NiaARM

Release 0.3.2

Žiga Stupan, Iztok Fister Jr.

May 30, 2023

USER DOCUMENTATION

NiaARM is a minimalistic framework for numerical association rule mining.

- **Free software:** MIT license
- **Github repository:** https://github.com/firefly-cpp/NiaARM
- **Python versions:** 3.7.x, 3.8.x, 3.9.x, 3.10.x

CHAPTER

ONE

GENERAL OUTLINE OF THE FRAMEWORK

NiaARM is a framework for Association Rule Mining based on nature-inspired algorithms for optimization. The framework is written fully in Python and runs on all platforms. NiaARM allows users to preprocess the data in a transaction database automatically, to search for association rules and provide a pretty output of the rules found. This framework also supports numerical and real-valued types of attributes besides the categorical ones. Mining the association rules is defined as an optimization problem, and solved using the nature-inspired algorithms that come from the related framework called NiaPy.

CHAPTER

TWO

DETAILED INSIGHTS

The current version includes (but is not limited to) the following functions:

- loading datasets in CSV format,
- preprocessing of data,
- searching for association rules,
- providing output of mined association rules,
- generating statistics about mined association rules,
- visualization of association rules,
- association rule text mining (experimental).

CHAPTER

THREE

DOCUMENTATION

The main documentation is organized into a couple of sections:

- *User Documentation*
- *Developer Documentation*
- *API Reference*
- *About*

3.1 Getting Started

This section is going to show you how to use the NiaARM framework.

3.1.1 Installation

You can install NiaARM package using the following command:

```
pip install niaarm
```

3.1.2 Usage

Loading Data

In NiaARM, data loading is done via the `Dataset` class. There are two options for loading data:

Option 1: Directly from file

```
from niaarm import Dataset

dataset = Dataset('Abalone.csv')
print(dataset)
```

Option 2: From a pandas DataFrame (recommended)

This option is recommended, as it allows you to preprocess the data before mining.

```
import pandas as pd
from niaarm import Dataset


df = pd.read_csv('Abalone.csv')
# Preprocess the dataframe...
dataset = Dataset(df)
print(dataset)
```

Output:

```
DATASET INFO:
Number of transactions: 4177
Number of features: 9

FEATURE INFO:

                    Sex Length Diameter Height Whole weight Shucked weight Viscera␣
↪weight Shell weight Rings
dtype       categorical  float    float  float        float          float         ␣
↪float        float   int
min_val              N/A  0.075    0.055    0.0        0.002          0.001         0.
↪0005       0.0015     1
max_val              N/A  0.815     0.65   1.13       2.8255          1.488           0.
↪76        1.005    29
categories     [M, F, I]    N/A      N/A    N/A          N/A            N/A            N/
↪A          N/A   N/A
```

Preprocessing

The *preprocessing* module provides functions for preprocessing transaction data. The only preprocessing method currently implemented is the *squash()* method, presented in this paper.

```
from niaarm.dataset import Dataset
from niaarm.preprocessing import squash


dataset = Dataset('datasets/Abalone.csv')
squashed = squash(dataset, threshold=0.9, similarity='euclidean')
print(squashed)
```

Output:

```
DATASET INFO:
Number of transactions: 626
Number of features: 9

FEATURE INFO:

                  Sex Length Diameter Height Whole weight Shucked weight Viscera weight␣
↪Shell weight Rings
dtype        category  float    float  float        float          float           float ␣
```

(continues on next page)

(continued from previous page)

```
↪      float   int
min_val           N/A  0.075    0.055    0.0        0.002          0.001          0.0005 ␣
↪     0.0015     1
max_val           N/A  0.815     0.65   1.13       2.8255          1.488           0.76 ␣
↪      1.005    29
categories  [F, I, M]    N/A      N/A    N/A          N/A            N/A            N/A ␣
↪        N/A   N/A
```

Mining Association Rules

Once the data has been loaded, we can run our mining algorithm.

The key component here is our *NiaARM* class, which inherits from NiaPy's Problem class. It implements numerical association rule mining as a real valued, single objective, unconstrained maximization problem (more details on this approach can be found here and here). To summarize, for each solution vector a *Rule* is built, and it's fitness is computed as a weighted sum of selected interest measures (metrics). The rule is then appended to a list of rules, which can be accessed through the NiaARM class.

The *NiaARM* class takes the dataset's dimension (calculated dimension of the optimization problem), features, and transactions (all attributes of the *Dataset* class) and the metrics selected for the fitness function. The metrics can either be passed in as a sequence of strings, in which case the weights of the metrics will be set to 1, or you can pass in a dict containing pairs of `{'metric_name': weight}`. You can also enable logging of fitness improvements by setting the `logging` parameter to `True`.

Bellow is a simple example of mining association rules on the Abalone dataset that we loaded above. For this example we picked Differential Evolution, specifically DE/rand/1/bin, which we'll be running for 50 iterations. All available algorithms can be found in the NiaPy documentation. We've selected the metrics: 'support', 'confidence', 'inclusion' and 'amplitude' for the fitness function. We then sort the rules by fitness in descending order and export them to csv.

```
from niaarm import NiaARM
from niapy.task import OptimizationType, Task
from niapy.algorithms.basic import DifferentialEvolution


# DE/rand/1/bin
algorithm = DifferentialEvolution(population_size=50,
                                  differential_weight=0.8,
                                  crossover_probability=0.9)

metrics = ('support', 'confidence', 'inclusion', 'amplitude')

problem = NiaARM(dataset.dimension, dataset.features, dataset.transactions, metrics,␣
↪logging=True)
task = Task(problem, max_iters=50, optimization_type=OptimizationType.MAXIMIZATION)

algorithm.run(task)

problem.rules.sort(by='fitness', reverse=True)
problem.rules.to_csv('output.csv')
```

The mined rules are stored in `problem.rules`, a *RuleList*. A RuleList is a thin wrapper around a normal python list, with the added functionalities of sorting by metric, exporting rules to csv, and properties for getting statistical data about the rules. Printing a RuleList prints a statistical report of the rules in it.

Output:

```
Fitness: 0.4421065111459649, Support: 0.00023940627244433804, Confidence: 1.0,␣
↪Inclusion: 0.3333333333333333, Amplitude: 0.43485330497808217
Fitness: 0.5363319939110781, Support: 0.006942781900885803, Confidence: 0.
↪9354838709677419, Inclusion: 0.5555555555555556, Amplitude: 0.6473457672201293
Fitness: 0.5395969006117709, Support: 0.1812305482403639, Confidence: 0.9895424836601308,
↪ Inclusion: 0.4444444444444444, Amplitude: 0.5431701261021447
Fitness: 0.5560783231641568, Support: 0.0023940627244433805, Confidence: 1.0, Inclusion:␣
↪0.6666666666666666, Amplitude: 0.5552525632655172
Fitness: 0.5711107256845077, Support: 0.5997127124730668, Confidence: 1.0, Inclusion: 0.
↪3333333333333333, Amplitude: 0.3513968569316307
Fitness: 0.5970815767218225, Support: 0.8099114196791956, Confidence: 0.9955856386109476,
↪ Inclusion: 0.3333333333333333, Amplitude: 0.2494959152638132
Fitness: 0.6479501714015481, Support: 0.7455111323916687, Confidence: 0.9860671310956302,
↪ Inclusion: 0.3333333333333333, Amplitude: 0.5268890887855602
Fitness: 0.6497709183879634, Support: 0.9820445295666747, Confidence: 1.0, Inclusion: 0.
↪4444444444444444, Amplitude: 0.17259469954073503
Fitness: 0.6522418829904134, Support: 0.9176442422791478, Confidence: 0.9422320550639135,
↪ Inclusion: 0.4444444444444444, Amplitude: 0.304646790174148
Fitness: 0.6600433108204055, Support: 0.9762987790280105, Confidence: 1.0, Inclusion: 0.
↪5555555555555556, Amplitude: 0.1083189086980556
Fitness: 0.6625114159138297, Support: 0.9209959300933684, Confidence: 1.0, Inclusion: 0.
↪3333333333333333, Amplitude: 0.39571640022861654
Fitness: 0.6748446186051374, Support: 0.9916207804644481, Confidence: 0.9916207804644481,
↪ Inclusion: 0.4444444444444444, Amplitude: 0.27169246904720923
Fitness: 0.6868285539707781, Support: 0.949006463969356, Confidence: 0.9927372902579514,␣
↪Inclusion: 0.5555555555555556, Amplitude: 0.25001490610024923
Rules exported to output.csv
```

Mining Association Rules (Simplified)

In addition to the above interface, we provide a much simpler one in the form of a simple function: *get_rules*. The function accepts a dataset object, an algorithm, sequence or dict of metrics, a stopping condition (either `max_evals` or `max_iters`) and a `logging` flag. The algorithm can either be a NiaPy Algorithm instance, or a string, in which case it's parameters can be passed in to the function as additional keyword arguments.

The *get_rules* function returns a named tuple of (rules, run_time), where rules is a *RuleList* and run_time is the run time of the algorithm in seconds.

The same example as above, using *get_rules*:

```
from niaarm import get_rules
from niapy.algorithms.basic import DifferentialEvolution


# DE/rand/1/bin
algorithm = DifferentialEvolution(population_size=50,
                                  differential_weight=0.8,
                                  crossover_probability=0.9)

metrics = ('support', 'confidence', 'inclusion', 'amplitude')
rules, run_time = get_rules(dataset, algorithm, metrics, max_iters=50)
```

(continues on next page)

(continued from previous page)

```
print(rules)
print(f'Run Time: {run_time:.4f} seconds')
rules.to_csv('output.csv')
```

Output:

```
STATS:
Total rules: 1153
Average fitness: 0.47320577312454054
Average support: 0.3983325861836626
Average confidence: 0.7050696319555724
Average lift: 1.8269022321777044
Average coverage: 0.5791478590164908
Average consequent support: 0.6708142990119975
Average conviction: 80294763647830.92
Average amplitude: 0.33832710930158877
Average inclusion: 0.45109376505733834
Average interestingness: 0.4107718184209992
Average comprehensibility: 0.6225319999993354
Average netconf: 0.08165217509315073
Average Yule's Q: 0.2631267094311884
Average length of antecedent: 2.248048568950564
Average length of consequent: 1.8117953165654814
Run Time: 6.9498 seconds
Rules exported to output.csv
```

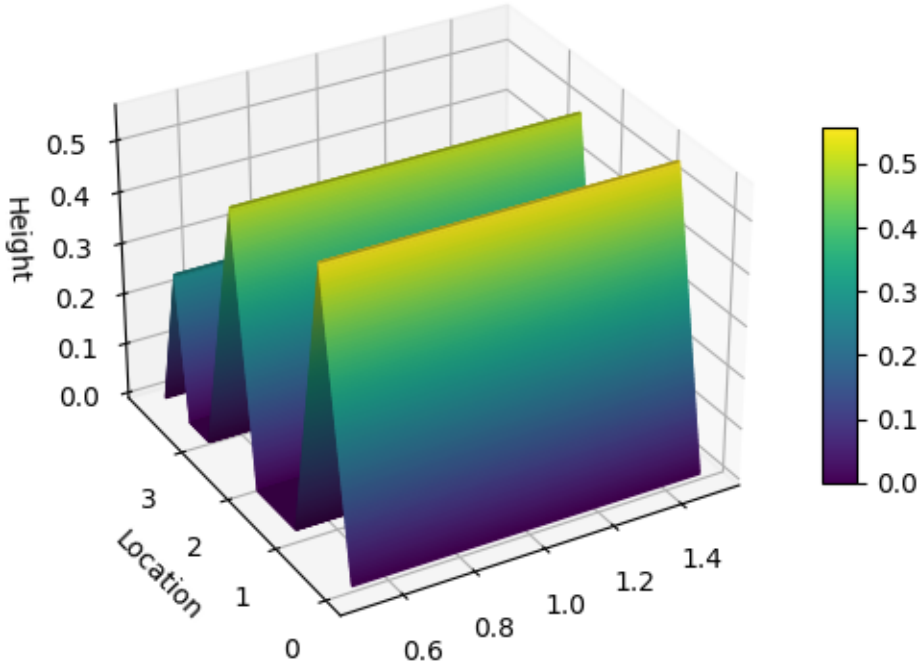
Visualization

The *visualize* module provides functions for plotting association rules. The only visualization method currently implemented is the *hill_slopes()* method, presented in this paper.

```
from matplotlib import pyplot as plt
from niaarm import Dataset, RuleList, get_rules
from niaarm.visualize import hill_slopes

dataset = Dataset('datasets/Abalone.csv')
metrics = ('support', 'confidence')
rules, _ = get_rules(dataset, 'DifferentialEvolution', metrics, max_evals=1000,␣
↪seed=1234)
some_rule = rules[150]
hill_slopes(some_rule, dataset.transactions)
plt.show()
```

Output:



Text Mining (Experimental)

An experimental implementation of association rule text mining using nature-inspired algorithms is also provided. The *niaarm.text* module contains the *Corpus* and *Document* classes for loading and preprocessing corpora, a *TextRule* class, representing a text rule, and the *NiaARTM* class, implementing association rule text mining as a continuous optimization problem. The *get_text_rules()* function, equivalent to *get_rules()*, but for text mining, was also added to the *niaarm.mine* module.

```
import pandas as pd
from niaarm.text import Corpus
from niaarm.mine import get_text_rules
from niapy.algorithms.basic import ParticleSwarmOptimization

df = pd.read_json('datasets/text/artm_test_dataset.json', orient='records')
documents = df['text'].tolist()
corpus = Corpus.from_list(documents)

algorithm = ParticleSwarmOptimization(population_size=200, seed=123)
metrics = ('support', 'confidence', 'aws')
rules, time = get_text_rules(corpus, max_terms=5, algorithm=algorithm, metrics=metrics,␣
↪max_evals=10000, logging=True)

if len(rules):
```

(continues on next page)

(continued from previous page)

```
    print(rules)
    print(f'Run time: {time:.2f}s')
    rules.to_csv('output.csv')
else:
    print('No rules generated')
    print(f'Run time: {time:.2f}s')
```

Note: You may need to download stopwords and the punkt tokenizer from nltk by running *import nltk; nltk.download('stopwords'); nltk.download('punkt')*.

Output:

```
Fitness: 0.53345778328699, Support: 0.1111111111111111, Confidence: 1.0, Aws: 0.
↪48926223874985886
Fitness: 0.7155830770302328, Support: 0.1111111111111111, Confidence: 1.0, Aws: 1.
↪0356381199795872
Fitness: 0.7279963436805833, Support: 0.1111111111111111, Confidence: 1.0, Aws: 1.
↪072877919930639
Fitness: 0.7875917299029188, Support: 0.1111111111111111, Confidence: 1.0, Aws: 1.
↪251664078597645
Fitness: 0.8071206688346807, Support: 0.1111111111111111, Confidence: 1.0, Aws: 1.
↪310250895392931
STATS:
Total rules: 52
Average fitness: 0.5179965084882088
Average support: 0.11538461538461527
Average confidence: 0.7115384615384616
Average lift: 5.524038461538462
Average coverage: 0.17948717948717943
Average consequent support: 0.1517094017094015
Average conviction: 1568561408678185.8
Average amplitude: nan
Average inclusion: 0.007735042735042727
Average interestingness: 0.6170069642291859
Average comprehensibility: 0.6763685578758655
Average netconf: 0.6675824175824177
Average Yule's Q: 0.9670329670329672
Average antecedent length: 1.6346153846153846
Average consequent length: 1.8461538461538463

Run time: 13.37s
Rules exported to output.csv
```

3.1.3 Interest Measures

The framework currently implements the following interest measures (metrics):

- Support
- Confidence
- Lift[1]
- Coverage
- RHS Support
- Conviction[1]
- Inclusion
- Amplitude
- Interestingness
- Comprehensibility
- Netconf[1]
- Yule's Q[1]

More information about these interest measures can be found in the API reference of the *Rule* class.

3.1.4 Examples

You can find the full code and usage examples here.

3.2 Command Line Interface

We provide a simple command line interface, which allows you to easily mine association rules on any input dataset, output them to a csv file and/or perform a simple statistical analysis on them.

3.2.1 Usage

```
niaarm -h  # or python -m niaarm -h
```

```
usage: niaarm [-h] [-v] -i INPUT_FILE [-o OUTPUT_FILE] -a ALGORITHM [-s SEED]
              [--max-evals MAX_EVALS] [--max-iters MAX_ITERS] --metrics
              METRICS [METRICS ...] [--weights WEIGHTS [WEIGHTS ...]] [--log]
              [--show-stats]

Perform ARM, output mined rules as csv, get mined rules' statistics

options:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
  -i INPUT_FILE, --input-file INPUT_FILE
```

(continues on next page)

[1] Not available as fitness metrics.

(continued from previous page)

```
                        Input file containing a csv dataset
  -o OUTPUT_FILE, --output-file OUTPUT_FILE
                        Output file for mined rules
  -a ALGORITHM, --algorithm ALGORITHM
                        Algorithm to use (niapy class name, e.g.
                        DifferentialEvolution)
  -s SEED, --seed SEED  Seed for the algorithm's random number generator
  --max-evals MAX_EVALS
                        Maximum number of fitness function evaluations
  --max-iters MAX_ITERS
                        Maximum number of iterations
  --metrics METRICS [METRICS ...]
                        Metrics to use in the fitness function.
  --weights WEIGHTS [WEIGHTS ...]
                        Weights in range [0, 1] corresponding to --metrics
  --log                 Enable logging of fitness improvements
  --stats               Display stats about mined rules
```

Exporting Rules to CSV

Mine Association rules on the Abalone dataset (available here) and output them to a csv file. We'll run Differential evolution for 30 iterations, logging fitness improvements. We selected the support and confidence metrics, their weights will defaulting to 1.

```
niaarm -i Abalone.csv -a DifferentialEvolution --max-iters 30 --metrics support␣
↪confidence -o output.csv --log
```

After running the above command we are prompted to edit the algorithms parameters in a text editor (vi or nano on unix, notepad on windows):

After we're done editing the parameters, we save the file and exit the editor, so the algorithm can run. The output should look like this:

```
Fitness: 0.006713839591358101, Support: 0.006703375628441465, Confidence: 0.
↪0067243035542747355
Fitness: 0.011814753063668868, Support: 0.00574575053866411 3, Confidence: 0.
↪01788375558867362
Fitness: 0.4774755380849042, Support: 0.027531721331098876, Confidence: 0.
↪9274193548387096
Fitness: 0.47886170567035946, Support: 0.24323677280344744, Confidence: 0.
↪7144866385372715
Fitness: 0.5001197031362221, Support: 0.00023940627244433804, Confidence: 1.0
Fitness: 0.5002394062724443, Support: 0.00047881254488867607, Confidence: 1.0
Fitness: 0.6182100887777294, Support: 0.2824994014843189, Confidence: 0.9539207760711399
Fitness: 0.7280954808121962, Support: 0.7115154417045727, Confidence: 0.7446755199198196
Fitness: 0.9669248968790327, Support: 0.9492458702418003, Confidence: 0.9846039235162652
Fitness: 1.0, Support: 1.0, Confidence: 1.0

Rules exported to output.csv
```

The first 10 rules of the generated output.csv file:

antecedent	consequent	fitness	support	confidenc	lift	coverage	rhs_s	conviction	amplitude	inclusion	interestingness	comprehensibility	netconf	yulesq	
[Height([0.0, 1.13])]	[Shell weight([0.001 1.005])]	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.222	0.999	0.630	0.0	-1.0	74
[Shucked weight([0.001 1.488]), Viscera weight([0.000 0.76])]	[Rings([1, 29])]	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.333	0.999	0.5	0.0	-1.0	
[Shucked weight([0.001 1.488])]	[Viscera weight([0.000 0.76])]	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.222	0.999	0.630	0.0	-1.0	74
[Rings([1, 29])]	[Viscera weight([0.000 0.76])]	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.222	0.999	0.630	0.0	-1.0	74
[Rings([1, 29])]	[Viscera weight([0.000 0.76]), Shucked weight([0.001 1.488])]	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.333	0.999	0.792	0.0	-1.0	81
[Shucked weight([0.001 1.488]), Rings([1, 29])]	[Viscera weight([0.000 0.76])]	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.333	0.999	0.5	0.0	-1.0	
[Rings([1, 29]), Shucked weight([0.001 1.488])]	[Viscera weight([0.000 0.76])]	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.333	0.999	0.5	0.0	-1.0	
[Shucked weight([0.001 1.488]), Rings([1, 29])]	[Whole weight([0.002 2.8255]), Viscera weight([0.000 0.76])]	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.444	0.999	0.682	0.0	-1.0	54
[Rings([1, 29]), Whole weight([0.002 2.8255])]	[Viscera weight([0.000 0.76])]	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.333	0.999	0.5	0.0	-1.0	
[Shucked weight([0.001 1.488])]	[Height([0.0, 1.13]), Viscera weight([0.000 0.76])]	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.333	0.999	0.792	0.0	-1.0	81

Displaying Statistics

With the `--stats` flag we can print basic statistics about the mined association rules. E.g. (for the above run):

```
STATS:
Total rules: 571
Average fitness: 0.41468758207787
Average support: 0.2218516293252978
Average confidence: 0.6075235348304421
Average lift: 4.594338596271166
Average coverage: 0.42734229269031015
Average consequent support: 0.5431864178239016
Average conviction: 206259068653654.78
Average amplitude: 0.42957104685221487
Average inclusion: 0.39307258221443864
Average interestingness: 0.23451084908249198
Average comprehensibility: 0.6063087509647604
Average netconf: 0.07274126434826349
Average Yule's Q: 0.779065174397917
Average length of antecedent: 1.97723292469352
Average length of consequent: 1.5604203152364273
Run Time: 6.4538s
```

3.3 Installation

3.3.1 Setup development environment

Requirements

- Poetry: https://python-poetry.org/docs/

After installing Poetry and cloning the project from GitHub, you should run the following command from the root of the cloned project:

```
$ poetry install
```

All of the project's dependencies should be installed and the project ready for further development. **Note that Poetry creates a separate virtual environment for your project.**

Dependencies

Package	Version	Platform
niapy	^2.0.1	All
pandas	^1.3.5	All
numpy	^1.21.5	All

Development Dependencies

Package	Version	Platform
Pytest	^7.0.1	Any

Extras

Package	Version	Platform
Sphinx	^4.4.0	Any
sphinx-rtd-theme	^1.0.0	Any
sphinxcontrib-bibtex	^2.4.1	Any

3.4 Testing

Before making a pull request, if possible provide tests for added features or bug fixes.

In case any of the test cases fails, those should be fixed before we merge your pull request to master branch.

For the purpose of checking if all test are passing locally you can run following command:

```
$ poetry run pytest
```

3.5 Documentation

To locally generate and preview documentation run the following commands in the project root folder:

```
poetry install --extras docs
poetry run sphinx-build ./docs ./docs/_build
```

If the build of the documentation is successful, you can preview the documentation in the docs/_build folder by clicking the `index.html` file.

3.6 API Reference

3.6.1 Dataset

class niaarm.dataset.**Dataset**(*path_or_df*, *delimiter=','*, *header=0*, *names=None*)

Bases: `object`

Class for working with a dataset.

Parameters

- **path_or_df** (*Union[str, os.PathLike, pandas.DataFrame]*) – Path to the dataset (csv) file or a pandas DataFrame.
- **delimiter** (*str*) – The delimiter in the csv file.

- **header** (*Optional[int]*) – Row to use as header (zero-based). Default: 0. Pass `header=None` if the file doesn't contain a header.
- **names** (*Optional[list[str]]*) – List of feature names to use. If the file already contains a header row, pass `header=0` to override the feature names.

transactions

Transactional data.

Type
pandas.DataFrame

header

Feature names.

Type
list[str]

features

List of features.

Type
list[*Feature*]

dimension

Dimension of the optimization problem for the dataset.

Type
int

3.6.2 Preprocessing

niaarm.preprocessing.**squash**(*dataset*, *threshold*, *similarity='euclidean'*)

Squash dataset.

Parameters

- **dataset** (Dataset) – Dataset to squash.
- **threshold** (*float*) – Similarity threshold. Should be between 0 and 1.
- **similarity** (*str*) – Similarity measure for comparing transactions (euclidean or cosine). Default: 'euclidean'.

Returns
Squashed dataset.

Return type
Dataset

3.6.3 Feature

class niaarm.feature.**Feature**(*name*, *dtype*, *min_val=None*, *max_val=None*, *categories=None*)

Bases: `object`

Class representing a feature.

Parameters

- **name** (*str*) – Name of the feature.
- **dtype** (*str*) – Datatype of the feature.
- **min_val** (*Optional[float]*) – Minimum value of the feature in the transaction database.
- **max_val** (*Optional[float]*) – Maximum value of the feature in the transaction database.
- **categories** (*Optional[list[str]]*) – Possible categorical feature's values.

3.6.4 Mine

class niaarm.mine.**Result**(*rules*, *run_time*)

Result of an algorithm run as a `namedtuple`.

rules

A list of mined association rules.

Type
RuleList

run_time

The run time of the algorithm in seconds.

Type
float

niaarm.mine.**get_rules**(*dataset*, *algorithm*, *metrics*, *max_evals=inf*, *max_iters=inf*, *logging=False*, ***kwargs*)

Mine association rules on a dataset.

Parameters

- **dataset** (Dataset) – Dataset to mine rules on.
- **algorithm** (*Union[niapy.algorithms.Algorithm, str]*) – Algorithm to use. Can be either an Algorithm object or the class name as a string. In that case, algorithm parameters can be passed in as keyword arguments.
- **metrics** (*Union[Dict[str, float], Sequence[str]]*) – Metrics to take into account when computing the fitness. Metrics can either be passed as a Dict of pairs {'metric_name': <weight of metric>} or a sequence of metrics as strings, in which case, the weights of the metrics will be set to 1.
- **max_evals** (*Optional[int]*) – Maximum number of iterations. Default: `inf`. At least one of `max_evals` or `max_iters` must be provided.
- **max_iters** (*Optional[int]*) – Maximum number of fitness evaluations. Default: `inf`.
- **logging** (*bool*) – Enable logging of fitness improvements. Default: `False`.

Returns
A named tuple containing the list of mined rules and the algorithm's run time in seconds.

Return type
Result

niaarm.mine.**get_text_rules**(*corpus*, *max_terms*, *algorithm*, *metrics*, *smooth=True*, *norm=2*, *threshold=0*, *max_evals=inf*, *max_iters=inf*, *logging=False*, ***kwargs*)

Mine association rules in a text corpus.

Parameters

- **corpus** (Corpus) – Dataset to mine rules on.
- **max_terms** (*int*) – Maximum number of terms in association rule.
- **algorithm** (*Union[niapy.algorithms.Algorithm, str]*) – Algorithm to use. Can be either an Algorithm object or the class name as a string. In that case, algorithm parameters can be passed in as keyword arguments.
- **metrics** (*Union[Dict[str, float], Sequence[str]]*) – Metrics to take into account when computing the fitness. Metrics can either be passed as a Dict of pairs {'metric_name': <weight of metric>} or a sequence of metrics as strings, in which case, the weights of the metrics will be set to 1.
- **smooth** (*bool*) – Smooth idf to prevent division by 0 error. Default: True.
- **norm** (*int*) – Order of norm for normalizing the tf-idf matrix. Default: 2.
- **threshold** (*Optional[float]*) – Threshold of tf-idf weights. If a weight is less than or equal to the threshold, the term is not included in the transaction. Default: 0.
- **max_evals** (*Optional[int]*) – Maximum number of iterations. Default: inf. At least one of max_evals or max_iters must be provided.
- **max_iters** (*Optional[int]*) – Maximum number of fitness evaluations. Default: inf.
- **logging** (*bool*) – Enable logging of fitness improvements. Default: False.

Returns
A named tuple containing the list of mined rules and the algorithm's run time in seconds.

Return type
Result

3.6.5 NiaARM

class niaarm.niaarm.**NiaARM**(*dimension*, *features*, *transactions*, *metrics*, *logging=False*)

Bases: Problem

Representation of Association Rule Mining as an optimization problem.

The implementation is composed of ideas found in the following papers:

- I. Fister Jr., A. Iglesias, A. Gálvez, J. Del Ser, E. Osaba, I Fister. [Differential evolution for association rule mining using categorical and numerical attributes] (http://www.iztok-jr-fister.eu/static/publications/231.pdf) In: Intelligent data engineering and automated learning - IDEAL 2018, pp. 79-88, 2018.
- I. Fister Jr., V. Podgorelec, I. Fister. [Improved Nature-Inspired Algorithms for Numeric Association Rule Mining] (https://link.springer.com/chapter/10.1007/978-3-030-68154-8_19) In: Vasant P., Zelinka I., Weber GW. (eds.) Intelligent Computing and Optimization. ICO 2020. Advances in Intelligent Systems and Computing, vol 1324. Springer, Cham.

Parameters

- **dimension** (*int*) – Dimension of the optimization problem for the dataset.
- **features** (*list[Feature]*) – List of the dataset's features.
- **transactions** (*pandas.Dataframe*) – The dataset's transactions.
- **metrics** (*Union[Dict[str, float], Sequence[str]]*) – Metrics to take into account when computing the fitness. Metrics can either be passed as a Dict of pairs {'metric_name': <weight of metric>} or a sequence of metrics as strings, in which case, the weights of the metrics will be set to 1.
- **logging** (*bool*) – Enable logging of fitness improvements. Default: `False`.

rules

A list of mined association rules.

Type
RuleList

3.6.6 Rule

class niaarm.rule.**Rule**(*antecedent*, *consequent*, *fitness=0.0*, *transactions=None*)

Bases: `object`

Class representing an association rule.

Parameters

- **antecedent** (*list[Feature]*) – A list of antecedents of the association rule.
- **consequent** (*list[Feature]*) – A list of consequents of the association rule.
- **fitness** (*Optional[float]*) – Fitness value of the association rule.
- **transactions** (*Optional[pandas.DataFrame]*) – Transactional database.

cls.**metrics**

List of all available interest measures.

Type
tuple[str]

support

Support is defined on an itemset as the proportion of transactions that contain the attribute X.

$supp(X) = \frac{n_X}{|D|}$,

where $|D|$ is the number of records in the transactional database.

For an association rule, support is defined as the support of all the attributes in the rule.

$supp(X \implies Y) = \frac{n_{XY}}{|D|}$

Range: $[0, 1]$

Reference: Michael Hahsler, A Probabilistic Comparison of Commonly Used Interest Measures for Association Rules, 2015, URL: https://mhahsler.github.io/arules/docs/measures

`confidence`

Confidence of the rule, defined as the proportion of transactions that contain the consequent in the set of transactions that contain the antecedent. This proportion is an estimate of the probability of seeing the consequent, if the antecedent is present in the transaction.

$conf(X \implies Y) = \frac{supp(X \implies Y)}{supp(X)}$

Range: $[0, 1]$

Reference: Michael Hahsler, A Probabilistic Comparison of Commonly Used Interest Measures for Association Rules, 2015, URL: https://mhahsler.github.io/arules/docs/measures

`lift`

Lift measures how many times more often the antecedent and the consequent Y occur together than expected if they were statistically independent.

$lift(X \implies Y) = \frac{conf(X \implies Y)}{supp(Y)}$

Range: $[0, \infty]$ (1 means independence)

Reference: Michael Hahsler, A Probabilistic Comparison of Commonly Used Interest Measures for Association Rules, 2015, URL: https://mhahsler.github.io/arules/docs/measures

`coverage`

Coverage, also known as antecedent support, is an estimate of the probability that the rule applies to a randomly selected transaction. It is the proportion of transactions that contain the antecedent.

$cover(X \implies Y) = supp(X)$

Range: $[0, 1]$

Reference: Michael Hahsler, A Probabilistic Comparison of Commonly Used Interest Measures for Association Rules, 2015, URL: https://mhahsler.github.io/arules/docs/measures

`rhs_support`

Support of the consequent.

$RHSsupp(X \implies Y) = supp(Y)$

Range: $[0, 1]$

Reference: Michael Hahsler, A Probabilistic Comparison of Commonly Used Interest Measures for Association Rules, 2015, URL: https://mhahsler.github.io/arules/docs/measures

`conviction`

Conviction can be interpreted as the ratio of the expected frequency that the antecedent occurs without the consequent.

$conv(X \implies Y) = \frac{1-supp(Y)}{1-conf(X \implies Y)}$

Range: $[0, \infty]$ (1 means independence, ∞ means the rule always holds)

Reference: Michael Hahsler, A Probabilistic Comparison of Commonly Used Interest Measures for Association Rules, 2015, URL: https://mhahsler.github.io/arules/docs/measures

`inclusion`

Inclusion is defined as the ratio between the number of attributes of the rule and all attributes in the database.

$inclusion(X \implies Y) = \frac{|X \cup Y|}{m}$,

where m is the total number of attributes in the transactional database.

Range: $[0, 1]$

Reference: I. Fister Jr., V. Podgorelec, I. Fister. Improved Nature-Inspired Algorithms for Numeric Association Rule Mining. In: Vasant P., Zelinka I., Weber GW. (eds) Intelligent Computing and Optimization. ICO 2020. Advances in Intelligent Systems and Computing, vol 1324. Springer, Cham.

`amplitude`

Amplitude measures the quality of a rule, preferring attributes with smaller intervals.

$ampl(X \implies Y) = 1 - \frac{1}{n} \sum_{k=1}^{n} \frac{Ub_k - Lb_k}{max(o_k) - min(o_k)}$,

where n is the total number of attributes in the rule, Ub_k and Lb_k are upper and lower bounds of the selected attribute, and $max(o_k)$ and $min(o_k)$ are the maximum and minimum feasible values of the attribute o_k in the transactional database.

Range: $[0, 1]$

Reference: I. Fister Jr., I. Fister A brief overview of swarm intelligence-based algorithms for numerical association rule mining. arXiv preprint arXiv:2010.15524 (2020).

`interestingness`

Interestingness of the rule, defined as:

$interest(X \implies Y) = \frac{supp(X \implies Y)}{supp(X)} \cdot \frac{supp(X \implies Y)}{supp(Y)} \cdot (1 - \frac{supp(X \implies Y)}{|D|})$

Here, the first part gives us the probability of generating the rule based on the antecedent, the second part gives us the probability of generating the rule based on the consequent and the third part is the probability that the rule won't be generated. Thus, rules with very high support will be deemed uninteresting.

Range: $[0, 1]$

Reference: I. Fister Jr., I. Fister A brief overview of swarm intelligence-based algorithms for numerical association rule mining. arXiv preprint arXiv:2010.15524 (2020).

`comprehensibility`

Comprehensibility of the rule. Rules with fewer attributes in the consequent are more comprehensible.

$comp(X \implies Y) = \frac{log(1+|Y|)}{log(1+|X \cup Y|)}$

Range: $[0, 1]$

Reference: I. Fister Jr., I. Fister A brief overview of swarm intelligence-based algorithms for numerical association rule mining. arXiv preprint arXiv:2010.15524 (2020).

`netconf`

The netconf metric evaluates the interestingness of association rules depending on the support of the rule and the support of the antecedent and consequent of the rule.

$netconf(X \implies Y) = \frac{supp(X \implies Y) - supp(X)supp(Y)}{supp(X)(1 - supp(X))}$

Range: $[-1, 1]$ (Negative values represent negative dependence, positive values represent positive dependence and 0 represents independence)

Reference: E. V. Altay and B. Alatas, "Sensitivity Analysis of MODENAR Method for Mining of Numeric Association Rules," 2019 1st International Informatics and Software Engineering Conference (UBMYK), 2019, pp. 1-6, doi: 10.1109/UBMYK48245.2019.8965539.

`yulesq`

The Yule's Q metric represents the correlation between two possibly related dichotomous events.

$yulesq(X \implies Y) = \frac{supp(X \implies Y)supp(\neg X \implies \neg Y) - supp(X \implies \neg Y)supp(\neg X \implies Y)}{supp(X \implies Y)supp(\neg X \implies \neg Y) + supp(X \implies \neg Y)supp(\neg X \implies Y)}$

Range: $[-1, 1]$ (-1 reflects total negative association, 1 reflects perfect positive association and 0 reflects independence)

Reference: E. V. Altay and B. Alatas, "Sensitivity Analysis of MODENAR Method for Mining of Numeric Association Rules," 2019 1st International Informatics and Software Engineering Conference (UBMYK), 2019, pp. 1-6, doi: 10.1109/UBMYK48245.2019.8965539.

3.6.7 RuleList

class niaarm.rule_list.**RuleList**(*initlist=None*)

Bases: UserList

A list of rules.

get(*metric*)

Get values of *metric* for each rule as a numpy array.

Parameters
metric (*str*) – Metric.

Returns
Array of *metric* for all rules.

Return type
numpy.ndarray

max(*metric*)

Get max value of metric.

Parameters
metric (*str*) – Metric.

Returns
Max value of metric in rule list.

Return type
float

mean(*metric*)

Get mean value of metric.

Parameters
metric (*str*) – Metric.

Returns
Mean value of metric in rule list.

Return type
float

min(*metric*)

Get min value of metric.

Parameters
metric (*str*) – Metric.

Returns
Min value of metric in rule list.

Return type
float

sort(*by='fitness'*, *reverse=True*)

Sort rules by metric.

Parameters

- **by** (*str*) – Metric to sort rules by. Default: `'fitness'`.
- **reverse** (*bool*) – Sort in descending order. Default: `True`.

std(*metric*)

Get standard deviation of metric.

Parameters
metric (*str*) – Metric.

Returns
Standard deviation of metric in rule list.

Return type
float

to_csv(*filename*)

Export rules to csv.

Parameters
filename (*str*) – File to save the rules to.

3.6.8 Text

class niaarm.text.**Corpus**(*documents=None*)

Bases: `object`

The text corpus class.

Parameters
documents (*Optional[list[Document]]*) – List of documents. If `None`, an empty list will be created. Default: `None`.

append(*document*)

Add a document to the corpus.

Parameters
document (`Document`) – Document to append.

classmethod from_directory(*path*, *encoding='utf-8'*, *language='english'*, *remove_stopwords=True*, *lowercase=True*)

Construct corpus from a directory containing plain text files.

Parameters

- **path** (*str*) – Path to directory.
- **encoding** (*str*) – Encoding of the files. Default: 'utf-8'.
- **language** (*str*) – Language of the files. Default: 'english'.
- **remove_stopwords** (*bool*) – If `True`, remove stopwords from text. Default: `True`.
- **lowercase** (*bool*) – If `True`, make text lowercase. Default: `True`.

Returns
The constructed corpus.

Return type
Corpus

classmethod from_list(*lst*, *language='english'*, *remove_stopwords=True*, *lowercase=True*)

Construct corpus from a list of strings.

Parameters

- **lst** (*list[str]*) – List of documents as strings.
- **language** (*str*) – Language of the file. Default: 'english'.
- **remove_stopwords** (*bool*) – If True, remove stopwords from text. Default: True.
- **lowercase** (*bool*) – If True, make text lowercase. Default: True.

Returns
The constructed corpus.

Return type
Corpus

terms()

Get a list of unique terms in the corpus

Returns
List of unique terms in the corpus.

Return type
list[str]

tf_idf_matrix(*smooth=True*, *norm=2*)

Get the tf-idf weights matrix as a pandas DataFrame.

Parameters

- **smooth** (*bool*) – Smooth idf by adding one to the numerator and the denominator to prevent division by 0 errors. Default: True.
- **norm** (*int*) – Order of the norm to normalize the matrix with. Default: 2.

Returns
The tf-idf matrix.

Return type
pd.DataFrame

class niaarm.text.**Document**(*text*, *language='english'*, *remove_stopwords=True*, *lowercase=True*)

Bases: object

A text document class.

Parameters

- **text** (*str*) – Document text.
- **language** (*str*) – Document language. Used for tokenization and stopword removal. Default: 'english'.
- **remove_stopwords** (*bool*) – If True, remove stopwords from text. Default: True.
- **lowercase** (*bool*) – If True, make text lowercase. Default: True.

frequency(*term*)

Get the frequency of a term,

Parameters
term (*str*) – Term to get frequency of.

Returns
Frequency of the term.

Return type
float

classmethod from_file(*path*, *encoding='utf-8'*, *language='english'*, *remove_stopwords=True*, *lowercase=True*)

Construct document from a plain text file.

Parameters

- **path** (*str*) – Path to file.
- **encoding** (*str*) – Encoding of the file. Default: 'utf-8'.
- **language** (*str*) – Language of the file. Default: 'english'.
- **remove_stopwords** (*bool*) – If `True`, remove stopwords from text. Default: `True`.
- **lowercase** (*bool*) – If `True`, make text lowercase. Default: `True`.

Returns
The constructed document.

Return type
Document

class niaarm.text.**NiaARTM**(*max_terms*, *terms*, *transactions*, *metrics*, *threshold=0*, *logging=False*)

Bases: *NiaARM*

Representation of Association Rule Text Mining as an optimization problem.

The implementation is composed of ideas found in the following paper:

- I. Fister, S. Deb, I. Fister, „Population-based metaheuristics for Association Rule Text Mining", in Proceedings of the 2020 4th International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence, New York, NY, USA, mar. 2020, pp. 19–23. doi: 10.1145/3396474.3396493.

Parameters

- **max_terms** (*int*) – Maximum number of terms in association rule..
- **features** (*list[str]*) – List of unique terms in the corpus.
- **transactions** (*pandas.Dataframe*) – The tf-idf matrix.
- **metrics** (*Union[Dict[str, float], Sequence[str]]*) – Metrics to take into account when computing the fitness. Metrics can either be passed as a Dict of pairs {'metric_name': <weight of metric>} or a sequence of metrics as strings, in which case, the weights of the metrics will be set to 1.
- **threshold** (*Optional[float]*) – Threshold of tf-idf weights. If a weight is less than or equal to the threshold, the term is not included in the transaction. Default: 0.
- **logging** (*bool*) – Enable logging of fitness improvements. Default: `False`.

rules

A list of mined text rules.

Type
RuleList

class niaarm.text.**TextRule**(*antecedent*, *consequent*, *fitness=0.0*, *transactions=None*, *threshold=0*)

Bases: *Rule*

Class representing a text association rule.

The class contains all the metrics in *Rule*, except for amplitude, which returns nan.

Parameters

- **antecedent** (*list[str]*) – A list of antecedent terms of the text rule.
- **consequent** (*list[str]*) – A list of consequent terms of the text rule.
- **fitness** (*Optional[float]*) – Fitness value of the text rule.
- **transactions** (*Optional[pandas.DataFrame]*) – The tf-idf matrix as a pandas DataFrame.
- **threshold** (*Optional[float]*) – Threshold of tf-idf weights. If a weight is less than or equal to the threshold, the term is not included in the transaction. Default: 0.

aws

The sum of tf-idf values for all the terms in the rule.

See also:

niaarm.rule.Rule

3.6.9 Visualize

niaarm.visualize.**hill_slopes**(*rule*, *transactions*)

Visualize rule as hill slopes.

Reference: Fister, I. et al. (2020). Visualization of Numerical Association Rules by Hill Slopes. In: Analide, C., Novais, P., Camacho, D., Yin, H. (eds) Intelligent Data Engineering and Automated Learning – IDEAL 2020. IDEAL 2020. Lecture Notes in Computer Science(), vol 12489. Springer, Cham. https://doi.org/10.1007/978-3-030-62362-3_10

Parameters

- **rule** (Rule) – Association rule to visualize.
- **transactions** (*pandas.DataFrame*) – Transactions as a DataFrame.

Returns
Figure and Axes of plot.

Return type
tuple[matplotlib.figure.Figure, matplotlib.axes.Axes]

3.7 Contributing to NiaARM

First off, thanks for taking the time to contribute!

3.7.1 Code of Conduct

This project and everyone participating in it is governed by the *Contributor Covenant Code of Conduct*. By participating, you are expected to uphold this code. Please report unacceptable behavior to iztok.fister1@um.si.

3.7.2 How Can I Contribute?

Reporting Bugs

Before creating bug reports, please check existing issues list as you might find out that you don't need to create one. When you are creating a bug report, please include as many details as possible in the issue template.

Suggesting Enhancements

Open new issue using the feature request template.

Pull requests

Fill in the pull request template and make sure your code is documented.

3.8 Contributor Covenant Code of Conduct

3.8.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

3.8.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

3.8.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

3.8.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

3.8.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at iztok.fister1@um.si. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

3.8.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

3.8.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

3.9 License

```
MIT License

Copyright (c) 2021-2022 Žiga Stupan and Iztok Fister Jr.

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.
```

(continues on next page)

(continued from previous page)

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

References

BIBLIOGRAPHY

[1] Iztok Fister, Andres Iglesias, Akemi Galvez, Javier Del Ser, Eneko Osaba, and Iztok Fister. Differential Evolution for Association Rule Mining Using Categorical and Numerical Attributes. In Hujun Yin, David Camacho, Paulo Novais, and Antonio J. Tallón-Ballesteros, editors, *Intelligent Data Engineering and Automated Learning – IDEAL 2018*, 79–88. Cham, 2018. Springer International Publishing. doi:10.1007/978-3-030-03493-1_9.

[2] Iztok Fister Jr., Vili Podgorelec, and Iztok Fister. Improved Nature-Inspired Algorithms for Numeric Association Rule Mining. In Pandian Vasant, Ivan Zelinka, and Gerhard-Wilhelm Weber, editors, *Intelligent Computing and Optimization*, 187–195. Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-68154-8_19.

[3] Iztok Fister Jr. and Iztok Fister. A brief overview of swarm intelligence-based algorithms for numerical association rule mining. *arXiv:2010.15524 [cs]*, October 2020. doi:10.48550/ARXIV.2010.15524.

[4] Iztok Fister, Suash Deb, and Iztok Fister. Population-based metaheuristics for Association Rule Text Mining. In *Proceedings of the 2020 4th International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence*, ISMSI '20, 19–23. New York, NY, USA, March 2020. Association for Computing Machinery. doi:10.1145/3396474.3396493.

[5] Iztok Fister, Dušan Fister, Andres Iglesias, Akemi Galvez, Eneko Osaba, Javier Del Ser, and Iztok Fister. Visualization of Numerical Association Rules by Hill Slopes. In Cesar Analide, Paulo Novais, David Camacho, and Hujun Yin, editors, *Intelligent Data Engineering and Automated Learning – IDEAL 2020*, 101–111. Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-62362-3_10.

PYTHON MODULE INDEX

n

INDEX

R

S

T

Y

Some radii options for box corners used; they were ignored as pict2e was not found