A.4 User Documentation

This document describes the installation and usage of the Rule Extraction Assistant (REA).

Required Software

This project is based on Python and requires `python3.9`.

In order to install and run REA, the following python packages are needed: - `pandas` - `numpy` - `scikit-learn` - `tensorflow` - `rpy2`

They can be installed using `pip`, `venv`, `virtualenv`, etc. and the provided `Pifile` or `requirements.txt`.

The following non-python software is needed for the rule extraction with ALPA or DNNRE: - `R` (programming language/interpreter) - R `C50` package

Example Setup

- Install python using your Linux distributions package manager or from the official website
- Install R with the C50 library
  - intsall `R` using your Linux distributions package manager or download it from the official website
  - to install `C5.0`, invoke the `R` REPL on a command prompt (root privileges might be necessary to write to `usr/lib`)
  - type `install.packages("C50")`
  - for this, you may need `build-essentials` (debian) or at least `gcc` and `gfortran`
- Set up the virtual environment
  - run `pipenv install --python 3.9` in the source folder

Usage

You can use the tool either through its API or through its CLI.

API

The api is documented in the API Documentation, which is also contained in the `docs` folder of the implementation.

CLI

We provide a CLI, which accepts a list of configuration files and executes the specified pipeline run. Different runs can be achieved by executing the CLI multiple times with different configuration file(s). The generated output files of each module can then also be used by other programs (provided that they can read the format). Some (advanced) examples for the usage of the CLI can be found in the experiments folder and the `run.sh` scripts in the hypothesis folders.

The next section provides you with an overview of all the configuration parameters, their data-type, restrictions and also whether they are required.

Execute the CLI by running `python -m rea -h` (in the source folder or after installing). This will provide you with some help.

Configuration Format

Global Keys

Key name	Name in Code	optional/required	description
logging	GlobalKeys.LOGGING	optional	output verbosity. One of the python logging modules log levels
seed	GlobalKeys.SEED	optional	seed used by all modules for reproducibility
metrics_filename	GlobalKeys.METRICS_FILENAME	optional	filename for the metrics generated by the extraction module
rules_filename	GlobalKeys.RULES_FILENAME	optional	filename for the rules generated by the extraction module

Key name	Name in Code	optional/required	description
predict_instance_filename	PREDICT_INSTANCE_FILENAME	optional	name of the output pickle instance, is being used in extraction and evaluation

Keys and Outputs of the Data-Module

Key name	Name in Code	optional/required	description
input_path	DataKeys.INPUT_PATH	optional	path to the dataset from which rules are to be extracted
output_path	DataKeys.OUTPUT_PATH	optional	Path to the folder to fill with output. If directory doesn't exist, one will be created.
label_col	DataKeys.LABEL_COL	optional	index or name of the column containing the labels, i.e. the feature to be predicted
orig_shape	DataKeys.ORIG_SHAPE	optional	specifies the original shape of the dataset before any conversion
test_size	DataKeys.TEST_SIZE	optional	percentage of the data used for testing
dataset_name	DataKeys.DATASET_NAME	optional	friendly name to give to the dataset
cat_conv_method	DataKeys.CAT_CONV_METHOD	optional	Method for categorical conversion
categorical_columns	DataKeys.CATEGORICAL_COLUMNS	optional	List of categorical columns to be converted in the dataset
scale_data	DataKeys.SCALE_DATA	optional	flag to MinMaxScale the input data

File	Description
`x_train.npy`	Training data in numpy format
`y_train.npy`	Trainin labels in numpy format
`x_test.npy`	Test data in numpy format
`y_test.npy`	Test labels in numpy format
`encoder.pickle`	Lable encoder, instance of the scikit `LabelEncoder` serialized with pickle
`metadata.json`	Remaining important fields, like `original_size`

Keys and Output for the Model-Module

Key name	Name in Code	optional/required	description
nwtype	ModelKeys.TYPE	required	The type of network to use ("ff" or "conv")
hidden_layer_units	ModelKeys.HIDDEN_LAYERS	required	the number layers and number of units for each hidden layer
hidden_layer_activations	ModelKeys.HIDDEN_LAYER_ACTIVATIONS	required	specifies the activation function used for each hidden layer
conv_layer_kernels	ModelKeys.CONV_LAYER_KERNELS	required	kernel to be used in each layer of a convolutional network
use_class_weights	ModelKeys.USE_CLASS_WEIGHTS	optional	flag that enables or disables precomputed class_weights
batch_size	ModelKeys.BATCH_SIZE	optional	number of samples per gradient update
epochs	ModelKeys.EPOCHS	optional	number of epochs to train the model
output_path	ModelKeys.OUTPUT_PATH	required	path where to save the model

Key name	Name in Code	optional/required	description
learning_rate	ModelKeys.LEARNING_RATE	optional	Learning rate for adam optimizer or initial learning rate for exponential decay
use_decay	ModelKeys.USE_DECAY	optional	flag that enables use of adam with exponential decay
dropout	ModelKeys.DROPOUT	optional	Rate for keras `Dropout` layer
val_split	ModelKeys.VAL_SPLIT	optional	percentage of the training data used for validation
data_path	ModelKeys.DATA_PATH	required	path to the `Data` output folder

File	Description
`history.png`	Visualization of training-process measures (validation loss/accuracy)
Keras Model Files	The trained tensorflow model

Keys and Output for the Extraction-Module

Key name	Name in Code	optional/required	description
trained_model_path	ExtractionKeys.MODEL_PATH	required	path where to load the model from
data_path	ExtractionKeys.DATA_PATH	required	path to the `Data` output folder
algorithm	ExtractionKeys.ALGORITHM	required	extraction algorithm to use, either "alpa" or "dnnre"
rules_dir	ExtractionKeys.OUTPUT_PATH	required	path of folder to save rules and metrics

File	Description
`eval_metrics.json`	Contains metrics, such as execution time, memory and best rho for ALPA
`rule_classifier.pickle`	R C5.0 prediction instance serialized with pickle
`rules.bin`	Extracted rules, serialized with pickle from the internal format

Keys and Output for the Evaluation-Module

Key name	Name in Code	optional/required	description
trained_model_path	EvaluationKeys.MODEL_PATH	required	path to the trained tensorflow model
rules_dir	EvaluationKeys.RULES_DIR	required	path to the rules folder
data_path	EvaluationKeys.DATA_PATH	required	path to the `Data` output folder
evaluation_dir	EvaluationKeys.OUTPUT_PATH	required	path of folder to save results of evaluation

File	Description
`test_eval.json`	Raw data produced by the evaluation on test set
`test_eval.md`	Evaluation report (based on raw data) for test set
`train_eval.json`	Raw data produced by the evaluation on train set
`train_eval.md`	Evaluation report (based on raw data) for train set
confusion matrices	pngs of the confusion matrices