

OpenBudgets.eu: Fighting Corruption with Fiscal Transparency

Project Number: 645833

Start Date of Project: 01.05.2015

Duration: 30 months

D2.5 Data Mining Interfaces

Dissemination Level	Public
Due Date of Deliverable	31.5.2017
Actual Submission Date	9.6.2017
Work Package	WP 2, Data Collection and Mining
Task	T2.5 Data Mining Interfaces
Type	Demo
Approval Status	Final
Version	1.1
Number of Pages	45
Filename	D2.5 Data Mining Interfaces



Abstract:

In this deliverable we describe the interfaces of the OpenBudgets.eu components for data mining and statistical analysis, already described in Deliverable 2.4. Dependently on the nature of the components, the functionalities are available via API endpoint, UI or via direct integration into other systems (for example Python modules or R packages).

The information in this document reflects only the author's views and the European Community is not liable for any use that may be made of the information contained therein. The information in this document is provided "as is" without guarantee or warranty of any kind, express or implied, including but not limited to the fitness of the information for a particular purpose. The user thereof uses the information at his/ her sole risk and liability.

History

Version	Date	Reason	Revised by
1.0	31.5.2017	First version for internal review	Stanislav Vojříř, Tiansi Dong
1.1	9.6.2017	Final version for submission	Fabrizio Orlandi

Author List

Organisation	Name	Contact Information
UEP	Jaroslav Kuchař	jaroslav.kuchar@fit.cvut.cz
UEP	Stanislav Vojříř	stanislav.vojir@vse.cz
UEP	Václav Zeman	vaclav.zeman@vse.cz
UEP	Jindřich Mynarz	jindrich.mynarz@vse.cz
UEP	Vojtěch Svátek	svatek@vse.cz
OKFGR	Kleanthis Koupidis	koupidis@okfn.gr
OKFGR	Aikaterini Chatzopoulou	kchatzopoul@okfn.gr
OKFGR	Charalampos Bratsas	charalampos.bratsas@okfn.org
UBONN	Tiansi Dong	tdong@uni-bonn.de
UBONN	Fathoni Musyaffa	musyaffa@iai.uni-bonn.de
UBONN	Kang Wang	wk0206@gmail.com
Fraunhofer	Fabrizio Orlandi	fabrizio.orlandi@iais.fraunhofer.de
Fraunhofer	Yakun Li	Yakun.Li@iais.fraunhofer.de

Executive Summary

In this deliverable, the authors describe the interfaces of the OpenBudgets.eu components for data mining and statistical analysis described in previous Deliverable 2.4. Dependently on the nature of the components, the functionalities are available via API endpoint, UI or via direct integration into other systems (for example Python modules or R packages).

The main part of this deliverable contains a description of software interfaces of components. For each component, it is provided a short description of the functionality, information about the implementation and the description of both - outer interface and internal architecture of the component.

The document is structured as follows: Chapter 2 describes the interfaces of DAM; Chapter 3 describes the interfaces of data pre-processing components; Chapter 4 describes the interfaces of data mining components running at the three data mining servers, the OKFGR server, UEP server, and Fraunhofer server. For complex data mining services, we also present the internal architectures.

Abbreviations and Acronyms

WP	Work Package
OS	OpenSpending
OBEU	OpenBudgets.eu
OKGR	Open Knowledge Greece
UEP	University of Economics, Prague

Table of Contents

1 Introduction	9
2 The Interface of Data Analysis and Mining (DAM)	10
2.1 Functional Interfaces of DAM	110
2.2 Communication Interface with UEP	12
2.2.1 Data Flow Inside DAM and uep_dm Module	12
2.2.2 Sample Usage	15
2.3 Communication Interfaces with Fraunhofer Server and OKFGR Server	16
2.3.1 Example: Communication with OKFGR server	16
2.3.2 Example: Communication with Fraunhofer Server	17
2.4 Meta-information of Algorithm for Indigo	18
3 Interfaces of Data Pre-processing	20
3.1 Discretization via SPARQL	20
4 Interfaces of the Three Data Mining Servers	22
4.1 EasyMiner - UEP Data Mining Server	22
4.1.1 Architecture & Internal APIs	22
4.1.2 Association Rule Mining API	24
4.1.3 Outlier Detection API	27
4.1.4 Association Rule Mining UI	28
4.1.5 fpmoutliers - R Package	29
4.2 OpenCPU Server - OKFGR Data Mining Server	31
4.2.1 DescriptiveStats.OBeu - R Package	31
4.2.2 TimeSeries.OBeu - R Package	34
4.2.3 Cluster.OBeu - R Package	37
4.3 Outlier_dm Lib - Fraunhofer Server	40
4.3.1 Input	40
4.3.2 Main Function	41
4.3.3 Output	42
5 Conclusion and Future Work	43

List of Figures

Figure 1: Architecture of DAM and its functional communication with other modules	10
Figure 2: Endpoint of DAM	11
Figure 3: EasyMiner - Architecture of services	23
Figure 4: EasyMiner - User interface for association rule mining	29
Figure 5: OpenCPU Server Interface	32
Figure 6: OpenCPU Server - Descriptive Statistics Input Example	33
Figure 7: OpenCPU Server- Snapshot of Descriptive Statistics Output Example	34
Figure 8: OpenCPU Server- Time Series Input Example	36
Figure 9: OpenCPU Server - Time Series Request Example	37
Figure 10: OpenCPU Server - Snapshot of Time Series Output Example	38
Figure 11: OpenCPU Server - Cluster Analysis Input Example	39
Figure 12: OpenCPU Server - Cluster Analysis Request Example	40
Figure 13: OpenCPU Server - Snapshot of Time Series Output Example	41
Figure 14: Structure of the input CSV to outlier-detection based on LOF	42
Figure 15: Sample of the output CSV of the outlier detection based on LOF	43

List of Tables

Table 1: Data mining module and its interfaces	9
Table 2: List of data mining endpoints and the processing place	11
Table 3: Input params for Figure 7	33
Table 4: Input params for Figure 9	36
Table 5: Input params for Figure 12	39
Table 6: The meaning of the main parameters of the main function of outlier detection based LOF	42

1 Introduction

This deliverable consists in interfaces of data mining and analytical tools mainly produced in the context of task T2.4 and also in testing the tools (software components). This content of this deliverable follows the previously submitted deliverable D2.4. We describe the functional interfaces of the data analysis and mining tools and the functional communication interfaces with front-end users. General issues on graphical user interfaces are reported in deliverables of WP3, graphical interfaces of data mining are presented in this deliverable.

To achieve the extensibility of data mining services, we use the distributed architecture. That is, data mining algorithms are allowed to be processed in different locations. Currently, we have three locations: Thessaloniki (the OKFGR server in Greece), Prague (the UEP server in Czech), Sankt Augustin (the Fraunhofer server in Germany). New data mining services can be easily integrated into the existing platform.

The data mining service receives requests from Indigo-user interfaces, forwards the requests to one or more of the three possible servers, and returns the results back to Indigo and its graphical user interface. Functionally, the Data Analysis and Mining (DAM) module receives requests from Indigo, performs some data pre-processing, sends the pre-processed data mining requests to one of the data mining servers, and publishes the results at DAM end-point for Indigo to fetch.

The rest of the document is structured as follows: Chapter 2 describes the interfaces of DAM; Chapter 3 describes the interfaces of data pre-processing components; Chapter 4 describes the interfaces of data mining components running at the three data mining servers, the OKFGR server, UEP server, and Fraunhofer server. The components are listed in Table 1. For complex data mining services, we also present the internal architectures.

Table 1: Data mining module and its interfaces

Data Mining Module	Interfaces	Provided by server	Chapters
Descriptive statistics	DescriptiveStats.OBeu - R package	OKFGR	4.2.1
Time series analysis, predictions	TimeSeries.OBeu - R package	OKFGR	4.2.2
Clustering and Similarity learning	Cluster.OBeu - R package	OKFGR	4.2.3
Rule/pattern mining	EasyMiner API EasyMiner UI EasyMiner Services Internal APIs	UEP	4.1.2 4.1.4 4.1.1
Outlier/anomaly detection	EasyMiner API jaroslav-kuchar/fpmoutliers - R package outlier_dm - python module	UEP Fraunhofer	4.1.4 4.1.5 4.3

2 The Interface of Data Analysis and Mining (DAM)

Data analysis and mining modules of the OBEU project are located in three places: (1) UEP server, (2) OKFGR server, (3) Fraunhofer server. The interface of Data Analysis and Mining (DAM)¹ provides a unified functional interface to process users' requests. The DAM is implemented using Python Flask. The DAM architecture and the functional communication with other modules are illustrated in Figure 1.

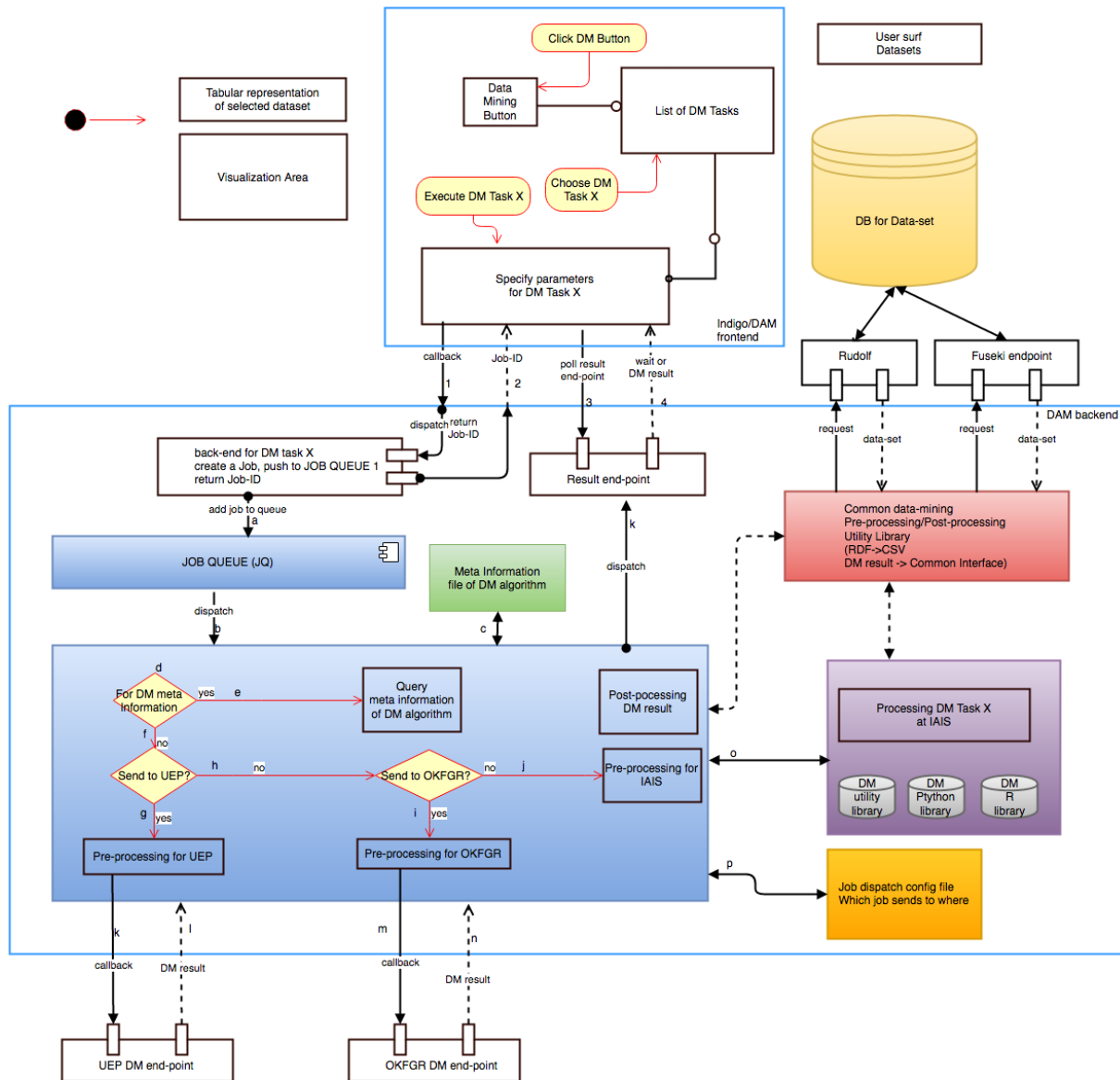


Figure 1: Architecture of DAM and its functional communication with other modules

¹ <https://github.com/openbudgets/DAM>

2.1 Functional Interfaces of DAM

The root DAM endpoint is currently located at the Fraunhofer server and publicly reachable at the following URL: <http://dam-obeu.iais.fraunhofer.de/>, as illustrated in Figure 2.

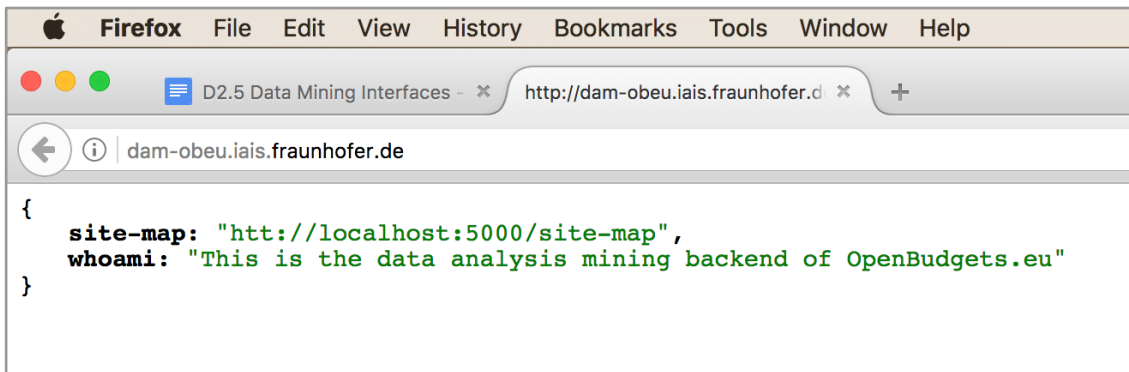


Figure 2: Endpoint of DAM²

The endpoint of each specific data mining task (algorithm) is named according to its function name. The mechanism of forwarding the requests to different servers is transparent to the end-users. Users do not know by which server the task is processed. All the endpoints are listed in Table 2.

Table 2: List of data mining endpoints and the processing place

Data mining function	End-point	Processing place
statistics	http://dam-obeu.iais.fraunhofer.de/statistics	OKFGR server
time series	http://dam-obeu.iais.fraunhofer.de/time_series	OKFGR server
comparative analysis	http://dam-obeu.iais.fraunhofer.de/KPI	OKFGR server
outlier-detection (LOF)	http://dam-obeu.iais.fraunhofer.de/outlier_detection/LOF	Fraunhofer server
outlier-detection (FQR)	http://dam-obeu.iais.fraunhofer.de/outlier_detection/FQR	UEP server
rule-mining	http://dam-obeu.iais.fraunhofer.de/rule_mining	UEP server

² This is the simplest way how to check whether DAM server works.

2.2 Communication Interface with UEP

2.2.1 Data Flow Inside DAM and uep_dm Module

The uep_dm module provides a unified interface for DAM to send data mining requests to the UEP server.

The DAM end-point of rule-mining is

http://dam-obeu.iais.fraunhofer.de/rule_mining

The data format, which DAM receives from Indigo, is in JSON format, while the UEP server needs CSV format. DAM will do data-preprocessing to transform JSON into CSV, construct a data mining request based on the format required by the UEP server, and send both the data and mining request to UEP server. Parameters required by the UEP server are as follows.

1. Set default value

- apiURL
 - constant value which is the easymining server URL
 - CONSTANT
 - value: <https://br-dev.lmcloud.vse.cz/easyminercenter/api>
- apiKey
 - unique Key bound to user, for the identify of task
 - default value
 - value: <your api Key>
- taskName
 - UEP server runs several tasks, this is to identify which task user request.
 - CONSTANT
 - value: **simple**
- outputFormat
 - CONSTANT
 - **json**
- antecedentColumns
 - default value
 - value: **[]**
- consequentColumns
 - default value
 - value: **["amount.sum"]**
- minConfidence
 - default value
 - value: **0.7**
- minSupport
 - default value
 - value: **0.1**
- csvSeparator
 - default value

- value: ,
- csvEncoding
 - default value
 - value: **utf8**

Data-set is sent by Indigo with a link, such as: <http://dam-obeu.iais.fraunhofer.de/sample-data/aggregate.json>

This link points to the data-set in the JSON format with the parameter name “BABBAGE_FACT_URI” or “BABBAGE_AGGREGATE_URI”.

2. Get “filename” parameter

- default: <http://dam-obeu.iais.fraunhofer.de/sample-data/aggregate.json>
- get from BABBAGE_FACT_URI
- get from BABBAGE_AGGREGATE_URI

As the EasyMiner API requires CSV data format as input, a preprocessing step is applied to the JSON source.

3. Construct input CSV by “filename”

- from JSON file to CSV by preprocessing
- get name “inputCSVFileName”

Now we have the default values from step 1 and the CSV file from step 3 which is actually from the source in step 2. We can send these parameters to the package `uep_dm`, calling the function [send_request_to_UEP_server](#).

4. Call `send_request_to_UEP_server` with parameter “inputCSVFileName” and all parameters.

- `job = q_dm.enqueue_call(func=uep_dm.send_request_to_UEP_server, args=[inputCSVFileName, taskName, apiURL, apiKey, outputFormat, antecedentColumns, consequentColumns, minConfidence, minSupport, csvSeparator, csvEncoding], result_ttl=5000)`

Then, we are jumping from DAM to `uep_dm` module, inside the [send_request_to_UEP_server](#) function, the processing flow is as follows. First step is the upload of the CSV file to the UEP server.

5. Upload CSV file to UEP server use `upload_data_set`

- parameter
 - `csv_file`
 - `csv_separator`
 - `csvEncoding`
- POST URL
 - `requests.post(API_URL + '/datasources?separator=' + urllib.parse.quote(csv_separator) + '&encoding=' + csvEncoding`

```
+ '&type=limited&apiKey=' + API_KEY, files=files,
headers=headers)
```

- get “dataSourceId”

Then second step is to create the miner instance and create the identification for the CSV dataset.

6. Create data mining task by *create_miner*

- parameter
 - dataSourceId
 - minerName
 - apiURL
 - apiKey
- POST URL
 - requests.post(apiURL + `"/miners?apiKey="` + apiKey, headers=headers, data=json_data.encode())
- get “minerId”

7. Get data column names by *preprocess_data_fields_to_attributes*

While the “taskName” parameter is “simple”, that means the task is a rule mining task, call `define_data_mining_task` to send all requirement of the task to server. Inside the function, it will first compose a JSON file `json_data` with some parameters like below, then post this `json_data` to server.

8. Start task by *define_data_mining_task*

- parameter
 - apiUrl, apiKey, taskName, minerId, minerName, antecedentColumns, consequentColumns, attributesColumnsMap, minConfidence, minSupport
- `json_data` = `json.dumps`({`"miner"`: minerId, `"name"`: minerName, `"limitHits"`: 1000, `"IMs"`: [


```

          {
              "name": "CONF",
              "value": minConfidence
          },
          {
              "name": "SUPP",
              "value": minSupport
          }
      ],
      "antecedent": antecedent,
      "consequent": consequent
    })
```
- POST URL

- requests.post(apiUrl + `"/tasks/"`+taskName+`"?apiKey="` + apiKey,
 - headers=headers, data=json_data.encode())
 - get `"task_id"`

After we get the `"task_id"`, we can call the final function `export_rules_in_JSON`, inside this function, it will start the task on server and then get response.

9. Get result from server by *export_rules_in_JSON*

- start task
- get result
- check result by URL

2.2.2 Sample Usage

Suppose a user chooses (1) a dataset pointed by the link:

http://ws307.math.auth.gr/rudolf/public/api/3/cubes/budget-kilkis-expenditure-2015_74025/aggregate?drilldown=administrativeClassification.prefLabel%7CeconomicClassification.prefLabel%7CbudgetPhase.prefLabel&aggregates=amount.sum

and (2) rule-mining as the data mining service.

The communication interface between Indigo and DAM is that Indigo knows that the DAM endpoint for rule mining, which is http://dam-obeu.iais.fraunhofer.de/rule_mining, and that the link of the dataset shall be stored in `BABBAGE_FACT_URI` or `BABBAGE_AGGREGATE_URI` variable.

Indigo can send the following curl command to DAM.

```
curl -H "Content-Type:application/json; charset=UTF-8" --request POST
'http://dam-obeu.iais.fraunhofer.de/rule_mining?BABBAGE_FACT_URI=http://ws307.math.auth.gr/rudolf/public/api/3/cubes/budget-kilkis-expenditure-2015__74025/aggregate?drilldown=administrativeClassification.prefLabel%7CeconomicClassification.prefLabel%7CbudgetPhase.prefLabel&aggregates=amount.sum'
```

When DAM receives the above request, it will first do data pre-processing. In this case, it will extract data from the link and save the data in a CSV file. Then, it will call the UEP interface, push a new task in the job-queue, and create an endpoint for Indigo to fetch results. The endpoint has the format: <http://dam-obeu.iais.fraunhofer.de/results/<job-id>>, for example: <http://dam-obeu.iais.fraunhofer.de/results/3ffecc31-e14a-4289-ae7-833f3fdebc28>

Indigo can fetch the data mining result through a curl command:

```
curl -H "Host:sub.domain.com" http://dam-  
obeu.iais.fraunhofer.de/results/3ffecc31-e14a-4289-ae7-833f3fdebc28
```

If the job has not been finished, DAM will return the 'wait' status in a JSON format.

```
{"status": "Wait!"}
```

If the job is finished, DAM will return the data mining result in the JSON format.

2.3 Communication Interfaces with Fraunhofer Server and OKFGR Server

DAM provides a unified interface for data mining requests which might be carried out at Fraunhofer Server or OKFGR server. The unified interface for data mining request is: [http://dam-obeu.iais.fraunhofer.de/<task>\[/subtask\]](http://dam-obeu.iais.fraunhofer.de/<task>[/subtask])

We provide two examples as follows.

2.3.1 Example: Communication with OKFGR server

Suppose a user chooses:

1. a JSON dataset pointed by the link
2. dimensions as follows:
functional_classification_2.Function|functional_classification_2.Code
3. 'Revised' amount,
4. coef.out1 value is 0.8,
5. set box.outliers = TRUE,
6. box.wdth value is 0.2,
7. 'spearman' method for cor.method for statistic analysis.

The communication interface between Indigo and DAM is that Indigo knows that the DAM endpoint for statistical analysis, which is:

<http://dam-obeu.iais.fraunhofer.de/statistics>,

and that the link of the dataset shall be stored in `BABBAGE_FACT_URI` or `BABBAGE_AGGREGATE_URI` variable. Other variables are: dimensions, amount, coef.out1, box.outliers, box.wdth, cor.method.

Indigo can send the following curl command to DAM.

```
curl -H "Content-Type:application/json; charset=UTF-8" --request POST  
'http://dam-  
obeu.iais.fraunhofer.de/statistics?json_data=sample_json_link_openspending&  
dimensions='functional_classification_2.Function|functional_classification_  
2.Code'&amount='Revised'&coef.out1=0.8&box.outliers=TRUE&box.wdth=0.2&cor.m  
ethod='spearman'
```


When DAM receives the above request, it will first do data pre-processing. In this case, it will extract data from the link and save the data in a CSV file. Then, it will call the OKFGR interface, push a new task in the job-queue, and create an endpoint for Indigo to fetch results. The endpoint has the format: <http://dam-obeu.iais.fraunhofer.de/results/<job-id>>, for example <http://dam-obeu.iais.fraunhofer.de/results/3ffecc31-e14a-4289-ae7-833f3fdebc28>

Indigo can fetch the data mining result through a curl command:

```
curl -H "Host:sub.domain.com" http://dam-obeu.iais.fraunhofer.de/results/3ffecc31-e14a-4289-ae7-833f3fdebc28
```

If the job has not been finished, DAM will return the 'wait' status in a JSON format:

```
{"status": "Wait!"}
```

If the job is finished, DAM will return the data mining result in the JSON format.

2.3.2 Example: Communication with Fraunhofer Server

Suppose a user chooses (1) a dataset pointed by the link:

http://ws307.math.auth.gr/rudolf/public/api/3/cubes/budget-kilkis-expenditure-2015_74025/aggregate?drilldown=administrativeClassification.prefLabel%7CeconomicClassification.prefLabel%7CbudgetPhase.prefLabel&aggregates=amount.sum

and (2) the LOF outlier-detection as the data mining service.

The communication interface between Indigo and DAM is that Indigo knows that the DAM endpoint for LOF outlier-detection, which is:

http://dam-obeu.iais.fraunhofer.de/outlier_detection/LOF/sample
or http://dam-obeu.iais.fraunhofer.de/outlier_detection/LOF,

and that the link of the dataset shall be stored in `BABBAGE_FACT_URI` variable.

Indigo can send the following curl command to DAM:

```
curl -H "Host:sub.domain.com" http://dam-obeu.iais.fraunhofer.de/outlier_detection/LOF/sample?BABBAGE_FACT_URI=http://ws307.math.auth.gr/rudolf/public/api/3/cubes/budget-kilkis-expenditure-2015_74025/aggregate?drilldown=administrativeClassification.prefLabel%7CeconomicClassification.prefLabel%7CbudgetPhase.prefLabel&aggregates=amount.sum
```

When DAM receives the above request, it will first do data pre-processing. In this case, it will extract data from the link and save the data in a CSV file. Then, it will call the LOF outlier-detection interface, push a new task in the job-queue, and create an endpoint for Indigo to fetch results. The endpoint has the format:

<http://dam-obeu.iais.fraunhofer.de/results/<job-id>>, for example: <http://dam-obeu.iais.fraunhofer.de/results/3ffecc31-e14a-4289-ae7-833f3fdebc28>

Indigo can fetch the data mining result through a curl command:

```
curl -H "Host:sub.domain.com" http://dam-obeu.iais.fraunhofer.de/results/3ffecc31-e14a-4289-ae7-833f3fdebc28
```

If the job has not been finished, DAM will return the 'wait' status in a JSON format.

```
{"status": "Wait!"}
```

If the job is finished, DAM will return the data mining result in the JSON format.

2.4 Meta-information of Algorithm for Indigo

To dynamically construct user interface, DAM provides Indigo with meta-information of each data mining algorithms. The interface is structured in a JSON file as follows.

```
"<function_name>": {
  "configurations": {
    "facts": {
      "inputs": {
        "BABBAGE_FACT_URI": {
          "name": "BABBAGE_FACT_URI",
          "title": "",
          "cardinality": <a natural number>,
          "guess": <boolean>,
          "required": <boolean>,
          "type": "URI pointing to a Babbage compliant
facts API request"
        }
      },
      "outputs": {
        "output": {
          "name": "output",
          "cardinality": <a natural number>,
          "type": <value|collection of objects>
        }
      },
      "prompt": "...",
      "method": 0,
      "endpoint": "",
      "name": "facts",
      "title": ""
    }
  },
  "name": "",
  "title": "",
  "description": ""
}
```

For example, the metadata information of **outlier detection** is described as below.

```

{ "outlier_detection": {
  "configurations": {
    "facts": {
      "inputs": {
        "BABBAGE_FACT_URI": {
          "name": "BABBAGE_FACT_URI",
          "title": "Data coming from an aggregation",
          "cardinality": "1",
          "guess": false,
          "required": true,
          "type": "URI pointing to a Babbage compliant
facts API request"
        }
      },
      "outputs": {
        "output": {
          "name": "output",
          "cardinality": 1,
          "type": "collection of objects"
        }
      },
      "prompt": "Build an aggregate, with a time-related drill-down
and then enter the prediction steps parameter from the left and click
on the execute button on top right.",
      "method": 0,
      "endpoint": "http://dam-
obeu.iais.fraunhofer.de/outlier_detection/LOF",
      "name": "facts",
      "title": "Facts outlier detection"
    }
  },
  "name": "outlier_detection",
  "title": "Outlier Detection",
  "description": ""
}

```

3 Interfaces of Data Pre-processing

3.1 Discretization via SPARQL

In order to enable discretization of RDF data we implemented a command-line tool that allows to discretize numeric literals in RDF via SPARQL. Discretization, which is also known as binning, converts continuous numeric values into discrete intervals. This allows to treat numbers as categorical data, which is required by some data mining tools. For example, discretization is used for association rule mining with EasyMiner (4.1), outlier detection with FPM (4.1.5), or tensor factorization with RESCAL.³ While EasyMiner has discretization built in, discretization via SPARQL can be incorporated directly in data preprocessing for generic data mining tasks. This is why we decided to wrap the EasyMiner-Discretization⁴ library and allow to apply it via SPARQL Update.

The tool supports three discretization methods: equidistance, equifrequency, and equisize. Equidistant discretization creates intervals of the same size, equifrequent discretization creates intervals with approximately the same number of members, and equisized discretization creates intervals based on minimum support. Equidistant and equifrequent discretization requires user to specify the desired integer number of intervals (bins) to generate. Equisized discretization requires user to provide the minimum support ($\in (0, 1)$) that a generated interval must have.

Application of discretization is guided via a SPARQL Update operation that uses *wishful thinking*. The operation contains a placeholder variable `?interval`, to which the tool binds the generated intervals by rewriting the operation. This allows to specify many ways in which the generated intervals should be used. For example, the intervals can replace the discretized numeric literals or they can be inserted as objects of an additional property. Here is an example of such operation:

```
PREFIX : <http://example.com/>
PREFIX obeu-measure: <http://data.openbudgets.eu/ontology/dsd/measure/>

WITH <http://data.openbudgets.eu/resource/dataset/budget-athens-
expenditure-2015>
DELETE {
  ?observation obeu-measure:amount ?value .
}
INSERT {
  ?observation :discretizedAmount ?interval .
}
WHERE {
  ?observation obeu-measure:amount ?value .
}
```

³ <https://github.com/mnick/rescal.py>

⁴ <https://github.com/KIZI/EasyMiner-Discretization>

In this operation the `WHERE` clause select the values to discretize, the `INSERT` clause inserts the generated intervals as objects of the `:discretizedAmount` property, and the `DELETE` clause deletes the original numeric values. In the background, the operation is rewritten to paged `SELECT` queries to fetch the values to be discretized. Once discretization generates the intervals, the operation is rewritten to a `SPARQL Update` operation that implements its specified transformation.

The generated intervals are represented as instances of `schema:QuantitativeValue`. The bounds of the intervals are described using `schema:minValue` for the lower bound and `schema:maxValue` for the upper bound. Classes from the SemanticScience Integrated Ontology⁵ are used to determine whether the bounds are open or closed. The intervals are identified with UUID-based URNs. The following listing shows an example interval.

```
@prefix schema: <http://schema.org/> .
@prefix sio:    <http://semanticscience.org/resource/SIO_> .

<urn:uuid:4E98F3EE-2861-4A4B-A39C-487A7018165E> a schema:QuantitativeValue,
                                                sio:001254, # Left-closed interval
                                                sio:001252 ; # Right-open interval

    schema:minValue 1000 ;
    schema:maxValue 3000 .
```

The intervals are loaded into a named graph provided via the `graph` CLI parameter. If this parameter is missing, the tool attempts to guess a named graph to load the interval to. It uses the graph specified by `WITH`, `USING`, or in the `INSERT` clause. If no graph is found, the tool asks you to provide it explicitly via `graph`.

If not all values to be discretized are numeric, the non-numeric values are ignored unless the `strict` CLI parameter is used. In such case the discretization fails if it encounters a non-numeric value. An example invocation of the tool is shown here:

```
discretize_sparql --endpoint http://localhost:8890/sparql-auth \
                 --auth dba:dba \
                 --update discretize_amounts.ru \
                 --method equipfrequency \
                 --bins 15 \
                 --strict
```

A special treatment is applied if OpenLink Virtuoso⁶ is used as the queried RDF store. Since Virtuoso has limited support for decimal digits in `xsd:decimal`, the tool rounds the generated intervals to this supported maximum precision to avoid values left outside of the generated intervals.

The tool is released as open source and is available at:

<https://github.com/jindrichmynarz/discretize-sparql>

⁵ <http://semanticscience.org>

⁶ <https://virtuoso.openlinksw.com>

4 Interfaces of the Three Data Mining Servers

This chapter contains description of interfaces of the three currently used data mining servers: [4.1](#) UEP EasyMiner Server, [4.2](#) OKFGR OpenCPU Server and [4.3](#) Outlier_dm lib at Fraunhofer Server.

4.1 EasyMiner - UEP Data Mining Server

EasyMiner is a complex web data mining system for data mining of association rules and outlier detection. The system is being developed at the University of Economics, Prague.

The system is based on a web service architecture supporting the individual steps of the data mining process (data upload, data preprocessing, execution of data mining algorithms, testing).

The end user does not have to use all the individual web services and their APIs. In the integration component, EasyMinerCenter, there is a complex API supporting the full functionality. There is also a GUI available (see [Figure 4](#)).

4.1.1 Architecture & Internal APIs

Data mining system EasyMiner is based on connection of RESTful web services architecture. This architecture is shown in [Figure 3](#). The main components are EasyMinerCenter, EasyMiner-Data, EasyMiner-Preprocessing, EasyMiner-Miner and EasyMiner-Scorer. The frontend component (service) is EasyMinerCenter, other components belong to the backend.

Each component has documented RESTful API. In case of need, it is able to call directly a selected backend service. The API documentation is available in Swagger form. In the following text, the paths to the documentation are written for the default installation architecture⁷ of EasyMiner.

⁷ The default installation is available using Docker images - see <https://github.com/kizi/easyminer>

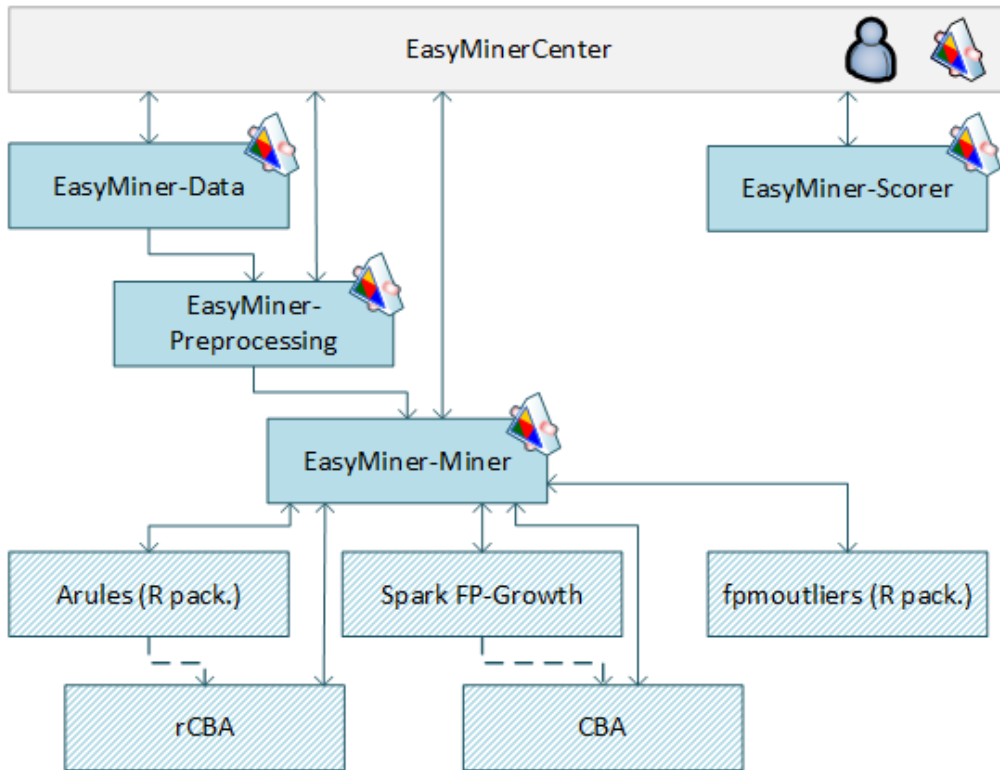


Figure 3: EasyMiner - Architecture of services

EasyMinerCenter

- Front the user's perspective, this is the only interface, with which the user must communicate. This component provides graphical user interface (usable in each modern web browser) and RESTful API for integration with other systems (or simple user scripting).
- The main functionality of this component is integration of other (back-end) services and user and task management.
- UI: <http://<server>/easyminercenter>
- API: <http://<server>/easyminercenter/api>
- API documentation: <http://<server>/easyminercenter/api>

EasyMiner-Data

- Service for management of data sources. This service supports upload data in CSV or RDF and its storage into database. The data are stored in database - MySQL (MariaDB) or Hive.
- API: <http://<server>/easyminer-data/api/v1>
- API documentation: <http://<server>/easyminer-data/index.html>

EasyMiner-Preprocessing

- The preprocessing service supports creation of datasets for data mining. It takes data fields from datasource stores using EasyMiner-Data and prepares data fields using one of these preprocessing algorithms: each value-one bin, intervals enumeration, nominal enumeration, equidistant intervals, equisized intervals.

- This service also provides the “hashing” functionality for support of special characters in data values within the run of data mining tools in EasyMiner-Miner.
- API: server/easyminer-preprocessing/api/v1
- API documentation: server/easyminer-preprocessing/index.html

EasyMiner-Miner

- The service EasyMiner-Miner support the run of data mining algorithms for data mining of association rules (optionally with pruning) and outlier detection. The service executes algorithms Apriori, FP-Growth, rCBA and fpmoutlier (described in deliverable D2.4).
- API: server/easyminer-miner/api/v1
- API documentation: server/easyminer-miner/index.html

EasyMiner-Scorer

- EasyMiner-Scorer is a web service for testing of classification models based on association rules.
- API: server/easyminer-scorer/v0.3/
- API documentation: server/easyminer-scorer/index.html

4.1.2 Association Rule Mining API

For usage of API, the user must have an own user account in the EasyMinerCenter. For the purpose of authentication, each user account has an own API key. The API key can be generated using GUI or using a POST request to API:

server/easyminercenter/api/users

An example of usage of the association rule mining API (written in Python) is available on the URL: <http://www.easyminer.eu/api-tutorial>

Data mining process using the main EasyMiner API endpoint⁸:

1. Upload data in CSV

- the data can optionally be zipped
- suitable for files of limited size (up to 50MB), for larger files, the user must use a cyclical post call directly on data service

```
headers = {"Accept": "application/json"}
files = {"file", open(CSV_FILE, 'rb')}
r = requests.post(API_URL + '/datasources?separator=' +
urllib.parse.quote(
    CSV_SEPARATOR) + '&encoding=' + CSV_ENCODING +
'&type=limited&apiKey=' + API_KEY, files=files, headers=headers)
datasource_id = r.json()["id"]
```

2. Create miner

```
headers = {'Content-Type': 'application/json', "Accept":
"application/json"}
```

⁸ EasyMiner API endpoint is currently available at: <https://br-dev.lmcloud.vse.cz/easyminercenter/api>


```

json_data = json.dumps({"name": "TEST MINER", "type": "cloud",
"datasourceId": datasource_id})
r = requests.post(API_URL + "/miners?apiKey=" + API_KEY, headers=headers,
data=json_data.encode())
miner_id = r.json()["id"]

```

3. Preprocess data – generate data fields from data fields stored in a data source

- the user defines preprocessing algorithm for each attribute; it is also possible to generate more attributes from one data field

```

headers = {'Content-Type': 'application/json', "Accept":
"application/json"}
r = requests.get(API_URL + '/datasources/' + str(datasource_id) +
'?apiKey=' + API_KEY, headers=headers)
datasource_columns = r.json()['column']
attributes_columns_map = {}
for col in datasource_columns:
    column = col["name"]
    json_data = json.dumps(
        {"miner": miner_id, "name": column, "columnName": column,
"specialPreprocessing": "eachOne"})
    r = requests.post(API_URL + "/attributes?apiKey=" + API_KEY,
headers=headers, data=json_data.encode())
    if r.status_code != 201:
        break # error occurred
    attributes_columns_map[column] = r.json()['name'] # map of created
attributes (based on the existing data fields)

```

4. Define association rule mining task

- attributes for the antecedent and consequent parts of association rules (with any value or with a fixed value)
- definition of threshold values of requested interest measures (confidence, support, lift)

```

# define data mining task
antecedent = []
consequent = []

# prepare antecedent pattern
if len(ANTECEDENT_COLUMNS):
    # add to antecedent only fields defined in the constant
    for column in ANTECEDENT_COLUMNS:
        antecedent.append({"attribute":attributes_columns_map[column]})
else:
    # add to antecedent all fields not used in consequent

```

```
for (column, attribute_name) in attributes_columns_map.items():
    if not(column in CONSEQUENT_COLUMNS):
        antecedent.append({"attribute": attribute_name})

# prepare consequent pattern
for column in CONSEQUENT_COLUMNS:
    consequent.append({"attribute": attributes_columns_map[column]})

json_data = json.dumps({"miner": miner_id,
                        "name": "Test task",
                        "limitHits": 1000,
                        "IMS": [
                            {
                                "name": "CONF",
                                "value": MIN_CONFIDENCE
                            },
                            {
                                "name": "SUPP",
                                "value": MIN_SUPPORT
                            }
                        ],
                        "antecedent": antecedent,
                        "consequent": consequent
                    })

# define new data mining task
r = requests.post(API_URL + "/tasks/simple?apiKey=" + API_KEY,
                 headers=headers, data=json_data.encode())
print("create task response code:" + str(r.status_code))
task_id = str(r.json()["id"])
```

5. Execute the mining task

```
r = requests.get(API_URL + "/tasks/" + task_id + "/start?apiKey=" +
API_KEY, headers=headers)
while True:
    time.sleep(1)
    # check state
    r = requests.get(API_URL + "/tasks/" + task_id + "/state?apiKey=" +
API_KEY, headers=headers)
    task_state = r.json()["state"]
    print("task_state:" + task_state)
    if task_state == "solved":
        break
    if task_state == "failed":
        print("task failed executing")
```

```
break
```

6. Export the results (in PMML AssociationModel, GUHA PMML or simple JSON)

```
# export rules in JSON format
headers = {"Accept": "application/json"}
r = requests.get(API_URL + '/tasks/' + task_id + '/rules?apiKey=' +
API_KEY, headers=headers)
task_rules = r.json()

# export of standardized PMML AssociationModel
r = requests.get(API_URL + '/tasks/' + task_id +
'/pmml?model=associationmodel&apiKey=' + API_KEY)
pmml = r.text

# export of GUHA PMML
r = requests.get(API_URL + '/tasks/' + task_id +
'/pmml?model=guha&apiKey=' + API_KEY)
guha_pmml = r.text
```

The described, main RESTful API of EasyMiner is the integration interface for other software tools developed in the OpenBudgets.eu Project.

The functionality of association rules mining and building of classification models based on association rules was completely tested using standard datasets from UCI repository.

4.1.3 Outlier Detection API

The outlier detection is integrated with other services of the data mining system EasyMiner. The data mining process for outlier detection tasks is described in the following list. The first three steps are the same as in the process for association rule mining. It is also possible to use the same prepared dataset for both tasks - for outlier detection and also for association rule mining.

1. **Upload data in CSV**
2. **Create miner**
3. **Preprocess data** – generate data fields from data fields stored in a data source
 - the user defines preprocessing algorithm for each attribute
 - opposite to the association rule mining tasks, it is necessary to create only attributes, which should be used for outlier detection task (it is not possible to select only a subset of attributes in the task definition)

4. **Define outlier detection mining task**

```
headers = {'Content-Type': 'application/json', "Accept":
"application/json"}
json_data = json.dumps({"miner": miner_id, "minSupport": min_support})
```

```
r = requests.post(API_URL + "/outliers-tasks?apiKey=" + API_KEY,
headers=headers, data=json_data.encode())
outlier_task_id = r.json()["id"]
```

5. Execute the mining task

```
r = requests.get(API_URL + "/outliers-tasks/" + outlier_task_id +
"/start?apiKey=" + API_KEY, headers=headers)
while True:
    time.sleep(1)
    # check state
    r = requests.get(API_URL + "/outliers-tasks/" + outlier_task_id +
"/state?apiKey=" + API_KEY, headers=headers)
    task_state = r.json()["state"]
    print("task_state:" + task_state)
    if task_state == "solved":
        break
    if task_state == "failed":
        print("task failed executing")
        break
```

6. Read the results

```
offset = 0
limit = 10

headers = {"Accept": "application/json"}
r = requests.get(API_URL + '/outliers-tasks/' + outlier_task_id +
'/outliers?apiKey=' + API_KEY + '&offset=' + offset + '&limit=' + limit,
headers=headers)
outliers = r.json()['outlier']
```

4.1.4 Association Rule Mining UI

EasyMiner provides to the users also graphical web user interface. The user can use it in each modern web browser. The GUI is available on the URL <server>/easyminercenter.

Figure 4 shows the “main” UI for data mining of association rules. The full UI is based on drag&drop operations. On the right side, there are pallets of data fields (original data columns from datasource; A) and preprocessed attributes (B) usable in association rules. The user defines a “pattern” of association rules (C) – dropping the attributes in the antecedent and consequent part of the pattern. The results are then shown in the section “Discovered rules” (D). In the development of this deliverable, the UI was modified for the support of special and non-ASCII characters in names and values of data fields and attributes.

For this deliverable, it is important, that the main RESTful API endpoint and graphical UI are fully compatible. The user can for example define the preprocessing and a “testing” task using

UI and then run more tasks using API calls - on the same prepared (preprocessed) datasets, collecting the results into one repository.

Figure 4: EasyMiner - User interface for association rule mining

4.1.5 fpmoutliers - R Package

This section documents the R implementation of algorithms for detection of outliers based on frequent pattern mining. The package is developed on the University of Economics, Prague. The current versions supports multiple algorithms for the outlier detection based on frequent pattern mining. There are implementations of six existing algorithms as baselines (FPCOF, FPOF, LFPOF, MFPOF, WCFPOF, WFPOF) and **one innovative approach (FPI)**.

All implemented methods require input data as a data frame in R and parameter `minSupport` - minimum support interest measure. It is the same measure as explained in EasyMiner section. Lower value will cause revealing of less frequent patterns in data and improve the quality and readability of provided outputs. However, lower values also lead to higher complexity of the computation.

The output is a list that mainly contains outlier scores - one value for each input row from the data frame:

- `minSupport` - minimum support setting for frequent itemsets mining
- `maxlen` - maximum length of frequent itemsets
- `model` - frequent itemset model represented as `itemsets-class` from R `arules` package
- `scores` - outlier/anomaly scores for each observation/row of the input dataframe

Example of a basic usage for *FPI*:

```
library(fpmoutliers)
dataFrame <- read.csv(system.file("extdata", "fp-outlier-customer-
data.csv", package = "fpmoutliers"))
model <- FPI(dataFrame, minSupport = 0.001)
```

```
dataFrame <- dataFrame[order(model$scores, decreasing = TRUE),]  
print(dataFrame[1,]) # instance with the highest anomaly score  
print(dataFrame[nrow(dataFrame),]) # instance with the lowest anomaly  
score
```

The package is also focused on explanations of outlier scores. There is a function `visualizeInstance(dataFrame, index)`, where *index* is index of instance in the data frame that we would like to visualize using bar plots. Example of usage:

```
library("fpmoutliers")  
dataFrame <- read.csv(  
  system.file("extdata", "fp-outlier-customer-data.csv", package =  
  "fpmoutliers"))  
model <- FPI(dataFrame, minSupport = 0.001)  
# sort data by the anomaly score  
dataFrame <- dataFrame[order(model$scores, decreasing = TRUE),]  
visualizeInstance(dataFrame, 1) # instance with the highest anomaly score  
visualizeInstance(dataFrame, nrow(dataFrame)) # instance with the lowest  
anomaly score
```

The visual explanations using bar plots are limited by the number of columns in the input dataframe. The visualization is suitable up to 6-8 columns. The module also provides implementation of textual explanations: `describeInstance(dataFrame, model, index)`, where *model* is the model provided by the outlier detection method (e.g. *FPI*). The output is a list that describes the instance: overall outlier score (parameter *score*), frequent itemsets that match the instance (parameter *itemsets*) and also provides information about contributions of attributes to the overall outlier score (parameter *scores*):

```
library("fpmoutliers")  
dataFrame <- read.csv(  
  system.file("extdata", "fp-outlier-customer-data.csv", package =  
  "fpmoutliers"))  
model <- FPI(dataFrame, minSupport = 0.001)  
# sort data by the anomaly score  
dataFrame <- dataFrame[order(model$scores, decreasing = TRUE),]  
# instance with the highest anomaly score  
out <- describeInstance(dataFrame, model, 1)  
# instance with the lowest anomaly score  
out <- describeInstance(dataFrame, model, nrow(dataFrame))
```

Experimental implementation of automatic build. The algorithm uses simple heuristics to automatically set optimal values for the *minSupport* parameter. The output is the same model as for other outlier detection implementations. Currently suitable and tested only on small datasets:

```
library("fpmoutliers")
data("iris")
model <- fpmoutliers::build(iris)
```

The tool is released as open source and is available at: <https://github.com/jaroslav-kuchar/fpmoutliers>

4.2 OpenCPU Server - OKFGR Data Mining Server

4.2.1 DescriptiveStats.OBeu - R Package

“DescriptiveStats.OBeu” package was built in R Software environment and it's publicly available on Github⁹. It enables the calculation of descriptive statistical measures in Budget data of municipalities across Europe, according to the OpenBudgets.eu data model. It provides the parameters of simple visualizations to meet the tasks of users' needs that were described in detail in Deliverable 2.3 - “Requirements for Statistical Analytics and Data Mining Techniques”.

“DescriptiveStats.OBeu” includes functions for measuring central tendency and dispersion of numeric variables along with their distributions and correlations and the frequencies of categorical variables for a given dataset of the input OpenBudgets.eu fiscal datasets.

The input dataset of the main function is a JSON link or a JSON file or a text in JSON format. There are different parameters that a user could specify (see the package specification) and interact with the results, but the at least should be defined the “dimensions”, “measured.dimensions” and “amounts” parameters to form the dataset. Next, there is an automated process that calculates the basic descriptive measures of tendency and spread, boxplot and histogram parameters to describe and visualize the distribution characteristics of the desired fiscal dataset.

The final returns of this process are contained in a list of parameters in JSON format that are needed to form summary tables of central tendency and dispersion measures and visualize boxplots, histograms, barplots and correlation matrices of the input fiscal data (for further details about the package see ds.analysis function¹⁰).

⁹ <https://github.com/okgreece/DescriptiveStats.OBeu>

¹⁰ <https://github.com/okgreece/DescriptiveStats.OBeu/blob/master/R/ds.analysis.R>

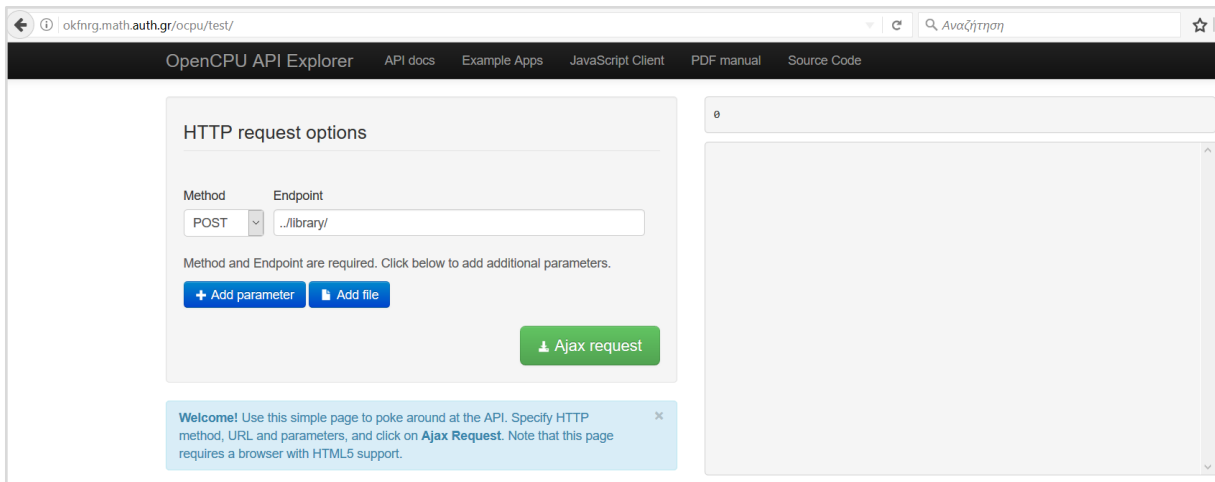


Figure 5: OpenCPU Server Interface

The main parts of the returned result of a process are contained in a list that consists of:

- Descriptives - the central tendency and spread measures (min, max, range, mean, median, quantiles, variance, standard deviation, skewness, kurtosis)
- Boxplot - boxplot parameters (lo.whisker, lo.hinge, median, up.hinge, up.whisker, box.width, lo.out, up.out, n)
- Histogram - histogram parameters (cuts, counts, normal.curve, mean, median)
- Frequencies - barplot parameters (frequencies, relative.frequencies)
- Correlation - correlation matrix (cor.matrix)

Example of usage

The user should load the library, specify the function “open_spending.ds” in the endpoint and the method as “POST”.

The next step is to specify the name of the parameters and their values, described before.

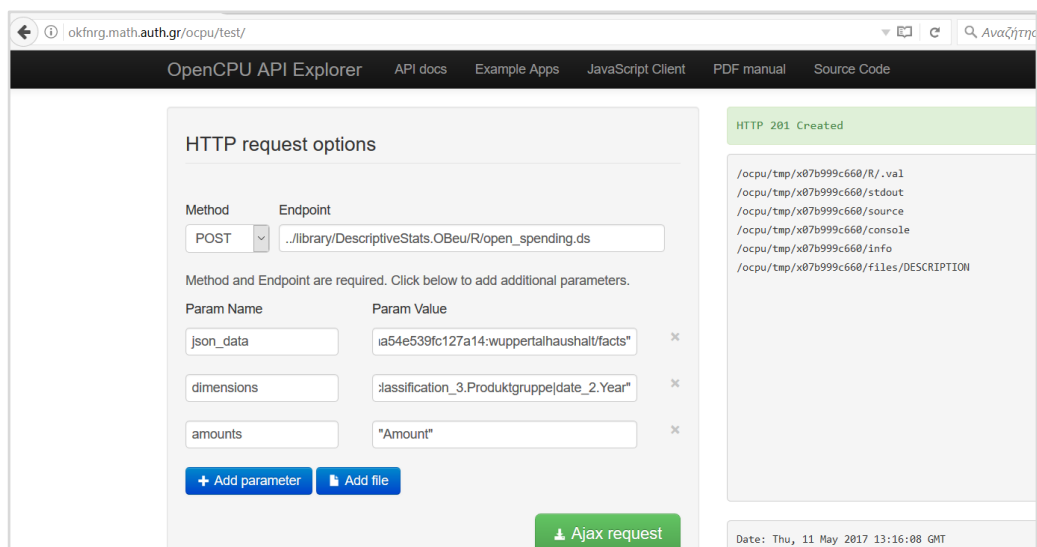


Figure 6: OpenCPU Server - Descriptive Statistics Input Example

The example in the following figure includes these inputs:

Table 3: Input params for Figure 7

Param Name	Param Value
json_data	"http://next.openspending.org/api/3/cubes/4b6d969e07ef7a86aa54e539fc127a14:wuppertalhaushalt/facts"
dimensions	"functional_classification_3.Produktgruppe date_2.Year"
amounts	"Amount"

The HTTP request options return the results in the right panel:

"/ocpu/tmp/x07b999c660/R/.val". The option with ".val" includes the desired results.

```

{
  "descriptives": {
    "Min": {
      "Amount": [
        -2040680.54
      ]
    },
    "Max": {
      "Amount": [
        507995000
      ]
    },
    "Range": {
      "Amount": [
        510035680.54
      ]
    },
    "Mean": {
      "Amount": [
        6171229.3658
      ]
    },
    "Median": {
      "Amount": [
        736038.09
      ]
    },
    "Quantiles": {
      "Amount": [
        243696.13,
        2653000
      ]
    },
    "Variance": {
      "Amount": [
        777106882358169
      ]
    },
    "StandardDeviation": {
      "Amount": [
        27876636.8552
      ]
    }
  }
}

```

Figure 7: OpenCPU Server- Snapshot of Descriptive Statistics Output Example

The output of this process are the parameters described before in JSON format. These parameters will be used further as the input for the visualizations.

4.2.2 TimeSeries.OBeu - R Package

There are OpenBudgets.eu fiscal datasets that consist of values as an ordered sequence of equally spaced time intervals and have an internal structure such as autocorrelation, trend or seasonal variance. “TimeSeries.OBeu” package developed to meet the tasks of users’ needs that were described in detail in Deliverable 2.3 - “Requirements for Statistical Analytics and Data Mining Techniques”. It was built in R Software environment and it's available in Github¹¹.

“TimeSeries.OBeu” package includes functions that automatically analyze the input univariate time series data using methods and techniques, such as local regressions and models from arima family, that consider this internal structure in different levels of OpenBudgets.eu fiscal datasets to extract meaningful characteristics and fit a model to predict the future behavior of such data. A set of tests are implemented in the input time series data to assess the stationarity for further analysis. Depending the nature of the time series data and the stationary tests there are different methods and techniques to extract the most appropriate meaningful characteristics and fit the best model to predict the future behavior of such data

The input dataset of the main function is a JSON link or a JSON file or a text in JSON format. The algorithm requires at least the “time”, “amount” parameters to form the time series data and the “prediction_steps” parameter to predict the future steps. The default order of the model, is fixed to fit the best model through some conditions and diagnostic tests but the user can also interact with the selection of the model’s order to fit the data and specify the “order” parameter.

The final returns of this process is a list of parameters in JSON format that are needed to visualize the time series data with the specified predictions, the decomposition components and the comparison measures of the input fiscal time series data.

The main components of the results are included in a list with subcomponents that consists of:

- `acf.param` - the information about the autocorrelation and partial autocorrelation function of the input data and the residuals after fitting a model (`acf.parameters`, `pacf.parameters`, `acf.residuals.parameters`, `pacf.residuals.parameters`).
- `decomposition` - all the details concerning the decomposition of time series data (`stl.plot`, `stl.general`, `residuals_fitted`, `compare`)
- `model.param` - (`model`, `residuals_fitted`, `compare`)

Example of usage

The user should load the library, specify the function “`open_spending.ts`” in the endpoint and the method as “POST”.

The next step is to specify the name of the parameters and their values, described before.

¹¹ <https://github.com/okgreece/TimeSeries.OBeu>

HTTP request options

Method: POST | Endpoint:

Method and Endpoint are required. Click below to add additional parameters.

Param Name	Param Value	
<input type="text" value="json_data"/>	<input type="text" value="order=adjusted.sum:desc&pagesize=2000"/>	✕
<input type="text" value="time"/>	<input type="text" value="date_2.year"/>	✕
<input type="text" value="amount"/>	<input type="text" value="executed.sum"/>	✕
<input type="text" value="prediction_steps"/>	<input type="text" value="3"/>	✕

+ Add parameter
📎 Add file

⬇ Ajax request

Figure 8: OpenCPU Server- Time Series Input Example

The example in the following figure includes these inputs:

Table 4: Input params for Figure 9

Param Name	Param Value
json_data	"https://next.openspending.org/api/3/cubes/boost:boost-moldova-2005-2014/aggregate?drilldown=date_2.year&order=adjusted.sum:desc&pagesize=2000"
time	"date_2.year"
amount	"executed.sum"
prediction_steps	3

The HTTP request options return the results in the right panel:

"/ocpu/tmp/x060c100f21/R/.val". The option with ".val" includes the desired results.

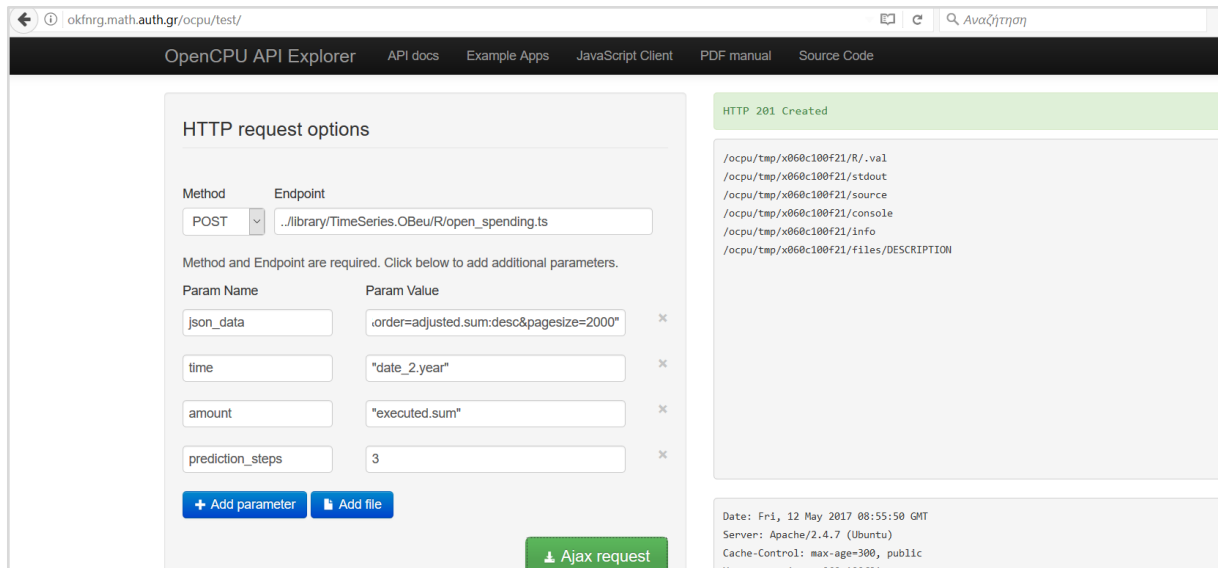
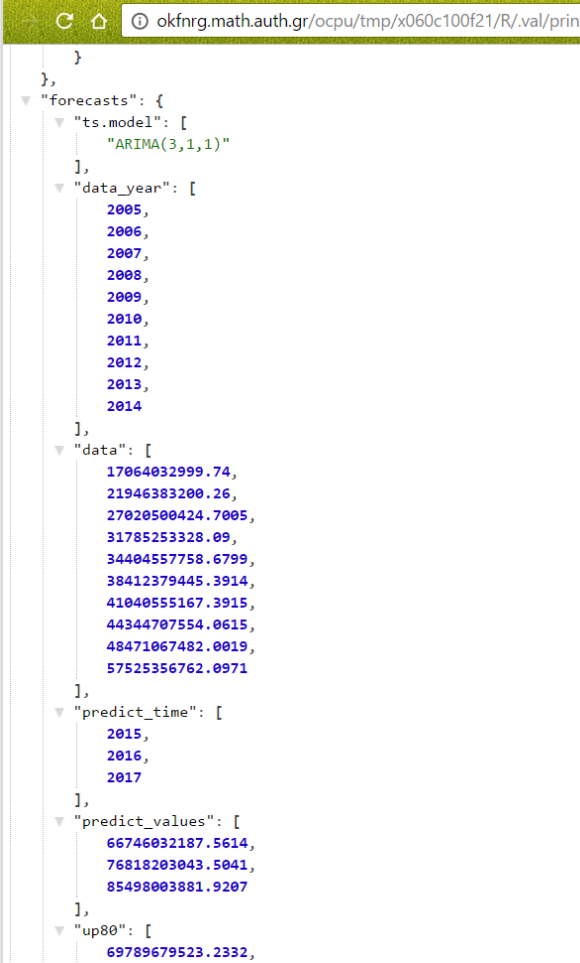


Figure 9: OpenCPU Server - Time Series Request Example

The output of this process are the parameters described before in JSON format, that will be used further and consist the inputs for visualizations.



```

}
},
"forecasts": {
  "ts.model": [
    "ARIMA(3,1,1)"
  ],
  "data_year": [
    2005,
    2006,
    2007,
    2008,
    2009,
    2010,
    2011,
    2012,
    2013,
    2014
  ],
  "data": [
    17064032999.74,
    21946383200.26,
    27020500424.7005,
    31785253328.09,
    34404557758.6799,
    38412379445.3914,
    41040555167.3915,
    44344707554.0615,
    48471067482.0019,
    57525356762.0971
  ],
  "predict_time": [
    2015,
    2016,
    2017
  ],
  "predict_values": [
    66746032187.5614,
    76818203043.5041,
    85498003881.9207
  ],
  "up80": [
    69789679523.2332,
  ]
}

```

Figure 10: OpenCPU Server - Snapshot of Time Series Output Example

4.2.3 Cluster.OBeu - R Package

“Cluster.OBeu” package is developed to find patterns of budget data and divide them into groups of similar observations (clusters). There are various algorithms available that differ significantly in their notion of how to form and define a cluster, and depending the nature of the problem to be solved there is an automated process that selects the appropriate clustering algorithm and number of clusters.

This package provides the needed parameters to visualize the results to meet the tasks of users’ needs that were described in detail in Deliverable 2.3 - “Requirements for Statistical Analytics and Data Mining Techniques”. It was built in R Software environment and it’s available in Github ¹².

There are different clustering models to be selected through an evaluation process. The user should define the “dimensions”, “measured.dim” and “amount” parameters to form the structure of cluster data. This package includes functions¹³ that automatically analyzes the input cluster

¹² <https://github.com/okgreece/Cluster.OBeu>

¹³ <https://github.com/okgreece/Cluster.OBeu/blob/master/R/cl.analysis.r>

data. The clustering algorithm, the number of clusters and the distance metric of the clustering model are set to the best selection using internal and stability measures. The end user can also interact with the cluster analysis and these parameters by specifying the “cl.method”, “cl.num” and “cl.dist” parameters respectively.

The final returns are the parameters needed for visualizing the cluster data depending on the selected algorithm and the specification parameters, as long as some comparison measure matrices.

The main components of the result are a list that consists of:

- cl.meth - Label of the clustering algorithm.
- clust.num - The number of clusters.
- data.pca - The principal components to visualize the input data.
- modelparam - The results of this parameter depend of the selected clustering model.

Example of usage

The user should load the library, specify the function “open_spending.ts” in the endpoint and the method as “POST”.

The next step is to specify the name of the parameters and their values, described before.

Figure 11: OpenCPU Server - Cluster Analysis Input Example

The example in the following figure includes these inputs:

Table 5: Input params for Figure 12

Param Name	Param Value
------------	-------------

json_data	"http://ws307.math.auth.gr/rudolf/public/api/3/cubes/budget-kalamaria-expenditure-2016_87f97/aggregate?drilldown=budgetPhase.prefLabel%7CadministrativeClassification.prefLabel&aggregate_s=amount.sum&pagesize=150"
dimensions	"administrativeClassification.prefLabel"
amounts	"amount.sum"
measured.dim	"budgetPhase.prefLabel"
cl.method	"pam"

The HTTP request options return the results in the right panel “/ocpu/tmp/x093a4b4254/R/.val”. The option with “.val” includes the desired results.

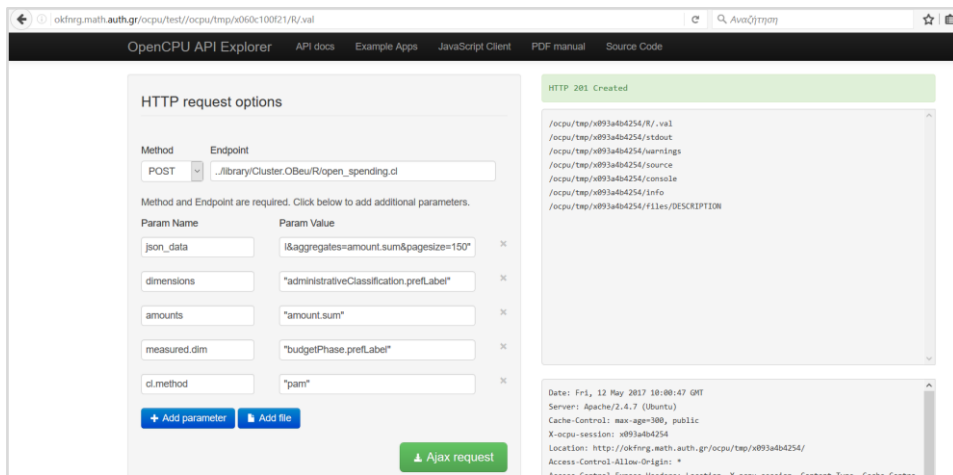


Figure 12: OpenCPU Server - Cluster Analysis Request Example

The output of this process are the parameters described before in JSON format, that will be used further and consist the inputs for visualizations.

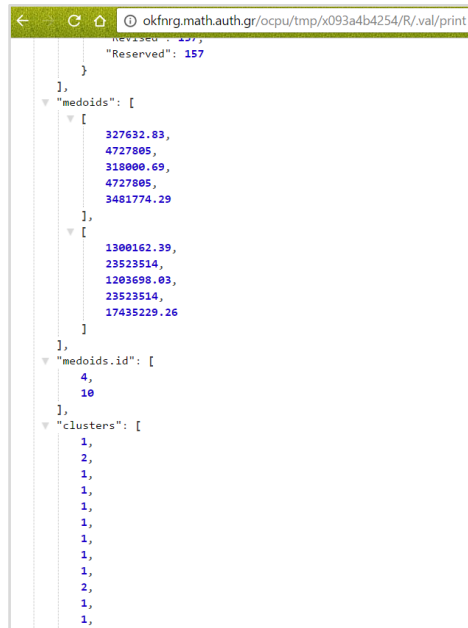


Figure 13: OpenCPU Server - Snapshot of Time Series Output Example

4.3 Outlier_dm Lib - Fraunhofer Server

The main server with DAM (Fraunhofer server) also processes data mining requests locally. The `outlier_dm`¹⁴ package is used for outlier-detection based on LOF¹⁵. This data mining tool developed following Fleischhacker, et al. (2014) was described in Deliverable D2.4. We describe its interface as follows:

4.3.1 Input

Input must be a CSV file: the first row is dimension names, the second row indicates which column is the target, from the third row is the observations, as illustrated in following figure:

¹⁴ https://github.com/openbudgets/outlier_dm

¹⁵ LOF - Local Outliers Factors based on Subpopulation


```

data — less Kilkis_neu.csv — 80x24
year , adminClass , economicClass , budgetPhase , sum ,
nom,nom,nom,nom,target,
2013 , 25 , 6061 , executed , 0.0 ,
2014 , 40 , 6243 , approved , 0.0 ,
2013 , 30 , 7112 , executed , 0.0 ,
2014 , 50 , 6253 , approved , 0.0 ,
2015 , 50 , 6252 , approved , 0.0 ,
2015 , 40 , 6242 , approved , 0.0 ,
2014 , 45 , 6114 , approved , 0.0 ,
2014 , 70 , 6266 , executed , 0.0 ,
2014 , 80 , 8222 , draft , 1400.0 ,
2014 , 30 , 6233 , approved , 0.0 ,
2015 . 30 . 6232 . approved . 0.0 .
    
```

Figure 14: Structure of the input CSV to outlier-detection based on LOF

The first row is the dimensions of the dataset, the second indicates 'sum' is the target, from the third row is the observations.

4.3.2 Main Function

The main function is:

```

detect_outliers_subpopulation_lattice(filename, output='Result',
output_path = '', full_output=False, delimiter=',', quotechar='|',
limit=25000, outlier_method='Outlier_LOF', min_population_size=30,
threshold=3, threshold_avg=3, num_outliers=25, k=5)
    
```

The meaning of the main parameters are described in Table 6.

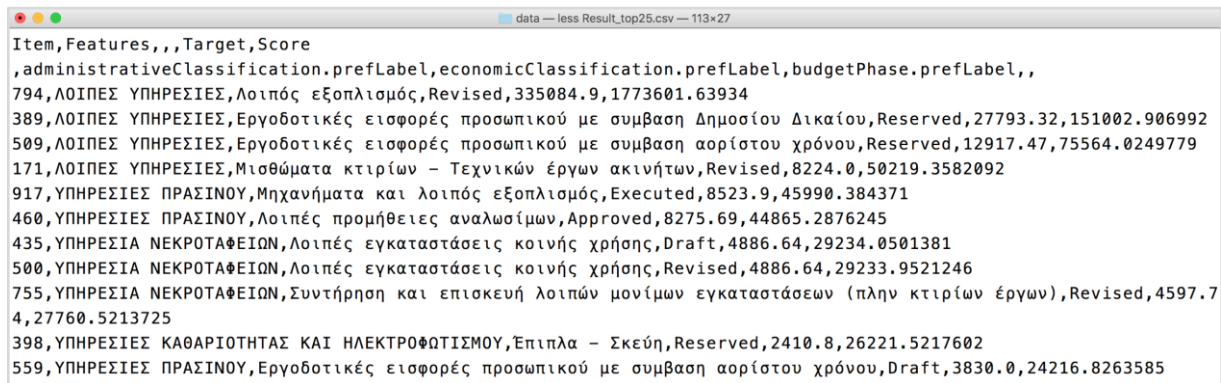
Table 6: The meaning of the main parameters of the main function of outlier detection based LOF

Parameter	Description	Default
outlier_method	Specifies which outlier method to use on the vertices of the lattice (1D outlier step on target attribute).	'Outlier_LOF'
min_population_size	Minimum population size for a subpopulation in the lattice.	30
threshold	Threshold for outputting an item as outlier (in a subpopulation).	3
threshold_avg	Threshold for outputting an item as outlier (average outlier score). In lattice structure, each sub population has different scores	1.8
num_outliers	Output the top number outliers (user want 25 outliers)	25

k	LOF method specific parameter specifying the number of neighbors used to calculate the local densities.	5
output	file name of the output	'Result'
output_path	path of the output file	' '

4.3.3 Output

Output is also a CSV file. Its first row indicates whether a column is an 'item' or 'feature', or 'target', or the outlier score. The second row indicates the corresponding name of a feature. From the third is the observations. Sample output structure is illustrated in Figure 15.



```

data -- less Result_top25.csv -- 113x27
Item,Features,,Target,Score
,administrativeClassification.prefLabel,economicClassification.prefLabel,budgetPhase.prefLabel,,
794,ΛΟΙΠΕΣ ΥΠΗΡΕΣΙΕΣ,Λοιπός εξοπλισμός,Revised,335084.9,1773601.63934
389,ΛΟΙΠΕΣ ΥΠΗΡΕΣΙΕΣ,Εργοδοτικές εισφορές προσωπικού με σύμβαση Δημοσίου Δικαίου,Reserved,27793.32,151002.906992
509,ΛΟΙΠΕΣ ΥΠΗΡΕΣΙΕΣ,Εργοδοτικές εισφορές προσωπικού με σύμβαση αορίστου χρόνου,Reserved,12917.47,75564.0249779
171,ΛΟΙΠΕΣ ΥΠΗΡΕΣΙΕΣ,Μισθώματα κτιρίων - Τεχνικών έργων ακινήτων,Revised,8224.0,50219.3582092
917,ΥΠΗΡΕΣΙΕΣ ΠΡΑΣΙΝΟΥ,Μηχανήματα και λοιπός εξοπλισμός,Executed,8523.9,45990.384371
460,ΥΠΗΡΕΣΙΕΣ ΠΡΑΣΙΝΟΥ,Λοιπές προμήθειες αναλωσίμων,Approved,8275.69,44865.2876245
435,ΥΠΗΡΕΣΙΑ ΝΕΚΡΟΤΑΦΕΙΩΝ,Λοιπές εγκαταστάσεις κοινής χρήσης,Draft,4886.64,29234.0501381
500,ΥΠΗΡΕΣΙΑ ΝΕΚΡΟΤΑΦΕΙΩΝ,Λοιπές εγκαταστάσεις κοινής χρήσης,Revised,4886.64,29233.9521246
755,ΥΠΗΡΕΣΙΑ ΝΕΚΡΟΤΑΦΕΙΩΝ,Συντήρηση και επισκευή λοιπών μονίμων εγκαταστάσεων (πλην κτιρίων έργων),Revised,4597.7
4,27760.5213725
398,ΥΠΗΡΕΣΙΕΣ ΚΑΘΑΡΙΟΤΗΤΑΣ ΚΑΙ ΗΛΕΚΤΡΟΦΩΤΙΣΜΟΥ,Έπιπλα - Σκεύη,Reserved,2410.8,26221.5217602
559,ΥΠΗΡΕΣΙΕΣ ΠΡΑΣΙΝΟΥ,Εργοδοτικές εισφορές προσωπικού με σύμβαση αορίστου χρόνου,Draft,3830.0,24216.8263585

```

Figure 15: Sample of the output CSV of the outlier detection based on LOF

5 Conclusion and Future Work

In this deliverable we summarized the information about interfaces of the data mining tools developed for data mining and analysis described in deliverable D2.4.

The main interface for integration into the OpenBudgets.eu infrastructure is the interface of DAM (Data Analysis and Mining) - described in chapter [2](#). The API endpoint of DAM is available at: <http://dam-obeu.iais.fraunhofer.de/>

The data mining tools are available not only using the DAM API, but also using the proprietary interfaces with more (specific) functionality. The tools are currently running on three servers - UEP EasyMiner server, OKFGR OpenCPU server and the main Fraunhofer DAM server - described in chapter [4](#). This chapter contains description of external as well as internal APIs available for developers. Some data mining packages written for the R system are also described.

For some data mining algorithms it is necessary to discretize the data values. The discretization developed for OpenBudgets.eu is available as part of EasyMiner (for rule or outlier detection algorithms) or as discretization using SPARQL - described in chapter [3](#).

In the future development, the integration with visualisation tools will be completed. The data mining and data analysis tools should be also tested within the Large Scale Trials.

References

Fayyad, U. M. and Irani, K. B. (1992), On the handling of continuous-valued attributes in decision tree generation, *Machine learning* 8, pp. 87-102.

Fleischhacker, et al. (2014). Paulheim, H. Bryl, Völker, J., Bizer, Ch. Detecting Errors in Numerical Linked Data using Cross-Checked Outlier Detection. In: 13th International Semantic Web Conference, pp 357-372, Riva del Garda, Italy, October, 2014. Proceedings, Part I, pp. 19-23.

Liu B. et al. (1998), Hsu, W. and Ma, Y., Integrating classification and association rule mining, in: *KDD'98: Proceedings of the fourth international conference on Knowledge Discovery and Data mining*, pp. 80-86.

OpenBudgets.EU (2016), Deliverable D2.3 - Requirements for Statistical Analytics and Data Mining, <https://drive.google.com/file/d/0Bx9zPWPFoVRaR2Rb1I2d21rZTQ/view>

OpenBudgets.EU (2017), Deliverable D2.4 - Data Mining and Statistical Analytics Techniques, <https://drive.google.com/file/d/0Bx9zPWPFoVRaYIY0YktQNVFmdkU/view>

Source Codes of Described Software

<https://github.com/openbudgets/DAM>

<https://github.com/okgreece/DescriptiveStats.OBeu>

<https://github.com/okgreece/TimeSeries.OBeu>

<https://github.com/okgreece/Cluster.OBeu>

<https://github.com/KIZI/EasyMiner>

<https://github.com/jindrichmynarz/discretize-sparql>

<https://github.com/jaroslav-kuchar/fpmoutliers>

https://github.com/openbudgets/outlier_dm