_paras=opt_paras, verbose=False, seed=42)

selector = RFE(estimator=model)
selector.fit(X_train, y_train)

# get the final dataset
data.X_train = data.X_train[selector.support_]
data.X_test = data.X_test[selector.support_]
print(f'Ranking of features from Recursive FS: {selector.ranking_}')
```

CHAPTER

THREE

INTELELM LIBRARY

3.1 intelelm.base_elm module

class intelelm.base_elm.**BaseElm**(*hidden_size=10*, *act_name='elu'*)

Bases: sklearn.base.BaseEstimator

Defines the most general class for ELM network that inherits the BaseEstimator class of Scikit-Learn library.

Parameters

- **hidden_size** (*int, default=10*) – The number of hidden nodes
- **act_name** (*{"relu", "leaky_relu", "celu", "prelu", "gelu", "elu", "selu", "rrelu", "tanh", "hard_tanh", "sigmoid",}*) – "hard_sigmoid", "log_sigmoid", "silu", "swish", "hard_swish", "soft_plus", "mish", "soft_sign", "tanh_shrink", "soft_shrink", "hard_shrink", "softmin", "softmax", "log_softmax" }, default='sigmoid' Activation function for the hidden layer.

CLS_OBJ_LOSSES = None

SUPPORTED_CLS_METRICS = {'AS': 'max', 'BSL': 'min', 'CEL': 'min', 'CKS': 'max', 'F1S': 'max', 'F2S': 'max', 'FBS': 'max', 'GINI': 'min', 'GMS': 'max', 'HL': 'min', 'HS': 'max', 'JSI': 'max', 'KLDL': 'min', 'LS': 'max', 'MCC': 'max', 'NPV': 'max', 'PS': 'max', 'ROC-AUC': 'max', 'RS': 'max', 'SS': 'max'}

SUPPORTED_REG_METRICS = {'A10': 'max', 'A20': 'max', 'A30': 'max', 'ACOD': 'max', 'APCC': 'max', 'AR': 'max', 'AR2': 'max', 'CI': 'max', 'COD': 'max', 'COR': 'max', 'COV': 'max', 'CRM': 'min', 'DRV': 'min', 'EC': 'max', 'EVS': 'max', 'GINI': 'min', 'GINI_WIKI': 'min', 'JSD': 'min', 'KGE': 'max', 'MAAPE': 'min', 'MAE': 'min', 'MAPE': 'min', 'MASE': 'min', 'ME': 'min', 'MRB': 'min', 'MRE': 'min', 'MSE': 'min', 'MSLE': 'min', 'MedAE': 'min', 'NNSE': 'max', 'NRMSE': 'min', 'NSE': 'max', 'OI': 'max', 'PCC': 'max', 'PCD': 'max', 'R': 'max', 'R2': 'max', 'R2S': 'max', 'RAE': 'min', 'RMSE': 'min', 'RSE': 'min', 'RSQ': 'max', 'SMAPE': 'min', 'VAF': 'max', 'WI': 'max'}

create_network(*X*, *y*)

evaluate(*y_true*, *y_pred*, *list_metrics=None*)

Return the list of performance metrics of the prediction.

Parameters

- **y_true** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True values for *X*.
- **y_pred** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – Predicted values for *X*.

- **list_metrics** (*list*) – You can get metrics from Permetrics library: https://github.com/thieu1995/permetrics

Returns **results** – The results of the list metrics

Return type dict

fit(*X*, *y*)

static load_model(*load_path='history'*, *filename='model.pkl'*)

predict(*X*, *return_prob=False*)

Inherit the predict function from BaseElm class, with 1 more parameter *return_prob*.

Parameters

- **X** (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – The input data.
- **return_prob** (*bool, default=False*) – It is used for classification problem:
  - If True, the returned results are the probability for each sample
  - If False, the returned results are the predicted labels

save_loss_train(*save_path='history'*, *filename='loss.csv'*)

Save the loss (convergence) during the training process to csv file.

Parameters

- **save_path** (*saved path (relative path, consider from current executed script path)*) –
- **filename** (*name of the file, needs to have ".csv" extension*) –

save_metrics(*y_true*, *y_pred*, *list_metrics=('RMSE', 'MAE')*, *save_path='history'*, *filename='metrics.csv'*)

Save evaluation metrics to csv file

Parameters

- **y_true** (*ground truth data*) –
- **y_pred** (*predicted output*) –
- **list_metrics** (*list of evaluation metrics*) –
- **save_path** (*saved path (relative path, consider from current executed script path)*) –
- **filename** (*name of the file, needs to have ".csv" extension*) –

save_model(*save_path='history'*, *filename='model.pkl'*)

Save model to pickle file

Parameters

- **save_path** (*saved path (relative path, consider from current executed script path)*) –
- **filename** (*name of the file, needs to have ".pkl" extension*) –

save_y_predicted(*X*, *y_true*, *save_path='history'*, *filename='y_predicted.csv'*)

Save the predicted results to csv file

Parameters

- **X** (*The features data, nd.ndarray*) –

- **y_true** (*The ground truth data*) –
- **save_path** (*saved path (relative path, consider from current executed script path)*) –
- **filename** (*name of the file, needs to have ".csv" extension*) –

score(*X*, *y*, *method=None*)

Return the metric of the prediction.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Test samples. For some estimators this may be a precomputed kernel matrix or a list of generic objects instead with shape (n_samples, n_samples_fitted), where n_samples_fitted is the number of samples used in the fitting for the estimator.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True values for *X*.
- **method** (*str, default="RMSE"*) – You can get metrics from Permetrics library: https://github.com/thieu1995/permetrics

Returns **result** – The result of selected metric

Return type float

scores(*X*, *y*, *list_methods=None*)

Return the list of metrics of the prediction.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Test samples. For some estimators this may be a precomputed kernel matrix or a list of generic objects instead with shape (n_samples, n_samples_fitted), where n_samples_fitted is the number of samples used in the fitting for the estimator.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True values for *X*.
- **list_methods** (*list, default=("MSE", "MAE")*) – You can get metrics from Permetrics library: https://github.com/thieu1995/permetrics

Returns **results** – The results of the list metrics

Return type dict

set_predict_request(*, *return_prob: Union[bool, None, str] = '$UNCHANGED$'*) → *intelelm.base_elm.BaseElm*

Request metadata passed to the predict method.

Note that this method is only relevant if enable_metadata_routing=True (see sklearn.set_config()). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- True: metadata is requested, and passed to predict if provided. The request is ignored if metadata is not provided.
- False: metadata is not requested and the meta-estimator will not pass it to predict.
- None: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- str: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (sklearn.utils.metadata_routing.UNCHANGED) retains the existing request. This allows you to change the request for some parameters and not others.

New in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a Pipeline. Otherwise it has no effect.

Parameters **return_prob** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for return_prob parameter in predict.

Returns **self** – The updated object.

Return type object

set_score_request(*, *method: Union[bool, None, str] = '$UNCHANGED$'*) → *intelelm.base_elm.BaseElm*

Request metadata passed to the score method.

Note that this method is only relevant if enable_metadata_routing=True (see sklearn.set_config()). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- True: metadata is requested, and passed to score if provided. The request is ignored if metadata is not provided.
- False: metadata is not requested and the meta-estimator will not pass it to score.
- None: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- str: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (sklearn.utils.metadata_routing.UNCHANGED) retains the existing request. This allows you to change the request for some parameters and not others.

New in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a Pipeline. Otherwise it has no effect.

Parameters **method** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for method parameter in score.

Returns **self** – The updated object.

Return type object

class intelelm.base_elm.**BaseMhaElm**(*hidden_size=10*, *act_name='elu'*, *obj_name=None*, *optimizer='BaseGA'*, *optimizer_paras=None*, *verbose=True*, *seed=None*)

Bases: *intelelm.base_elm.BaseElm*

Defines the most general class for Metaheuristic-based ELM model that inherits the BaseELM class

Parameters

- **hidden_size** (*int, default=10*) – The number of hidden nodes
- **act_name** (*{"relu", "leaky_relu", "celu", "prelu", "gelu", "elu", "selu", "rrelu", "tanh", "hard_tanh", "sigmoid",}*) – "hard_sigmoid", "log_sigmoid", "silu", "swish", "hard_swish", "soft_plus", "mish", "soft_sign", "tanh_shrink", "soft_shrink", "hard_shrink", "softmin", "softmax", "log_softmax" }, default='sigmoid' Activation function for the hidden layer.
- **obj_name** (*None or str, default=None*) – The name of objective for the problem, also depend on the problem is classification and regression.
- **optimizer** (*str or instance of Optimizer class (from Mealpy library), default = "BaseGA"*) – The Metaheuristic Algorithm that use to solve the feature selection problem. Current supported list, please check it here: https://github.com/thieu1995/mealpy. If a custom optimizer is passed, make sure it is an instance of *Optimizer* class.
- **optimizer_paras** (*None or dict of parameter, default=None*) – The parameter for the *optimizer* object. If *None*, the default parameters of optimizer is used (defined in https://github.com/thieu1995/mealpy.) If *dict* is passed, make sure it has at least *epoch* and *pop_size* parameters.
- **verbose** (*bool, default=True*) – Whether to print progress messages to stdout.
- **seed** (*int, default=None*) – Determines random number generation for weights and bias initialization. Pass an int for reproducible results across multiple function calls.

```
SUPPORTED_CLS_OBJECTIVES = {'AS': 'max', 'BSL': 'min', 'CEL': 'min', 'CKS': 'max',
'F1S': 'max', 'F2S': 'max', 'FBS': 'max', 'GINI': 'min', 'GMS': 'max', 'HL': 'min',
'HS': 'max', 'JSI': 'max', 'KLDL': 'min', 'LS': 'max', 'MCC': 'max', 'NPV': 'max',
'PS': 'max', 'ROC-AUC': 'max', 'RS': 'max', 'SS': 'max'}
```

SUPPORTED_OPTIMIZERS = ['OriginalABC', 'OriginalACOR', 'AugmentedAEO',
'EnhancedAEO', 'ImprovedAEO', 'ModifiedAEO', 'OriginalAEO', 'MGTO', 'OriginalAGTO',
'DevALO', 'OriginalALO', 'OriginalAO', 'OriginalAOA', 'IARO', 'LARO', 'OriginalARO',
'OriginalASO', 'OriginalAVOA', 'OriginalArchOA', 'AdaptiveBA', 'DevBA',
'OriginalBA', 'DevBBO', 'OriginalBBO', 'OriginalBBOA', 'OriginalBES', 'ABFO',
'OriginalBFO', 'OriginalBMO', 'DevBRO', 'OriginalBRO', 'OriginalBSA', 'ImprovedBSO',
'OriginalBSO', 'CleverBookBeesA', 'OriginalBeesA', 'ProbBeesA', 'OriginalCA',
'OriginalCDO', 'OriginalCEM', 'OriginalCGO', 'DevCHIO', 'OriginalCHIO',
'OriginalCOA', 'OCRO', 'OriginalCRO', 'OriginalCSA', 'OriginalCSO',
'OriginalCircleSA', 'OriginalCoatiOA', 'JADE', 'OriginalDE', 'SADE', 'SAP_DE',
'DevDMOA', 'OriginalDMOA', 'OriginalDO', 'DevEFO', 'OriginalEFO', 'OriginalEHO',
'AdaptiveEO', 'ModifiedEO', 'OriginalEO', 'OriginalEOA', 'LevyEP', 'OriginalEP',
'CMA_ES', 'LevyES', 'OriginalES', 'Simple_CMA_ES', 'OriginalESOA', 'OriginalEVO',
'OriginalFA', 'DevFBIO', 'OriginalFBIO', 'OriginalFFA', 'OriginalFFO',
'OriginalFLA', 'DevFOA', 'OriginalFOA', 'WhaleFOA', 'OriginalFOX', 'OriginalFPA',
'BaseGA', 'EliteMultiGA', 'EliteSingleGA', 'MultiGA', 'SingleGA', 'OriginalGBO',
'DevGCO', 'OriginalGCO', 'OriginalGJO', 'OriginalGOA', 'DevGSKA', 'OriginalGSKA',
'Matlab101GTO', 'Matlab102GTO', 'OriginalGTO', 'GWO_WOA', 'IGWO', 'OriginalGWO',
'RW_GWO', 'OriginalHBA', 'OriginalHBO', 'OriginalHC', 'SwarmHC', 'OriginalHCO',
'OriginalHGS', 'OriginalHGSO', 'OriginalHHO', 'DevHS', 'OriginalHS', 'OriginalICA',
'OriginalINFO', 'OriginalIWO', 'DevJA', 'LevyJA', 'OriginalJA', 'DevLCO',
'ImprovedLCO', 'OriginalLCO', 'OriginalMA', 'OriginalMFO', 'OriginalMGO',
'OriginalMPA', 'OriginalMRFO', 'WMQIMRFO', 'OriginalMSA', 'DevMVO', 'OriginalMVO',
'OriginalNGO', 'ImprovedNMRA', 'OriginalNMRA', 'OriginalNRO', 'OriginalOOA',
'OriginalPFA', 'OriginalPOA', 'AIW_PSO', 'CL_PSO', 'C_PSO', 'HPSO_TVAC', 'LDW_PSO',
'OriginalPSO', 'P_PSO', 'OriginalPSS', 'DevQSA', 'ImprovedQSA', 'LevyQSA',
'OppoQSA', 'OriginalQSA', 'OriginalRIME', 'OriginalRUN', 'GaussianSA', 'OriginalSA',
'SwarmSA', 'DevSARO', 'OriginalSARO', 'DevSBO', 'OriginalSBO', 'DevSCA',
'OriginalSCA', 'QleSCA', 'OriginalSCSO', 'ImprovedSFO', 'OriginalSFO', 'L_SHADE',
'OriginalSHADE', 'OriginalSHIO', 'OriginalSHO', 'ImprovedSLO', 'ModifiedSLO',
'OriginalSLO', 'DevSMA', 'OriginalSMA', 'DevSOA', 'OriginalSOA', 'OriginalSOS',
'DevSPBO', 'OriginalSPBO', 'OriginalSRSR', 'DevSSA', 'OriginalSSA', 'OriginalSSDO',
'OriginalSSO', 'OriginalSSpiderA', 'OriginalSSpiderO', 'OriginalSTO',
'OriginalSeaHO', 'OriginalServalOA', 'OriginalTDO', 'DevTLO', 'ImprovedTLO',
'OriginalTLO', 'OriginalTOA', 'DevTPO', 'OriginalTS', 'OriginalTSA', 'OriginalTSO',
'EnhancedTWO', 'LevyTWO', 'OppoTWO', 'OriginalTWO', 'DevVCS', 'OriginalVCS',
'OriginalWCA', 'OriginalWDO', 'OriginalWHO', 'HI_WOA', 'OriginalWOA',
'OriginalWaOA', 'OriginalWarSO', 'OriginalZOA']

SUPPORTED_REG_OBJECTIVES = {'A10': 'max', 'A20': 'max', 'A30': 'max', 'ACOD':
'max', 'APCC': 'max', 'AR': 'max', 'AR2': 'max', 'CI': 'max', 'COD': 'max', 'COR':
'max', 'COV': 'max', 'CRM': 'min', 'DRV': 'min', 'EC': 'max', 'EVS': 'max', 'GINI':
'min', 'GINI_WIKI': 'min', 'JSD': 'min', 'KGE': 'max', 'MAAPE': 'min', 'MAE': 'min',
'MAPE': 'min', 'MASE': 'min', 'ME': 'min', 'MRB': 'min', 'MRE': 'min', 'MSE': 'min',
'MSLE': 'min', 'MedAE': 'min', 'NNSE': 'max', 'NRMSE': 'min', 'NSE': 'max', 'OI':
'max', 'PCC': 'max', 'PCD': 'max', 'R': 'max', 'R2': 'max', 'R2S': 'max', 'RAE':
'min', 'RMSE': 'min', 'RSE': 'min', 'RSQ': 'max', 'SMAPE': 'min', 'VAF': 'max',
'WI': 'max'}

fit(*X*, *y*, *lb=(- 10.0,)*, *ub=(10.0,)*, *mode='single'*, *n_workers=None*, *termination=None*, *save_population=False*)

Parameters

- **X** (*The features data, np.ndarray*) –

- **y** (*The ground truth data*) –
- **lb** (*The lower bound for decision variables in optimization problem (The weights and biases of network)*) –
- **ub** (*The upper bound for decision variables in optimization problem (The weights and biases of network)*) –
- **mode** (*Parallel: 'process', 'thread'; Sequential: 'swarm', 'single'.*) –
  - 'process': The parallel mode with multiple cores run the tasks
  - 'thread': The parallel mode with multiple threads run the tasks
  - 'swarm': The sequential mode that no effect on updating phase of other agents
  - 'single': The sequential mode that effect on updating phase of other agents, this is default mode
- **n_workers** (*The number of workers (cores or threads) to do the tasks (effect only on parallel mode)*) –
- **termination** (*The termination dictionary or an instance of Termination class in Mealpy library*) –
- **save_population** (*Save the population of search agents (Don't set it to True when you don't know how to use it)*) –

fitness_function(*solution=None*)

set_fit_request(**, lb: Union[bool, None, str] = '$UNCHANGED$', mode: Union[bool, None, str] = '$UNCHANGED$', n_workers: Union[bool, None, str] = '$UNCHANGED$', save_population: Union[bool, None, str] = '$UNCHANGED$', termination: Union[bool, None, str] = '$UNCHANGED$', ub: Union[bool, None, str] = '$UNCHANGED$'*) → *intelelm.base_elm.BaseMhaElm*

Request metadata passed to the `fit` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `fit` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `fit`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

New in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters

- **lb** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `lb` parameter in `fit`.
- **mode** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `mode` parameter in `fit`.
- **n_workers** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `n_workers` parameter in `fit`.
- **save_population** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `save_population` parameter in `fit`.
- **termination** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `termination` parameter in `fit`.
- **ub** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `ub` parameter in `fit`.

Returns **self** – The updated object.

Return type object

set_predict_request(*, *return_prob: Union[bool, None, str] = '$UNCHANGED$'*) → *intelelm.base_elm.BaseMhaElm*

Request metadata passed to the `predict` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `predict` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `predict`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

New in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters **return_prob** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `return_prob` parameter in `predict`.

Returns **self** – The updated object.

Return type object

set_score_request(*, *method: Union[bool, None, str] = '$UNCHANGED$'*) → *intelelm.base_elm.BaseMhaElm*

Request metadata passed to the `score` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `score` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `score`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

New in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters **method** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `method` parameter in `score`.

Returns **self** – The updated object.

Return type object

class intelelm.base_elm.**ELM**(*size_input=5*, *size_hidden=10*, *size_output=1*, *act_name='sigmoid'*, *seed=None*)

Bases: `object`

Extreme Learning Machine

This class defines the general ELM model

Parameters

- **size_input** (*int, default=5*) – The number of input nodes
- **size_hidden** (*int, default=10*) – The number of hidden nodes
- **size_output** (*int, default=1*) – The number of output nodes
- **act_name** ({*"relu", "leaky_relu", "celu", "prelu", "gelu", "elu", "selu", "rrelu", "tanh", "hard_tanh", "sigmoid",*}) – "hard_sigmoid", "log_sigmoid", "silu", "swish", "hard_swish", "soft_plus", "mish", "soft_sign", "tanh_shrink", "soft_shrink", "hard_shrink", "softmin", "softmax", "log_softmax" }, default='sigmoid' Activation function for the hidden layer.
- **seed** (*int, default=None*) – Determines random number generation for weights and bias initialization. Pass an int for reproducible results across multiple function calls.

fit(*X*, *y*)

Fit the model to data matrix X and target(s) y.

Parameters

- **X** (*ndarray or sparse matrix of shape (n_samples, n_features)*) – The input data.
- **y** (*ndarray of shape (n_samples,) or (n_samples, n_outputs)*) – The target values (class labels in classification, real numbers in regression).

Returns **self** – Returns a trained ELM model.

Return type object

get_weights()

get_weights_size()

predict(*X*)

Predict using the Extreme Learning Machine model.

Parameters **X** (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – The input data.

Returns **y** – The predicted values.

Return type ndarray of shape (n_samples, n_outputs)

set_weights(*weights*)

update_weights_from_solution(*solution*, *X*, *y*)

3.2 Subpackages

3.2.1 intelelm.model package

3.2.1.1 intelelm.model.standard_elm module

class intelelm.model.standard_elm.**ElmClassifier**(*hidden_size=10*, *act_name='elu'*, *seed=None*)

Bases: *intelelm.base_elm.BaseElm*, sklearn.base.ClassifierMixin

Defines the general class of Metaheuristic-based ELM model for Classification problems that inherit the BaseElm and ClassifierMixin classes.

Parameters

- **hidden_size** (*int, default=10*) – The number of hidden nodes
- **act_name** (*{"relu", "leaky_relu", "celu", "prelu", "gelu", "elu", "selu", "rrelu", "tanh", "hard_tanh", "sigmoid",}*) – "hard_sigmoid", "log_sigmoid", "silu", "swish", "hard_swish", "soft_plus", "mish", "soft_sign", "tanh_shrink", "soft_shrink", "hard_shrink", "softmin", "softmax", "log_softmax" }, default='sigmoid' Activation function for the hidden layer.
- **seed** (*int, default=None*) – Determines random number generation for weights and bias initialization. Pass an int for reproducible results across multiple function calls.

Examples

```
>>> from intelelm import Data, ElmClassifier
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification(n_samples=100, random_state=1)
>>> data = Data(X, y)
>>> data.split_train_test(test_size=0.2, random_state=1)
>>> model = ElmClassifier(hidden_size=10, act_name="elu")
>>> model.fit(data.X_train, data.y_train)
>>> pred = model.predict(data.X_test)
>>> print(pred)
array([1, 0, 1, 0, 1])
```

CLS_OBJ_LOSSES = ['CEL', 'HL', 'KLDL', 'BSL']

create_network(*X*, *y*)

evaluate(*y_true*, *y_pred*, *list_metrics=('AS', 'RS')*)

Return the list of performance metrics on the given test data and labels.

Parameters

- **y_true** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True values for *X*.
- **y_pred** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – Predicted values for *X*.
- **list_metrics** (*list, default=("AS", "RS")*) – You can get metrics from Permetrics library: https://github.com/thieu1995/permetrics

Returns **results** – The results of the list metrics

Return type dict

score(*X*, *y*, *method='AS'*)

Return the metric on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Test samples.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True labels for *X*.
- **method** (*str, default="AS"*) – You can get all of the metrics from Permetrics library: https://github.com/thieu1995/permetrics

Returns **result** – The result of selected metric

Return type float

scores(*X*, *y*, *list_methods=('AS', 'RS')*)

Return the list of metrics on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Test samples.

- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True labels for *X*.
- **list_methods** (*list, default=("AS", "RS")*) – You can get all of the metrics from Permetrics library: https://github.com/thieu1995/permetrics

Returns **results** – The results of the list metrics

Return type dict

set_predict_request(*, *return_prob: Union[bool, None, str] = '$UNCHANGED$'*) → *intelelm.model.standard_elm.ElmClassifier*

Request metadata passed to the `predict` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `predict` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `predict`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

New in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters **return_prob** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `return_prob` parameter in `predict`.

Returns **self** – The updated object.

Return type object

set_score_request(*, *method: Union[bool, None, str] = '$UNCHANGED$'*) → *intelelm.model.standard_elm.ElmClassifier*

Request metadata passed to the `score` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `score` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `score`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.

- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

New in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters **method** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `method` parameter in `score`.

Returns **self** – The updated object.

Return type object

class intelelm.model.standard_elm.**ElmRegressor**(*hidden_size=10*, *act_name='elu'*, *seed=None*)

Bases: *intelelm.base_elm.BaseElm*, `sklearn.base.RegressorMixin`

Defines the ELM model for Regression problems that inherit the BaseElm and RegressorMixin classes.

Parameters

- **hidden_size** (*int, default=10*) – The number of hidden nodes
- **act_name** (*{"relu", "leaky_relu", "celu", "prelu", "gelu", "elu", "selu", "rrelu", "tanh", "hard_tanh", "sigmoid",}*) – "hard_sigmoid", "log_sigmoid", "silu", "swish", "hard_swish", "soft_plus", "mish", "soft_sign", "tanh_shrink", "soft_shrink", "hard_shrink", "softmin", "softmax", "log_softmax" }, default='sigmoid' Activation function for the hidden layer.
- **seed** (*int, default=None*) – Determines random number generation for weights and bias initialization. Pass an int for reproducible results across multiple function calls.

Examples

```
>>> from intelelm import ElmRegressor, Data
>>> from sklearn.datasets import make_regression
>>> X, y = make_regression(n_samples=200, random_state=1)
>>> data = Data(X, y)
>>> data.split_train_test(test_size=0.2, random_state=1)
>>> model = ElmRegressor(hidden_size=10, act_name="elu")
>>> model.fit(data.X_train, data.y_train)
>>> pred = model.predict(data.X_test)
>>> print(pred)
```

create_network(*X*, *y*)

evaluate(*y_true*, *y_pred*, *list_metrics=('MSE', 'MAE')*)

Return the list of performance metrics of the prediction.

Parameters

- **y_true** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True values for *X*.

- **y_pred** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – Predicted values for *X*.
- **list_metrics** (*list, default=("MSE", "MAE")*) – You can get metrics from Permetrics library: https://github.com/thieu1995/permetrics

Returns **results** – The results of the list metrics

Return type dict

score(*X*, *y*, *method='RMSE'*)

Return the metric of the prediction.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Test samples. For some estimators this may be a precomputed kernel matrix or a list of generic objects instead with shape `(n_samples, n_samples_fitted)`, where `n_samples_fitted` is the number of samples used in the fitting for the estimator.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True values for *X*.
- **method** (*str, default="RMSE"*) – You can get all of the metrics from Permetrics library: https://github.com/thieu1995/permetrics

Returns **result** – The result of selected metric

Return type float

scores(*X*, *y*, *list_methods=('MSE', 'MAE')*)

Return the list of metrics of the prediction.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Test samples. For some estimators this may be a precomputed kernel matrix or a list of generic objects instead with shape `(n_samples, n_samples_fitted)`, where `n_samples_fitted` is the number of samples used in the fitting for the estimator.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True values for *X*.
- **list_methods** (*list, default=("MSE", "MAE")*) – You can get all of the metrics from Permetrics library: https://github.com/thieu1995/permetrics

Returns **results** – The results of the list metrics

Return type dict

set_predict_request(**, return_prob: Union[bool, None, str] = '$UNCHANGED$'*) → *intelelm.model.standard_elm.ElmRegressor*

Request metadata passed to the `predict` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `predict` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `predict`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.

- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

New in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters **return_prob** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `return_prob` parameter in `predict`.

Returns **self** – The updated object.

Return type object

set_score_request(*, *method: Union[bool, None, str] = '$UNCHANGED$'*) → *intelelm.model.standard_elm.ElmRegressor*

Request metadata passed to the `score` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `score` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `score`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

New in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters **method** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `method` parameter in `score`.

Returns **self** – The updated object.

Return type object

3.2.1.2 intelelm.model.mha_elm module

class intelelm.model.mha_elm.**MhaElmClassifier**(*hidden_size=10*, *act_name='elu'*, *obj_name=None*, *optimizer='BaseGA'*, *optimizer_paras=None*, *verbose=False*, *seed=None*)

Bases: *intelelm.base_elm.BaseMhaElm*, sklearn.base.ClassifierMixin

Defines the general class of Metaheuristic-based ELM model for Classification problems that inherit the BaseMhaElm and ClassifierMixin classes.

Parameters

- **hidden_size** (*int, default=10*) – The number of hidden nodes
- **act_name** (*{"relu", "leaky_relu", "celu", "prelu", "gelu", "elu", "selu", "rrelu", "tanh", "hard_tanh", "sigmoid",}*) – "hard_sigmoid", "log_sigmoid", "silu", "swish", "hard_swish", "soft_plus", "mish", "soft_sign", "tanh_shrink", "soft_shrink", "hard_shrink", "softmin", "softmax", "log_softmax" }, default='sigmoid' Activation function for the hidden layer.
- **obj_name** (*None or str, default=None*) – The name of objective for the problem, also depend on the problem is classification and regression.
- **optimizer** (*str or instance of Optimizer class (from Mealpy library), default = "BaseGA"*) – The Metaheuristic Algorithm that use to solve the feature selection problem. Current supported list, please check it here: https://github.com/thieu1995/mealpy. If a custom optimizer is passed, make sure it is an instance of *Optimizer* class.
- **optimizer_paras** (*None or dict of parameter, default=None*) – The parameter for the *optimizer* object. If *None*, the default parameters of optimizer is used (defined in https://github.com/thieu1995/mealpy.) If *dict* is passed, make sure it has at least *epoch* and *pop_size* parameters.
- **verbose** (*bool, default=False*) – Whether to print progress messages to stdout.
- **seed** (*int, default=None*) – Determines random number generation for weights and bias initialization. Pass an int for reproducible results across multiple function calls.

Examples

```
>>> from intelelm import Data, MhaElmClassifier
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification(n_samples=100, random_state=1)
>>> data = Data(X, y)
>>> data.split_train_test(test_size=0.2, random_state=1)
>>> opt_paras = {"name": "GA", "epoch": 10, "pop_size": 30}
>>> print(MhaElmClassifier.SUPPORTED_CLS_OBJECTIVES)
{'PS': 'max', 'NPV': 'max', 'RS': 'max', ...., 'KLDL': 'min', 'BSL': 'min'}
>>> model = MhaElmClassifier(hidden_size=10, act_name="elu", obj_name="BSL",␣
↪optimizer="BaseGA", optimizer_paras=opt_paras)
>>> model.fit(data.X_train, data.y_train)
>>> pred = model.predict(data.X_test)
>>> print(pred)
array([1, 0, 1, 0, 1])
```

CLS_OBJ_LOSSES = ['CEL', 'HL', 'KLDL', 'BSL']

create_network(*X*, *y*)

evaluate(*y_true*, *y_pred*, *list_metrics=('AS', 'RS')*)

Return the list of performance metrics on the given test data and labels.

Parameters

- **y_true** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True values for *X*.
- **y_pred** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – Predicted values for *X*.
- **list_metrics** (*list, default=("AS", "RS")*) – You can get metrics from Permetrics library: https://github.com/thieu1995/permetrics

Returns **results** – The results of the list metrics

Return type dict

fitness_function(*solution=None*)

Evaluates the fitness function for classification metric

Parameters **solution** (*np.ndarray, default=None*) –

Returns **result** – The fitness value

Return type float

score(*X*, *y*, *method='AS'*)

Return the metric on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Test samples.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True labels for *X*.
- **method** (*str, default="AS"*) – You can get all of the metrics from Permetrics library: https://github.com/thieu1995/permetrics

Returns **result** – The result of selected metric

Return type float

scores(*X*, *y*, *list_methods=('AS', 'RS')*)

Return the list of metrics on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Test samples.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True labels for *X*.
- **list_methods** (*list, default=("AS", "RS")*) – You can get all of the metrics from Permetrics library: https://github.com/thieu1995/permetrics

Returns **results** – The results of the list metrics

Return type dict

set_fit_request(*, *lb: Union[bool, None, str] = '$UNCHANGED$'*, *mode: Union[bool, None, str] = '$UNCHANGED$'*, *n_workers: Union[bool, None, str] = '$UNCHANGED$'*, *save_population: Union[bool, None, str] = '$UNCHANGED$'*, *termination: Union[bool, None, str] = '$UNCHANGED$'*, *ub: Union[bool, None, str] = '$UNCHANGED$'*) → *intelelm.model.mha_elm.MhaElmClassifier*

Request metadata passed to the `fit` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `fit` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `fit`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

New in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters

- **lb** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `lb` parameter in `fit`.
- **mode** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `mode` parameter in `fit`.
- **n_workers** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `n_workers` parameter in `fit`.
- **save_population** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `save_population` parameter in `fit`.
- **termination** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `termination` parameter in `fit`.
- **ub** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `ub` parameter in `fit`.

Returns **self** – The updated object.

Return type object

set_predict_request(*, *return_prob: Union[bool, None, str] = '$UNCHANGED$'*) → *intelelm.model.mha_elm.MhaElmClassifier*

Request metadata passed to the `predict` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `predict` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `predict`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

New in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters **return_prob** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `return_prob` parameter in `predict`.

Returns **self** – The updated object.

Return type object

set_score_request(*, *method: Union[bool, None, str] = '$UNCHANGED$'*) → *intelelm.model.mha_elm.MhaElmClassifier*

Request metadata passed to the `score` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `score` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `score`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

New in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters **method** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `method` parameter in `score`.

Returns **self** – The updated object.

Return type object

class intelelm.model.mha_elm.**MhaElmRegressor**(*hidden_size=10*, *act_name='elu'*, *obj_name=None*, *optimizer='BaseGA'*, *optimizer_paras=None*, *verbose=False*, *seed=None*, *obj_weights=None*)

Bases: *intelelm.base_elm.BaseMhaElm*, `sklearn.base.RegressorMixin`

Defines the general class of Metaheuristic-based ELM model for Regression problems that inherit the BaseMhaElm and RegressorMixin classes.

Parameters

- **hidden_size** (*int, default=10*) – The number of hidden nodes
- **act_name** (*{"relu", "leaky_relu", "celu", "prelu", "gelu", "elu", "selu", "rrelu", "tanh", "hard_tanh", "sigmoid",}*) – "hard_sigmoid", "log_sigmoid", "silu", "swish", "hard_swish", "soft_plus", "mish", "soft_sign", "tanh_shrink", "soft_shrink", "hard_shrink", "softmin", "softmax", "log_softmax" }, default='sigmoid' Activation function for the hidden layer.
- **obj_name** (*None or str, default=None*) – The name of objective for the problem, also depend on the problem is classification and regression.
- **optimizer** (*str or instance of Optimizer class (from Mealpy library), default = "BaseGA"*) – The Metaheuristic Algorithm that use to solve the feature selection problem. Current supported list, please check it here: https://github.com/thieu1995/mealpy. If a custom optimizer is passed, make sure it is an instance of *Optimizer* class.
- **optimizer_paras** (*None or dict of parameter, default=None*) – The parameter for the *optimizer* object. If *None*, the default parameters of optimizer is used (defined in https://github.com/thieu1995/mealpy.) If *dict* is passed, make sure it has at least *epoch* and *pop_size* parameters.
- **verbose** (*bool, default=False*) – Whether to print progress messages to stdout.
- **seed** (*int, default=None*) – Determines random number generation for weights and bias initialization. Pass an int for reproducible results across multiple function calls.

Examples

```
>>> from intelelm import MhaElmRegressor, Data
>>> from sklearn.datasets import make_regression
>>> X, y = make_regression(n_samples=200, random_state=1)
>>> data = Data(X, y)
>>> data.split_train_test(test_size=0.2, random_state=1)
>>> opt_paras = {"name": "GA", "epoch": 10, "pop_size": 30}
>>> model = MhaElmRegressor(hidden_size=10, act_name="elu", obj_name="RMSE",␣
↪optimizer="BaseGA", optimizer_paras=opt_paras)
>>> model.fit(data.X_train, data.y_train)
>>> pred = model.predict(data.X_test)
>>> print(pred)
```

create_network(*X*, *y*)

evaluate(*y_true*, *y_pred*, *list_metrics=('MSE', 'MAE')*)
Return the list of performance metrics of the prediction.

Parameters

- **y_true** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True values for *X*.
- **y_pred** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – Predicted values for *X*.
- **list_metrics** (*list, default=("MSE", "MAE")*) – You can get metrics from Permetrics library: https://github.com/thieu1995/permetrics

Returns **results** – The results of the list metrics

Return type dict

fitness_function(*solution=None*)
Evaluates the fitness function for regression metric

Parameters **solution** (*np.ndarray, default=None*) –

Returns **result** – The fitness value

Return type float

score(*X*, *y*, *method='RMSE'*)
Return the metric of the prediction.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Test samples. For some estimators this may be a precomputed kernel matrix or a list of generic objects instead with shape `(n_samples, n_samples_fitted)`, where `n_samples_fitted` is the number of samples used in the fitting for the estimator.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True values for *X*.
- **method** (*str, default="RMSE"*) – You can get all of the metrics from Permetrics library: https://github.com/thieu1995/permetrics

Returns **result** – The result of selected metric

Return type float

scores(*X*, *y*, *list_methods=('MSE', 'MAE')*)
Return the list of metrics of the prediction.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Test samples. For some estimators this may be a precomputed kernel matrix or a list of generic objects instead with shape (n_samples, n_samples_fitted), where n_samples_fitted is the number of samples used in the fitting for the estimator.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True values for *X*.
- **list_methods** (*list, default=("MSE", "MAE")*) – You can get all of the metrics from Permetrics library: https://github.com/thieu1995/permetrics

Returns **results** – The results of the list metrics

Return type dict

set_fit_request(**, lb: Union[bool, None, str] = '$UNCHANGED$', mode: Union[bool, None, str] = '$UNCHANGED$', n_workers: Union[bool, None, str] = '$UNCHANGED$', save_population: Union[bool, None, str] = '$UNCHANGED$', termination: Union[bool, None, str] = '$UNCHANGED$', ub: Union[bool, None, str] = '$UNCHANGED$'*) → *intelelm.model.mha_elm.MhaElmRegressor*
Request metadata passed to the fit method.

Note that this method is only relevant if enable_metadata_routing=True (see sklearn.set_config()). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- True: metadata is requested, and passed to fit if provided. The request is ignored if metadata is not provided.
- False: metadata is not requested and the meta-estimator will not pass it to fit.
- None: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- str: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (sklearn.utils.metadata_routing.UNCHANGED) retains the existing request. This allows you to change the request for some parameters and not others.

New in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a Pipeline. Otherwise it has no effect.

Parameters

- **lb** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for lb parameter in fit.
- **mode** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for mode parameter in fit.
- **n_workers** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for n_workers parameter in fit.

- **save_population** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `save_population` parameter in `fit`.
- **termination** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `termination` parameter in `fit`.
- **ub** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `ub` parameter in `fit`.

Returns **self** – The updated object.

Return type object

set_predict_request(*, *return_prob: Union[bool, None, str] = '$UNCHANGED$'*) → *intelelm.model.mha_elm.MhaElmRegressor*

Request metadata passed to the `predict` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `predict` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `predict`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

New in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters **return_prob** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `return_prob` parameter in `predict`.

Returns **self** – The updated object.

Return type object

set_score_request(*, *method: Union[bool, None, str] = '$UNCHANGED$'*) → *intelelm.model.mha_elm.MhaElmRegressor*

Request metadata passed to the `score` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config()`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `score` if provided. The request is ignored if metadata is not provided.

- `False`: metadata is not requested and the meta-estimator will not pass it to `score`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

New in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `Pipeline`. Otherwise it has no effect.

Parameters **method** (*str, True, False, or None, default=sklearn.utils.metadata_routing.UNCHANGED*) – Metadata routing for `method` parameter in `score`.

Returns **self** – The updated object.

Return type object

3.2.2 intelelm.utils package

3.2.2.1 intelelm.utils.activation module

intelelm.utils.activation.**celu**(*x*, *alpha=1.0*)

intelelm.utils.activation.**elu**(*x*, *alpha=1*)

intelelm.utils.activation.**gelu**(*x*, *alpha=0.044715*)

intelelm.utils.activation.**hard_shrink**(*x*, *alpha=0.5*)

intelelm.utils.activation.**hard_sigmoid**(*x*, *lower=- 2.5*, *upper=2.5*)

intelelm.utils.activation.**hard_swish**(*x*, *lower=- 3.0*, *upper=3.0*)

intelelm.utils.activation.**hard_tanh**(*x*, *lower=- 1.0*, *upper=1.0*)

intelelm.utils.activation.**leaky_relu**(*x*, *alpha=0.01*)

intelelm.utils.activation.**log_sigmoid**(*x*)

intelelm.utils.activation.**log_softmax**(*x*)

intelelm.utils.activation.**mish**(*x*, *beta=1.0*)

intelelm.utils.activation.**prelu**(*x*, *alpha=0.5*)

intelelm.utils.activation.**relu**(*x*)

intelelm.utils.activation.**rrelu**(*x*, *lower=0.125*, *upper=0.3333333333333333*)

intelelm.utils.activation.**selu**(*x*, *alpha=1.67326324*, *scale=1.05070098*)

intelelm.utils.activation.**sigmoid**(*x*)

intelelm.utils.activation.**silu**(*x*)

intelelm.utils.activation.**soft_plus**(*x*, *beta=1.0*)

intelelm.utils.activation.**soft_shrink**(*x*, *alpha=0.5*)

intelelm.utils.activation.**soft_sign**(*x*)

intelelm.utils.activation.**softmax**(*x*)

intelelm.utils.activation.**softmin**(*x*)

intelelm.utils.activation.**swish**(*x*)

intelelm.utils.activation.**tanh**(*x*)

intelelm.utils.activation.**tanh_shrink**(*x*)

3.2.2.2 intelelm.utils.data_loader module

class intelelm.utils.data_loader.**Data**(*X=None*, *y=None*, *name='Unknown'*)

Bases: `object`

The structure of our supported Data class

Parameters

- **X** (*np.ndarray*) – The features of your data
- **y** (*np.ndarray*) – The labels of your data

SUPPORT = {'scaler': ['standard', 'minmax', 'max-abs', 'log1p', 'loge', 'sqrt', 'sinh-arc-sinh', 'robust', 'box-cox', 'yeo-johnson']}

static encode_label(*y*)

static scale(*X*, *scaling_methods=('standard',)*, *list_dict_paras=None*)

set_train_test(*X_train=None*, *y_train=None*, *X_test=None*, *y_test=None*)

Function use to set your own X_train, y_train, X_test, y_test in case you don't want to use our split function

Parameters

- **X_train** (*np.ndarray*) –
- **y_train** (*np.ndarray*) –
- **X_test** (*np.ndarray*) –
- **y_test** (*np.ndarray*) –

split_train_test(*test_size=0.2*, *train_size=None*, *random_state=41*, *shuffle=True*, *stratify=None*, *inplace=True*)

The wrapper of the split_train_test function in scikit-learn library.

intelelm.utils.data_loader.**get_dataset**(*dataset_name*)

Helper function to retrieve the data

Parameters **dataset_name** (*str*) – Name of the dataset

Returns **data** – The instance of Data class, that hold X and y variables.

Return type *Data*

3.2.2.3 intelelm.utils.encoder module

class intelelm.utils.encoder.**LabelEncoder**
Bases: object

Encode categorical features as integer labels.

fit(*y*)
Fit label encoder to a given set of labels.

y [array-like] Labels to encode.

fit_transform(*y*)
Fit label encoder and return encoded labels.

Parameters **y** (*array-like of shape (n_samples,)*) – Target values.

Returns **y** – Encoded labels.

Return type array-like of shape (n_samples,)

inverse_transform(*y*)
Transform integer labels to original labels.

y [array-like] Encoded integer labels.

original_labels [array-like] Original labels.

transform(*y*)
Transform labels to encoded integer labels.

y [array-like] Labels to encode.

encoded_labels [array-like] Encoded integer labels.

class intelelm.utils.encoder.**ObjectiveScaler**(*obj_name='sigmoid'*, *ohe_scaler=None*)
Bases: object

For label scaler in classification (binary and multiple classification)

inverse_transform(*data*)

transform(*data*)

3.2.2.4 intelelm.utils.evaluator module

intelelm.utils.evaluator.**get_all_classification_metrics**()

intelelm.utils.evaluator.**get_all_regression_metrics**()

intelelm.utils.evaluator.**get_metrics**(*problem*, *y_true*, *y_pred*, *metrics=None*, *testcase='test'*)

3.2.2.5 intelelm.utils.validator module

intelelm.utils.validator.**check_bool**(*name: str*, *value: bool*, *bound=(True, False)*)

intelelm.utils.validator.**check_float**(*name: str*, *value: int*, *bound=None*)

intelelm.utils.validator.**check_int**(*name: str*, *value: int*, *bound=None*)

intelelm.utils.validator.**check_str**(*name: str*, *value: str*, *bound=None*)

intelelm.utils.validator.**check_tuple_float**(*name: str*, *values: tuple*, *bounds=None*)

intelelm.utils.validator.**check_tuple_int**(*name: str*, *values: tuple*, *bounds=None*)

intelelm.utils.validator.**is_in_bound**(*value*, *bound*)

intelelm.utils.validator.**is_str_in_list**(*value: str*, *my_list: list*)

CHAPTER

FOUR

CITATION REQUEST

Note: If you want to understand how Metaheuristic is applied to Extreme Learning Machine, you need to read the paper titled "A new workload prediction model using extreme learning machine and enhanced tug of war optimization". The paper can be accessed at the following this link

Please include these citations if you plan to use this library:

```
@software{nguyen_van_thieu_2023_8249046,
  author       = {Nguyen Van Thieu},
  title        = {IntelELM: A Python Framework for Intelligent Metaheuristic-based␣
↪Extreme Learning Machine},
  month        = aug,
  year         = 2023,
  publisher    = {Zenodo},
  doi          = {10.5281/zenodo.8249045},
  url          = {https://github.com/thieu1995/IntelELM}
}

@article{nguyen2020new,
  title={A new workload prediction model using extreme learning machine and enhanced tug␣
↪of war optimization},
  author={Nguyen, Thieu and Hoang, Bao and Nguyen, Giang and Nguyen, Binh Minh},
  journal={Procedia Computer Science},
  volume={170},
  pages={362--369},
  year={2020},
  publisher={Elsevier},
  doi={10.1016/j.procs.2020.03.063}
}

@article{van2023mealpy,
  title={MEALPY: An open-source library for latest meta-heuristic algorithms in Python},
  author={Van Thieu, Nguyen and Mirjalili, Seyedali},
  journal={Journal of Systems Architecture},
  year={2023},
  publisher={Elsevier},
  doi={10.1016/j.sysarc.2023.102871}
}
```

```
```

If you have an open-ended or a research question, you can contact me via nguyenthieu2102@gmail.com

CHAPTER

FIVE

IMPORTANT LINKS

- Official source code repo: https://github.com/thieu1995/intelelm
- Official document: https://intelelm.readthedocs.io/
- Download releases: https://pypi.org/project/intelelm/
- Issue tracker: https://github.com/thieu1995/intelelm/issues
- Notable changes log: https://github.com/thieu1995/intelelm/blob/master/ChangeLog.md
- **This project also related to our another projects which are "optimization" and "machine learning", check it here:**
  - https://github.com/thieu1995/mealpy
  - https://github.com/thieu1995/metaheuristics
  - https://github.com/thieu1995/opfunu
  - https://github.com/thieu1995/enoppy
  - https://github.com/thieu1995/permetrics
  - https://github.com/thieu1995/MetaCluster
  - https://github.com/thieu1995/pfevaluator
  - https://github.com/aiir-team

CHAPTER

SIX

LICENSE

The project is licensed under GNU General Public License (GPL) V3 license.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX

T

U