nta-meta-analysis Documentation

Daniel C. Ferreira

Nov 12, 2018

current version:

In order to keep data curation homogeneity, some global rules must be defined. They are:

- The JSON files must use utf-8 encoding.
- `null` is a default value for fields that have not been checked in the curation process.
- `"missing"` replaces `null` when the required information for a specific fields has been checked but not found in the paper.
- To avoid confusion, every field with a finite set of values should use only lower case characters. A relevant exception is the set of IANA IPFIX features.
- Any value that is not common or included among the predefined options must be preceded by '_', e.g., `"_new_approach"`.
- Curated files must be named according to the following nomenclature: [first_author_surname]_[first_paper_title_word].json; e.g., *iglesias_time-activity.json*. More title words can be added to avoid matching.
- Whenever a field allows an array of values, the name finishes with *s*. Otherwise, it only allows one value. For example:

```
"1st_author": "Chekhov, A.",
"authors": ["Chekhov, Anton", "de Maupassant, Guy", "Stifter, Adalbert"]
```

- Fields are established according to two granularity levels for the curation process: *basic* and *complete*. Second level fields are distinguished by an "(*optional*)" next to their name. First level fields (basic, priority) are all the remaining fields, which are mandatory.
- Additional fields for human readability can be added (e.g., for leaving comments), and the name of this field should be preceded by '_', e.g., `"_comment"`.
- The information in each main block is always to be taken in its context. For example, when looking at whether the data is `"raw"` or `"preprocessed"` in the Data block, consider the state of the data before the preprocessing phase.
- In addition to the documentation, please check the supporting examples and templates for clarification:
  - *example_basic.json*
  - *example_complete.json*
  - *template_basic.json*
  - *template_complete.json*

CHAPTER 1

Repository Organization

We are currently in the process of converting all the v1 files into the latest version of our format. This documentation includes the content for both versions, and the repository itself also supports both formats at the moment. However, when we have converted all the files to the current version, the v1 files will be removed from the repository and documentation.

This is the current structure of the repository:

```
.
├── datasets.json          # v1 datasets
├── papers                 # v1 papers
│   └── ...
├── tools.json             # v1 tools
└── v2_papers              # papers in current version
    └── ...
```

CHAPTER 2

Versions

2.1 Old Versions

Currently there are 2 simultaneous major versions, v1 and the current version. No more work should be done for the v1 version, and it should be deleted when all the curated papers have been converted to the current version. However, until then some useful information might be extracted from the existing v1 papers.

2.2 Versioning System

From v2 onwards, we use semantic versioning. Papers follow the `vMAJOR.MINOR.PATCH` versioning. For example, a valid version for a paper is `v2.1.1`. However external tools should only need to look at the `MAJOR.MINOR` version, and the patch number should not be fundamental to their behavior.

An increase in the MAJOR version reveals a complete change of the NTARC format. An increate in the MINOR version indicates some small changes in the NTARC specification, that break backwards compatibility. An increase in the PATCH version occurs every time something is changed in the specification, that doesn't break backwards compatibility. Increases in the PATCH version should be frequent, and should happen when someone adds a new feature to our list of Information Elements, or when a new value is added to the list of previously allowed values for any specific field.

There is an exception to these rules for versions `v2` and `v2.1`, since they were introduced before the decision to follow the semantic versioning system. Some discussion on this topic can be found at https://github.com/CN-TU/nta-meta-analysis-specification/issues/14.

To summarize, we have:

- One branch for each minor version, named `rMAJOR.MINOR` (e.g., `r2.1`)
- Starting from v2.1.1, we use semantic versioning
- Each change that doesn't break backwards compatibility increments the PATCH number of the version (e.g., adding possible values to fields with finite possible values; adding new IEs to `own_ies.csv`)
- This version is included in the NTARC `version` field

- External tools usually only need to care about MINOR version, as everything should be backwards compatible (in particular, this should be true for the verifier)
- Tags with PATCH should be very frequent; they should be used for individual commits, or for multiple commits in a short time
- The r2 branch (corresponding to v2.0.X) is exempt from the proposed branch naming scheme, to keep compatibility with existing tools

CHAPTER 3

Main Blocks

The main blocks are ordered according to the typical processing pipeline::

```
Data > Preprocessing > Analysis Method > Evaluation > Result
```

The **Reference** block contains information about the publication itself (title, authors, etc) and the curator of the paper.

The **Data** block contains information about the datasets used in the publication.

The **Preprocessing** block contains information about the processing done to the data before passing it on to an analysis method. This process usually consists of extracting vectors of features from the original data.

The **Analysis Method** block contains information about what analytical methods were used in the publication.

The **Evaluation** block contains in information about how the performance of the methods in the previous section was evaluated.

The **Result** block contains information about the conclusions of the paper. The information for this block is usually completely contained in the conclusion section of the paper.

Additionally to the main blocks, in order to avoid confusion with past/future versions of the format, there is a mandatory field for version (which is always "v2").

3.1 JSON example

Summary with only the main blocks:

```
{
  "version": "v2.1",
  "reference": {
    ...
  },
  "data": {
    ...
  },
```

(continues on next page)

(continued from previous page)

```
  "preprocessing": {
    ...
  },
  "analysis_method": {
    ...
  },
  "evaluation": {
    ...
  },
  "result": {
    ...
  }
}
```

Example of a complete file:

```
{
  "version": "v3.0.0",
  "reference": {
    "title": "time-activity footprints in ip traffic",
    "authors": ["Iglesias, Félix", "Zseby, Tanja"],
    "publication_name": "computer networks",
    "publication_type": "peer_reviewed_journal",
    "year": 2016,
    "organization_publishers": [
      "elsevier"
    ],
    "pages_number_of": 12,
    "bibtex": {
        "type": "article",
        "volume": "107, Part 1",
        "issue": "missing",
        "pages": "64--75"
    },
    "access_open": false,
    "curated_by": "felix",
    "curated_last_revision": "10-04-2017",
    "curated_revision_number": 1
  },
  "data": {
    "datasets": [
      {
        "name": "mawi-2015",
        "availability": "public",
        "format": "packet",
        "types": [
          "ip"
        ],
        "generation": "captured",
        "generation_year": 2015,
        "covered_period": "minutes",
        "details": [
          "raw",
          "no_payload"
        ],
        "subsets": [
          "01-01-2015",
```

(continues on next page)

(continued from previous page)

```
          "15-04-2015",
          "31-07-2015"
        ],
        "anonymized": true
      }
    ]
  },
  "preprocessing": {
    "performed_feature_selection": true,
    "packet_analysis_oriented": false,
    "flow_analysis_oriented": true,
    "flow_aggregation_analysis_oriented": false,
    "tools": [
      {
        "name": "tshark",
        "detail": "v2.0.0",
        "availability": "public"
      },
      {
        "name": "own_perl_scripts",
        "detail": "none",
        "availability": "private"
      }
    ],
    "normalization_type": "range",
    "transformations": [
      "flow_extraction",
      "log",
      "time_series",
      "feature_operation",
      "class_separation"
    ],
    "final_data_format": "numerical_vectors",
    "feature_selections": [
      {
        "name": "max-relevance min-redundancy filter (correlation and MI based)",
        "type": "filter",
        "classifier": "none",
        "role": "main"
      }
    ],
    "packets": "none",
    "flows": [
      {
        "selection": "expert_knowledge",
        "role": "main",
        "main_goal": "traffic_classification",
        "active_timeout": 60,
        "idle_timeout": 60,
        "bidirectional": false,
        "features": [
          {
            "log": [
              "octetTotalCount"
            ]
          },
          {
```

(continues on next page)

(continued from previous page)

```
            "log": [
              "packetTotalCount"
            ]
          },
          "_activeForSeconds",
          {
            "log": [
              {
                "divide": [
                  "octetTotalCount",
                  "_activeForSeconds"
                ]
              }
            ]
          },
          {
            "log": [
              {
                "divide": [
                  "packetTotalCount",
                  "_activeForSeconds"
                ]
              }
            ]
          },
          "__maximumConsecutiveSeconds",
          "__minimumConsecutiveSeconds",
          {
            "maximum": [
              "_interPacketTimeMicroseconds"
            ]
          },
          {
            "minimum": [
              "_interPacketTimeMicroseconds"
            ]
          },
          "__numberOfActivityIntervals"
        ],
        "key_features": [
          "sourceIPv4Address",
          "destinationIPv4Address",
          "protocolIdentifier"
        ]
      },
      {
        "selection": "feature_selection",
        "role": "main",
        "main_goal": "traffic_classification",
        "active_timeout": 60,
        "idle_timeout": 60,
        "bidirectional": false,
        "features": [
          {
            "log": [
              "octetTotalCount"
            ]
```

(continues on next page)

(continued from previous page)

```
          },
          {
            "log": [
              {
                "divide": [
                  "octetTotalCount",
                  "_activeForSeconds"
                ]
              }
            ]
          },
          {
            "maximum": [
              "_interPacketTimeMicroseconds"
            ]
          },
          {
            "minimum": [
              "_interPacketTimeMicroseconds"
            ]
          }
        ],
        "key_features": [
          "sourceIPv4Address",
          "destinationIPv4Address",
          "protocolIdentifier"
        ]
      }
    ],
    "flow_aggregations": "none"
  },
  "analysis_method": {
    "supervised_learning": false,
    "unsupervised_learning": true,
    "semisupervised_learning": true,
    "anomaly_detection": true,
    "tools": [
      {
        "name": "matlab_fuzzyclusteringtoolbox",
        "availability": "public",
        "detail": "none"
      },
      {
        "name": "own_matlab_scripts",
        "availability": "private",
        "detail": "none"
      }
    ],
    "algorithms": [
      {
        "family": "fuzzy_clustering",
        "detail": "Gustafson-kessel fuzzy clustering",
        "learning": "unsupervised",
        "role": "main",
        "type": "clustering",
        "metric/decision_criteria": "mahalanobis",
        "tools": [
```

(continues on next page)

(continued from previous page)

```
          {
            "name": "matlab_fuzzyclusteringtoolbox",
            "detail": "none",
            "availability": "public"
          }
        ],
        "source": "referenced",
        "parameters_provided": false
      },
      {
        "family": "statistics",
        "detail": "Mad-based outlier removal",
        "learning": "statistics/model_fit",
        "role": "main",
        "type": "outlier_detection",
        "metric/decision_criteria": "mahalanobis",
        "tools": [
          {
            "name": "own_matlab_scripts",
            "detail": "none",
            "availability": "private"
          }
        ],
        "source": "referenced",
        "parameters_provided": false
      }
    ]
  },
  "evaluation": {
    "algorithm_comparison": false,
    "internal_validation": true,
    "external_validation": true,
    "dpi-based_validation": false,
    "port-based_validation": false,
    "pre-knowledge-based_validation": false,
    "manual_verification": true,
    "implementation_in_real_scenario": false,
    "train_test_separation": false,
    "methods": [
      {
        "name": "manual verification",
        "type": "external",
        "metrics": [
          "heuristic"
        ],
        "source": "popular"
      },
      {
        "name": "weighted vote",
        "type": "nest",
        "metrics": [
          "vote"
        ],
        "source": "popular"
      },
      {
        "name": "classification entropy",
```

(continues on next page)

(continued from previous page)

```
        "type": "internal",
        "metrics": [
          "clustering_metrics"
        ],
        "source": "referenced"
      },
      {
        "name": "partition index",
        "type": "internal",
        "metrics": [
          "clustering_metrics"
        ],
        "source": "referenced"
      },
      {
        "name": "xie and benix index",
        "type": "internal",
        "metrics": [
          "clustering_metrics"
        ],
        "source": "referenced"
      },
      {
        "name": "clustering gain",
        "type": "internal",
        "metrics": [
          "clustering_metrics"
        ],
        "source": "referenced"
      },
      {
        "name": "own cluster validity",
        "type": "internal",
        "metrics": [
          "clustering_metrics"
        ],
        "source": "missing"
      }
    ]
  },
  "result": {
    "main_goal": "detect_anomalies",
    "focus_main": "methodology/framework",
    "claimed_improvements": [
      "improved_data_description",
      "improved_traffic_classification",
      "fast_processing",
      "_flaw_detection"
    ],
    "reproducibility": "replicable",
    "subgoals": [
      "traffic_classification"
    ]
  }
}
```

CHAPTER 4

Features

The features are the hardest part of this format, as they are the most complex. However, having a complete description of the feature-set used in a paper in this format allows the use of an extractor tool to automatically reproduce the feature vectors used in the paper.

4.1 Concepts

- **Features** A feature represents a value which can be extracted from a packet/flow/flow aggregation. These can be base features (often can be computed by looking only at packet headers), or some more complicated things (like the entropy of a value that can be found in the packet headers). In our format, each feature is a combination of operations applied to base features. Each feature must be only one scalar value (as opposed to a vector).
- **Base features** Base features are the basic elements of our features format. These are always represented by strings.

  Examples: `"packetTotalCount"`, `"octetTotalCount"`, `"sourceIPv4Address"`
- **Operations** We have multiple operations defined, which receive features as arguments. These are defined as a JSON dictionary with one key, in which the key is the name of the operation, and the corresponding value is a list of arguments.

  Examples: `"mean"`, `"add"`, `"log"`
- **Selections** We also have an option to filter out specific packets in a flow/flow aggregation. This allows us for example to count the number of packets with a specific property (e.g., packet size larger than some threshold).

4.2 Base Features

We call base features those which are not obtained by combining other features. These are represented in this format by JSON strings.

We try to use the names of the IPFIX information elements defined by IANA. For features that we can not get out of combining IANA features with our limited set of operations, we have two naming options:

- if the feature is expected to be used many times (e.g.: there are some KDD '99 features which we cannot represent using IANA features and operations, but they are used in many papers), use a _ as prefix to a descriptive feature name. This features are listed in `own_ies.csv`. If you want to specify a new _ feature, you need to add it there.
- if the feature is very specific to this paper, use __ (double _) as prefix to a descriptive feature name

In both of this cases, try to give descriptive feature names, similar to the the ones used by IANA. Names should use camel case and start with a lower-case character. They should follow the following regex: `^[_]{1,2}[a-z0-9]+([A-Z][a-z0-9]*)*$`.

This means that **all base features that do not start with _ have to be IPFIX information elements defined by IANA**.

There is still another case, which is features that are repeated often, and are a combination of IANA features. In this case, use a descriptive feature name which starts with _ as an alias for it. A complete list of aliases is in `feature_aliases.json`; please add additional aliases there.

4.3 Operations

Besides the base features, we also have some `operations`, by which we can get new features.

We can have two kinds of operations:

- **value** The output is a single scalar value.
- **values** The output is a vector of values (possibly of variable size).

Note: The highest level operation in a feature **cannot** be one that is defined in the `<values>` directive, as that outputs multiple values.

Below is a grammar defining the list of possible operations, and their respective arguments:

Value:

```
# <value> always outputs a single number (a <value>)
<value> -> {"mean": [<values>]}
<value> -> {"stdev": [<values>]}
<value> -> {"variance": [<values>]}
<value> -> {"median": [<values>]}
<value> -> {"quantile": [<values>, <value>]} # second argument is a number from 0 to␣
↪1, where 0 is the minimum and 1 the maximum
<value> -> {"minimum": [<values>]} | {"minimum": [<value>, <value>+]}
<value> -> {"maximum": [<values>]} | {"maximum": [<value>, <value>+]}
<value> -> {"argmin": [<values>]} | {"argmin": [<value>, <value>+]}
<value> -> {"argmax": [<values>]} | {"argmax": [<value>, <value>+]}
<value> -> {"floor": [<value>]}
<value> -> {"ceil": [<value>]}
<value> -> {"mode": [<values>]} # returns the most frequent element in <values>
<value> -> {"mad": [<values>]} # returns the mean absolute deviation of <values>
<value> -> {"moment": [<values>, <value>]} # returns the <value>-th standardized␣
↪moment of <values>
<value> -> {"count": [<selection>]}  # returns number of selected objects
```

(continues on next page)

(continued from previous page)

```
<value> -> {"length": [<values>]}  # returns number of values (useful to use with␣
↪quantile_range)
<value> -> {"distinct": [<values>]}  # returns number of distinct values in <values>␣
↪in the selected objects
<value> -> {"apply": [<value>, <selection>]}  # returns a single feature value for␣
↪the selection of objects
<value> -> {"add": [<value>, <value>+]} | {"add": [<values>]}
<value> -> {"subtract": [<value>, <value>]}
<value> -> {"multiply": [<value>, <value>+]} | {"multiply": [<values>]}
<value> -> {"divide": [<value>, <value>]}
<value> -> {"pow": [<value>, <value>]}  # raises the first value to the power of the␣
↪second value (e.g., pow(octetTotalCount, 2) == octetTotalCount^2)
<value> -> {"log": [<value>]}
<value> -> {"exp": [<value>]}
<value> -> {"entropy": [<values>]}
<value> -> {"get": [<value>, <values>]}  # gets the <value>-th element of the second␣
↪argument; indexing is like in Python
<value> -> {"get_bits": [<value>, <value>]}  # gets the <value>-th bit of the second␣
↪argument; indexing is like in Python
<value> -> {"slice_bits": [<value>, <value>, <value>]}  # gets third_argument[first_
↪argument : second_argument], considering the third argument as an array of bits,␣
↪and returns it as a value; indexing is like in Python
<value> -> {"ifelse": [<logic>, <value>, <value>]}  # if the condition is true,␣
↪return the first argument else the second
<value> -> {"left_shift": [<value>, <value>]}  # shift the bits in the first value␣
↪left by the second value
<value> -> {"right_shift": [<value>, <value>]}  # shift the bits in the first value␣
↪right by the second value
```

Values:

```
# <values> outputs a list of <value>
<values> -> {"map": [<down>, <selection>]}  # returns a feature value for each object␣
↪in selection
<values> -> {"slice": [<value>, <value>, <values>]}  # gets third_argument[first_
↪argument, second_argument]; indexing is like in Python
<values> -> {"quantile_range": [<values>, <value>, <value>]} # e.g. {"quantile_range
↪": [<values>, 0, 0.25]} returns all values in the first quartile
<values> -> {"flat_map": [<down2>, <selection>]} | {"flat_map": [<down2>, <selection>,
↪ <selection>]}  # only applicable for flow-aggregations; just one selection applies␣
↪same selection for both flows and packets; two selections applies the 1st selection␣
↪for flows and the second for packets
<values> -> <down>  # features from one level-down (in flows, packet features; in␣
↪flow-aggregations, flow features)
```

4.4 Selections

The `selection` directive is useful for filtering out packets or any other information which might not be interesting for a particular feature. Intuitively, using selection on a flow will select packets (that is, the result will be the packets that fulfill the conditions in the selection), and in a flow_aggregation will output either flows or packets, depending on the selection used. Because of this distinction, for each selection that outputs packets, there is another selection that outputs flows, and contains `"_flows"` in its name.

This distinction between outputting flows or packets is necessary, since you can select objects with "octetTotalCount" > 1000, and in this case it's ambiguous whether you want to select all packets with more than 1000 bytes, or all the flows with more than 1000 bytes. Note that some features only make sense for flows (e.g., "packetTotalCount").

Its syntax is the following:

```
# <selection> outputs a list of objects (packets, flows or aggregations, depending on␣
↪what kind of feature is used)
<selection> -> {"select": [<logic-down>]}
<selection> -> {"select_slice": [<value>, <value>]} | {"select_slice": [<value>,
↪<value>, <selection>]}  # selects a slice from the first value to the second value,␣
↪with Python-like indexing (if a <selection is not provided, default to selecting␣
↪everything)
<selection> -> "forward" | "backward"  # special cases for selection; select objects␣
↪in the forward (or backward) direction
```

The logic directive contains the test to decide what gets or not filtered. Definition of logic:

```
# <logic> is used for selection, should be evaluated for each object
<logic> -> {"and": [<logic>+]}
<logic> -> {"or": [<logic>+]}
<logic> -> {"geq": [<value>, <value>]}
<logic> -> {"leq": [<value>, <value>]}
<logic> -> {"less": [<value>, <value>]}
<logic> -> {"greater": [<value>, <value>]}
<logic> -> {"equal": [<value>, <value>]}
<logic> -> true | false
<logic-down> -> {"and": [<logic-down>+]}
<logic-down> -> {"or": [<logic-down>+]}
<logic-down> -> {"geq": [<down>, <value>]}
<logic-down> -> {"leq": [<down>, <value>]}
<logic-down> -> {"less": [<down>, <value>]}
<logic-down> -> {"greater": [<down>, <value>]}
<logic-down> -> {"equal": [<down>, <value>]}
<logic-down> -> true | false
```

4.5 Example Features

The following are examples of the features directive.

```
"features": [
  "protocolIdentifier",
  "sourceTransportPort",
  "destinationTransportPort",
  "octetTotalCount",
  "packetTotalCount",
  "_activeForSeconds",
  {"divide": ["octetTotalCount", "_activeForSeconds"]},
  {"divide": ["packetTotalCount", "_activeForSeconds"]},
  "__maximumConsecutiveSeconds",
  "__minimumConsecutiveSeconds",
  {"maximum": ["_interPacketTimeMicroseconds"]},
  {"minimum": ["_interPacketTimeMicroseconds"]},
```

(continues on next page)

(continued from previous page)

```
  {"count": [{"select": [{"geq": ["_interPacketTimeMicroseconds", 1000000]}]}]}
]
```

```
"features": [
  {"entropy": ["sourceIPv4Address"]},
  {"entropy": ["destinationIPv4Address"]},
  {"entropy": ["destinationTransportPort"]},
  {"entropy": ["_flowDurationSeconds"]},
  {"multiply": [{"argmax": [{"count": [{"select": [{"less": ["ipTotalLength", 128]}]}
↪]}, {"count": [{"and": [{"select": [{"geq": ["ipTotalLength", 128]}]}, {"select": [{
↪"less": ["ipTotalLength", 256]}]}]}]}, {"count": [{"and": [{"select": [{"geq": [
↪"ipTotalLength", 256]}]}, {"select": [{"less": ["ipTotalLength", 512]}]}]}]}, {
↪"count": [{"and": [{"select": [{"geq": ["ipTotalLength", 512]}]}, {"select": [{"less
↪": ["ipTotalLength", 1024]}]}]}]}, {"count": [{"and": [{"select": [{"geq": [
↪"ipTotalLength", 1024]}]}, {"select": [{"less": ["ipTotalLength", 1500]}]}]}]}]}, {
↪"add": [{"entropy": [{"count": [{"select": [{"less": ["ipTotalLength", 128]}]}]}]},
↪{"entropy": [{"count": [{"and": [{"select": [{"geq": ["ipTotalLength", 128]}]}, {
↪"select": [{"less": ["ipTotalLength", 256]}]}]}]}]}, {"entropy": [{"count": [{"and
↪": [{"select": [{"geq": ["ipTotalLength", 256]}]}, {"select": [{"less": [
↪"ipTotalLength", 512]}]}]}]}]}, {"entropy": [{"count": [{"and": [{"select": [{"geq
↪": ["ipTotalLength", 512]}]}, {"select": [{"less": ["ipTotalLength", 1024]}]}]}]}]},
↪ {"entropy": [{"count": [{"and": [{"select": [{"geq": ["ipTotalLength", 1024]}]}, {
↪"select": [{"less": ["ipTotalLength", 1500]}]}]}]}]}]}]},
  {"get": [14, "tcpControlBits"]}
]
```

```
"features": ["_KDD5", "_KDD23", "_KDD3", "_KDD6", "_KDD35", "_KDD1"]
```

CHAPTER 5

Reference

5.1 Properties

5.1.1 title

(*string*) The title of the paper.

```
"title": "time-activity footprints in ip traffic"
```

5.1.2 authors

(*array* of *strings*) Array with the authors' list. Format as in BibTeX. Check this StackExchange thread for details.

Warning: Make sure to keep the order of the names the same as the paper!

Example:

```
"authors": ["Chekhov, Anton", "de Maupassant, Guy", "Stifter, Adalbert"]
```

Example with only one author:

```
"authors": ["Chekhov, Anton"]
```

5.1.3 publication_name

(*string*) Name of the publication (journal or conference). Please, use the most common name used in scientific citations (without abbreviations and in lower case). Example:

```
"publication_name": "ieee transactions on networking"
```

5.1.4 publication_type

(*string*) It marks the type of publication. Please, consider carefully if the publication fits any of the following default labels (values):

- `"peer_reviewed_journal"` journal with a peer review process.
- `"peer_reviewed_conference"` conference with a peer review process.
- `"arxiv"` paper is published only in arxiv.
- `"technical_report"` technical report, usually without peer review, and published in the author's/university's page.

Example:

```
"publication_type": "peer_reviewed_journal"
```

5.1.5 year

(*numerical*) The year of the publication release. Example:

```
"year": 2016
```

5.1.6 organization_publishers (*optional*)

(*array* of *strings*) Please, consider carefully if the publication fits one or more of the following default organizations (values):

- `"ieee"`
- `"elsevier"`
- `"acm"`
- `"springer"`
- `"wiley"`
- `"taylor_&_francis"`
- `"mdpi"`

Example:

```
"organization_publishers": ["acm"]
```

5.1.7 pages_number_of (*optional*)

(*numerical*) The total number of pages of the paper. Example:

```
"pages_number_of": 8
```

5.1.8 bibtex

(*object*) Various BibTeX-related fields. All fields in this object are strings.

Note: This object supports extra fields, so you are free to add other BibTeX properties.

type

(*string*, for *bibtex* citation compatibility) Please, consider carefully if the publication fits one or more of the following default bibtex types (values):

- `"article"`
- `"inproceedings"`
- `"techreport"`
- `"inbook"`
- `"misc"`

Example:

```
"type": "article"
```

volume (*optional*)

(*string*, for *bibtex* citation compatibility) The volume of the related multi-volume publication or book. If there is no volume, write `"missing"`. Example:

```
"volume": "8"
```

issue (*optional*)

(*string*, for *bibtex* citation compatibility) The issue or number of the related publication or book. If there is no issue or number, write `"missing"`. Example:

```
"issue": "5"
```

pages (*optional*)

(*string*, for *bibtex* citation compatibility) The page range of the paper. If there is no page range, write `"missing"`. Write "–" between page numbers. Example:

```
"pages": "102--114"
```

5.1.9 access_open (*optional*)

(*boolean*) Is the paper open access for any normal Internet user? Example:

```
"access_open": true
```

5.1.10 curated_by

(*string*) Last person who reviewed/curated/modified this JSON file. Example:

```
"curated_by": "ferreira, d."
```

5.1.11 curated_last_revision

(*string*, format: *dd-mm-yyyy*) Date of the last revision/modification of this JSON file. Example:

```
"curated_last_revision": "10-01-2017"
```

5.1.12 curated_revision_number

(*numerical*) Number of the total revisions/modification/updates carried out on this specific JSON file. Minimum number is 1. Example:

```
"curated_revision_number": 3
```

5.2 JSON example (reference, complete)

```
"reference": {
  "title": "time-activity footprints in ip traffic",
  "authors": ["Iglesias, Félix", "Zseby, Tanja"],
  "publication_name": "computer networks",
  "publication_type": "peer_reviewed_journal",
  "year": 2016,
  "organization_publishers": ["elsevier"],
  "pages_number_of": "12",
  "bibtex": {
    "type": "article",
    "volume": "107, Part 1",
    "issue": "missing",
    "pages": "64--75"
  },
  "access_open": false,
  "curated_by": "iglesias, f.",
  "curated_last_revision": "10-04-2017",
  "curated_revision_number": 2
}
```

CHAPTER 6

Data

6.1 Properties

6.1.1 datasets

(*array* of *objects*) *datasets* can contain several *dataset-objects*. A *dataset-object* is composed of several fields.

name

(*string*) The name that identifies the dataset. By default we use a *source_year* nomenclature. Example:

```
"name": "mawi-2015"
```

availability

(*string*) It establishes how a normal Internet user can access the specific dataset. Please, consider carefully if the dataset-accesibility fits any of the following default labels (values):

- `"public"`
- `"public_on_demand"`
- `"private"`
- `"lost_source"` when the paper provides the source/link of the dataset but this is not valid any more.

Example:

```
"availability": "public"
```

format (*optional*)

(*string*) It specifically addresses if the dataset contains packet or flow descriptions. Therefore, the options by default are: `"packet"` and `"flow"`. Example:

```
"format": "flow"
```

types (*optional*)

(*array* of *strings*) It specifically addresses if the dataset has been pre-filtered and only contains some types of data based on protocols, versions, etc. Consider labels (values) as filter keys (e.g., if `"ipv4"` is used, there is no need to add `"tcp"` or `"udp"` too). Please, check if the dataset-type fits any of the following default labels (values):

- `"ip"`
- `"ipv4"`
- `"ipv6"`
- `"tcp"`
- `"http"`
- `"udp"`
- `"icmp"`
- `"dns"`
- `"tls"`
- `"ipsec"`

Note: The most general should be used when all of its subsets are used. For example, `["ipv4", "ipv6"]` is the same as `["ip"]`.

Example:

```
"types": ["ipv4"]
```

generation (*optional*)

(*string*) It contains information about how the dataset was generated. Please, consider carefully if the dataset-generation fits any of the following default labels (values):

- `"captured"` when the dataset has been directly captured from network sensors.
- `"synthetic"` when the dataset has been generated by algorithms for artificial traffic generation. This includes capturing data in the network, if the packets were algorithmically generated.
- `"altered_captured"` when the dataset is modeled/based on real captures, but manipulated to fulfill some specific criteria (e.g., increase the presence of certain attacks). Also includes datasets generated by running the actual application and capturing its traffic.
- `"mixed"` whenever real captures or capture-based traffic is mixed with synthetic traffic.

Note: There is a slight interception between `"synthetic"` and `"altered_captured"`. Hopefully common sense is enough to disambiguate between them for each paper. If this is not the case for a particular paper, a consensus among experts is necessary.

Example:

```
"generation": "captured"
```

generation_year

(*numerical* or *array* of *numberical*) The year the dataset was captured or generated. Example:

```
"generation_year": 2015
```

covered_period (*optional*)

(*string*) It tries to give an approximate impression about the time covered by the used dataset during analysis. As a criterion, if the *covered_period* is below two times the unity, the selected label should be the immediately below, e.g., if the dataset covers 90 minutes, *covered_period* should be `"minutes"`; however, if the dataset covers 120 minutes, *covered_period* should be `"hours"`. Please, consider carefully if the covered period fits any of the following default labels (values):

- `"minutes"`
- `"hours"`
- `"days"`
- `"weeks"`
- `"months"`
- `"years"`

Example:

```
"covered_period": "hours"
```

details (*optional*)

(*array* of *string*) Suitable to make a record of special characteristics of the dataset that are worth considering in meta-analysis. Please, consider carefully if any of the following default labels (values) are applicable:

- `"raw"` data is shown as came directly from sensors or generators with no shape/format transformation. Includes both packet captures (e.g., tcpdump) and flow records (e.g., NetFlow).
- `"preprocessed"` data has been transformed/mapped during a preprocessing step. Such preprocessing must have changed somehow the data format, for example, transforming it in structured vectors (i.e., filtered data is still `"raw"`).
- `"no_payload"` when payload has been removed from data. Payload removal does not make data *preprocessed*.

When no relevant details exist, write `"none"`.

Example:

```
"details": ["raw", "no_payload"]
```

subsets

(*array* of *strings*) The dataset might consist of diverse subsets. Here we specify which subsets have been used during the analysis. If it is not clearly specified in the paper with a proper name, the default nomenclature of the subsets refer to the date if possible (format: *hh-dd-mm-yyyy*). If there are no relevant subsets, write `"none"`, and if the subsets are not specified (or if it is not clear whether subsets are used or not), write `"missing"`. Example:

Note: You can also use this field when a dataset has been divided into constant time pieces (for example, when a one-hour long dataset was divided into 60 1-second long datasets)

```
"subsets": ["03-11-2014", "30-06-2015", "27-12-2016"]
```

anonymized (*optional*)

(*boolean*) Whether the dataset is anonymized or not.

Example:

```
"anonymized": true
```

6.2 JSON example (data, complete)

```
"data": {
  "datasets": [
    {
      "name": "mawi-2015",
      "availability": "public",
      "format": "packet",
      "types": "ip",
      "generation": "captured",
      "generation_year": 2015,
      "covered_period": "minutes",
      "details": ["raw","no_payload"],
      "subsets": ["01-01-2015","15-04-2015","31-07-2015"]
    },
    {
      "name": "kddcup-1999",
      "availability": "public",
      "format": "packet",
      "types": "ipv4",
      "generation": "altered_captured",
      "generation_year": 1999,
      "covered_period": "missing",
      "details": ["preprocessed"],
      "subsets": ["original","original_10_percent","corrected"],
      "anonymized": true
    }
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

CHAPTER 7

Preprocessing

7.1 Properties

7.1.1 performed_feature_selection

(*boolean*) It states if a feature selection process is carried out in the paper to select the suitable set of features for the analysis. Example:

```
"performed_feature_selection": true
```

7.1.2 packet_analysis_oriented

(*boolean*) It states if, after the preprocessing phase, the data to analyze is intended to be explored packet by packet (e.g., by methods that perform deep packet inspection). Example:

```
"packet_analysis_oriented": false
```

7.1.3 flow_analysis_oriented

(*boolean*) It states if, after the preprocessing phase, the data to analyze is intended to be explored flow by flow. Example:

```
"flow_analysis_oriented": true
```

7.1.4 flow_aggregation_analysis_oriented

(*boolean*) It states if, after the preprocessing phase, the data to analyze has been aggregated according to either features or flows. Therefore, the final analysis will not explore flows or packets, but usually networks as a whole by studying the aggregated values (e.g., time series showing the use of network resources). Example:

```
"flow_aggregation_analysis_oriented": false
```

7.1.5 tools

(*array* of *objects*) Here we describe the tools used for the preprocessing (data extraction, feature generation and transformations).

Example:

```
"tools": [
    {
        "name": "tshark",
        "detail": "v2.0.0",
        "availability": "public"
    },
    {
        "name": "own_python_scripts",
        "detail": "none",
        "availability": "private"
    },
    {
        "name": "own_perl_scripts",
        "detail": "none",
        "availability": "private"
    }
]
```

name

(*string*) We use the following keys for the nomenclature:

1. if they are released tools, software or suites, they must be appear with the corresponding name; e.g., *tshark*, *silk*, *tcpdump*.
2. if they consist on scripts or plugins for well-known programming languages, suites, packages or environments, the name must reflect such dependency; e.g., *matlab_scripts*, *java_scripts*, *python_scripts*.
3. only the top-dependency must be shown (e.g., *matlab*). Additional *relevant* packages running under the same environment should be also added as *tools*.
4. names start with *own_* if they are presented in the paper or referred to previous publications by the same authors (and they do not fit case 'a.'); e.g., *own_matlab_scripts*.

Example:

```
"name": "tshark"
```

detail (*optional*)

(*string*) This field expresses important details about the referred tools (e.g., version, release). If no details are required, `"none"` should be written in the corresponding place. Example:

```
"detail": "v2.0.0"
```

availability

(*strings*) This field expresses the availability of the referred tool. Please, consider carefully the following default labels (values):

- `"public"`
- `"private"`
- `"public_on_demand"`
- `"commercial"`

Example:

```
"availability": "public"
```

7.1.6 normalization_type

(*string*) This field saves information about possible normalization of numerical data. `"no"` stands for cases where no normalization is applied but numerical attributes are used. `"not_applicable"` is for cases where normalization makes no sense (e.g., all analyzed fields are nominal or categories). Please, consider carefully the following default labels (values):

`"no"`, `"not_applicable"`, `"range"`, `"zscore"`, `"decimal_scaling"`, `"quartile"`

Note: do not confuse `"quartile"` with `"quantile"`. `"quartile"` normalization uses *Q1* (25th percentile) and *Q3* (75th percentile) for normalization.

Example:

```
"normalization_type": "range"
```

7.1.7 transformations

(*array* of *strings*) This field collects all transformations that are performed after the dataset retrieval and previous to the analysis phase (i.e., they are part of the data preparation). Please, consider carefully the following listed operations (values):

`"sampling"`, `"filtering"`, `"log"`, `"map"`, `"graph"`, `"feature_aggregation"`, `"flow_extraction"`, `"entropy"`, `"time_series"`, `"feature_operation"`, `"class_separation"`

Note: This field is redundant with the features in the packets/flows/flow aggregations. However, this field is mandatory while the feature fields are optionals.

Example:

```
"transformations": ["sampling", "flow_extraction", "class_separation"]
```

7.1.8 final_data_format

(*string*) It collects the format of data after the preprocessing and previous to the analysis phase. Please, consider carefully the following default labels (values):

- `"numerical_vectors"`
- `"nominal_vectors"`
- `"mixed_vectors"`
- `"strings"`
- `"time_series"`

Example:

```
"final_data_format": "numerical_vectors"
```

7.1.9 feature_selections (*optional*)

(*array* of *objects*) *feature_selections* can contain several *feature_selection-objects*. When no *flow_aggregation-object* exists in the paper, write `"none"`. A *feature_selection-object* is composed of several fields:

name

(*string*) The name that identifies the feature selection technique. Example:

```
"name": "forward_selection"
```

type (*optional*)

(*string*) It identifies the type of feature selection method. Please, consider carefully the following default labels (values):

- `"wrapper"` see a description here.
- `"filter"` see a description here.
- `"hybrid"` see a description here.
- `"nest"` when it embeds or operates in a higher level than other nested methods.
- `"feature_reduction"` when it refers to methods that change the space and transform the initial set of features into a new set of features with less dimensions (e.g., PCA, LDA).

Example:

```
"type": "wrapper"
```

classifier (*optional*)

(*string*) It identifies the wrapped classifier that is used to evaluate the subset performance. If *classifier* is not applicable (e.g., for filters), write `"none"`. Example:

```
"classifier": "naive_bayes"
```

role (*optional*)

(*string*) This field is meaningful when diverse feature selection methods are compared. Default values are: `"main"`, when the method led to the best solutions; and `"competitor"` for other cases. If only one feature selection method is used, it is always `"main"`. Example:

```
"role": "main"
```

7.1.10 packets (*optional*)

(*array* of *objects*) *packets* can contain several *packet-objects*. A *packet-object* is defined when analysis in the paper are conducted on packets, i.e., analysis tools check packets independently or/and packet contents. Use this if you have a feature-vector for each packet. When no *packet-object* exists in the paper, write `"none"`. A *packet-object* is composed of several fields:

selection (*optional*)

(*string*) It identifies how the features extracted to analyze packets were selected. Please, consider carefully the following default labels (values):

- `"in_dataset"` if the analyzed feature set is exactly the same feature set of the dataset before preprocessing.
- `"feature_selection"` if a feature selection process was conducted and led to the current feature subset.
- `"study_based"` if the selected features are taken from a previous study referred in the paper.
- `"tool_based"` if the selected features are obtained from an extraction or preprocessing tool.
- `"expert_knowledge"` if the selection of features is endorsed by reasoning and proper explanations in the paper.

Example:

```
"selection": "in_dataset"
```

role (*optional*)

(*string*) This field is meaningful when diverse preprocessing methods are compared.

Default values are:

- `"main"` when the method led to the best solutions.
- `"validation"` for the specific case of *packets*, when packet inspection is used as baseline or ground truth for validating flow-based analysis.
- `"intermediate"` if this method of aggregation is only used as an intermediate step on the way to further aggregation. E.g. if flows are used but only for the purpose of being aggregated to flow aggregations.
- `"competitor"` otherwise.

Example:

```
"role": "validation"
```

main_goal (*optional*)

(*string*) This field saves the main goal of preparing the data according to this packet-based format. Please, consider the following possible labels (values):

- `"anomaly_detection"`
- `"application_classification"`
- `"attack_classification"`
- `"botnet_detection"`
- `"classification_for_qos"`
- `"classification_of_encrypted_traffic"`
- `"ddos_detection"`
- `"dos_detection"`
- `"http_intrusion_detection"`
- `"network_properties_monitoring"`
- `"p2p_botnet_detection"`
- `"p2p_traffic_classification"`
- `"probe_detection"`
- `"remote_to_local_detection"`
- `"specific_malware_detection"`
- `"traffic_classification"`
- `"traffic_rate_prediction"`
- `"traffic_visualization"`
- `"user_to_root_detection"`

Example:

```
"main_goal": "traffic_classification"
```

features (*optional*)

(*array* of *objects*)

Describes the features used in the paper. See *Features* for complete information.

7.1.11 flows (*optional*)

(*array* of *objects*) *flows* can contain several *flow-objects*. A *flow-object* is defined when analysis in the paper are conducted on flows, i.e., analysis tools check the behaviour of connection and connection attempts. Use this if you have a feature-vector for each flow. When no *flow-object* exists in the paper, write `"none"`. A *flow-object* is composed of several fields:

selection (*optional*)

Like in *packet-object.selection*.

role (*optional*)

Like in *packet-object.role*.

main_goal (*optional*)

Like in *packet-object.main_goal*.

active_timeout (*optional*)

(*numerical*, in seconds) This field defines the maximum duration of a flow. Example:

```
"active_timeout": 60
```

idle_timeout (*optional*)

(*numerical*, in seconds) This field defines the time in which, if no activity has been detected, the flow is considered as finished. Example:

```
"idle_timeout": 5
```

bidirectional (*optional*)

(*boolean*) This field marks if transmissions between two devices A and B are considered monodirectional (`false`), i.e., A>B and A<B are two different flows; or bidirectional (`true`), i.e., A>B and A<B belong to the same flow . Example:

```
"bidirectional": true
```

features (*optional*)

(see *features*)

key_features (*optional*)

(*array* of *objects*)

Describes the features used to aggregate the packets. That is, packets which share these features will be put in the same flow. In case all packets should be in the same flow, use `"none"`.

For the features, see *features*.

7.1.12 flow_aggregations (*optional*)

(*array* of *objects*) *flow_aggregation* can contain several *flow_aggregation-objects*. A *flow_aggregation-object* is defined when analysis in the paper are conducted on aggregation of features or flows, i.e., analysis tools usually describe networks as a whole. Use this if you have a feature-vector for each set of flows. When no *flow_aggregation-object* exists in the paper, write `"none"`. A *flow_aggregation-object* is composed of several fields:

selection (*optional*)

Like in *packet-object.selection*.

role (*optional*)

Like in *packet-object.role*.

main_goal (*optional*)

Like in *packet-object.main_goal*.

active_timeout (*optional*)

Like in *flow-object.active_timeout*.

bidirectional (*optional*)

Like in *flow-object.bidirectional*.

features (*optional*)

(see *features*)

key_features (*optional*)

(*array* of *objects*)

Describes the features used to aggregate the flows. That is, flows which share these features will be put in the same flow aggregation. In case all flows should be in the same flow, use `"none"`.

For the features, see *features*.

7.2 JSON example (preprocessing, complete)

```
"preprocessing": {
  "performed_feature_selection": true,
  "packet_analysis_oriented": false,
  "flow_analysis_oriented": true,
  "flow_aggregation_analysis_oriented": false,
  "tools": [
```

(continues on next page)

(continued from previous page)

```
        {
            "tool": "tshark",
            "detail": "v2.0.0",
            "availability": "public"
        },
        {
            "tool": "own_perl_scripts",
            "detail": "none",
            "availability": "private"
        }
    ],
    "normalization_type": "range",
    "transformations": ["flow_extraction","log","time_series", "feature_operation",
↪"class_separation"],
    "final_data_format": "numerical_vectors",
    "feature_selections": [
        {
            "name": "max-relevance min-redundancy filter (correlation and MI based)",
            "type": "filter",
            "classifier": "none",
            "role": "main"
        }
    ],
    "flows": [
        {
            "selection": "expert_knowledge",
            "role": "main",
            "main_goal": "traffic_classification",
            "active_timeout": 60,
            "idle_timeout": 60,
            "bidirectional": false,
            "features": [
                {"log": ["octetTotalCount"]},
                {"log": ["packetTotalCount"]},
                "_activeForSeconds",
                {"log": [{"divide": ["octetTotalCount", "_activeForSeconds"]}]},
                {"log": [{"divide": ["packetTotalCount", "_activeForSeconds"]}]},
                "__maximumConsecutiveSeconds",
                "__minimumConsecutiveSeconds",
                {"maximum": ["_interPacketTimeMicroseconds"]},
                {"minimum": ["_interPacketTimeMicroseconds"]},
                "__numberof_activity_intervals",
            ],
            "key_features": [
                "sourceIPv4Address",
                "destinationIPv4Address",
                "protocolIdentifier"
            ]
        },
        {
            "selection": "feature_selection",
            "role": "main",
            "main_goal": "traffic_classification",
            "active_timeout": 60,
            "idle_timeout": 60,
            "bidirectional": false,
            "features": [
```

(continues on next page)

(continued from previous page)

```
                {"log": ["octetTotalCount"]},
                {"log": [{"divide": ["octetTotalCount", "_activeForSeconds"]}]},
                {"maximum": ["_interPacketTimeMicroseconds"]},
                {"minimum": ["_interPacketTimeMicroseconds"]},
            ],
            "key_features": [
                "sourceIPv4Address",
                "destinationIPv4Address",
                "protocolIdentifier"
            ]
        }
    ]
},
```

CHAPTER 8

Analysis Method

8.1 Properties

8.1.1 supervised_learning

(*boolean*) It marks if a classification or regression algorithm (or any technique known as supervised learning) was used during the analysis part (e.g., a decision tree). This field specifically refers to algorithms, not methodologies or frameworks. Example:

```
"supervised_learning": true
```

8.1.2 unsupervised_learning

(*boolean*) It marks if a clustering algorithm (or any technique known as unsupervised learning) was used during the analysis part (e.g., DBSCAN). This field specifically refers to algorithms, not methodologies or frameworks. Example:

```
"unsupervised_learning": false
```

8.1.3 semisupervised_learning

(*boolean*) It marks if a algorithm known as semisupervised learning was used during the analysis part (e.g., Transductive SVM). This field specifically refers to algorithms, not methodologies or frameworks. Example:

```
"semisupervised_learning": true
```

8.1.4 anomaly_detection

(*boolean*) It marks if a algorithm known as an anomaly detection technique was used during the analysis part (e.g., LOF). This field specifically refers to algorithms, not methodologies or frameworks. Example:

```
"anomaly_detection": true
```

8.1.5 tools

(*array* of *objects*) Here we describe the tools used for the preprocessing (data extraction, feature generation and transformations).

Example:

```
"tools": [
    {
        "name": "tshark",
        "detail": "v2.0.0",
        "availability": "public"
    },
    {
        "name": "own_python_scripts",
        "detail": "none",
        "availability": "private"
    },
    {
        "name": "own_perl_scripts",
        "detail": "none",
        "availability": "private"
    }
]
```

name

(*string*) We use the following keys for the nomenclature:

1. if they are released tools, software or suites, they must be appear with the corresponding name; e.g., *tshark*, *silk*, *tcpdump*.
2. if they consist on scripts or plugins for well-known programming languages, suites, packages or environments, the name must reflect such dependency; e.g., *matlab_scripts*, *java_scripts*, *python_scripts*.
3. only the top-dependency must be shown (e.g., *matlab*). Additional *relevant* packages running under the same environment should be also added as *tools*.
4. names start with *own_* if they are presented in the paper or referred to previous publications by the same authors (and they do not fit case 'a.'); e.g., *own_matlab_scripts*.

Example:

```
"name": "tshark"
```

detail (*optional*)

(*string*) This field expresses important details about the referred tools (e.g., version, release). If no details are required, `"none"` should be written in the corresponding place. Example:

```
"detail": "v2.0.0"
```

availability

(*strings*) This field expresses the availability of the referred tool. Please, consider carefully the following default labels (values):

- `"public"`
- `"private"`
- `"public_on_demand"`
- `"commercial"`

Example:

```
"availability": "public"
```

8.1.6 algorithms (*optional*)

(*array* of *objects*) *algorithms* can contain several *algorithm-objects*. An *algorithm-object* is composed of several fields:

family

(*string*) The family to which the algorithm belongs. Please, consider carefully the following default labels (values):

- `"autoregressive_models"`
- `"bayesian"`
- `"cluster_validation"`
- `"crossvalidation"`
- `"decision_tree"`
- `"ensemble"`
- `"entropy_based"`
- `"fuzzy_clustering"`
- `"glm_regression"`
- `"graph_modeling"`
- `"hierarchical_clustering"`
- `"kmeans_clustering"`
- `"kmedoids_clustering"`
- `"knn"`
- `"markov_process"`
- `"mixture_model"`
- `"neural_networks"`
- `"parameter_search"`
- `"pca"`
- `"random_forest"`

- "rule_extraction"
- "signature"
- "statistics"
- "stream_clustering"
- "svm"
- "two_step_clustering"
- "wavelet_transform"

Example:

```
"name": "fuzzy_clustering"
```

detail (*optional*)

(*string*) Additional details about the algorithm name. Example:

```
"subname": "Gustafson-kessel fuzzy clustering"
```

learning (*optional*)

(*string*) It identifies the learning approach of the algorithm. Please, consider carefully the following default labels (values):

- "supervised"
- "unsupervised"
- "semisupervised"
- "statistics/model_fit" the method uses predefined models, distributions and statistics and tries to check how real data fit such assumed models, i.e., it finds model parameters, gives summary values or discovers outliers based on distances to models.
- "nest" when it embeds or operates in a higher level than other nested methods.
- "no" it is somehow not possible to apply the word *learning* to the used algorithm

Example:

```
"learning": "supervised"
```

role (*optional*)

(*string*) This field is meaningful when diverse algorithms are compared. Default values are:

- "main" the method led to the best solution.
- "validation" the algorithm is used to establish a ground truth.
- "competitor" for all other cases.

If only one algorithm is used, it is always "main".

Example:

```
"role": "main"
```

type (*optional*)

(*string*) It refers to the normal use of the defined algorithms, i.e., the algorithm is generally considered as a method for <type_value>. Please consider carefully the following default labels (values):

- `"classification"`

The algorithm is supervised and originally intended to perform classification and class-prediction. Based on a labeled set of training data, the algorithm predicts the labels of a set of test data. * `"regression"` Similar to classification, the algorithm is supervised but, instead of predicting or guessing a class/label, it outputs a predicted numerical value. * `"clustering"` The algorithm is unsupervised and originally intended to find clusters within the data. Results are based on the intrinsic properties of the analyzed data and the input space drawn by such data (without resorting to any external partition or knowledge) * `"modeling"` Based on a set of given assumptions and pre-knowledge, the algorithm uses some training data to create a model that summarizes a given process. The difference with other algorithm types–which might and generally do also use models–is that here the "model-according-to-some-initial-assumptions" is the focus of the process (i.e., knowledge discovery), whereas other algorithms are focused on the predicted values (i.e., problem solving), being the model a mean or secondary otuput. * `"outlier_detection"` The algorithm is unsupervised and originally created to rank data points based on their outlierness properties when compared with the rest of data points in the same dataset. Clustering algorithms can be used for outlier detection, but purely outlier detection algorithms are not forced to create a cluster-like representation of the data. * `"space_transformation"` The algorithm transforms the input space into an output space that simplifies subsequent analysis, improves the visualization or understanding of the data, or removes constraints. * `"specific_detection"` The algorithm is specifically crafted to capture, detect, or identify a specific type of phenomenon. These algorithms are usually based on heuristics and ad-hoc steps. * `"validation_optimization"` The algorithm contributes to validate or optimize the analysis process. Such algorithms validate the outcomes of other algorithms, performs the search of optimal parameters, or nest algorithms to enhance some properties.

Example:

```
"type": "clustering"
```

metric/decision_criteria (*optional*)

(*string*) It assesses the used metric, similarity or dissimilarity distance, also the core of the decision making criteria. Please, consider carefully the following default labels (values):

- `"error/fitting_function"`
- `"euclidean"`
- `"mutual_information"`
- `"correlation"`
- `"jaccard"`
- `"mahalanobis"`
- `"hamming"`
- `"exact_matching"`
- `"manhattan"`
- `"probabilistic"`

- "vote"

Example:

```
"metric/decision_criteria": "euclidean"
```

tools (*optional*)

(see *tools*)

source (*optional*)

(*string*) It identifies the origin of the algorithm. Please, consider carefully the following default labels (values):

- "own_proposed" if authors developed and present the algorithm in the paper.
- "own_referenced" if authors developed the algorithm but presented it in a previous publication.
- "referenced" if authors took the method from the literature or known sources.

Example:

```
"source": "referenced"
```

parameters_provided (*optional*)

(*boolean* or *string*) This field expresses if the required parameters for reproducing the analysis are provided. In addition to `true` and `false`, "partially" is also possible when authors provide some parameters but some of them is missing or, for any reason, the experiment seems to be not reproducible.

Note: If the method has no parameters, use `true`, since you have enough information to replicate it.

Example:

```
"parameters_provided": "partially"
```

8.2 JSON example (analysis_method, complete)

```
"analysis_method": {
  "supervised_learning": false,
  "unsupervised_learning": true,
  "semisupervised_learning": true,
  "anomaly_detection": true,
  "tools": [
      {
          "tool": "matlab_fuzzyclusteringtoolbox",
          "detail": "none",
          "availability": "public"
      },
      {
          "tool": "own_matlab_scripts",
```

(continues on next page)

(continued from previous page)

```
            "detail": "none",
            "availability": "private"
        }
    ],
    "algorithms": [
        {
            "family": "fuzzy_clustering",
            "detail": "Gustafson-kessel fuzzy clustering",
            "learning": "unsupervised",
            "role": "main",
            "type": "clustering",
            "metric/decision_criteria": "mahalanobis",
            "tools": [
                {
                    "tool": "matlab_fuzzyclusteringtoolbox",
                    "detail": "none",
                    "availability": "public"
                }
            ],
            "source": "referenced",
            "parameters_provided": false
        },
        {
            "family": "statistics",
            "detail": "Mad-based outlier removal",
            "learning": "statistics/model_fit",
            "role": "main",
            "type": "outlier_detection",
            "metric_distance": "mahalanobis",
            "tools": [
                {
                    "tool": "own_matlab_scripts",
                    "detail": "none",
                    "availability": "private"
                }
            ],
            "source": "referenced",
            "parameters_provided": false
        }
    ]
},
```

CHAPTER 9

Evaluation

9.1 Properties

9.1.1 algorithm_comparison

(*boolean*) It marks if different algorithms are compared in the paper in an attempt to establish the best one to fulfill a specific goal. Example:

```
"algorithm_comparison": false
```

9.1.2 internal_validation

(*boolean*) It marks if algorithms were evaluated by means of internal validation methods, i.e., scores are provided based on the intrinsic properties of data under analysis (e.g., Silhouette). Example:

```
"internal_validation": false
```

9.1.3 external_validation

(*boolean*) It marks if algorithms were evaluated by means of external validation methods, e.g., baseline partitions, ground truth, pre-labeled data, dpi-classes. Example:

```
"internal_validation": false
```

9.1.4 dpi-based_validation

(*boolean*) It marks if algorithms were evaluated by using the results from deep packet inspection (dpi) as benchmark. Example:

```
"dpi-based_validation": true
```

9.1.5 port-based_validation

(*boolean*) It marks if algorithms were evaluated by using the results from port-based identification (typically TCP and UDP destination ports) as benchmark. Example:

```
"port-based_validation": false
```

9.1.6 pre-knowledge-based_validation

(*boolean*) It marks if algorithms were evaluated by using some kind of pre-knowledge as benchmark. Such pre-knowledge usually means that authors prepared the data previous to the analysis according to predefined classes (e.g., `"synthetic"` or `"altered_captured"`) Example:

```
"pre-knowledge-based_validation": true
```

9.1.7 manual_verification

(*boolean*) It marks if results (e.g., outliers, classes, patterns) obtained by algorithms were manually/visually checked after the analysis to see/discover if results represent specific phenomena/events. Example:

```
"manual_verification": true
```

9.1.8 implementation_in_real_scenario

(*boolean*) It marks if authors mentioned/explained/tried an actual implementation of the proposed methodology/framework/algorithms in a real scenario. If the proposed system is running in a real environment and derived experiences are documented in the paper. Example:

```
"implementation_in_real_scenario": false
```

9.1.9 train_test_separation

(*boolean*) It marks if datasets were clearly separated in independent train and test sets for the analysis. In other words, `true` if none of the testing data was used in training, `false` otherwise. Example:

```
"train_test_separation": true
```

9.1.10 methods (*optional*)

(*array* of *objects*) *methods* can contain several *method-objects*. A *method-object* represents a technique used for the analysis evaluation or algorithm validation. A *method-object* is composed of several fields:

name

(*string*) The name that identifies the evaluation method. Example:

```
"name": "normal classification metrics"
```

type (*optional*)

(*string*) It identifies the type of evaluation method. Please, consider carefully the following default labels (values):

- `"external"` the evaluation depends on labels or some other form of external ground truth.
- `"internal"` the evaluation does not depend on any ground truth (e.g. silhouette coefficient).
- `"external_and_internal"` both external and internal.
- `"nest"` the evaluation is a method that embeds other methods, or carries out some kind of bootstrapping (e.g. cross-validation analysis, ensemble learning

Note: In a particular paper, the usage of cross-validation could depend on the labels of the data. However, since cross-validation method itself is independent from the score used (you can do both supervised and unsupervised cross-validation, or even a mix of both), cross-validation is considered `"nest"`.

Example:

```
"type": "external"
```

metrics (*optional*)

(*array* of *string*) It assesses the used metrics for the evaluation. Please, consider carefully the following default labels (values):

- `"error_distance"` metric depends on the distance from the model to the data points. e.g. sum of squared error, absolute error, r^2, etc
- `"precision"` precision metric
- `"accuracy"` accuracy metric
- `"recall"` recall metric
- `"f-1"` f-1 metric
- `"false_negative_rate"` rate of false negatives
- `"false_positive_rate"` rate of false positives
- `"roc/auc"` roc-based metrics
- `"complete_confusion_matrix"` all information regarding the confusion matrix is provided.
- `"incomplete_confusion_matrix"` some information regarding the confusion matrix is missing and it is relevant for evaluating the quality of the classifier.
- `"classification_loss"` e.g. logistic regression loss function
- `"clustering_metrics"` e.g. silhouette coefficient
- `"time-based"` evaluation on the required time for running the method

- "other_computing_resources-based" evaluation of required computing resources (excluding time) for running the method (e.g. memory, cpu)
- "granularity-based" e.g. an algorithm provides more detailed information (classes, traffic types) than other algorithm.
- "heuristic" the metric is an heuristic developed specifically for the problem
- "vote" for nest methods (usually). The nest method integrates diverse validation techniques and the best result/algorithm is decided by means of consensus.

Example:

```
"metrics": ["error_distance"]
```

source (*optional*)

(*string*) It identifies the origin of the method. Please, consider carefully the following default labels (values):

- "own_proposed" if authors developed and present the algorithm in the paper.
- "own_referenced" if authors developed the algorithm but presented it in a previous publication.
- "referenced" if authors took the method from the literature or known sources.
- "popular" the method is popular enough to not require a reference (e.g., FP, FN).

Example:

```
"source": "referenced"
```

9.2 JSON example (evaluation, complete)

```
"evaluation": {
  "algorithm_comparison": false,
  "internal_validation": true,
  "external_validation": true,
  "dpi-based_validation": false,
  "port-based_validation": false,
  "pre-knowledge-based_validation": false,
  "manual_verification": true,
  "implementation_in_real_scenario": false,
  "train-test_separation": false,
  "methods": [
      {
          "name": "manual verification",
          "type": "external",
          "metrics": ["heuristics"],
          "source": "popular"
      },
      {
          "name": "weighted vote",
          "type": "nest",
          "metrics": ["vote"],
          "source": "popular"
      },
      {
```

(continues on next page)

(continued from previous page)

```
          "name": "classification entropy",
          "type": "internal",
          "metrics": ["clustering_metrics"[,
          "source": "referenced"
      },
      {
          "name": "partition index",
          "type": "internal",
          "metrics": ["clustering_metrics"],
          "source": "referenced"
      },
      {
          "name": "xie and benix index",
          "type": "internal",
          "metrics": ["clustering_metrics"],
          "source": "referenced"
      },
      {
          "name": "clustering gain",
          "type": "internal",
          "metrics": ["clustering_metrics"],
          "source": "referenced"
      },
      {
          "name": "own cluster validity",
          "type": "internal",
          "metrics": ["clustering_metrics"],
          "source": "missing"
      }
  ]
}
```

CHAPTER 10

Result

10.1 Properties

10.1.1 main_goal

(*string*) This field should contain the main paper goal, as intended by the authors. In case of doubt, abstract and conclusion sections should help to establish this value. Note that in some cases, datasets for specific goals are used when evaluating different goals (e.g., using KDD99 to evaluate anomaly detection). In these cases, this value should correspond to the intentions of the authors, and not to the used dataset. Please, consider carefully the following default labels (values):

- `"detect_anomalies"` The goal of the paper is to detect anomalies in network traffic. That means just to distinguish between normal traffic and traffic that does not fit the normal behavior model, no matter if this is due to an attack, failure or just changes in user behavior. Methods can include statistical models, supervised or unsupervised machine learning, outlier detection and others. Important to distinguish this category from other (e.g., attack detection or traffic classification) is that the authors look for any traffic that does not fit the normal behavior and do not have detecting a particular attack or traffic class in mind.
- `"detect_attacks"` The goal of the paper is to detect attacks or related behavior (e.g., botnet communication) in network traffic. That means that the authors look for traffic that can be associated with malicious behavior. This can be done for instance by searching for attack specific traffic patterns in the network. Attack detection can use signature-based or anomaly detection based methods. If a papers goal is stated as detecting attacks and it is using anomaly detection methods for it, it is classified as attack detection paper (with method anomaly detection) and not as anomaly detection paper.
- `"classify_traffic"` The goal of the paper is to identify specific traffic classes in the network traffic. This could be for instance classifying packets into traffic properties (encrypted/non-encrypted, fragmented/non-fragmented packets) or usage of different applications or protocols (HTTP, FTP), distinguishing different flows by their characteristics (long, short flows, bursty flows) and many more. If a paper is looking for attacks and aims at classifying different attack types, the goal for the paper is detect_attacks and not classify_traffic.

Example:

```
"main_goal": "detect_anomalies"
```

10.1.2 subgoals (*optional*)

(*array* of *strings*) Here additional paper goals are collected. Goals are usually aimed in the *abstract* and must be understood as the *motivations* that inspire and justify the research. Please consider the following possible labels (values):

- `"anomaly_detection"`
- `"application_classification"`
- `"attack_classification"`
- `"botnet_detection"`
- `"classification_for_qos"`
- `"classification_of_encrypted_traffic"`
- `"ddos_detection"`
- `"dos_detection"`
- `"http_intrusion_detection"`
- `"network_properties_monitoring"`
- `"p2p_botnet_detection"`
- `"p2p_traffic_classification"`
- `"probe_detection"`
- `"remote_to_local_detection"`
- `"specific_malware_detection"`
- `"traffic_classification"`
- `"traffic_rate_prediction"`
- `"traffic_visualization"`
- `"user_to_root_detection"`

Example:

```
"subgoals": ["traffic_classification", "dos_detection"]
```

10.1.3 focus_main

(*string*) This field tries to capture the main aspect where the paper focuses the efforts. In other words, where the main novelty/proposal of the paper is located. Please, consider carefully the following default foci (values):

- `"algorithm"` authors present a new algorithm that outperforms old approaches.
- `"methodology/framework"` the novelty is on the methodology or framework devised to properly deal with network traffic. i.e., a combination of steps, that can include: preprocessing, filtering, analysis methods, verification, etc.
- `"features"` the main contribution of the paper is on the selected features, the preprocessing or the methods presented to select features.

- "pattern_analysis" authors describe normal behavior in the data (either by textual descriptions or numerical estimates)
- "outlier_analysis" authors describe abnormal behavior in the data (either by textual descriptions or numerical estimates)
- "data_description" the nature of the paper is mostly descriptive, in a formal way. Authors try to explain the Internet, network traffic or a significant part of it by exploring and depicting one or some datasets, and presenting numbers/scores from mathematical analysis/statistics.

Note: "algorithm" and "methodology/framework" are very similar concepts. In general, a methodology/framework is a composition of algorithms (and of how they interact with one another), in which each algorithm can easily be replaced by some other with the same input/output. However, this distinction is not always clear.

A good way to distinguish between algorithm and methodology/framework is that usually an algorithm is limited to one of the Main Blocks (data/preprocessing/methods/evaluation), while a methodology/framework usually crosses the boundaries between the Main Blocks. Common sense should be enough to make the distinction. If not, consensus among experts is required.

Example:

```
"focus_main": "pattern_analysis"
```

10.1.4 claimed_improvements

(*array* of *strings*) We specifically refer to improvements claimed in the *conclusions* section. Please, consider carefully if the claimed improvements appear in the following default list:

- "improved_detection_rates" the proposed method is better at detecting its objective (e.g. attacks) than previous methods.
- "improved_traffic_classification" the proposed method is better at identifying its objective (e.g. attacks, applications) than previous methods.
- "new_phenomena_disclosed" new traffic phenomena was disclosed.
- "fast_processing" also referred as: lightweight approach, low time-complexity, etc.
- "reduced_computational_resources" in terms of memory, storage or dependencies.
- "good_transportability" as the capability of being integrated in diverse environments and structures, also compatibility, portability or usability.
- "enhanced_functionality" being a more complete option than competitors because additional or further functions are implemented or it gathers/integrate diverse solutions together.
- "improved_data_description" datasets (i.e. network traffic) are more accurately described or with a higher granularity, more phenomena or characteristics, better level of detail.
- "parallelization_oriented" the presented methods are designed for or ensured to be suitable for parallel computing structures.
- "big_data_oriented" the presented methods are claimed to be suitable for big data (aka large datasets).
- "data_stream_oriented" the presented methods are claimed to be suitable for data stream mining or analysis.

Example:

```
"claimed_improvements": ["improved_detection_rates","reduced_computational_resources"]
```

10.1.5 reproducibility

(*string*) This field states if, based on the opinion of the paper data curator, the experiments and analysis can be reproduced or repeated. Please, consider carefully the following default terms (values):

- `"reproducible"` experiments are fully reproducible by a different team after reading the paper. The setup, all parameters, tools and datasets are described and/or provided (references to valid links) in a clear and open way. Results are expected to be the same or very similar.
- `"replicable"` the experiment can be replicated by a different team but with a different setup. The methodology is clearly explain, at least in a theoretical level. Not all parameters or tools are provided, but readers have enough know-how in the paper and references to develop their own setups based on the provided descriptions. Therefore, they can replicate the experiments.
- `"repeteable"` methodologies and setups are clearly described with scientific rigor; however, experiments can only be repeated by the authors given that some resources are not publicly available (e.g., using own datasets).
- `"no"` important information about part of the methodology is missing in a way that the experiment cannot be repeated in comparable conditions. The paper show findings or results, but it is not clear how they were obtained (this information is hidden, omitted or just missing).

Example:

```
"repoducibility": "replicable"
```

10.2 JSON example (result, complete)

```
"result": {
  "main_goal": "detect_anomalies",
  "goals": ["traffic_classification"],
  "focus_main": "methodology/framework",
  "claimed_improvements": ["improved_data_description", "improved_traffic_
↪classification", "fast_processing", "_flaw_detection"]
  "reproducibility": "replicable"
}
```

CHAPTER 11

Paper Editor

The paper editor can be found in github: https://github.com/CN-TU/nta-meta-analysis-editor.

You can use the precompiled versions in the releases. Alternatively, you can clone the repository and run

```
npm run electron
```

to open up the editor.

CHAPTER 12

Overview

The specification of this format is written as a grammar, in which everything are terminal symbols, except those surronded by `<...>`. As this is a JSON file, the spaces/newlines specified in this grammar are irrelevant. The only thing relevant is that the final JSON file contains all the information you can gather, and that **it is JSON parseable**, with the structure defined in this document. For details on the JSON format in general, check JSON.

This is the high-level structure of a paper JSON file:

As is usual in JSON, all fields are optional, but it is good to write as complete specification of a paper as possible. On the other end, adding extra fields for specific papers is OK, for example for comments you want to make.

The next sections go into detail about the multiple directives introduced in the snippet above.

CHAPTER 13

Reference

The reference part needs just enough information to uniquely identify the paper. This is the main author name, title of the paper, and year it was published.

Definition:

Example:

```
"reference": {
  "author": "Iglesias et al.",
  "title": "Time-activity footprints in IP traffic",
  "year": 2016
}
```

CHAPTER 14

Aggregations, Flows & Packets

- The `packets` directive serves the purpose of representing feature-vectors which represent packets.
- The `flows` directive allows representing aggregations of packets, according to a specific key.
- The `flow-aggregations` directive is used for representing aggregations of flows, according to a specific key.

In short, packets are to flows as flows are to aggregations.

All of these directive refer to lists of `packet`/`flow`/`flow-aggregation` (respectively). This is because papers frequently try different feature vectors for different goals, and/or for comparison among them. This way, we can keep the information of the multiple feature-vectors, without having multiple different specifications for the same paper.

Each `packet`/`flow`/`flow-aggregation` contains a list of features (directive `features`), a list of free-text goals (directive `goals`), in which you can write the problem the authors were addressing, and a free-text tool (directive `tool`), in which you should put the tool that was used to extract the features (e.g.: tshark, yaf, etc). The `flow` and `flow-aggregation` directives contain additionally a specification of the key used (directive `key`), and a time window (directive `window`), in seconds.

14.1 Key

The `key` directive contains itself some more fields:

The `key_features` directive indicates the features used for a flow/flow-aggregation. If you do not know the features being used as key, use `null` or leave empty. If there is no key (as in, everything is aggregated together), use an empty list (`[]`).

The `bidirectional` directive indicates whether a flow is unidirectional (only has packets with the exact same key as in `key_features`), bidirectional (has packets with the same key as in `key_features`, and packets in the opposite direction) or "separate_directions" (has packets as if it was bidirectional, but the features in the `features` directive are evaluated twice, once for each direction; i.e. if you have octetTotalCount in the features list, and key has "separate_directions", you will get two features, one with the octetTotalCount in the packets in one direction, and another in the opposite direction).

Definition of `bidirectional`:

The following is an example of the very common unidirectional 5-tuple key:

```
"key": {
  "bidirectional": false,
  "key_features": [
    "protocolIdentifier",
    "sourceIPv4Address",
    "sourceTransportPort",
    "destinationIPv4Address",
    "destinationTransportPort"
  ]
}
```

14.1.1 Traffic Type

The `traffic_type` directive is to be used when only traffic of a certain type is used. Its definition follows:

14.2 Definitions

Definition of `packet`:

Definition of `flow`:

Definition of `flow-aggregation`:

CHAPTER 15

Features

The features are the main focus of this format, and also the most complex part of it.

15.1 Base Features

We call base features those which are not obtained by combining other features. These are represented in this format by JSON strings.

We try to use the names of the IPFIX information elements defined by IANA. Additionally to the names defined by IANA, we also have some `operations`, by which we can get new features. For features that we can not get out of combining IANA features with our limited set of operations, we have two naming options:

- if the feature is expected to be used many times (e.g.: there are some KDD '99 features which we cannot represent using IANA features and operations, but they are used in many papers), use a _ as prefix to a descriptive feature name
- if the feature is very specific to this paper, use __ (double _) as prefix to a descriptive feature name

In both of this cases, try to give descriptive feature names, similar to the the ones used by IANA.

This means that **all base features that do not start with _ have to be IPFIX information elements defined by IANA**.

There is still another case, which is features that are repeated often, and are a combination of IANA features. In this case, use a descriptive feature name which starts with _ as an alias for it. A complete list of aliases is in `../dict.json`; please add additional aliases there.

15.2 Operations

Below is a complete list of possible operations:

```
# <value> always outputs a single number (a <value>)
<value> -> {"mean": [<values>]}
<value> -> {"stdev": [<values>]}
<value> -> {"variance": [<values>]}
<value> -> {"median": [<values>]}
<value> -> {"quantile": [<values>, <value>]} # second argument is a number from 0 to␣
↪1, where 0 is the minimum and 1 the maximum
<value> -> {"minimum": [<values>]} | {"minimum": [<value>, <value>+]}
<value> -> {"maximum": [<values>]} | {"maximum": [<value>, <value>+]}
<value> -> {"argmin": [<values>]} | {"argmin": [<value>, <value>+]}
<value> -> {"argmax": [<values>]} | {"argmax": [<value>, <value>+]}
<value> -> {"floor": [<value>]}
<value> -> {"ceil": [<value>]}
<value> -> {"mode": [<values>]} # returns the most frequent element in <values>
<value> -> {"mad": [<values>]} # returns the mean absolute deviation of <values>
<value> -> {"moment": [<values>, <value>]} # returns the <value>-th standardized␣
↪moment of <values>
<value> -> {"count": [<selection>]}  # returns number of selected objects
<value> -> {"length": [<values>]}  # returns number of values (useful to use with␣
↪quantile_range)
<value> -> {"distinct": [<values>]}  # returns number of distinct values in <values>␣
↪in the selected objects
<value> -> {"apply": [<value>, <selection>]}  # returns a single feature value for␣
↪the selection of objects
<value> -> {"add": [<value>, <value>+]} | {"add": [<values>]}
<value> -> {"subtract": [<value>, <value>]}
<value> -> {"multiply": [<value>, <value>+]} | {"multiply": [<values>]}
<value> -> {"divide": [<value>, <value>]}
<value> -> {"pow": [<value>, <value>]}  # raises the first value to the power of the␣
↪second value (e.g., pow(octetTotalCount, 2) == octetTotalCount^2)
<value> -> {"log": [<value>]}
<value> -> {"exp": [<value>]}
<value> -> {"entropy": [<values>]}
<value> -> {"get": [<value>, <values>]}  # gets the <value>-th element of the second␣
↪argument; indexing is like in Python
<value> -> {"get_bits": [<value>, <value>]}  # gets the <value>-th bit of the second␣
↪argument; indexing is like in Python
<value> -> {"slice_bits": [<value>, <value>, <value>]}  # gets third_argument[first_
↪argument : second_argument], considering the third argument as an array of bits,␣
↪and returns it as a value; indexing is like in Python
<value> -> {"ifelse": [<logic>, <value>, <value>]}  # if the condition is true,␣
↪return the first argument else the second
<value> -> {"left_shift": [<value>, <value>]}  # shift the bits in the first value␣
↪left by the second value
<value> -> {"right_shift": [<value>, <value>]}  # shift the bits in the first value␣
↪right by the second value
```

15.2.1 Value & Values

The `value` directive represents a single value, while the `values` directive represents a list of values. This is necessary to distinguish the arguments to the operations.

15.2.2 Selection & Logic

The `selection` directive is useful for filtering out packets or any other information which might not be interesting for a particular feature. Intuitively, using selection on a flow will select packets (that is, the result will be the packets that fulfill the conditions in the selection), and in a flow_aggregation will select flows.

Its syntax is the following:

```
# <selection> outputs a list of objects (packets, flows or aggregations, depending on␣
↪what kind of feature is used)
<selection> -> {"select": [<logic-down>]}
<selection> -> {"select_slice": [<value>, <value>]} | {"select_slice": [<value>,
↪<value>, <selection>]}  # selects a slice from the first value to the second value,␣
↪with Python-like indexing (if a <selection is not provided, default to selecting␣
↪everything)
<selection> -> "forward" | "backward"  # special cases for selection; select objects␣
↪in the forward (or backward) direction
```

The `logic` directive contains the test to decide what gets or not filtered. Definition of `logic`:

```
# <logic> is used for selection, should be evaluated for each object
<logic> -> {"and": [<logic>+]}
<logic> -> {"or": [<logic>+]}
<logic> -> {"geq": [<value>, <value>]}
<logic> -> {"leq": [<value>, <value>]}
<logic> -> {"less": [<value>, <value>]}
<logic> -> {"greater": [<value>, <value>]}
<logic> -> {"equal": [<value>, <value>]}
<logic> -> true | false
<logic-down> -> {"and": [<logic-down>+]}
<logic-down> -> {"or": [<logic-down>+]}
<logic-down> -> {"geq": [<down>, <value>]}
<logic-down> -> {"leq": [<down>, <value>]}
<logic-down> -> {"less": [<down>, <value>]}
<logic-down> -> {"greater": [<down>, <value>]}
<logic-down> -> {"equal": [<down>, <value>]}
<logic-down> -> true | false
```

15.3 Feature Specification

The following is the specification for the `features` and `feature` directives:

```
from one level-down (in flows, packet features; in flow-aggregations, flow features)
```

The `packet-feature`, `flow-feature` and `aggregation-feature` are packet, flow and aggregation -level features (respectively), which are not compositions of other features/operations. That is, they should be strings from the IANA IPFIX information elements list, or strings that start with _ or __.

15.4 Example Features

The following are examples of the `features` directive.

```
"features": [
  "protocolIdentifier",
  "sourceTransportPort",
  "destinationTransportPort",
  "octetTotalCount",
  "packetTotalCount",
  "_activeForSeconds",
  {"divide": ["octetTotalCount", "_activeForSeconds"]},
  {"divide": ["packetTotalCount", "_activeForSeconds"]},
  "__maximumConsecutiveSeconds",
  "__minimumConsecutiveSeconds",
  {"maximum": ["_interPacketTimeMicroseconds"]},
  {"minimum": ["_interPacketTimeMicroseconds"]},
  {"count": [{"select": [{"geq": ["_interPacketTimeMicroseconds", 1000000]}]}]}
]
```

```
"features": [
  {"entropy": ["sourceIPv4Address"]},
  {"entropy": ["destinationIPv4Address"]},
  {"entropy": ["destinationTransportPort"]},
  {"entropy": ["_flowDurationSeconds"]},
  {"multiply": [{"argmax": [{"count": [{"select": [{"less": ["ipTotalLength", 128]}]}
↪]}, {"count": [{"and": [{"select": [{"geq": ["ipTotalLength", 128]}]}, {"select": [{
↪"less": ["ipTotalLength", 256]}]}]}]}, {"count": [{"and": [{"select": [{"geq": [
↪"ipTotalLength", 256]}]}, {"select": [{"less": ["ipTotalLength", 512]}]}]}]}, {
↪"count": [{"and": [{"select": [{"geq": ["ipTotalLength", 512]}]}, {"select": [{"less
↪": ["ipTotalLength", 1024]}]}]}]}, {"count": [{"and": [{"select": [{"geq": [
↪"ipTotalLength", 1024]}]}, {"select": [{"less": ["ipTotalLength", 1500]}]}]}]}]}, {
↪"add": [{"entropy": [{"count": [{"select": [{"less": ["ipTotalLength", 128]}]}]}]},
↪{"entropy": [{"count": [{"and": [{"select": [{"geq": ["ipTotalLength", 128]}]}, {
↪"select": [{"less": ["ipTotalLength", 256]}]}]}]}]}, {"entropy": [{"count": [{"and
↪": [{"select": [{"geq": ["ipTotalLength", 256]}]}, {"select": [{"less": [
↪"ipTotalLength", 512]}]}]}]}]}, {"entropy": [{"count": [{"and": [{"select": [{"geq
↪": ["ipTotalLength", 512]}]}, {"select": [{"less": ["ipTotalLength", 1024]}]}]}]}]},
↪ {"entropy": [{"count": [{"and": [{"select": [{"geq": ["ipTotalLength", 1024]}]}, {
↪"select": [{"less": ["ipTotalLength", 1500]}]}]}]}]}]}]},
  {"get": [14, "tcpControlBits"]}
]
```

```
"features": ["_KDD5", "_KDD23", "_KDD3", "_KDD6", "_KDD35", "_KDD1"]
```

CHAPTER 16

Methods

In this field you can put any methods the authors used, along with some properties.

CHAPTER 17

Evaluation

In this field you can put any evaluation metrics the authors used, along with some properties.

CHAPTER 18

Datasets

The information about the datasets is in a separate file `data/datasets.json`. When a new dataset is used, please add it to the list of datasets, using the specification described below.

18.1 Specification

```
{ <dataset>+ }
<dataset> -> <dataset-key>: {
  "name": <free-text>,
  "year": <free-integer>,
  "data-type": <data-type>,
  "type": <type>,
  "availability": <availability>
}

<dataset-key> -> <free-text>
<data-type> -> "packet_pcap" | "packet_other" | "flow_sflow" | "flow_netflow" | "flow_
↪ipfix" | "flow_other" | "other"
<type> -> "real" | "synthetic"
<availability> -> "public" | "private" | "on_demand"
```

CHAPTER 19

Tools

The information about the tools is in a separate file `data/tools.json`. When a new tool is used, please add it to the list of datasets, using the specification described below.

19.1 Specification

```
{ <tool>+ }
<tool> -> <tool-key>: {
  "url": <free-text>,
  "name": <free-text>,
  "availability": <availability>
}

<availability> -> "opensource" | "freeware" | "commercial" | "proprietary"
```

CHAPTER 20

Naming Conventions

20.1 Paper Names

Each JSON file should be called `data/papers/YEAR/LNAME_1STWORDS.json`, with `YEAR` the year of the paper, `LNAME` the main author's last name and `1STWORDS` for the first word (or first two/three words) of the title, and this filename should be completely lowercase.

Example filename: `data/papers/2016/iglesias_timeactivity.json`

20.2 Feature Names

Feature names should in principle use the names defined in IANA's website. However, IANA Information Elements do not cover all the features that can be extracted from packets. If this is the case for a specific feature, there are two options:

- If the feature is likely used by many other people: prefix the feature name with _ (underscore), and add it to `data/own_ies.csv`, if it is not already there.
- If the feature is not likely to be used by other people: prefix the feature name __ (double underscore). In this case, there is not need to add it to any list.

The naming convention in both cases should try to follow IANA's naming convention: feature names are to be descriptive and in camelcase.

Additionally, the the first case (with _) can also be used for aliases; that is, features that are used a lot but that are some kind of combination of other features. A list of aliases is defined in `data/feature_aliases.json`.

20.3 Specification

20.3.1 Non-terminal Symbols

Non-terminal symbols in the specification are lowercase words separated by - (hyphen).

20.3.2 Strings

Strings that are part of the specification are lowercase words separated by _ (underscore).

Free text strings have no convention.

CHAPTER 21

Full Format Specification

```
# Design notes:
#
# - why <values> -> <down>?
#     * you can have mean(base-feature)
# - why not <down> in mean?
#     * wouldn't allow map
# - why not <values> in map?
#     * would allow for map(map(map(map(...
# - why not <value> in map?
#     * would allow for mean(map(mean(map(mean(...
# - why {"minimum": [<value>]} not allowed?
#     * {"minimum": [<values>]} needs <down> to reach <value>, but {"minimum": [<value>
↪]} would bypass needing <down>
#
#######

#! <down> -> <value>  # (packet, flow, flow-agg) can use this rule (0, 1, 2) times
#! <down2> -> <down>  # only applicable for flow-aggregations; goes down twice (same␣
↪as executing down->value rule twice)

<value> -> <free-integer> | <base-feature> | <free-float> | <logic>

# operation
# <value> always outputs a single number (a <value>)
<value> -> {"mean": [<values>]}
<value> -> {"stdev": [<values>]}
<value> -> {"variance": [<values>]}
<value> -> {"median": [<values>]}
<value> -> {"quantile": [<values>, <value>]} # second argument is a number from 0 to␣
↪1, where 0 is the minimum and 1 the maximum
<value> -> {"minimum": [<values>]} | {"minimum": [<value>, <value>+]}
<value> -> {"maximum": [<values>]} | {"maximum": [<value>, <value>+]}
<value> -> {"argmin": [<values>]} | {"argmin": [<value>, <value>+]}
<value> -> {"argmax": [<values>]} | {"argmax": [<value>, <value>+]}
```

(continues on next page)

(continued from previous page)

```
<value> -> {"floor": [<value>]}
<value> -> {"ceil": [<value>]}
<value> -> {"mode": [<values>]} # returns the most frequent element in <values>
<value> -> {"mad": [<values>]} # returns the mean absolute deviation of <values>
<value> -> {"moment": [<values>, <value>]} # returns the <value>-th standardized␣
↪moment of <values>
<value> -> {"count": [<selection>]}  # returns number of selected objects
<value> -> {"length": [<values>]}  # returns number of values (useful to use with␣
↪quantile_range)
<value> -> {"distinct": [<values>]}  # returns number of distinct values in <values>␣
↪in the selected objects
<value> -> {"apply": [<value>, <selection>]}  # returns a single feature value for␣
↪the selection of objects
<value> -> {"add": [<value>, <value>+]} | {"add": [<values>]}
<value> -> {"subtract": [<value>, <value>]}
<value> -> {"multiply": [<value>, <value>+]} | {"multiply": [<values>]}
<value> -> {"divide": [<value>, <value>]}
<value> -> {"pow": [<value>, <value>]}  # raises the first value to the power of the␣
↪second value (e.g., pow(octetTotalCount, 2) == octetTotalCount^2)
<value> -> {"log": [<value>]}
<value> -> {"exp": [<value>]}
<value> -> {"entropy": [<values>]}
<value> -> {"get": [<value>, <values>]}  # gets the <value>-th element of the second␣
↪argument; indexing is like in Python
<value> -> {"get_bits": [<value>, <value>]}  # gets the <value>-th bit of the second␣
↪argument; indexing is like in Python
<value> -> {"slice_bits": [<value>, <value>, <value>]}  # gets third_argument[first_
↪argument : second_argument], considering the third argument as an array of bits,␣
↪and returns it as a value; indexing is like in Python
<value> -> {"ifelse": [<logic>, <value>, <value>]}  # if the condition is true,␣
↪return the first argument else the second
<value> -> {"left_shift": [<value>, <value>]}  # shift the bits in the first value␣
↪left by the second value
<value> -> {"right_shift": [<value>, <value>]}  # shift the bits in the first value␣
↪right by the second value
# end

# values
# <values> outputs a list of <value>
<values> -> {"map": [<down>, <selection>]}  # returns a feature value for each object␣
↪in selection
<values> -> {"slice": [<value>, <value>, <values>]}  # gets third_argument[first_
↪argument, second_argument]; indexing is like in Python
<values> -> {"quantile_range": [<values>, <value>, <value>]} # e.g. {"quantile_range
↪": [<values>, 0, 0.25]} returns all values in the first quartile
<values> -> {"flat_map": [<down2>, <selection>]} | {"flat_map": [<down2>, <selection>,
↪ <selection>]}  # only applicable for flow-aggregations; just one selection applies␣
↪same selection for both flows and packets; two selections applies the 1st selection␣
↪for flows and the second for packets
<values> -> <down>  # features from one level-down (in flows, packet features; in␣
↪flow-aggregations, flow features)
# end

# selection
# <selection> outputs a list of objects (packets, flows or aggregations, depending on␣
↪what kind of feature is used)
<selection> -> {"select": [<logic-down>]}
```

(continues on next page)

(continued from previous page)

```
<selection> -> {"select_slice": [<value>, <value>]} | {"select_slice": [<value>,
↪<value>, <selection>]}  # selects a slice from the first value to the second value,␣
↪with Python-like indexing (if a <selection is not provided, default to selecting␣
↪everything)
<selection> -> "forward" | "backward"  # special cases for selection; select objects␣
↪in the forward (or backward) direction
# end

# logic
# <logic> is used for selection, should be evaluated for each object
<logic> -> {"and": [<logic>+]}
<logic> -> {"or": [<logic>+]}
<logic> -> {"geq": [<value>, <value>]}
<logic> -> {"leq": [<value>, <value>]}
<logic> -> {"less": [<value>, <value>]}
<logic> -> {"greater": [<value>, <value>]}
<logic> -> {"equal": [<value>, <value>]}
<logic> -> true | false
<logic-down> -> {"and": [<logic-down>+]}
<logic-down> -> {"or": [<logic-down>+]}
<logic-down> -> {"geq": [<down>, <value>]}
<logic-down> -> {"leq": [<down>, <value>]}
<logic-down> -> {"less": [<down>, <value>]}
<logic-down> -> {"greater": [<down>, <value>]}
<logic-down> -> {"equal": [<down>, <value>]}
<logic-down> -> true | false
# end
```

CHAPTER 22

Indices and tables

- genindex
- modindex
- search