DELIVERABLE REPORT D4.3

nQSAR Modelling infrastructure

GRANT AGREEMENT:	604134
ACRONYM:	eNanoMapper
NAME:	eNanoMapper - A Database and Ontology Framework for Nanomaterials Design and Safety Assessment
PROJECT COORDINATOR:	Douglas Connect GmbH
START DATE OF PROJECT; DURATION:	1 February 2014; 36 months
PARTNER(s) RESPONSIBLE FOR THIS DELIVERABLE:	NTUA
DATE:	31.7.2015
VERSION:	[V.4.0.]

Call identifier	FP7-NMP-2013-SMALL-7
Document Type	Deliverable Report
WP/Task	WP4 /Task3
Document ID	eNanoMapper D4.3
Status	Final

Partner Organisations	• Douglas Connect, GmbH (DC) • National Technical University of Athens (NTUA) • In Silico Toxicology (IST) • Ideaconsult (IDEA) • Karolinska Institutet (KI) • VTT Technical Research Centre of Finland (VTT) • European Bioinformatics Institute (EMBL-EBI) • Maastricht University (UM) • Misvik Biology Oy (MB)
Authors	Charalambos Chomenidis Philip Doganis George Drakakis Georgia Tsiliki Haralambos Sarimveis Final review and edit by Barry Hardy
Purpose of the Document	To report on the progress of updating and extending statistical and machine learning methods and OpenTox services, for the development of nQSAR models.
Document History	1.Table of Contents, 10/03/2015 2.First draft, 02/05/2015 3. Second draft, 30/06/2015 4. Third draft, 16/07/2015

TABLE OF CONTENTS

TABLE OF FIGURES

GLOSSARY

Abbreviation / acronym	Description
1D	1-dimensional
3D	3-Dimensional
ANOVA	Analysis of Variance
API	Application Programming Interface
ARFF	Attribute-Relation File Format
BP	Biological Processes
CC	Cellular Components
CSV	Comma-separated values
CV	Cross-Validation
DFT	Density Functional Theory
DoA	Domain of Applicability
EJB	Enterprise JavaBeans
ENET	Elastic Net regression
ENM(s)	Engineered Nanomaterial(s)
FFNN	Feed Forward Neural Network
GLM	Generalized Linear Model with Stepwise Feature Selection
GNU GPL	GNU General Public License
GO	Gene Ontology
GPW	Gaussian and plane waves
GSEA	Gene Set Enrichment Analysis
GUI	Graphical user interface
HC	Hierarchical Clustering
HF	Hartree-Fock
HOMO	Highest Occupied Molecular Orbital
HTTP	Hypertext Transfer Protocol
ID3	Iterative Dichotomiser 3
in	K Nearest Neighbour
JPDI	Jaqpot Protocol of Data Interchange
JQ	Jaqpot Quattro
JSON	JavaScript Object Notation
KEGG	Kyoto Encyclopaedia of Genes and Genomes
KS-DFT	Kohn-Sham density functional theory
LASSO	Least Absolute Shrinkage and Selection Operator
LC-MS/MS	Liquid chromatography–Mass Spectrometry/ Mass Spectrometry
LM	Linear Model
LOO	Leave-One-Out
LOOCV	Leave one out cross validation
LUMO	Lowest Unoccupied Molecular Orbital
MeOx	Metal oxides
MF	Molecular Functions
MLR	Multiple Linear Regression
MS	Mass Spectrometry

MSigDB	Molecular Signature Databases
NanoQSAR , nQSAR	Nano Quantitative Structure Activity Relationship
NPs	Nanoparticles
OLS	Ordinary Least Squares
OpenTox	http://www.opentox.org/
PLS	Partial Least Squares
PMML	Predictive Model Markup Language
PubMed	http://www.ncbi.nlm.nih.gov/pubmed
QM	Quantum Mechanics
QSAR	Quantitative Structure Activity Relationship
RBF	Radial Basis Function
REST	Representational State Transfer
RF	Random Forest
RFE	Recursive Feature Elimination
RMSE	Root Mean Square Error
RSS	Rich Site Summary
SVM	Support Vector Machines
SVR	Support Vector Regression
TEM	Transmission Electron Microscopy
UniProt	Universal Protein Database
URI	Uniform Resource Identifier
VIP	Variable Importance in Projection
WS	Web Services
XML	Extensible Markup Language

1. EXECUTIVE SUMMARY

This deliverable describes the nQSAR Modelling infrastructure developed within Work Package 4 to support eNanoMapper modelling activities. First, we describe the novelties in the Application Programming Interface (API) that allow seamless integration of any web service or algorithm, independently of architecture or language of algorithm, with the sole condition that it complies with the Jaqpot API. Second, a complete walkthrough of the Jaqpot modelling workflow is provided, from experimental data bundle to dataset, models and predictions, followed by a presentation of the modelling algorithms that have been made available from R, Python and WEKA. Finally, we describe the RRegrs package for computer-aided model selection that gives access to standard and individually implemented methodologies utilized within the eNanoMapper computational infrastructure.

2. INTRODUCTION

Deliverable 4.3 describes how the OpenTox modelling resources have been extended to reflect the required adequate description of ENMs and integrate statistical and machine learning (ML) algorithms, able to exploit and integrate diverse data and metadata captured in the database warehouse of the project including images, high throughput screening and omics data. Such an extension was deemed necessary, as the OpenTox algorithm and modelling API originated as chemical structure-centric, and requires clean datasets in a specific form, while the eNanoMapper prototype database is explicitly designed to handle all peculiarities of experimental data, including replicates, range and error values.

The approach taken by eNanoMapper, resulted in the Jaqpot Quattro (JQ) web application, the API documentation of which can be found at http://app.jaqpot.org:8080/jaqpot/swagger. JQ is an extension of the Jaqpot web application, which was originally developed during the OpenTox project (Hardy et. al., 2010) and features improved efficiency and additional functionality. JQ is an open-source project, written in Java and licensed with the GNU GPL v3 license. It provides asynchronous execution of tasks submitted by users, as well as authentication and authorisation. JQ is part of the eNanoMapper framework and communicates with other web services in the framework via the common REST API described. The source code is publicly available from https://github.com/KinkyDesign/JaqpotQuattro.

Section 3 of this report describes the architecture and functionalities of the JQ web application and focuses on the following main features: Producing data sets suitable for modelling, data preprocessing procedures, APIs for dynamic algorithm integration, integration with third-party services and algorithm implementations. A full demonstration of the JQ modelling workflow is given section 4, with the help of the Swagger interface for visualization of the functionality of the web services. Section 5 describes the RRegrs (R Regressions) R package (Tsiliki et al., 2015), that has been developed in the context of the eNanoMapper project. RRegrs is a standardized framework that automates the development of a reliable and well-validated QSAR model or set of models.

3. THE JAQPOT QUATTRO WEB APPLICATION

JQ is a web application that currently supports data preprocessing, as well as statistical, data mining and machine learning algorithms and methods for defining the Domain of Applicability of a predictive nQSAR model. JQ has been developed following the extended OpenTox APIs that have been presented in detail in Deliverable 4.1. As already mentioned in the introduction, the JQ web services API has been documented using the Swagger framework. A list of REST endpoints is presented to the end-user; these correspond to the main entities/resources of eNanoMapper: datasets, models, algorithms, BibTeX entities, asynchronous tasks and more. The user can click on any of these endpoints to get a list of the available operations related to each entity. A screenshot of the JQ API is shown in Fig.1. The main features of the JQ web application are presented in the following subsections.

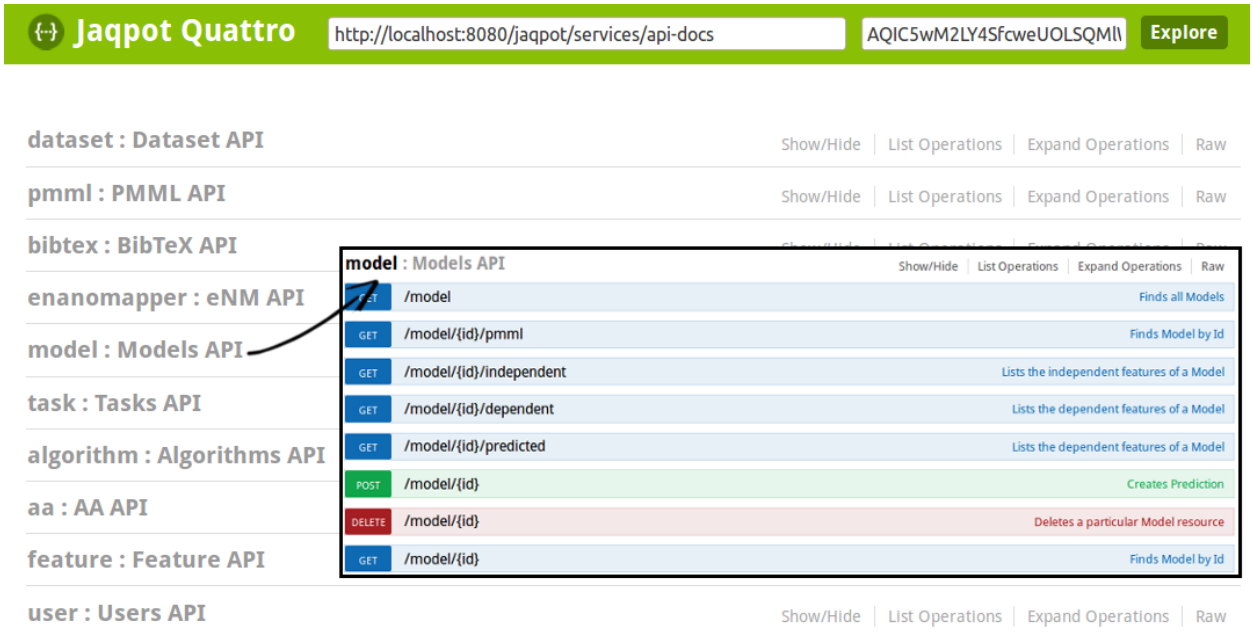


Figure 1 Screenshot of the JQ modelling web services API

3.1 PRODUCING DATASETS SUITABLE FOR MODELLING

JQ algorithm services require input data in a standardized format in order to generate a predictive model. Therefore, raw experimental data cannot be used directly for modelling purposes. The experimental data are, more often than not, heterogeneous by nature and properly structuring these is not a trivial task. To this end, a web service acting as a link between experimental data and data for modelling was introduced, which will be hereafter referred to as the ***Conjoiner service***. This service performs the task of transforming the experimental data into a modelling-friendly format, and producing standardized datasets as specified in the OpenTox API. One can initiate a conjoiner service operation by specifying a bundle URI. A bundle is an eNanoMapper resource that acts as an assortment of experimental effects, images and molecular structures, for nanomaterials, and the Conjoiner service's job is to combine all that disparate data into a dataset suitable to be fed to an algorithm service. Regarding experimental data, multiple individual measurements, interval-valued measurements (lower

and upper values), or values accompanied by a standard measurement error, may be included for the same endpoint in a bundle. These need to be aggregated into a single value, which is currently carried out by taking the average value of all experimental measurements having excluded outliers identified by a Dixon's q-test (Rorabacher, 1991), but different aggregation procedures can be implemented based on more elaborate outlier detection criteria and rejection/aggregation schemata. When bundles contain links to raw data of nanomaterials such as images, crystallographic data and proteomics data, the descriptor calculation web services described in Deliverable 4.2 are activated to produce the calculated descriptors. The overall procedure is illustrated in Figure 2. Data as heterogeneous as chemical structures, raw experimental measurements, spectra and microscopy images can be combined by the conjoiner service to produce a dataset for modelling purposes.

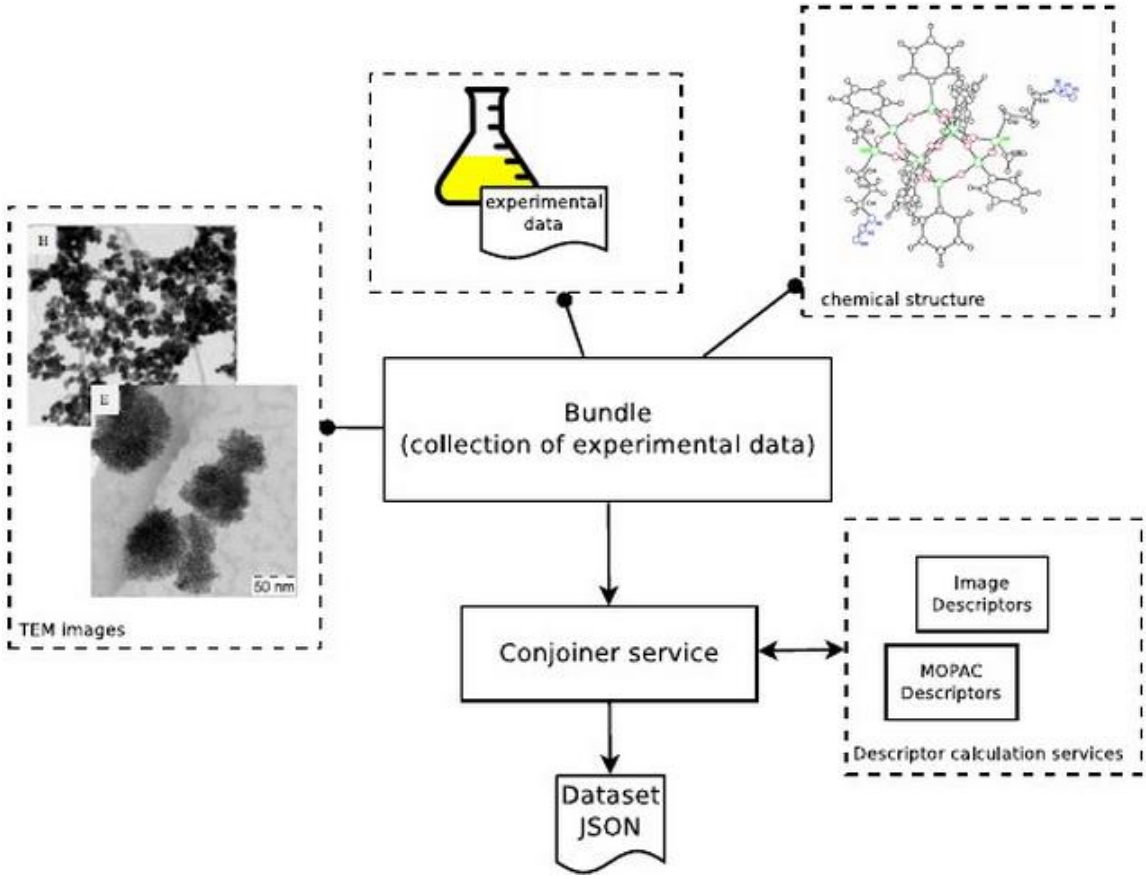
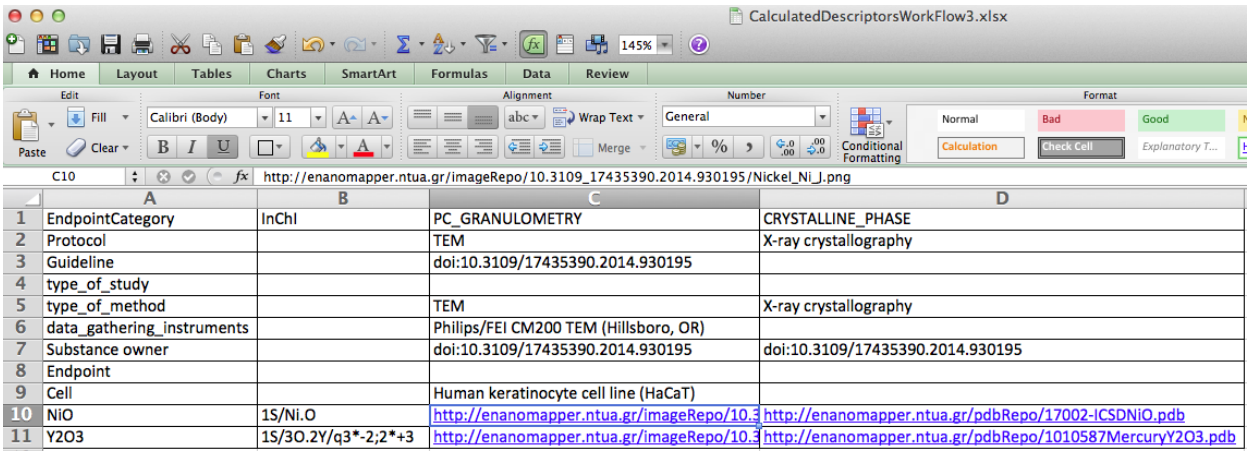


Figure 2 Conjoiner API: modelling-oriented information can be extracted from bundles of experimental data.

Here follows an example, where a dataset used for modelling is created from a spreadsheet that does not contain any numerical value, but only links to crystallographic data files and image files for two metal oxide nanoparticles: NiO and Y_2O_3. The example spreadsheet file in .xlsx format is shown in Figure 3. Cells C10:D11 contain links to private repositories where we store images and crystallographic (PDB) files, although the links could point to any internet location. The spreadsheet file is uploaded to the eNanoMapper database, by visiting the following address:
https://apps.ideaconsult.net/enmtest/ui/uploadsubstance1



	A	B	C	D
1	EndpointCategory	InChI	PC_GRANULOMETRY	CRYSTALLINE_PHASE
2	Protocol		TEM	X-ray crystallography
3	Guideline		doi:10.3109/17435390.2014.930195	
4	type_of_study			
5	type_of_method		TEM	X-ray crystallography
6	data_gathering_instruments		Philips/FEI CM200 TEM (Hillsboro, OR)	
7	Substance owner		doi:10.3109/17435390.2014.930195	doi:10.3109/17435390.2014.930195
8	Endpoint			
9	Cell		Human keratinocyte cell line (HaCaT)	
10	NiO	1S/Ni.O	http://enanomapper.ntua.gr/imageRepo/10.3	http://enanomapper.ntua.gr/pdbRepo/17002-ICSDNiO.pdb
11	Y2O3	1S/3O.2Y/q3*-2;2*+3	http://enanomapper.ntua.gr/imageRepo/10.3	http://enanomapper.ntua.gr/pdbRepo/1010587MercuryY2O3.pdb

Figure 3 Input spreadsheet for Crystallographic files and TEM images of NiO and Y_2O_3

In order for the eNanoMapper system to be able to comprehend the file described above and alter the input provided by the user to the appropriate category, the user needs to upload a JSON file that contains the mapping for the .xlsx file, which is presented on the next page:

```
{  "TEMPLATE_INFO": {
     "NAME": "CalculatedDescriptorsWorkFlow",
     "VERSION": "original",
     "TYPE": 1   },
   "DATA_ACCESS": {
     "ITERATION": "ROW_SINGLE",
     "SHEET_INDEX": 1,
     "START_ROW": 10,
"END_ROW": 11,
     "START_HEADER_ROW": 1,
     "END_HEADER_ROW": 9,
     "ALLOW_EMPTY": true,
     "RECOGNITION": "BY_INDEX"   },
   "SUBSTANCE_RECORD": {
     "COMPANY_UUID": {
       "COLUMN_INDEX": "A"      },
     "PUBLIC_NAME": {
       "COLUMN_INDEX": "A"      },
     "OWNER_NAME": {
       "ITERATION": "ABSOLUTE_LOCATION",
       "COLUMN_INDEX": "C",
       "ROW_INDEX": 7      },
     "SUBSTANCE_TYPE": {
       "ITERATION": "JSON_VALUE",
       "JSON_VALUE": "Nanoparticle"      },
     "REFERENCE_SUBSTANCE_UUID": {
       "COLUMN_INDEX": "B"      },
     "COMPOSITION": [         {
         "STRUCTURE_RELATION": "HAS_CORE",
         "CONTENT": {
           "COLUMN_INDEX": "B"           },
         "FORMAT": {
           "ITERATION": "JSON_VALUE",
           "JSON_VALUE": "INC"           },
         "InChI": {
           "COLUMN_INDEX": "B"           }        }     ]   },
   "PROTOCOL_APPLICATIONS": [
     {    "CITATION_TITLE": {
         "ITERATION": "ABSOLUTE_LOCATION",
         "COLUMN_INDEX": "C",
         "ROW_INDEX": 3        },
"CITATION_TITLE": {
         "ITERATION": "JSON_VALUE",
         "JSON_VALUE":
"http://dx.doi.org/10.3109/17435390.2014.930195" },
       "CITATION_YEAR": {
         "ITERATION": "JSON_VALUE",
         "JSON_VALUE": "2014"        },
       "CITATION_OWNER": {
         "ITERATION": "JSON_VALUE",
         "JSON_VALUE": "eNanoMapper"        },
       "PROTOCOL_CATEGORY_CODE": {
         "ITERATION": "ABSOLUTE_LOCATION",
         "COLUMN_INDEX": "C",
         "ROW_INDEX": 1        },
       "PROTOCOL_GUIDELINE": {
         "guideline1": {
           "ITERATION": "ABSOLUTE_LOCATION",
           "COLUMN_INDEX": "C",
           "ROW_INDEX": 2          }        },
       "PARAMETERS": {
         "type_of_method": {
           "ITERATION": "ABSOLUTE_LOCATION",
           "COLUMN_INDEX": "C",
           "ROW_INDEX": 5          },
         "data_gathering_instruments": {
           "ITERATION": "ABSOLUTE_LOCATION",
           "COLUMN_INDEX": "C",
           "ROW_INDEX": 6          }        },
       "EFFECTS": [
         {    "ENDPOINT": {
             "ITERATION": "JSON_VALUE",
             "JSON_VALUE": "IMAGE"             },
           "TEXT_VALUE": {
             "COLUMN_INDEX": "C"             },
           "CONDITIONS": {
             "Cell": {
               "ITERATION": "ABSOLUTE_LOCATION",
               "COLUMN_INDEX": "C",
               "ROW_INDEX": 9}    }    }        ]     },
     {    "CITATION_TITLE": {
         "ITERATION": "ABSOLUTE_LOCATION",
         "COLUMN_INDEX": "D",
         "ROW_INDEX": 3        },
       "CITATION_YEAR": {
         "ITERATION": "JSON_VALUE",
         "JSON_VALUE": "2014"        },
       "CITATION_OWNER": {
         "ITERATION": "JSON_VALUE",
         "JSON_VALUE": "eNanoMapper"
       },
       "PROTOCOL_CATEGORY_CODE": {
         "ITERATION": "ABSOLUTE_LOCATION",
         "COLUMN_INDEX": "D",
         "ROW_INDEX": 1        },
       "PROTOCOL_GUIDELINE": {
         "guideline1": {
           "ITERATION": "ABSOLUTE_LOCATION",
           "COLUMN_INDEX": "D",
           "ROW_INDEX": 2          }        },
       "PARAMETERS": {
         "type_of_method": {
           "ITERATION": "ABSOLUTE_LOCATION",
           "COLUMN_INDEX": "D",
           "ROW_INDEX": 5          }        },
       "EFFECTS": [
         {    "ENDPOINT": {
             "ITERATION": "JSON_VALUE",
             "JSON_VALUE": "PDB_CRYSTAL_STRUCTURE"
           },
           "TEXT_VALUE": {
             "COLUMN_INDEX": "D"
           }          }        ]     }   ]
}
```


The mapping provided in the JSON file provides the system with an association of the cells in the spreadsheet and the eNanoMapper database fields whose values should be input. The upload process leads to this dataset:

https://apps.ideaconsult.net/enmtest/substanceowner/XLSX-021e69c9-4fa3-3e1e-b2b7-871ceab93347/dataset

which is depicted in Figure 4, where the names of the imported metal oxides, the owner of the data (DOI of the publication), and the links for the images and crystallographic data are given.

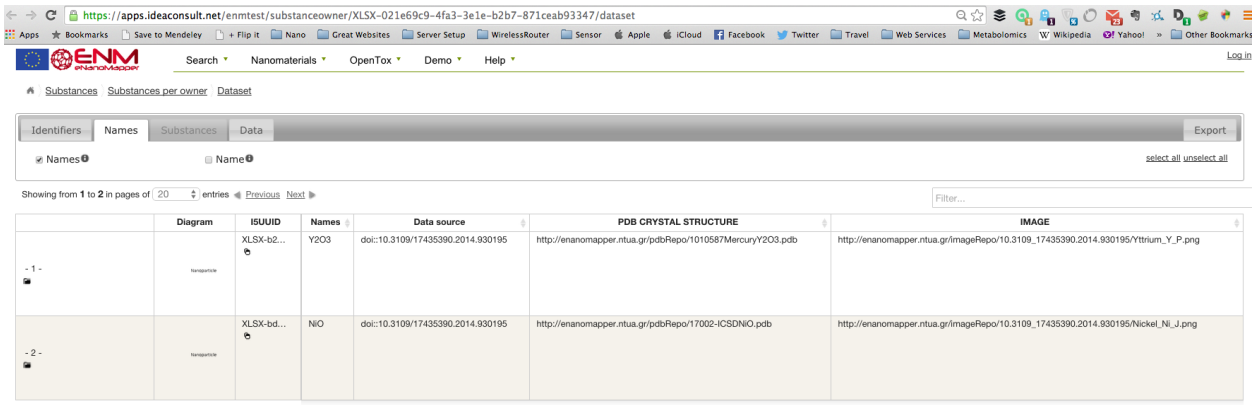


Figure 4 The uploaded dataset with links for images and PDBs.

Visiting the Create Bundle operation of the Swagger interface (via selection of the menu, or directly accessing http://app.jaqpot.org:8080/jaqpot/swagger/#!/enm/createBundle) and adding in the body the URI for the relevant substance owner (https://apps.ideaconsult.net/enmtest/substanceowner/XLSX-021e69c9-4fa3-3e1e-b2b7-871ceab93347), as shown in Figure 5, we get a resulting bundle URI, namely https://apps.ideaconsult.net/enmtest/bundle/81.

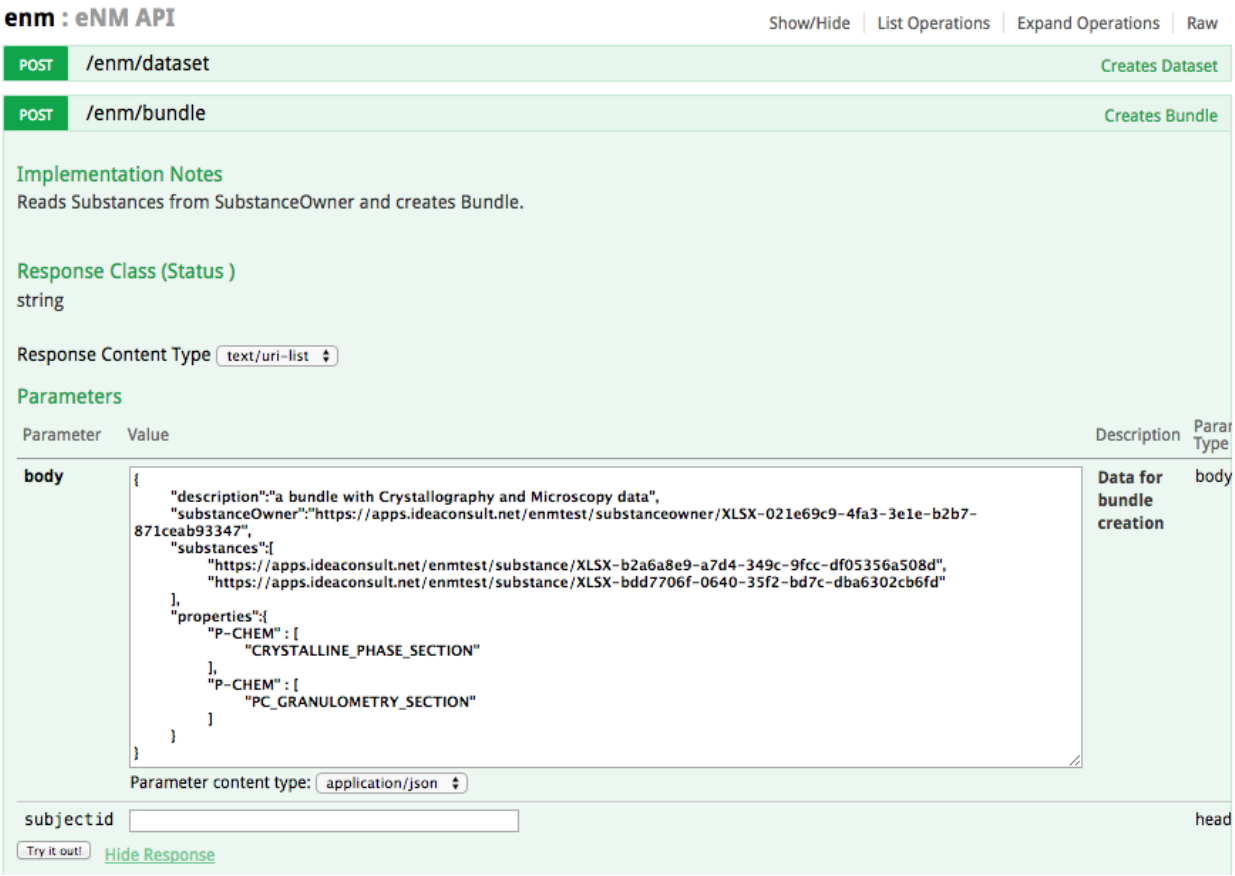


Figure 5 Create Bundle operation

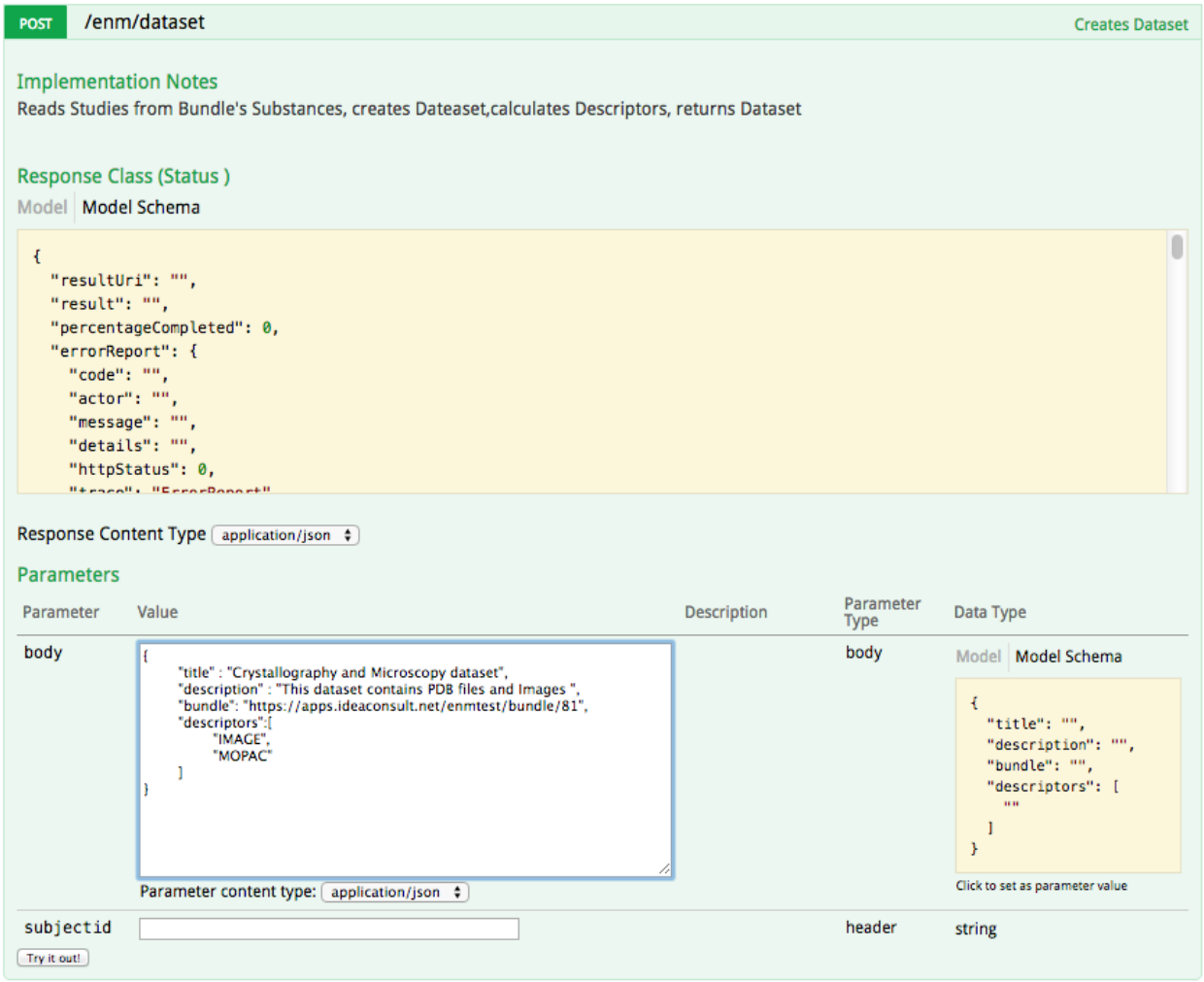


Figure 6 Dataset creation from bundle with the command to calculate Image and MOPAC descriptors.

Subsequently, we visit the Create Dataset operation, in Swagger: http://app.jaqpot.org:8080/jaqpot/swagger/#!/enm/createDataset. After input of the body text shown in Figure 6, we request a dataset to be created using the bundle we provided as input and also for Image and MOPAC descriptors to be calculated and added to the dataset. The result of this action is this dataset: http://app.jaqpot.org:8080/jaqpot/services/dataset/3MOwEZvFhnQ6.

We can observe in the following JSON representation of the aforementioned dataset that there are two groups of features in the "values" field: the first group is that of the image-derived descriptors, whose URIs begin with http://app.jaqpot.org:8080/jaqpot/services/feature/image+ and continue with the descriptor name. For instance, the URI that represents the particle area for the “average particle”, a virtual particle that contains the average values for all recognized particles is as follows: http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+area. The second group is that of the MOPAC derived descriptors, like the Electronic Energy, defined by the feature https://apps.ideaconsult.net/enmtest/feature/62.


```
{
 meta:
     {
      comments:
          [
           "Created by task TSKFzSxVqUStVCy"
          ],
      descriptions:
          [
```

```
            "This dataset contains PDB files and Images "
          ],
      titles:
          [
           "Crystallography and Microscopy dataset"
          ],
      creators:
          [
           "filipposd"
          ],
      hasSources:
          [
           "https://apps.ideaconsult.net/enmtest/bundle/81"
          ]
     },
dataEntry:
     [
      {
           compound:
               {
                URI: "https://apps.ideaconsult.net/enmtest/substance/XLSX-b2a6a8e9-a7d4-349c-
                9fcc-df05356a508d"
               },
           values:
               {
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+angle:
                149.23083,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+area: 8
                803,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+area_fr
                action: 43.979816,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+aspect_
                ratio: 1.3006951,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+circula
                rity: 0.34530884,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+feret:
                200.14246,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+feret_a
                ngle: 136.01219,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+feret_x
                : 0,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+feret_y
                : 0,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+integra
                ted_density: 827776,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+kurtosi
                s: -0.017311413,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+major:
                120.74194,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+max_gre
                y_value: 130,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+mean_gr
                ey_value: 94.0334,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+min_fer
                et: 139,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+min_gre
                y_value: 6,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+minor:
                92.82878,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+perimet
                er: 566,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+raw_int
                egrated_density: 827776,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+roundne
                ss: 0.7688197,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+skewnes
                s: -0.6812224,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+solidit
                y: 1,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+spheric
                ity: 0.0006815858230148812,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+std_dev
                : 24.328926,
                http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+surface
                Diameter: 52.94811785808232,
```

```
                    http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+volume:
                     0,
                    http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+volumeD
                    iameter: 0,
                    http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+volumeT
                    oSurface: 0,
                    http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+x_cente
                    r_of_mass: 67.20986,
                    http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+x_centr
                    oid: 66.68812,
                    http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+y_cente
                    r_of_mass: 49.1137,
                    http://app.jaqpot.org:8080/jaqpot/services/feature/image+average+particle+y_centr
                    oid: 45.340168
                   }
        },
…
```


3.2 API FOR DYNAMIC ALGORITHM INTEGRATION AND NANO-QSAR MODEL DEVELOPMENT

The Jaqpot Protocol of Data Interchange, in short JPDI, is a new feature of the JQ web services that allows developers of machine learning algorithms to integrate their implementations in the framework. This integration requires little engagement with intricate software development and allows algorithm developers to outsource their implementations and make them available to the nanomaterial design community through the eNanoMapper framework. The communication between eNanoMapper services and third-party JPDI services is carried out by exchanging JSON documents that contain no more information than what a modelling service needs to train a predictive model, calculate descriptors, perform a prediction, evaluate the domain of applicability of a model or perform other tasks. This is illustrated in Figure 7. Once a developer (possibly third-party) has prepared a JPDI-compliant web service, he/she needs to register it with the eNanoMapper framework by specifying (i) the name of the algorithm, (ii) meta-data for the algorithm, such as a description, tags, copyright notice and any other meta-data supported by the Dublin core ontology (http://dublincore.org/) and/or the OpenTox ontology (Tcheremenskaia et. al. 2012), (iii) the URI of their implementation to be used as an endpoint for training, (iv) the corresponding URI for the prediction web service, (v) an ontological characterization of the algorithm according to the OpenTox Algorithms ontology (e.g., ot:Regression or ot:Classification, or ot:Clustering; http://www.opentox.org/dev/apis/api-1.1/Algorithms). The algorithm is then registered by POSTing a JSON containing all this information to /algorithm. Once registered, the algorithm acquires a URI, and is exposed as a web service, that can be consumed by JQ.

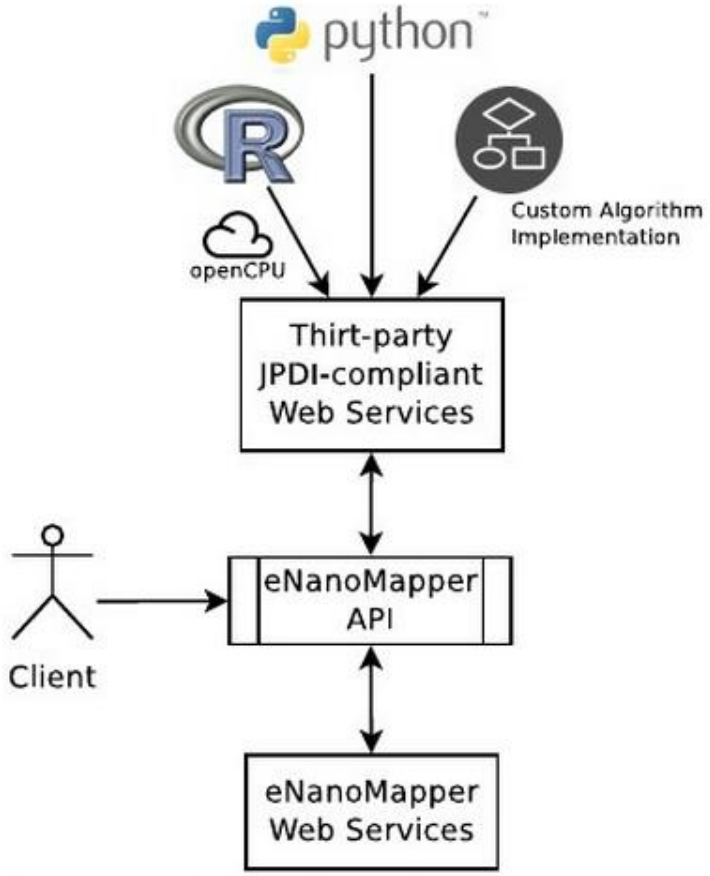


Figure 7 JPDI-compliant web services can be seamlessly incorporated into the eNanoMapper framework.

A JPDI request for training is presented below. This request is issued by an algorithm web service of eNanoMapper to a JPDI-compliant web service.

```
{
  "dataset": {
    "dataEntry": [
      {
        "compound": {
            "URI": "http://some.server.org/substance/1"
        },
        "values": {
          "http://some.server.org/property/1": 0.268,
          "http://some.server.org/property/2": 0.667,
          ...
        }
      },
      {
        "compound": {
          "URI": "http://some.server.org/substance/2"
        },
        "values": {
          "http://some.server.org/property/1": 0.115,
```

```
            "http://some.server.org/property/2": 0.759,
            ...
          }
        },
        ...
    ]
  },
  "predictionFeature": "http://some.server.org/feature/1",
  "parameters": {
      "theta": 49,
      "mvh": 1.0
  }
}
```


Once the model is trained, the JPDI service will return it to the caller in JSON format in which the actual model is encoded. Below is an example JSON response file from the JPDI service:


```
{
  "rawModel": "<BASE64>",
  "pmmlModel": "<PMML-XML>",
  "additionalInfo" : "<Extra information the algorithm service needs saved with the model>",
  "independentFeatures": [
      "http://some.server.org/property/1",
      "http://some.server.org/property/2"
  ]
}
```


Notice that the JPDI web service may select only some of the features of the initial dataset, which are defined in the PMML. Then, the JPDI service requires that a dataset containing these features be posted back to it, i.e., a JPDI service in order to perform predictions requires (i) the model it has previously produced and (ii) a dataset containing values for the features it has selected.

Upon training, the model returned to the caller is stored as-is by the called service and will be returned back to the JPDI-compliant service when the client requests a prediction. This way, as already mentioned, the JPDI service providers do not need to maintain a database, while at the same time the eNanoMapper services do not need to know how the third-party services perform computations.

Likewise, when JQ needs to consume a JPDI web service to perform predictions, it POSTs to it a JSON with (i) the input dataset containing substances and (ii) the model that was previously created by the JPDI service. An example of JSON prediction request is shown below:


```
{
 "dataset": {
  "datasetURI": ["http://some.server.org/dataset/1"],
  "dataEntry": [{
   "compound": {
    "URI": "http://some.server.org/substance/1"
          },
    "values": {
     "http://some.server.org/property/1": 0.268,
```

```
      "http://some.server.org/property/2": 0.667,
            ...
          }
          }, {
  "compound": {
   "URI": "http://some.server.org/substance/2"
          },
    "values": {
     "http://some.server.org/property/1": 0.115,
     "http://some.server.org/property/2": 0.759,
            ...
          }
        },
        ...
    ],
 "rawModel": ["<BASE64>"],
 "additionalInfo": ["<other info the JPDI algorithm service needs>"]
  }
}
```


3.3 API FOR NANO-QSAR MODEL DEVELOPMENT

Figure 8**Error! Reference source not found.** shows the interface that allows model creation by actually assembling a POST command according to the following parameters:

- ***dataset_uri***: URI of the training dataset.
- ***prediction_feature:*** URI of the end-point to be predicted.
- ***parameters:*** Algorithm-specific tuning parameters are specified in this field.
- ***scaling:*** Scaling or normalization can be applied to the training data.
- ***domain of applicability:*** An algorithm for defining the domain of applicability can be selected.
- ***transformations:*** JQ makes use of the Predictive Model Markup Language (PMML) file format that allows clients to define a data dictionary and a transformations dictionary, by providing the URI of a PMML document. The data dictionary selects a number of features out of the original dataset that will be provided as input to the modelling algorithm, while the transformation dictionary defines mathematical formulae to be applied on the selected features. Construction and use of PMML files has been described in detail in Section 3 of D4.1.
- ***id:*** The id of the desired algorithm must be entered here. As previously noted, all algorithms, even those not hosted at JQ but complying with its API can be used. A list of all available algorithms can be retrieved at http://app.jaqpot.org:8080/jaqpot/swagger/#!/algorithm/getAlgorithms.
- ***subjectid:*** Here users can supply their own token, in case the dataset is not available by the guest token, which is used by default.

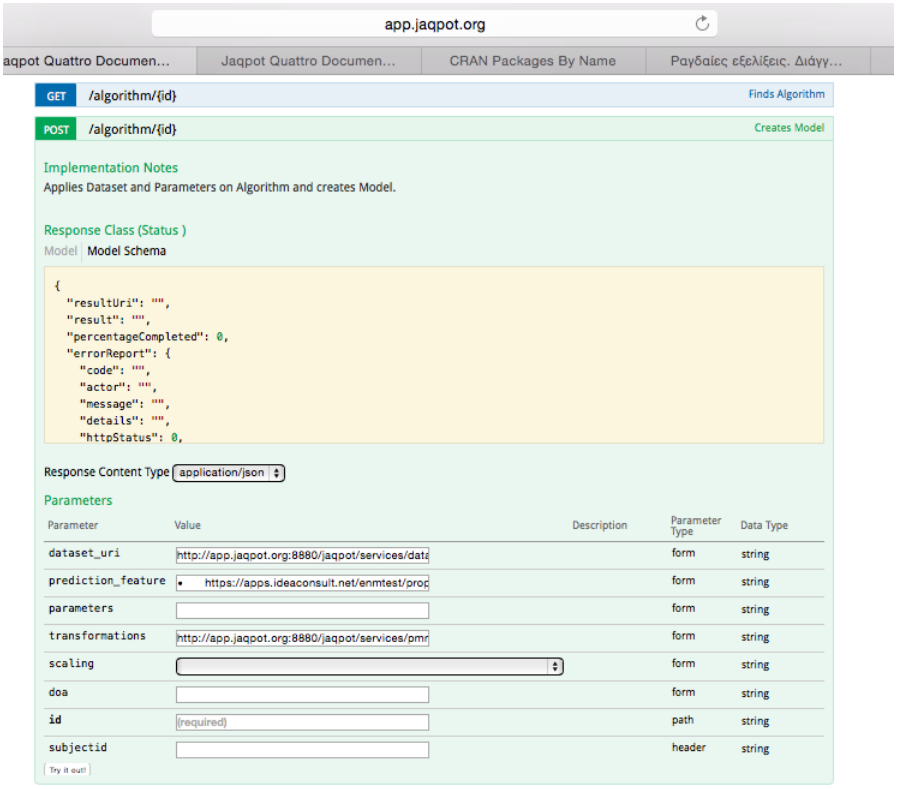


Figure 8 Creating a model using an algorithm

3.4 INTEGRATION WITH THIRD PARTY SERVICES – ALGORITHM IMPLEMENTATIONS

With the modification and extensions to the OpenTox API, all supported state-of-the-art statistical and machine learning methods can be applied to ENMs. The extension of the OpenTox modelling infrastructure now provides additional facilities for data analysis, as well as for building and validating new ENM predictive models, which may be applied to all new datasets incorporated into the eNanoMapper infrastructure. For this reason we have extensively studied the integration of the facilities provided by R, Python and WEKA which allow easy access to a wealth of additional algorithms and methods, as well as specially designed libraries for the analysis and interpretation of omics and biological data. The eNanoMapper computational infrastructure provides the means to simultaneously run numerous regression models and compare their performances, but also run individual algorithms to enable specially designed optimization for their parameter space.

The JDPI protocol allows to dynamically and seamlessly incorporate any custom algorithmic implementation into eNanoMapper without any need for resource management (i.e., the algorithm providers do not need to maintain a database system). The protocol specifies the form of data exchange between eNanoMapper services and third party algorithm web service implementations. The eNanoMapper framework already provides wrappers for WEKA (Hall et.al, 2009) the R language (R Development Core Team, 2012) and the Python language.

In addition, the leverage method for defining the *Domain of Applicability* (DoA) of NanoQSAR models has been implemented and offered as a service. The DoA is created by applying a POST to an instance of the DoA web service. Then, a predictive model can be linked to the DoA model in such a way that

predictions are accompanied by an indicator that informs the user whether the query compound is in or out of the DoA of the model.

3.4.1 Integration of R Language Algorithms into the eNanoMapper Computational Infrastructure using OpenCpu

R is a programming language and software environment (http://www.r-project.org/) that has become the most popular choice for users for statistical computing, graphics and data analysis, as shown by relevant literature and user surveys (Smith, 2015 & Piatetsky, 2015) Its power is largely owed to its cost-free structure and its community that drives constant development, update and scrutiny of R packages in a collaborative manner. R is widely used in the Bioinformatics domain and provides many alternative platforms and architectures for users in that field. Open source Bioconductor provides Bioinformatics capabilities using R (http://www.bioconductor.org/), focusing on high-throughput genomic data and making available many bioinformatics packages (1024 in June 2015), such as ChemmineR. R can also be used through the open source data analytics, reporting and integration platform KNIME (KNIME, 2013). Bioconductor is also available as images that allow users to deploy their own instances easily: an AMI (Amazon Machine Image, http://www.bioconductor.org/help/bioconductor-cloud-ami/) and a series of Docker images (http://www.bioconductor.org/help/docker/).

3.4.1.1 Service Technical Details

Integration with R is made possible through the OpenCPU (https://www.opencpu.org/) system, which defines a HTTP API for embedded scientific computing based on R, although this approach could easily be generalized to other computational back-ends (Ooms, 2014). OpenCPU acts as a wrapper to R that is readily able to expose R functions as RESTful HTTP resources. The OpenCPU server takes advantage of multi-processing in the Apache2 web server to handle concurrency. This implementation uses forks of the R process to serve concurrent requests immediately with little performance overhead. By doing so it enables access to those functions on simple HTTP calls converting R from a standalone application to a web service.

3.4.1.2 Algorithms available from CRAN Package Repository

Currently, the Comprehensive R Archive Network (CRAN) package repository is the most complete repository of algorithms for R, featuring 6784 available packages (accessed June 2015) that provide access to more than 300 regression algorithms that cover a wide array of approaches (for example, here are statistics on packages that involve some methods: 200+ Bayes, 52 Gaussian, 4 Ridge Regression, 8 Support Vector Machines), more than 84 classification packages and more than 120 packages for clustering.

3.4.1.3 R Algorithms Implemented as eNanoMapper Web Services

Tailor-made eNanoMapper R packages are now ready for Linear Regression, Lasso and Ridge Regression, and Elastic Net. Additionally, we have implemented clustering methodologies, i.e. hierarchical clustering and bi-clustering. All packages have the same structure: a function which creates the model, a predictive function to predict or produce new instances given any new data supplied, and two other auxiliary functions. In all cases we have maintained the flexibility provided by the original R packages used in terms of parameters and methodologies considered. In order to ensure that the R functions expect a ‘parameters’ argument, whenever more than one option are available, where the user can specify

methodologies or parameters of the functions based on the guidance from ours or R's help files. An example of such a help file can be found in http://147.102.82.122/ocpu/library/glmNETpkg/man/glmnet.funct/text.

Special features of the packages are their ability to accept JSON input files and produce JSON output files using the jsonlite R package (http://cran.r-project.org/web/packages/jsonlite/index.html), and also produce PMML using the pmml R package (http://cran.r-project.org/web/packages/pmml/index.html).

REGRESSION ALGORITHMS

The PLMplusPMMLpkg R package applies a linear model to the data and allows for prediction, since the model is stored in a serialized form in the system. More specifically we have wrapped the lm() and predict.lm() R functions in order to be able to accept JSON input files and produce JSON output files. The four functions of the package are:

- ***read.in.json***: reads in data in JSON format for linear modelling
- ***read.in.json.for.pred***: reads in R raw linear models in JSON format for prediction
- ***lm.funct***: performs linear modelling
- ***pred.funct***: predicts linear models supplied

The package is available at http://147.102.82.122/ocpu/library/PLMplusPMMLpkg/info and https://github.com/GTsiliki/PLMpkg, where more details can be found in the accompanied manual files, while the algorithm's web service URI is at: http://app.jaqpot.org:8080/jaqpot/services/algorithm/ocpu-lm.

The glmNETpkg R package is built on top of the glmnet R package (http://cran.r-project.org/web/packages/glmnet/index.html) which employs procedures for fitting the entire lasso and elastic-net regularization path for linear regression, logistic and multinomial regression models, Poisson regression, the Cox model, multiple-response Gaussian, and the grouped multinomial. The four functions of the package are:

- ***read.in.json***: reads in data in JSON format for glmnet package functions
- ***read.in.json.for.pred***: reads in R raw glmnet models in JSON format for prediction
- ***glmnet.funct***: performs cv.glmnet which based on the users parameters can run a lasso, ridge or elastic net model. In order to choose which type of regression, the user needs to specify the elastic net mixing parameter alpha, i.e. alpha=0 (ridge), alpha=0.5 (elastic net), alpha=1 (lasso). The function runs glmnet 10+1 times by default; the first to get the penalty parameter lambda sequence, and then the remainder to compute the fit with each of the folds omitted. The response (dependent) variable can be categorized based on the following statistical families of distributions: Gaussian, Binomial, Poisson, Multinomial, Cox, multinomial Gaussian.
- ***pred.funct***: predicts from a cross-validated glmnet model, using the stored glmnet R model, and the optimal value chosen for lambda. The user needs to specify values of the penalty parameter lambda at which predictions are required. The values accepted are either s="lambda.1se" stored on the CV object, or s="lambda.min" can be used. If s is numeric, it is taken as the values of lambda to be used.

The package is available at http://147.102.82.122/ocpu/library/glmNETpkg/info and https://github.com/GTsiliki/glmNet , where more details can be found in the accompanied manual files, while the algorithm's web service URI is at: http://app.jaqpot.org:8080/jaqpot/services/algorithm/ocpu-glmnet.

CLUSTERING ALGORITHMS

We have implemented an R package called clusteringPkg to estimate the clustering memberships for columns and rows of a given matrix. Data can be clustered by either employing hierarchical clustering algorithm (Kaufman and Rousseau, 1990) using distance measures form the vegan R package

(http://cran.r-project.org/web/packages/vegan/index.html) or bi-clustering (Cheng and Church, 2000; Lazzeroni and Owen, 2000) from the blockcluster R package (http://cran.r-project.org/web/packages/blockcluster/index.html).

The package is available at http://147.102.82.122/ocpu/library/clusteringPkg/info and https://github.com/GTsiliki/clustPkg, where more details can be found in the accompanied manual files, while the algorithm's web service URI is at: http://app.jaqpot.org:8080/jaqpot/services/algorithm/ocpu-clustering.

The three functions of the package are:

- ***generate.biclust.model***: performs co-clustering (simultaneous clustering of rows and columns) for Binary, Contingency and Continuous data-sets using latent block models. It can also be used to perform semi-supervised co-clustering. The functions used are cocluster and coclusterstrategy from the blockcluster R package. The user needs to specify the row and column names of the data or declare them as unknown, the data type (binary, contingency, continuous, categorical) and the number of clusters for the x and y axis respectively. The output is the clustered data matrix and memberships of columns and rows; a graph of the clustered data is also produced for a better comprehension of the results.
- ***generate.hierar.model***: performs hierarchical clustering using the hclust function and vegdist from the vegan R package. The user needs to specify the number of clusters or the height of the dendrogram. For the distance function the options provided are: "manhattan", "euclidean", "canberra", "bray", "kulczynski", "jaccard", "gower", "altGower", "morisita", "horn", "mountford", "raup", "binomial", "chao", "cao" or "mahalanobis"., and for the agglomeration clustering methodologies the options are: "ward.D", "ward.D2", "single", "complete", "average", "mcquitty", "median" or "centroid". The output is the clustered data matrix and memberships of columns or rows; a graph of the clustered data is also produced for a better comprehension of the results.
- ***pred.clusters***: returns cluster memberships as estimated by generate.hierar.model or generate.biclust.model functions.

3.4.2 Integration of Python into the eNanoMapper computational infrastructure - Development of a Modelling web service in the Python language

Python (https://www.python.org/) has recently become the one of the most widely used programming languages, primarily due to its straightforward syntax which allows large-scale implementations in less lines of code than most languages, as well as rapid and effective system integration, boasting numerous and increasing success stories (https://www.python.org/about/success/). Its usage in scientific computing has augmented since the introduction of the numpy (http://www.numpy.org/) package for numerical computation and the scipy (http://www.scipy.org/) library, which allows the application of algorithms and toolboxes applicable to a wide array of scientific disciplines such as mathematics, engineering, as well as computational biology and chemistry. Many cheminformatics toolkits are built in python, provide a python API or allow python calls into other implementations, which include RDKit (http://www.rdkit.org/), OpenBabel (http://openbabel.org/) and Indigo (http://lifescience.opensource.epam.com/indigo/). A popular python software for using OpenBabel libraries is PyBel (O'Boyle et.al., 2008), allowing access to the attributes of molecules in a wide variety of formats and the calculation of physicochemical descriptors and fingerprints such as Molprint 2D (http://www.molprint.com/) (Bender et. al., 2004). Apart from descriptor/fingerprint calculations,

python is often used by researchers to either build custom machine learning algorithms, or apply already implemented ones to their chemical/biological datasets such as those provided by SciKit learn (http://scikit-learn.org/).

3.4.2.1 ALGORITHMS AVAILABLE FROM SCIKIT LEARN

SciKit learn (http://scikit-learn.org/) is an open-source python library for machine learning based on numpy and scipy, emphasizing on easy-to-use classification, regression and clustering algorithms. Implementations include generalized linear models (such as Ordinary Least Squares, Ridge Regression, Lasso, Elastic Net, Logistic Regression etc.), models for linear and quadratic discriminant analysis, kernel ridge regression, Support Vector Machines, Stochastic Gradient Descent, Nearest Neighbors, Gaussian Processes, Naive Bayes variations, numerous Decision Trees, Ensemble methods as well as functions for Feature selection (http://scikit-learn.org/dev/).

3.4.2.2 SERVICE TECHNICAL DETAILS

BEHIND THE SCENES

The python web service implemented in this work was built using Flask (http://flask.pocoo.org/), a microframework based on the Jinja2 (http://jinja.pocoo.org/) and Werkzeug WSGI (http://werkzeug.pocoo.org/) libraries. Jinja2 is a template engine for python, whilst Werkzeug is a WSGI utility library, which incorporates very straightforward HTTP header parsing and easy-to-handle request/response objects. The entire web application fits nicely into a single python file.

DATA HANDLING

The application receives a JSON file containing a training request *via* a POST command containing a dataset, the feature for prediction and model parameters where applicable. The dataset field further contains the unique dataset URI and the data entry field filled with substance identifiers and property-value pairs. Below is an example of a training request in JSON format.


```
{
   "dataset": {
      "datasetURI": "https://apps.ideaconsult.net/ambit2/dataset/R545",
      "dataEntry": [
                  {
                  "compound": {
                  "URI": "https://apps.ideaconsult.net/ambit2/compound/17/conformer/419593"
                  },
                  "values": {
                  "https://apps.ideaconsult.net/ambit2/feature/22127": 0.268,
                  "https://apps.ideaconsult.net/ambit2/feature/22137": 0.667,
                  "https://apps.ideaconsult.net/ambit2/feature/22200": -5.331,
                  "https://apps.ideaconsult.net/ambit2/feature/22213": 2,
                  "https://apps.ideaconsult.net/ambit2/feature/22252": 0.352
                  }
                  },
                  {
                  "compound": {
                  "URI": "https://apps.ideaconsult.net/ambit2/compound/40746/conformer/419619"
                  },
                  "values": {
                  "https://apps.ideaconsult.net/ambit2/feature/22127": 0.115,
                  "https://apps.ideaconsult.net/ambit2/feature/22137": 0.759,
                  "https://apps.ideaconsult.net/ambit2/feature/22200": -5.342,
                  "https://apps.ideaconsult.net/ambit2/feature/22213": 1,
                  "https://apps.ideaconsult.net/ambit2/feature/22252": 0.213
                  }
                  },
                                                {...},
                                                ...
                  ]
   },
   "predictionFeature": "https://apps.ideaconsult.net/ambit2/feature/22137",
   "parameters": {
   }
```

```
}
```

The curl command for a local implementation looks like:

```
curl -i -H "Content-Type: application/json" -X POST -d @C:/Python27/Flask-0.10.1/python-api/train.json
http://localhost:5000/pws/1
```

The application converts the fields into python objects/variables and calls the algorithm for prediction. The web service response consists of the fields:

- PMML model (where applicable)
- Raw model: python model encoded in base64 format
- Predicted feature: new feature name for attribute for which the model is being built
- Independent features: Attributes used for building model and to be used for testing
- Additional information: Any supplemental information the model may need, ranging from the URI of the predicted feature to parameters or meta-information that cannot be passed through the base64 encoding.

An example of a training response is shown below:


```
{
  "pmmlModel": "",
  "rawModel": "Y2NvcHlfcmVnCl9yZWNvbnN0cnVj9yZS5tdWx0aWFycmF5CnNj [...] dHA0Mgpic2Iu",
  "predictedFeatures": ["https://apps.ideaconsult.net/ambit2/feature/22137 predicted"],
  "independentFeatures":
    [
    "https://apps.ideaconsult.net/ambit2/feature/22200",
    "https://apps.ideaconsult.net/ambit2/feature/22127",
    "https://apps.ideaconsult.net/ambit2/feature/22213",
    "https://apps.ideaconsult.net/ambit2/feature/22252"
            ],
  "additionalInfo": [{'predictedFeature':"https://apps.ideaconsult.net/ambit2/feature/22137 predicted"}]
 }
```


A similar POST command invokes the testing portion of the web service, from which the information from the training response JSON is incorporated into another dataset file with test instances. The “rawModel” field is decoded and any additional information is supplied to the predictive service along with the test instances. A test dataset example can be found below:


```
{
   "dataset": {
      "datasetURI": ["https://apps.ideaconsult.net/ambit2/dataset/R545"],
      "dataEntry": [{
            "compound": {
               "URI": "https://apps.ideaconsult.net/ambit2/compound/17/conformer/419593"
            },
            "values": {
               "https://apps.ideaconsult.net/ambit2/feature/22127": 0.268,
               "https://apps.ideaconsult.net/ambit2/feature/22200": -5.331,
               "https://apps.ideaconsult.net/ambit2/feature/22213": 2,
               "https://apps.ideaconsult.net/ambit2/feature/22252": 0.352
            }
         }, {
            "compound": {
               "URI": "https://apps.ideaconsult.net/ambit2/compound/40746/conformer/419619"
            },
            "values": {
               "https://apps.ideaconsult.net/ambit2/feature/22127": 0.115,
               "https://apps.ideaconsult.net/ambit2/feature/22200": -5.342,
               "https://apps.ideaconsult.net/ambit2/feature/22213": 1,
               "https://apps.ideaconsult.net/ambit2/feature/22252": 0.213
            }
         }, {
            ...
```

```
                                        }]
    },
    "rawModel": "Y2NvcHlJ1x4MTJceGY [...] veVx4OTc/JwpwNDEKdHA0Mgpic2Iu",
            "additionalInfo": [{
                    "predictedFeature": "https://apps.ideaconsult.net/ambit2/feature/22137 predicted"
        }]
}
```


The response once again consists of a JSON file containing a dictionary with property- predicted value pairs in the same order as the test file to be later appended as an extra field in JAQPOT. The test response looks like:


```
{
    "predictions": [{
            "https://apps.ideaconsult.net/ambit2/feature/22137 predicted": 0.715
        }, {
            "https://apps.ideaconsult.net/ambit2/feature/22137 predicted": 0.933
        }, {
                    ...
    }]
}
```


3.4.2.3 ALGORITHMS IMPLEMENTED IN THE PYTHON WEB SERVICE

The purpose behind the development of the python web service is to provide users with more modelling choices, as well as to allow developers/researchers to dynamically post their own algorithms and thus keep the service updated and up-to-speed with current developments through the passage of time. In the first instance, we incorporated the LinearRegression which implements the Ordinary Least Squares algorithm and the LASSO function from SciKit learn, as well Quinlan's ID3 decision tree and Partial Least Squares with Variable Importance in Projection scoring.

ORDINARY LEAST SQUARES

The LinearRegression function fits a linear model in order to minimize the residual sum of squares between the observed and predicted responses by linear approximation (http://scikit-learn.org/stable/modules/linear_model.html). More specifically, it takes a matrix *X* (array of 1D instance arrays per variable) such as **[[0, 0], [1, 1], [2, 2]]** and an array *Y* (prediction feature), for example **[0, 1, 2]** and fits a linear model with coefficients w_i (where *i* = number of variables) stored in an array (**[0.5, 0.5]**) and used later with the SciKit learn predict function.
The implementation can be found under:
http://app.jaqpot.org:8080/jaqpot/services/algorithm/python-lm

LASSO

Lasso (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html) is a linear model that estimates sparse coefficients. It tends to reach solutions with fewer parameter values, thus reducing the number of variables used for a prediction. Lasso and algorithms derived from it are claimed to be fundamental to the field of compressed sensing. It is trained with L1 prior as regularizer and attempts to minimize the following:
(1 / (2 * n_samples)) * ||y - Xw||^2_2 + alpha * ||w||_1
Where alpha is a user-specified constant and ||w||_1 is the L1 norm of the parameter vector. Lasso optimizes the same objective function as the Elastic Net with l1_ratio=1.0, which means no L2 penalty.

The implementation can be found under: http://app.jaqpot.org:8080/jaqpot/services/algorithm/python-lasso

Parameters:

- alpha (optional). Constant that multiplies the L1 term. Default value: 1

ID3 DECISION TREE

The ID3 classification algorithm builds a decision tree, which at each node recursively splits the data into subsets according to the values of a selected attribute. These splits occur until the termination criteria of the algorithm are met, where a classification outcome is assigned. The splits are made based on information gain, for which attributes are evaluated for their information content compared to the target variable or class. At each node the variable with highest information content is selected until the split does not yield any more information. Information gain has been implemented to handle numeric values for attributes.
By default, this algorithm cannot handle numeric data for its prediction feature, as this must be a class. However, according to Scott's normal reference rule (*Scott 1979*), the outcome variable values are evaluated during training and automatically put into bins. Therefore, even for continuous data, the algorithm can still yield a prediction within a range i.e. [0.5, 0.8) which is technically a nominal value.
The implementation can be found under:
http://app.jaqpot.org:8080/jaqpot/services/algorithm/python-id3

PARTIAL LEAST SQUARES With "VARIABLE IMPORTANCE IN PROJECTION" scores

PLS-VIP is an exhaustive search for the best PLS model, optimized for the number of attributes used, the choice of these attributes and the number of latent variables. At each iteration, the attributes are evaluated on their VIP score for the optimal number of latent variables. For this step all allowed values for latent variables are evaluated. The worst variable is eliminated based on the lowest VIP score and the algorithm continues until no attributes are left. The PLS R^2 score at each step (for each number of attributes and their optimal number latent variables) for the particular set of attributes is stored in a matrix. The highest performing PLS model is the resulting training model.
The implementation can be found under: http://app.jaqpot.org:8080/jaqpot/services/algorithm/python-pls-vip

Parameters:

- latentVariables (mandatory). Number of latent variables to use. Must be smaller than number of attributes. Default value: num_attr -1

3.4.3 *INTEGRATION OF WEKA INTO THE ENANOMAPPER COMPUTATIONAL INFRASTRUCTURE - DEVELOPMENT OF A MODELLING WEB SERVICE USING THE WEKA DATA MINING SOFTWARE*

The Waikato Environment for Knowledge Analysis tool (Weka; http://www.cs.waikato.ac.nz/ml/weka/) is an open source data mining software written in Java, boasting a vast collection of machine learning algorithms suitable for addressing a plethora of data mining problems. These include methods for classification, regression and clustering, as well as association rules among attributes. Weka also allows

both supervised and unsupervised data pre-processing, ranging from data discretization and mathematical operations, all the way to principal component projection and feature selection, using methods such as information gain. Finally, it includes a flexible data visualizer for both input and output. Weka is highly popular due to its straightforward GUI, which allows all operations to be completed within a few clicks, as well as the number of widely used algorithms implemented and the variety of readable file formats allowed (such as ARFF and CSV). Each algorithm is initialized with suggested default values for its parameters, thus allowing the software to also be used by non-experts. However, for those which know the intricate details of each algorithm, the parameters can be adjusted for optimized results. It includes options for both internal cross-validation and user-provided external validation test set, as well as statistical measures for assessing the derived model quality and predictive power.

3.4.3.1 SERVICE TECHNICAL DETAILS

BEHIND THE SCENES

JQ provides a collection of basic algorithmic implementations under the JPDI protocol as a module of the core system, in order to provide a starting functionality as well as demonstrating the use of JPDI. This module gets shipped inside the Jaqpot EAR bundle but does not utilize any of the EJB service layers of the core system, showing the exact same behaviour as expected of an external JPDI algorithm service, which sums up in being able to provide clean restful and stateless resources for each algorithm implementation that keep no internal state or data whatsoever.

The module consists of a set of preprocessing algorithms and a domain of applicability calculation algorithm, coded by the Jaqpot development team, and a set of training algorithms, powered by the well-established machine learning libraries Weka (v 3.6.12) and LibSVM (v 3.17). Each algorithm has its own resource URI under the path /algorithms and different functions for training and prediction mapped on /training and /prediction endpoints

DATA HANDLING

The application receives a JSON file containing a training request *via* a POST command containing a dataset, the feature for prediction and model parameters where applicable. The dataset field further contains the unique dataset URI and the data entry field filled with substance identifiers and property-value pairs.

The application converts the fields into Weka objects/variables and calls algorithm for prediction. The web service response consists of the fields:
-Pmml model (for model transferability)
-Raw model: Weka model encoded in base64 format
-Predicted feature: new feature name for attribute for which the model is being built
-Independent features: Attributes used for building model and to be used for testing
-Additional information: Any supplemental information the model may need, ranging from the URI of the predicted feature to parameters or meta-information that cannot be passed through the base64 encoding.

A similar POST command invokes the testing portion of the web service, from which the information from the training response JSON is incorporated into another dataset file with test instances. The “rawModel” field is decoded and any additional information is supplied to the predictive service along

with the test instances. The response once again consists of a JSON file containing a dictionary with property- predicted value pairs in the same order as the test file to be later appended as an extra field in JAQPOT.

3.4.3.2 ALGORITHMS IMPLEMENTED IN THE WEKA WEB SERVICE

LINEAR REGRESSION

Linear Regression that uses the Akaike criterion for model selection, and is able to deal with weighted instances. The implementation can be found under:
http://app.jaqpot.org:8080/jaqpot/services/algorithm/weka-mlr

PARTIAL LEAST SQUARES REGRESSION

Partial Least Squares Regression with Simpls and PLS1 supported. The implementation can be found under:
http://app.jaqpot.org:8080/jaqpot/services/algorithm/weka-pls

Parameters:

- algorithm (optional). The algorithm to use. Default value: "PLS1". Available choices: SIMPLS, PLS1.
- components (optional). The number of components to compute. Default value: 20

RADIAL BASIS FUNCTION NETWORK

RBFNetwork Implements a normalized Gaussian radial basis function network. It uses the k-means clustering algorithm to provide the basis functions and learns either a logistic regression (discrete class problems) or linear regression (numeric class problems) on top of that. The implementation can be found under:
http://app.jaqpot.org:8080/jaqpot/services/algorithm/weka-rbf

Parameters:

- seed (optional). Set the random seed to be used by K-means. Default value: 1
- maxClusters (optional). Maximum number of clusters to generate. Default value: 2
- minStdDev (optional). Sets the minimum standard deviation for the clusters. Default value: 0.1
- maxIts (optional). Maximum number of iterations for the logistic regression to perform Default value: -1
- ridge (optional). Set the ridge value for the logistic or linear regression. Default value: 1e-8

SUPPORT VECTOR MACHINES

Support Vector Machine LibSVM implementation that supports One-class SVM, Regressing SVM, and nu-SVM. The implementation can be found under:
http://app.jaqpot.org:8080/jaqpot/services/algorithm/weka-svm

Parameters:

- cost (optional). Set the parameter C of C-SVC, epsilon-SVR, and nu-SVR. Default value: 100
- gamma (optional). Set gamma in kernel function. Default value: 1.5
- epsilon (optional). Set the epsilon in loss function of epsilon-SVR. Default value: 0.1
- coeff0 (optional). Set coeff0 in kernel function. Default value: 0
- cacheSize (optional). The size of the cache (a prime number), 0 for full cache and -1 to turn it off. Default value: 25077

- kernel (optional). Set type of kernel function. Default value: “RBF”. Available choices: 0 = “linear”, 1 = “polynomial”, 2 = “radial basis function”, 3 = “sigmoid”
- loss (optional). Set the epsilon in loss function of epsilon-SVR. Default value: 0.1
- nu (optional). Set the parameter nu of nu-SVC, one-class SVM, and nu-SVR. Default value: 0.5
- degree (optional). Set degree in kernel function. Default value: 3
- type (optional). Type of SVM. Default value: “NU_SVR”. Available choices: 0=”C-SVC”, 1=”nu-SVC”, 2=”one-class SVM”, 3=”epsilon-SVR”, 4=”nu-SVR”

4. DEMONSTRATION OF THE JAQPOT MODELING WORKLFOW

A full demonstration of the JQ modelling workflow will be given in this section with the help of the Swagger interface for visualization of the functionality of the web services. Starting from a bundle with experimental data, a dataset with numerical data will be created. Mathematical operations specified by a PMML file will be performed on this dataset to produce the final dataset, which will be used to train a model using the JQ implementation of the R multivariable linear regression algorithm. The model will be applied to a test dataset in order to obtain predictions.

4.1 FROM EXPERIMENTAL DATA BUNDLE TO DATASET

For this demonstration, we will use the experimental results published by Walkey et. al. (2014) on gold nanoparticles, which are stored as a bundle at this URI: https://apps.ideaconsult.net/enmtest/bundle/27 (Figure 9). This bundle contains experimental data for 84 surface-modified gold nanoparticles, extracted from different characterization assays. The aim is to produce a multivariable linear nQSAR model that predicts cell association as a linear combination of two input parameters (zeta potential after synthesis and zeta potential after exposure) as well as four transformations on these parameters that are defined later in this section. Cell association is the end-point used in the reference publication, because of its relevance to inflammatory responses, biodistribution and toxicity in-vivo.

The Swagger API interface (http://swagger.io/) will be used to communicate with the web services and demonstrate their functionality. On the top of the Swagger page shown in Figure 1, we can see the URI for the web service documentation used and an alphanumeric string which is a token retrieved on behalf of the user. This token contains 'guest' privileges and allows access to all eNanoMapper services and databases. Users which want to access datasets available specifically to a user must supply a token for their account, which can be done at this page at http://app.jaqpot.org:8080/jaqpot/swagger/#!/aa/login (for details, please see Appendix I, in D4.2).

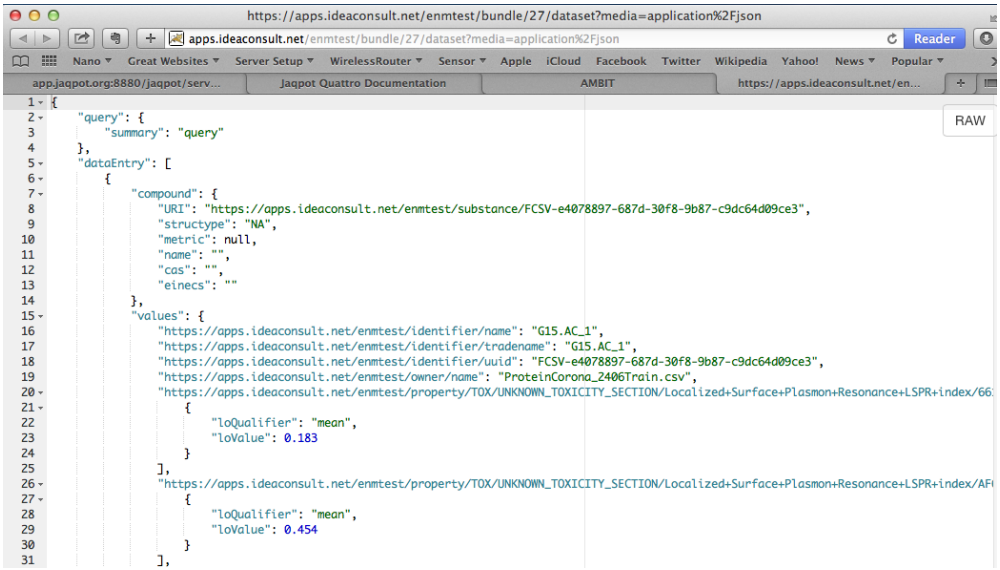


Figure 9 Details on content of a bundle

The function of "cleaning up" a bundle to create a dataset with only the relevant entries is implemented as shown in the Swagger interface in Figure 10. The user needs to input the "parent" bundle, from which the dataset will be produced.

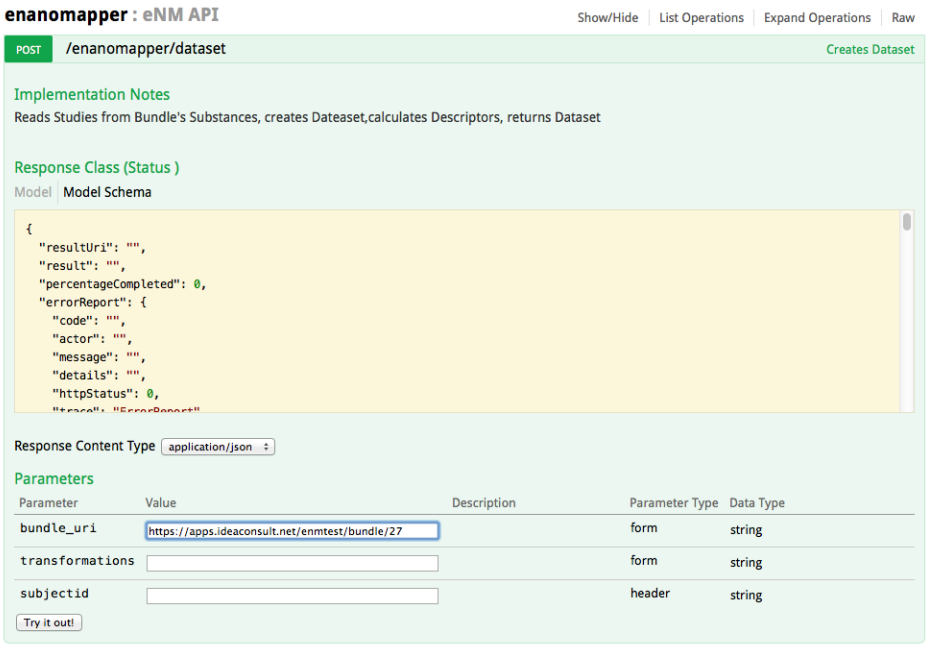


Figure 10 Swagger form for creation of dataset from bundle

Hitting the "Try it out!" button on the bottom of the page informs the user that a task has been queued; using the task id provided, the user can query its current status, as in Figure 11.

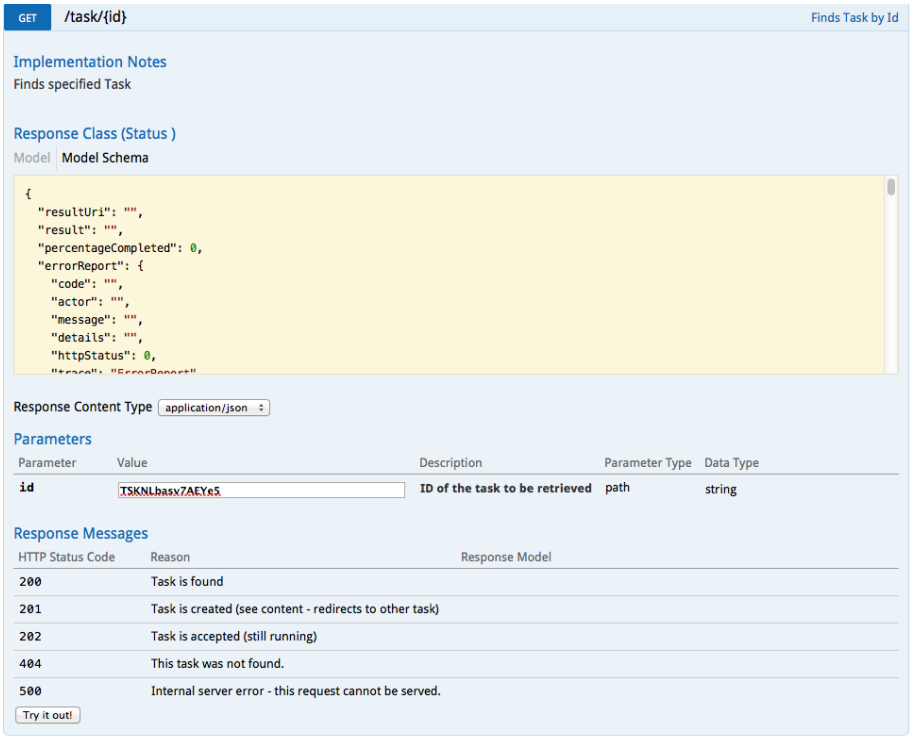


Figure 11 Query of task id

The output of the query is as follows, where we can see that the task has been completed. The URI of the dataset produced can be found in the resultUri field (see highlighted fields):


```
{
  "meta": {
    "comments": [
      "Preparation Task is now running with ID Thread-97 (HornetQ-client-global-threads-401950683)",
      "Starting Dataset preparation...",
      "Dataset ready.",
      "Saving to database...",
      "Dataset saved successfully.",
      "Preparation Task is now completed."
    ],
    "descriptions": [
      "A preparation procedure will return a Dataset if completed successfully. It may also initiate other procedures if desired."
    ],
    "titles": [
      "Preparation on bundle: https://apps.ideaconsult.net/enmtest/bundle/27"
    ],
    "hasSources": [
      null
    ],
    "date": "2015-06-25T11:13:11.570+0000"
  },
  "resultUri": "http://app.jaqpot.org:8080/jaqpot/services/dataset/SFz42r0XiMLO",
  "result": "dataset/SFz42r0XiMLO",
  "percentageCompleted": 100,
  "httpStatus": 200,
  "createdBy": "guest",
  "type": "PREPARATION",
  "_id": "TSKNLbasv7AEYe5",
  "status": "COMPLETED"
}
```


The produced dataset can be seen by clicking on the dataset URI (see Figure 12): http://app.jaqpot.org:8080/jaqpot/services/dataset/SFz42r0XiMLO

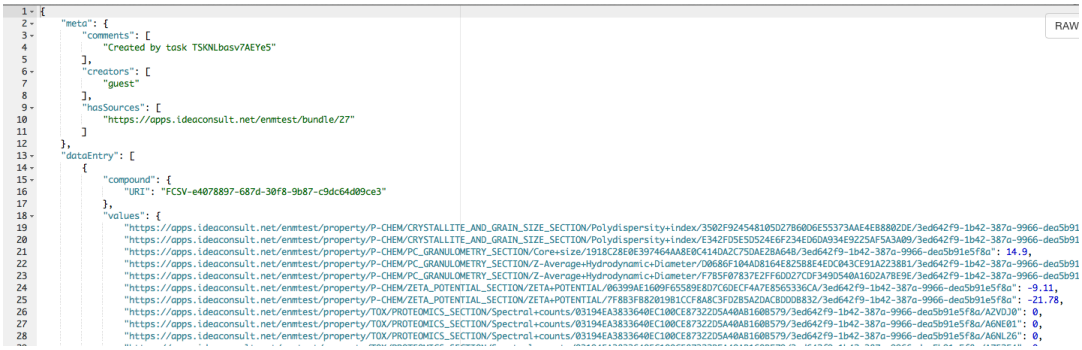


Figure 12 Dataset produced from bundle

4.2 USING THE DATASET FOR MODELLING

Using the input dataset created above and an algorithm of our choice, we can now create a model. The algorithm selection is carried out by simply adding its URI to the list of model parameters. The following user-specified parameters are provided in the modelling interface:

- ***dataset_uri***: (URI of the input dataset) http://app.jaqpot.org:8080/jaqpot/services/dataset/SFz42r0XiMLO
- ***prediction_feature:*** here we wish to create a model that predicts the log2 transformed value of the Cell Association, as reported in the source publication: https://apps.ideaconsult.net/enmtest/property/TOX/UNKNOWN_TOXICITY_SECTION/Log2+transformed/94D664CFE4929A0F400A5AD8CA733B52E049A688/3ed642f9-1b42-387a-9966-dea5b91e5f8a
- ***parameters:*** (Algorithm-specific parameters can be specified in this field) {blank} The R linear regression implementation is not associated with tuning parameters.
- ***scaling:*** {blank} The field was left blank as no scaling was used.
- ***domain of applicability:*** We used the *Leverage* algorithm for DoA calculations.
- ***transformations:*** The user can apply any of the transformations available in the PMML language (Pechter, 2009) and stored in a separate PMML file. The use of PMML has been first described in Deliverable 4.1 and here we will show its implementation. Users can upload a PMML file (it uses the .xml file format) to make it available to Jaqpot and other web services using the interface at http://app.jaqpot.org:8080/jaqpot/swagger/#!/pmml/createPMML. The file for our example has been made available at http://app.jaqpot.org:8080/jaqpot/services/pmml/tYpyGNyqPX and can be viewed below. In this case we use PMML to form a new dataset that consists of the absolute values of two variables (Zeta Potential with and without human serum), their difference and quotient. The predicted variable declared above is added automatically.
- ***id:*** We will use the linear modelling algorithm from R, available through OpenCPU, with ID: ocpu-lm.
- ***subjectid:*** {blank} (guest token will be used).

```
<PMML version="4.0"
   xsi:schemaLocation="http://www.dmg.org/PMML-4_0
   http://www.dmg.org/v4-0/pmml-4-0.xsd"
   xmlns="http://www.dmg.org/PMML-4_0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<DataDictionary numberOfFields="4" >
  <DataField name="https://apps.ideaconsult.net/enmtest/property/P-
CHEM/ZETA_POTENTIAL_SECTION/ZETA+POTENTIAL/7F8B3FB82019B1CCF8A8C3FD2B5A2DACBDDDB832/3ed642f9-1b42-
387a-9966-dea5b91e5f8a" optype="continuous" dataType="double" />

  <DataField name="https://apps.ideaconsult.net/enmtest/property/P-
CHEM/ZETA_POTENTIAL_SECTION/ZETA+POTENTIAL/06399AE1609F65589E8D7C6DECF4A7E8565336CA/3ed642f9-1b42-
387a-9966-dea5b91e5f8a" optype="continuous" dataType="double" />

</DataDictionary>
<TransformationDictionary>
        <DerivedField dataType="double" name="zp_ch" optype="categorical">
                <Apply function="-">
                        <FieldRef field="https://apps.ideaconsult.net/enmtest/property/P-
CHEM/ZETA_POTENTIAL_SECTION/ZETA+POTENTIAL/7F8B3FB82019B1CCF8A8C3FD2B5A2DACBDDDB832/3ed642f9-1b42-
387a-9966-dea5b91e5f8a"/>
                        <FieldRef field="https://apps.ideaconsult.net/enmtest/property/P-
CHEM/ZETA_POTENTIAL_SECTION/ZETA+POTENTIAL/06399AE1609F65589E8D7C6DECF4A7E8565336CA/3ed642f9-1b42-
387a-9966-dea5b91e5f8a"/>
                </Apply>
        </DerivedField>
        <DerivedField dataType="double" name="zp_rel" optype="categorical">
                <Apply function="/">
                        <FieldRef field="https://apps.ideaconsult.net/enmtest/property/P-
CHEM/ZETA_POTENTIAL_SECTION/ZETA+POTENTIAL/7F8B3FB82019B1CCF8A8C3FD2B5A2DACBDDDB832/3ed642f9-1b42-
387a-9966-dea5b91e5f8a"/>
                        <FieldRef field="https://apps.ideaconsult.net/enmtest/property/P-
CHEM/ZETA_POTENTIAL_SECTION/ZETA+POTENTIAL/06399AE1609F65589E8D7C6DECF4A7E8565336CA/3ed642f9-1b42-
387a-9966-dea5b91e5f8a"/>
                </Apply>
        </DerivedField>
        <DerivedField dataType="double" name="zp_synth_mag" optype="categorical">
                <Apply function="abs">
                        <FieldRef field="https://apps.ideaconsult.net/enmtest/property/P-
CHEM/ZETA_POTENTIAL_SECTION/ZETA+POTENTIAL/7F8B3FB82019B1CCF8A8C3FD2B5A2DACBDDDB832/3ed642f9-1b42-
387a-9966-dea5b91e5f8a"/>
                </Apply>
        </DerivedField>
        <DerivedField dataType="double" name="zp_serum_mag" optype="categorical">
                <Apply function="abs">
                        <FieldRef field="https://apps.ideaconsult.net/enmtest/property/P-
CHEM/ZETA_POTENTIAL_SECTION/ZETA+POTENTIAL/06399AE1609F65589E8D7C6DECF4A7E8565336CA/3ed642f9-1b42-
387a-9966-dea5b91e5f8a"/>
                </Apply>
        </DerivedField>
</TransformationDictionary>
  </PMML>
```

After, hitting the “Try it out!” button the user is notified that a task has been initiated with the following id: **JqDUnYgb3I4W**. Querying this task id at http://app.jaqpot.org:8080/jaqpot/swagger/#!/task/getTask shows the following information:

```
{
  "meta": {
    "comments": [
      "Training task created",
      "Training Task is now running.",
      "--",
      "Processing transformations...",
      "-",
      "Starting training on transformation algorithm:http://app.jaqpot.org:8080/jaqpot/services/algorithm/pmml",
      "Training task created:http://app.jaqpot.org:8080/jaqpot/services/task/GGxgel6nbzI4",
      "Transformation model created successfully:http://app.jaqpot.org:8080/jaqpot/services/model/SLnI8mUX9HJ9",
      "Prediction task created:http://app.jaqpot.org:8080/jaqpot/services/task/TSKlXhzZeCt41vj",
      "Transformed dataset created successfully:http://app.jaqpot.org:8080/jaqpot/services/dataset/rbLnSXJw3YwHRe",
```

```
    "-",
    "Starting training on linked algorithm:http://app.jaqpot.org:8080/jaqpot/services/algorithm/leverage",
    "Training task created:http://app.jaqpot.org:8080/jaqpot/services/task/D5cUFT8loUaY",
    "Linked model created successfully:http://app.jaqpot.org:8080/jaqpot/services/model/APCFYvgmxeNK",
    "--",
    "Training dataset URI is:http://app.jaqpot.org:8080/jaqpot/services/dataset/rbLnSXJw3YwHRe",
    "Attempting to download dataset...",
    "Dataset has been retrieved.",
    "Creating JPDI training request...",
    "Inserted dataset.",
    "Inserted prediction feature.",
    "Inserted parameters.",
    "Inserted algorithm id.",
    "Sending request to  algorithm service:http://app.jaqpot.org:8004/ocpu/library/PLMplusPMMLpkg/R/lm.funct/json",
    "Algorithm service responded with status:200",
    "Attempting to parse response...",
    "Response was parsed successfully",
    "Building model...",
    "Defining the prediction features",
    "Model was built successfully",
    "Model was built successfully. Now saving to database...",
    "Task Completed Successfully."
  ],
  "descriptions": [
    "Training task using algorithm ocpu-lm"
  ],
  "titles": [
    "Training on algorithm: ocpu-lm"
  ],
  "hasSources": [
    "algorithm/ocpu-lm"
  ],
  "date": "2015-07-14T08:36:51.390+0000"
 },
 "resultUri": "http://app.jaqpot.org:8080/jaqpot/services/model/VZ4xuG8WJUbn",
 "result": "model/VZ4xuG8WJUbn",
 "percentageCompleted": 100,
 "httpStatus": 201,
 "createdBy": "guest",
 "duration": 14111,
 "type": "TRAINING",
 "_id": "JqDUnYgb3I4W",
 "status": "COMPLETED"
}
```


The task has been completed and we are provided with a model URI: http://app.jaqpot.org:8080/jaqpot/services/model/VZ4xuG8WJUbn. We can use the model id (VZ4xuG8WJUbn) to send a query at http://app.jaqpot.org:8080/jaqpot/swagger/#!/model/getModelPmml (Figure 13) and receive the model in PMML form, as below:


```
<PMML version="4.2" xmlns="http://www.dmg.org/PMML-4_2" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.dmg.org/PMML-4_2 http://www.dmg.org/v4-2/pmml-4-2.xsd">
  <Header copyright="Copyright (c) 2015 www-data" description="Linear Regression Model">
    <Extension name="user" value="www-data" extender="Rattle/PMML"/>
    <Application name="Rattle/PMML" version="1.4"/>
    <Timestamp>2015-07-14 08:37:05</Timestamp>
  </Header>
  <DataDictionary numberOfFields="5">
    <DataField name="x1" optype="continuous" dataType="double"/>
    <DataField name="x2" optype="continuous" dataType="double"/>
    <DataField name="x3" optype="continuous" dataType="double"/>
    <DataField name="x4" optype="continuous" dataType="double"/>
    <DataField name="x5" optype="continuous" dataType="double"/>
  </DataDictionary>
  <RegressionModel modelName="Linear_Regression_Model" functionName="regression" algorithmName="least squares">
    <MiningSchema>
```

```
      <MiningField name="x1" usageType="predicted"/>
      <MiningField name="x2" usageType="active"/>
      <MiningField name="x3" usageType="active"/>
      <MiningField name="x4" usageType="active"/>
      <MiningField name="x5" usageType="active"/>
    </MiningSchema>
    <Output>
      <OutputField name="Predicted_x1" feature="predictedValue"/>
    </Output>
    <RegressionTable intercept="-8.49158823688487">
      <NumericPredictor name="x2" exponent="1" coefficient="0.131823732845061"/>
      <NumericPredictor name="x3" exponent="1" coefficient="0.233801765980121"/>
      <NumericPredictor name="x4" exponent="1" coefficient="0.136671740405479"/>
      <NumericPredictor name="x5" exponent="1" coefficient="0.12020574120013"/>
    </RegressionTable>
  </RegressionModel>
</PMML>
```

The PMML representation of the model provides full information on the algorithm used, the dependent and independent features and predicted variable and allows rapid deployment on any platform that supports PMML without additional code.

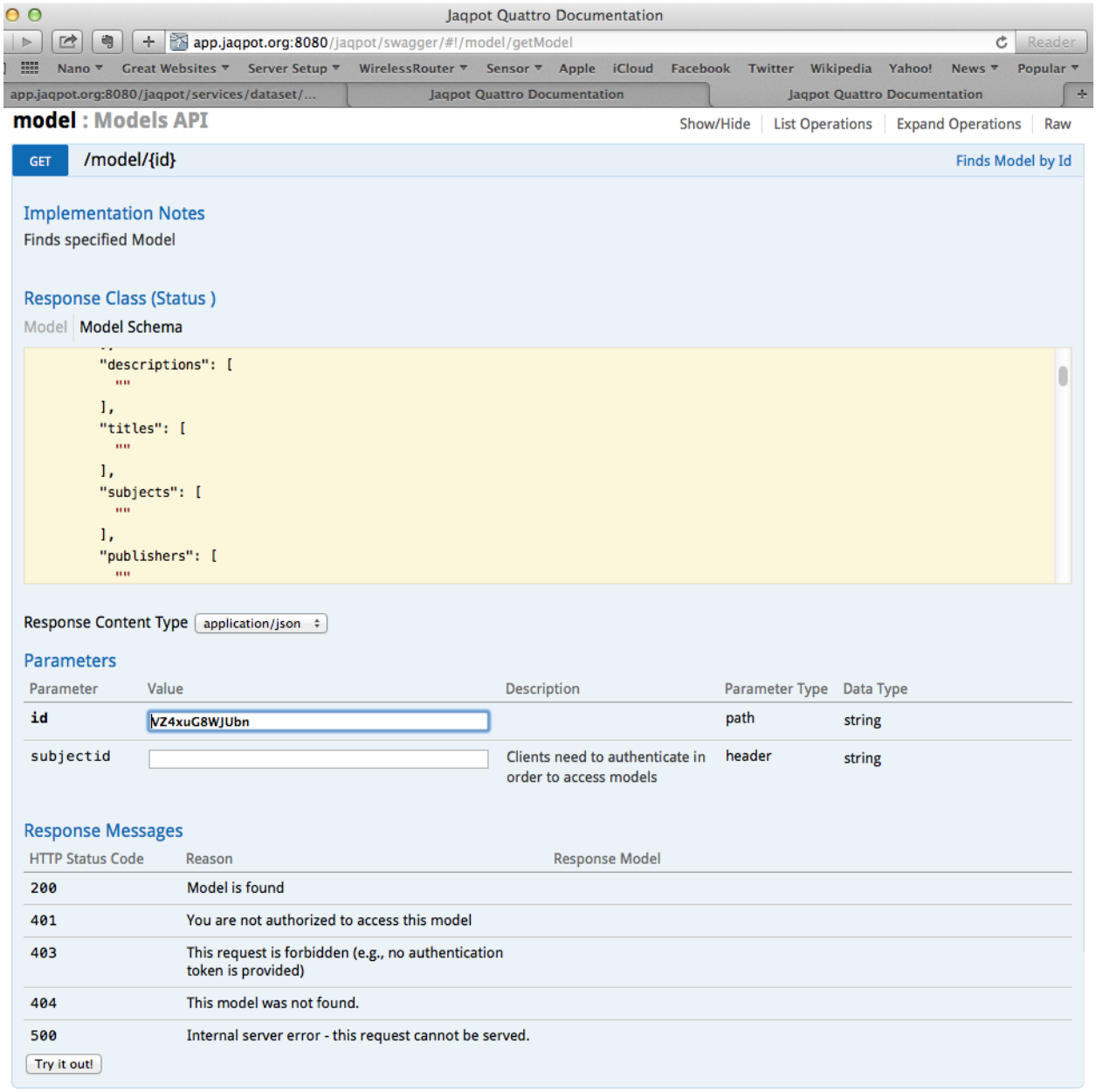


Figure 13 Querying the model

4.3 USING THE MODEL FOR PREDICTIONS

With the model at hand, predictions can be made on a different set of instances. A separate test dataset will be used for this example, which can be found at:
http://app.jaqpot.org:8080/jaqpot/services/dataset/lGJBKSO6Wr8X

As shown in Figure 14, the Swagger interface requires to be provided with the URI of the dataset on which the model will be applied and the id of the model that will be used. After the modelling has been initiated, we receive the following message of a new task that has been queued:


```
{
  "meta": {
    "comments": [],
```

```
    "descriptions": [
      "A prediction procedure will return a new Dataset if completed successfully."
    ],
    "titles": [
      "Prediction by model VZ4xuG8WJUbn"
    ],
    "hasSources": [
      null
    ],
    "date": "2015-07-14T08:43:23.617+0000"
  },
  "httpStatus": 202,
  "createdBy": "guest",
  "type": "PREDICTION",
  "_id": "TSKnY8pDQuN089o",
  "status": "QUEUED"
}
```

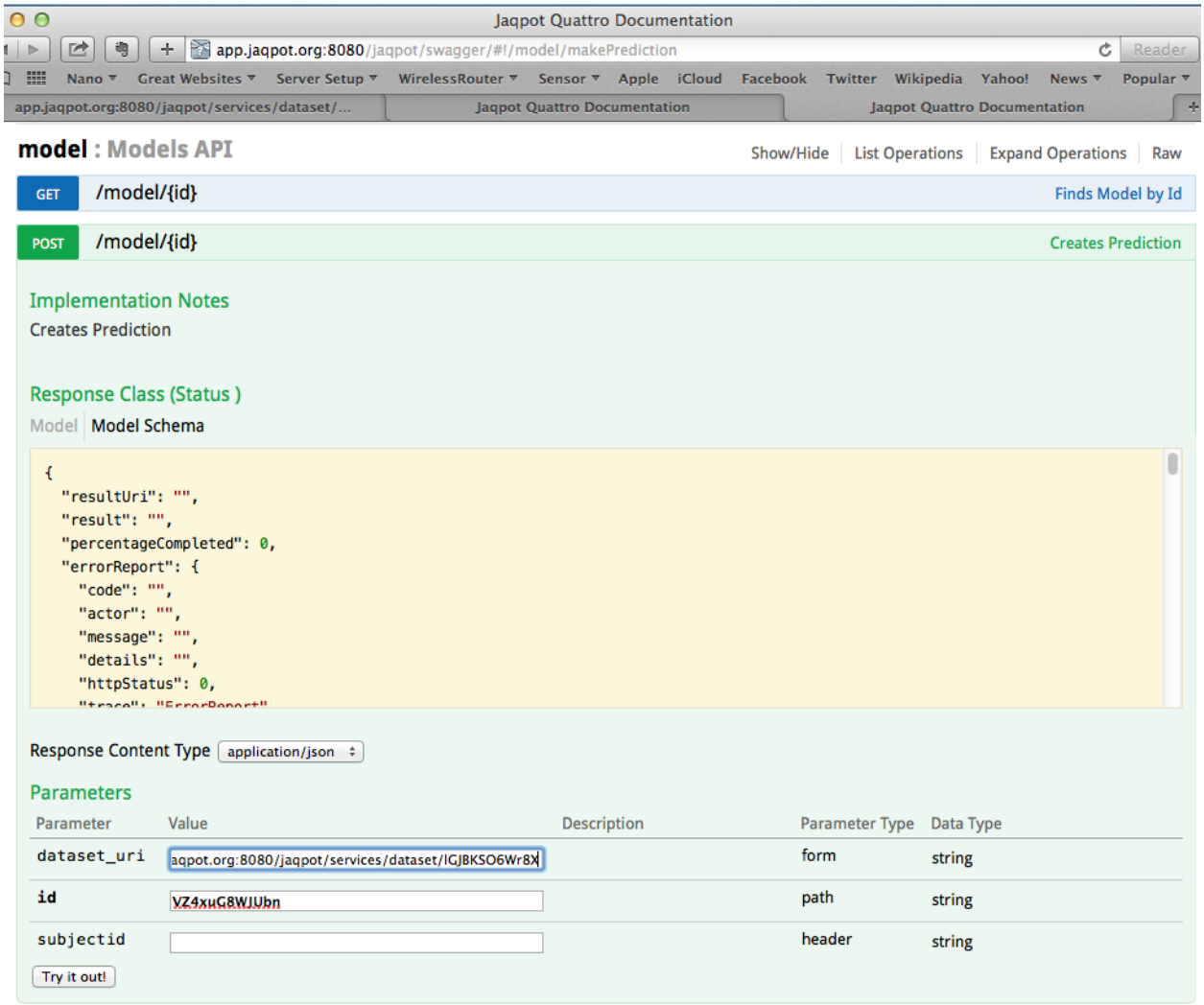



Figure 14 Applying the model to the test dataset

Querying the task id at http://app.jaqpot.org:8080/jaqpot/swagger/#!/task/getTask provides the user with information on the various stages of the modelling task and its final status.

```
{
  "meta": {
    "comments": [
      "Prediction Task is now running.",
      "Attempting to find model in database...",
      "Model retrieved successfully.",
      "--",
      "Processing transformations...",
      "Transformation task created:http://app.jaqpot.org:8080/jaqpot/services/task/TSKD1JzpipEiMT1",
      "Transformed dataset created successfully:http://app.jaqpot.org:8080/jaqpot/services/dataset/szu95fY8BCuTa9",
      "--",
      "--",
      "Processing linked models...",
      "Prediction task created:http://app.jaqpot.org:8080/jaqpot/services/task/TSKf9DQanOj4tbl",
      "Prediction dataset created successfully:http://app.jaqpot.org:8080/jaqpot/services/dataset/cN5kyfiilhotXT",
      "--",
      "Attempting to download dataset...",
      "Dataset has been retrieved.",
      "Dataset has been cleaned from unused values.",
      "Creating JPDI prediction request...",
      "Sending request to algorithm service:http://app.jaqpot.org:8004/ocpu/library/PLMplusPMMLpkg/R/pred.funct/json",
      "Algorithm service responded with status:200",
      "Attempting to parse response...",
      "Response was parsed successfully.",
      "Creating new Dataset for predictions...",
      "Dataset ready.",
      "Saving to database...",
      "Dataset saved...",
      "Task Completed Successfully."
    ],
    "descriptions": [
      "A prediction procedure will return a new Dataset if completed successfully."
    ],
    "titles": [
      "Prediction by model VZ4xuG8WJUbn"
    ],
    "hasSources": [
      null
    ],
    "date": "2015-07-14T08:43:23.617+0000"
  },
  "resultUri": "http://app.jaqpot.org:8080/jaqpot/services/dataset/JYKC24AkXw5Koa",
  "result": "dataset/JYKC24AkXw5Koa",
  "percentageCompleted": 100,
  "httpStatus": 201,
  "createdBy": "guest",
  "type": "PREDICTION",
  "_id": "TSKnY8pDQuN089o",
  "status": "COMPLETED"
}
```


The output dataset with the predictions is available at the following URI (Figure 15):
http://app.jaqpot.org:8080/jaqpot/services/dataset/JYKC24AkXw5Koa

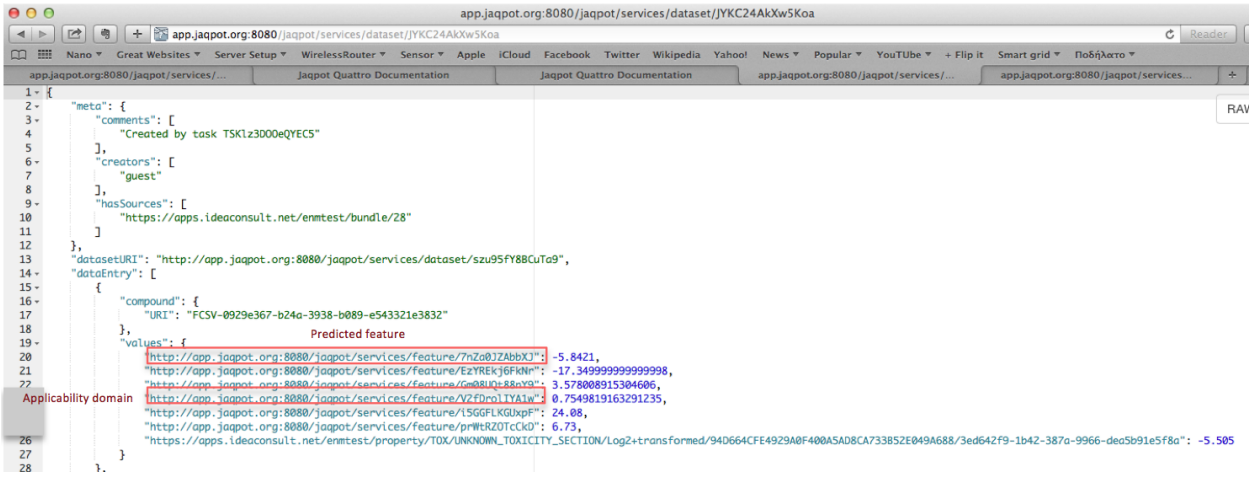


Figure 15 Output dataset with prediction feature

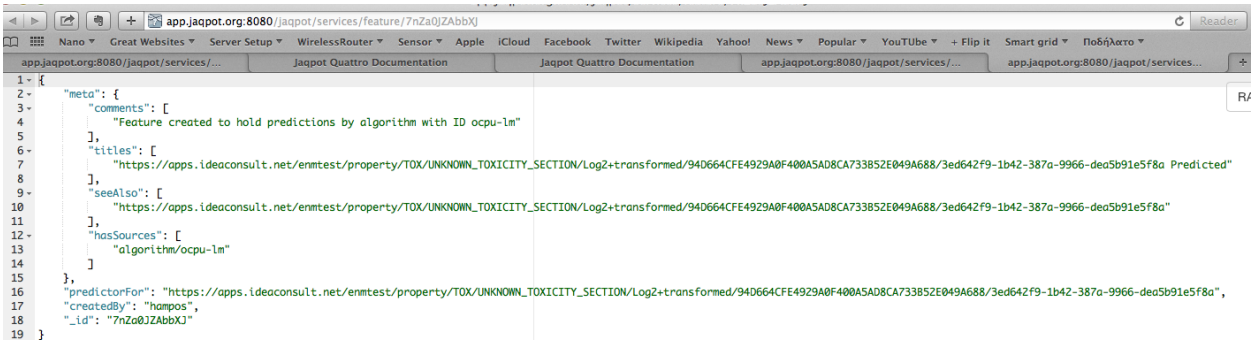


Figure 16 JSON representation of predicted feature

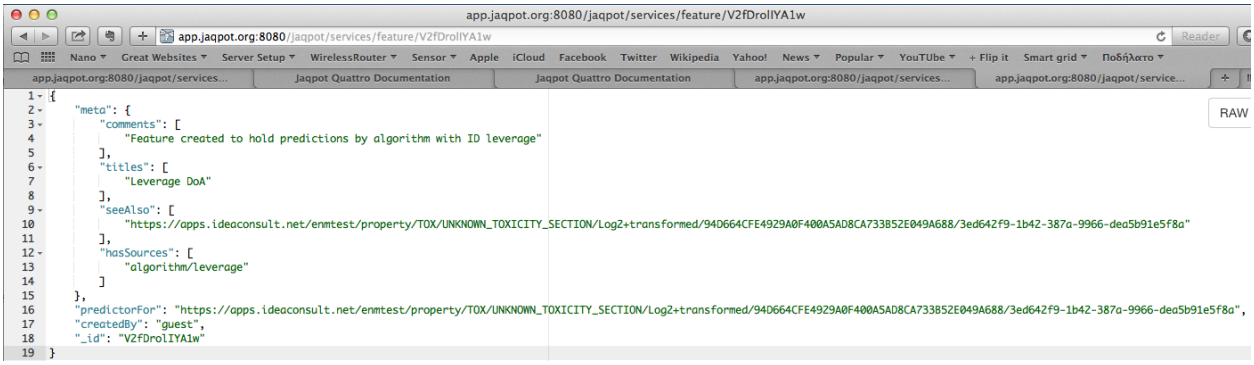


Figure 17 JSON representation of DoA feature

The new dataset consists of the original values from the input dataset plus the predictions and the domain of applicability values, which are represented by newly created features, which can be found at: http://app.jaqpot.org:8080/jaqpot/services/feature/7nZa0JZAbbXJ and http://app.jaqpot.org:8080/jaqpot/services/feature/V2fDrolIYA1w. Figure 15 highlights the prediction and the DoA value for a specific substance (a DoA value closer to 1 shows greater confidence for the prediction, while the 0 value means that the specific substance is outside the DoA of the model). The URIs of the newly created features (Figure 16, Figure 17) give us information on the features and the algorithms used to produce predictions or DoA values. It must be noted here that, instead of "algorithm/ocpu-lm" as in the prediction, the "hasSources" field contains "algorithm/leverage", to indicate the origin of its values.

5. RREGRS PACKAGE

We have implemented an R package, called RRegrs, for computer-aided model selection with regression algorithms, as well as individually implemented methodologies utilized within the eNanoMapper computational infrastructure. The RRegrs (R Regressions) tool (Tsiliki et al., 2015) is a standardized framework that automates the development of a reliable and well-validated QSAR model or set of models. RRegrs is mainly based on the R caret package and is focusing on regression modelling providing a standardization methodology to search across the model space and produce a validated predictive model. The main advantage of this package is that only one RRegrs call is needed (call RRegrs function) to run the entire workflow and obtain the produced QSAR model(s) in a reproducible format in contrast to the standard, inefficient and time-consuming QSAR modelling workflow, where the modeller tries many different algorithms and tuning parameter in each algorithm. RRegrs suggests an easy way to explore the models' search space for linear to non-linear models with special parameters specifications and cross validation schemes. Model outputs are easily accessible and readable, organized by methods, centralized and averaged by multiple reproducible data set splits. Summary files are also produced helping the user to easily access all methodology results. The current implementation of the RRegrs package contains ten different regression algorithms. Utilization of this package does not require advanced knowledge of R, but on the other hand an experienced R user can easily modify or extend the package, by including the algorithms of his/her choice. RRegrs call could be integrated into complex desktop/Web tools for QSAR and is available as an open repository at https://github.com/enanomapper/RRegrs or as a fist release with DOI 10.5281/zenodo.16446.

In Figure 18 we show the main flowchart of the RRegrs function. It can be seen that the main function of the package (RRegrs) contains several sections: loading parameters and dataset, remove near zero variance features, scaling dataset, remove correlated features, dataset splitting, run the ten regression methods, summary of statistics for all methods and splitting's, averages for each method and cross-validation type for all methods for splitting, automatic best model statistics, best model Y-randomization. Assessment of Domain of Applicability was included in each method.

In order to use RRegrs function, it is needed to specify a minimum set of parameters (the others will get default values). All the parameters will be written into a CSV file (ex: Parameters.csv), in the same working folder where it should be present the input dataset file and the outputs files. The dataset needs to have CSV format, with the first column as the dependent variable. The input and output files will be placed into a working folder. The output files are CSV statistics files, PDF and PNG plots. In the following sections we present the special features of RRegrs package and the flexibility they offer. However, as already specified earlier in the text, the user has the option to apply all functions using the default settings introduced in the manual and help files of the package.

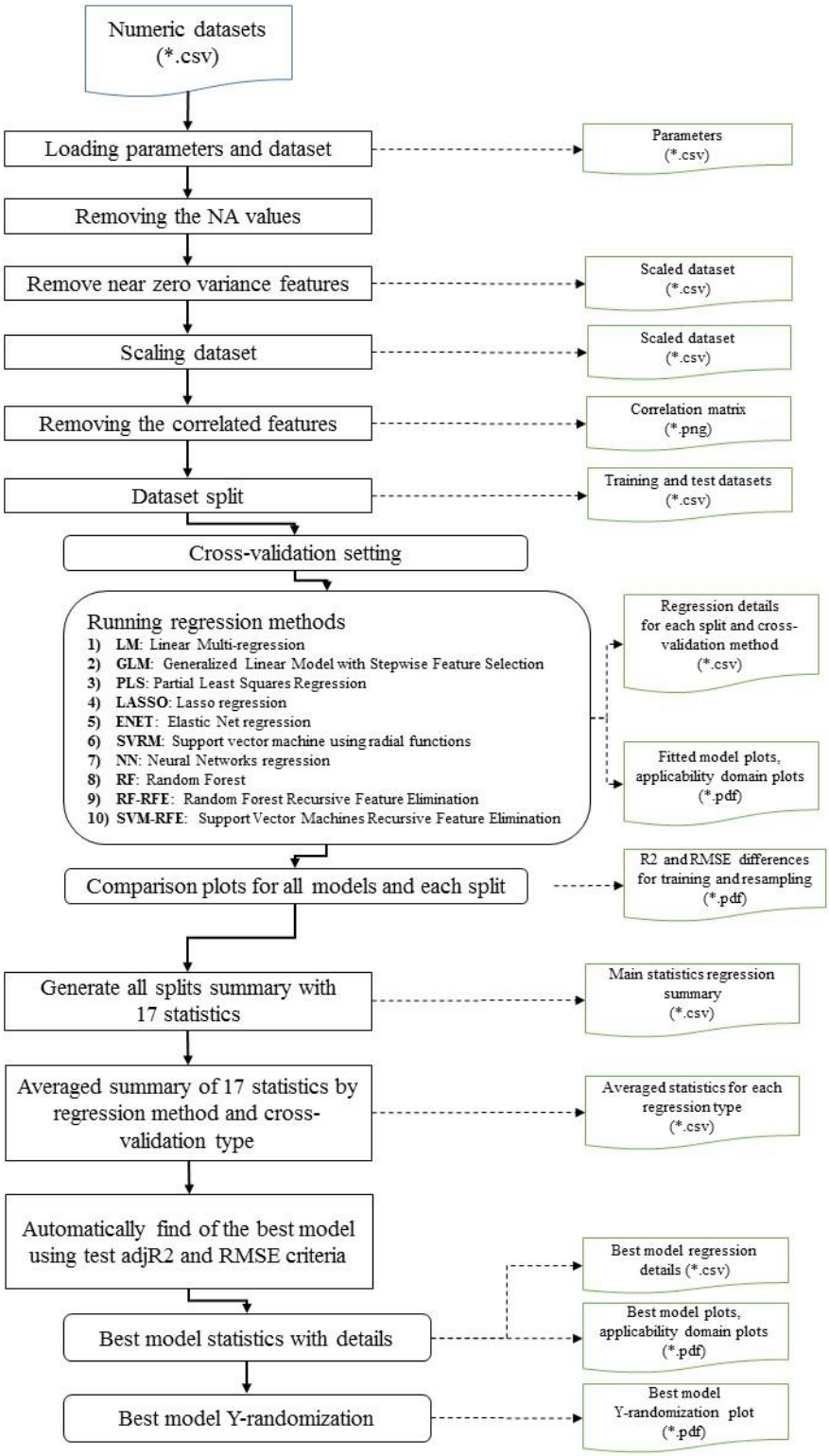


Figure 18 RRegrs methodology flowchart. The main steps followed by the RRegrs function are indicated as well as the input parameters needed and the format of the output produced

Embedded Regression Models

RRegrs represents a simple tool to screen any dataset for the best regression model using ten implemented regression methods:

1. Linear Multi-regression (LM)
2. Generalized Linear Model with Stepwise Feature Selection (GLM)
3. Partial Least Squares Regression (PLS)
4. Lasso regression (LASSO)
5. Elastic Net regression (ENET)
6. Support vector machine using radial functions (SVM radial)
7. Neural Networks regression (NN)
8. Random Forest (RF)
9. Random Forest-Recursive Feature Elimination (RF-RFE)
10. Support Vector Machines Recursive Feature Elimination (SVM-RFE)

The above regression methodologies can be categorized into two wide categories of linear and non-linear models since it was our intention to include a variety of algorithms that could well describe the submitted data. Three of the above methodologies were introduced in D4.1, namely LASSO, ENET, RF methodologies. Others, such as LM and PLS models were also introduced for the OpenTox infrastructure, however we are also including them into RRegrs to make use of the unified output produced as well as its model comparison capabilities.

Also, the RFE methodology was introduced in D4.1 as an important feature selection algorithm, particularly it is a backwards variable selection method that fits the model to all predictors and each predictor is ranked based on its importance to the model. Recursive feature elimination (RFE) is a backwards variable selection method. The RFE algorithm fits the model to all predictors, where each predictor is ranked using its importance to the model. During each iteration of the feature selection, the S top ranked predictors are retained, the model is refit and performance is assessed. The predictor rankings could be recomputed on each reduced feature set, which would generally increase performance, although it has been shown that in some algorithms (e.g. Random Forest) there is a decrease in performance[57]. Tuning parameters include the specification of the number of features that should be retained, and the external resampling method used (options are bootstrap, LOOCV, CV, repeated LOOCV).

Embedded Cross Validation Schemes

For each model, a CV scheme is introduced with two options: 10-fold repeated CV and LOO CV. In the case of repeated CV, we run 10 repeats of 10-fold CV for all models except SVM-RFE (3-folds, 1 repeat) and RF-RFE (5-folds, 1 repeat), which are considerable time-consuming methods. The procedure followed by caret and also introduced in RRegrs tool, randomly splits the data in K distinct blocks of roughly equal size (K = 10, 3, 5 depending on the method). Each block of data is left out sequentially, and the model is fit to the remaining of the data; this model is used to predict the held out block. The process is repeated where for each repetition a random proportion of the data are used to train the model (default value is 0.75) while the remainder is used for prediction. Average performance across the number of repeats, or across the LOO runs, are reported. When a model requires parameters selection, averaged values are returned per parameter (set) and the best model is selected based on the minimum RMSE statistic and the maximum R^2_{test} ≤0.005.

In order to further validate RRegrs test results, Y-randomization is applied to the best model found. For the last data split and the best model found, RRegrs performs Y-randomization for the 10- fold repeated CV scheme, and compares R2 test values to the best model corresponding value.

Parallel Computing Utility

RRegrs offers parallel support for calculations only in the more time-intense and computationally demanding methodologies, *i.e.* SVM, NN, RF, RF-RFE, SVM-RFE. A variable called *noCores* is included to indicate the number of CPU cores to be used for calculation with available options being 0=all available, 1=no parallel, n = specific number of cores. Depending on the operating system, different parallel R package will be needed, i.e. doMC package for Linux or Mac (http://cran.r-project.org/web/packages/doMC/index.html), doSNOW (http://cran.r-project.org/web/packages/doSNOW/index.html) and foreach (http://cran.r-project.org/web/packages/foreach/index.html) packages for Windows. When using RStudio in Windows, several processes will be created and if all the available cores will be used, the computer will become very slow (it is indicated the use of available cores-1 and the restart of RStudio to free the RAM between calculations).

Additional RRegrs Features

As was introduced in Figure 18, RRegrs includes scaling options, and particularly the following three options are available:

1. Normalization
2. Scaling
3. Custom filter function supplied by the user

Additionally, if prompted the data can be filtered by employing the near zero variance filter and the highly correlated features filter, which exclude independent variables when their variance is less than 0.05 or their pairwise correlation is higher than 0.9. The latter default value can be adjusted to user needs.

Applications

This section includes results from the application of RRegrs function to protein corona data (Walkey et al., 2014) and metal oxides (MeOx) data (Gajewicz et al. 2014, Puzyn et al.2011), as they were introduced in deliverables D4.1, D4.2. Further details are presented in (Tsiliki et al., 2015) which is currently under review. For the protein corona data, we present results on the initial set of 129 x 84 proteins to gold NPs data (21 neutral NPs were excluded from analysis as in Walkey et al.), and also on a set of 76x84 proteins to gold NPs data. These 76 proteins are selected from the authors with VIP≥0.6 threshold. For the MeOx data, we present results on the initial set of 32 parameters to the eighteen metal oxides. Because of the restricted number of samples and descriptors, RRegrs was applied without filtering options in this case, whereas the best model was selected between those that perform feature selection, i.e. GLM, LASSO, SVM-RFE, RF-RFE and ENET.

RRegrs was applied to 10 random splits of the data (75% train and 25% test) along with 10 Y randomization runs for the best model. Protein corona data were normalized and filtered using the RRegrs near zero variance and correlation filters, for that reason the 129 proteins are filtered to be 99 and the 76 proteins data set are reduced to 60 features. For metal oxides, data were normalized as in (Gajewicz et al. 2014). Table 1 shows the best model selected by RRegrs, its number of features, the adj.R^2 and the R^2 and RMSE values for the train and test sets, averaged over 10 random splits of the data. Table 2 shows the best model found in all data splits, i.e. we compare all methodologies and data splits to find the best R^2 test.

It can be observed from Table 2 that the highest value reported for protein corona data was R^2_{test} = 0.844 for individual split nine of the data set. For the data set with 129 proteins, the best model is an SVRM model with R^2_{test} = 0.631. When we study the set with 76 proteins, we find that the best model is an SVRM with averaged R^2_{test} = 0.728, whereas the best individual split value is R^2_{test} = 0.89. The corresponding RRegrs results for the PLS model are R^2_{test} = 0.7 (averaged over 10 data splits), whereas the highest values are reported for individual split five R^2_{test} = 0.885 (for repeated CV) and R^2_{test}= 0.873 (for LOO). Although the last number cannot be directly compared to R^2_{LOO} = 0.81 reported by the authors, it gives an indication of how our PLS implementation performs for the specific data set.

For MeOx data, the best performance model and the best averaged model is ENET, keeping on average 8.8 variables from the data including the two important variables (ΔH^c_f , χ^c) selected in the original publication. The ENET averaged statistics for 10 splits of the data are R^2_{test} = 0.746, R^2_{CV} = 0.933, which are very similar to the values reported by the authors. The best individual split value is equal to R^2_{test} = 0.998 for ENET model with eight variables including the final two suggested by the authors (LOO at the eighth split of the data).

Data set	Best model	Features no	adj.R^2	R^2_{CV}	R^2_{test}	$RMSE_{CV}$	$RMSE_{test}$
Protein corona (129x84)	SVRM	99	1.02	0.687	0.631	0.558	0.612
Protein corona (76x84)	SVRM	60	0.582	0.777	0.728	0.477	0.538
MeOX	ENET	8.8	>>1	0.933	0.746	0.639	0.639

Table 1: RRegrs averaged statistics reported for the three use cases, under the 10-fold repeated CV scheme. Averaged values are reported across the 10 different data splits.

Data set	Best model	Data split	Features no	Validation type	adj.R^2	$R^2_{CV/LOO}$	R^2_{test}	$RMSE_{CV/LOO}$	$RMSE_{test}$
Protein corona (129x84)	SVRM	5	99	LOO, CV	1.03	0.644/0.61	0.844	0.618/0.643	0.357
Protein corona (76x84)	SVRM	5	60	LOO, CV	0.407	0.767/0.741	0.89	0.525/0.527	0.296
MeOX	ENET	8	8	LOO	0.808	0.7	0.998	0.588	0.246

Table 2: RRegrs best model statistics. Both LOO and CV values are considered.

6. CONCLUSION

In this report we have described the infrastructure that has been developed for eNanoMapper nQSAR modelling purposes. The modifications and extensions to the API of the JQ web services have been detailed and the ease of incorporation of new machine learning algorithms in the framework has been emphasized. The modelling workflow for eNanoMapper users was presented using the Swagger interface, from experimental data bundle to a dataset and from there to building models and making predictions. Moreover, the array of modelling algorithms that have been made available from R, Python and WEKA has been presented. Lastly, the modelling infrastructure is complemented by the capabilities of RRegrs, a tool that aids users in model selection by scanning through a wide spectrum of options. A tutorial for RRegrs has been compiled and will be made available at the eNanoMapper website.

7. BIBLIOGRAPHY

1. Abraham, G.; Kowalczyk, A.; Loi, S.; Haviv, I.; Zobel, J. Prediction of breast cancer prognosis using gene set statistics provides signature stability and biological context. *BMC Bioinformatics* **2010**, *11*, 277 DOI: 10.1186/1471-2105-11-277.
2. Abràmoff, M. D.; Magalhães, P. J.; Ram, S. J. Image processing with imageJ. Biophotonics International, **2004**, 11, 36–41.
3. Aggarwal, C. C.; Gates, S. C.; Yu, P. S. On the merits of building categorization systems by supervised clustering. *Proc. fifth ACM SIGKDD Int. Conf. Knowl. Discov. data Min. - KDD '99* **1999**, 352–356 DOI: 10.1145/312129.312279.
4. Atkinson, A.C.; Donev, A.N. *Optimum experimental designs*. Clarendon Press, 1992; p. 328.
5. Balbin, O. A.; Prensner, J. R.; Sahu, A.; Yocum, A.; Shankar, S.; Malik, R.; Fermin, D.; Dhanasekaran, S. M.; Chandler, B.; Thomas, D.; et al. Reconstructing targetable pathways in lung cancer by integrating diverse omics data. *Nat. Commun.* **2013**, *4*, 2617 DOI: 10.1038/ncomms3617.
6. Bansal, N.; Blum, A.; Chawla, S. Correlation Clustering. *Mach. Learn.* **2004**, *56*, 89–113 DOI: 10.1023/B:MACH.0000033116.57574.95.
7. Bender, A.; Mussa, H. Y.; Glen, R. C.; Reiling, S. Similarity searching of chemical databases using atom environment descriptors (MOLPRINT 2D): Evaluation of performance. J. Chem. Inf. Comput. Sci. **2004**, *44*, 1708–1718 DOI: 10.1021/ci0498719.
8. Breiman, L. Random forests. *Mach. Learn.* **2001**, *45*, 5–32 DOI: 10.1023/A:1010933404324.
9. Brown, I. D.; McMahon, B. CIF: The computer language of crystallography. *Acta Crystallogr. Sect. B Struct. Sci.* **2002**, *58*, 317–324 DOI: 10.1107/S0108768102003464.
10. Burello, E.; Worth, A. P. A theoretical framework for predicting the oxidative stress potential of oxide nanoparticles. *Nanotoxicology*, **2011**, *5*, 228–235 DOI: 10.3109/17435390.2010.502980.
11. Chang, C.-C.; Lin, C.-J. LIBSVM: A Library for Support Vector Machines. *ACM Trans. Intell. Syst. Technol.* **2011**, *2*, 27:1–27:27 DOI: 10.1145/1961189.1961199.
12. Cheng, Y.; Church, G. M. Biclustering of expression data. *Proc. Int. Conf. Intell. Syst. Mol. Biol.* **2000**, *8*, 93–103.
13. Chin, L.; Gray, J. W. Translating insights from the cancer genome into clinical practice. *Nature* **2008**, *452*, 553–563 DOI: 10.1038/nature06914.
14. Fedorov, V. V. *Theory of optimal experiments*; Elsevier Science, 1972; p. 306.
15. Fourches, D.; Pu, D.; Tassa, C.; Weissleder, R.; Shaw, S. Y.; Mumper, R. J.; Tropsha, A. Quantitative nanostructure-activity relationship modelling. *ACS Nano*, **2010**, *4*, 5703–5712 DOI: 10.1021/nn1013484.
16. Frank *et al*, Data mining in bioinformatics using Weka, *Bioinformatics* **20**, 2479-81 (**2004**)
17. Free Software Foundation, GNU General Public License https://www.gnu.org/copyleft/gpl.html (accessed Nov 1, **2014**).
18. Gajewicz, A.; Schaeublin, N.; Rasulev, B.; Hussain, S.; Leszczynska, D.; Puzyn, T.; Leszczynski, J. Towards understanding mechanisms governing cytotoxicity of metal oxides nanoparticles: Hints from nano-QSAR studies. *Nanotoxicology* **2014**, *5390*, 1–13 DOI: 10.3109/17435390.2014.930195.
19. Ge, C.; Du, J.; Zhao, L.; Wang, L.; Liu, Y.; Li, D.; Yang, Y.; Zhou, R.; Zhao, Y.; Chai, Z.; et al. Binding of blood proteins to carbon nanotubes reduces cytotoxicity. *Proceedings of the National Academy of Sciences*, **2011**, *108*, 16968–16973 DOI: 10.1073/pnas.1105270108
20. Guazzelli, A. What is PMML ? Explore the power of predictive analytics and open standards. *IBM developerWorks*. **2010**, pp. 1–10 http://www.ibm.com/developerworks/library/ba-ind-PMML1/ba-ind-PMML1-pdf.pdf (accessed Oct 31, **2014**).
21. Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; Witten, I. H. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl*. **2009**, *11*, 10–18 DOI: 10.1145/1656274.1656278.
22. Hardy, B.; Douglas, N.; Helma, C.; Rautenberg, M.; Jeliazkova, N.; Jeliazkov, V.; Nikolova, I.; Benigni, R.; Tcheremenskaia, O.; Kramer, S.; et al. Collaborative development of predictive toxicology applications. J. Cheminform. **2010**, 2 DOI: 10.1186/1758-2946-2-7.
23. Hastie, T.; Tibshirani, R.; Friedman, J. *The elements of statistical learning data mining, inference, and prediction*; **2009**; p. 745 S.

24. Indahl, U. G.; Liland, K. H.; Næs, T. Canonical partial least squares-a unified PLS approach to classification and regression problems. *J. Chemom.* **2009**, *23*, 495–504 DOI: 10.1002/cem.1243.
25. Jamal *et al*, Cheminformatic models based on machine learning for pyruvate kinase inhibitors of *Leishmania mexicana*, *BMC Bioinformatics* **14**, 329 (**2013**)
26. Jaworska, J.; Nikolova-Jeliazkova, N.; Aldenberg, T. QSAR applicability domain estimation by projection of the training set in descriptor space: A review. *ATLA Alternatives to Laboratory Animals*, **2005**, *33*, 445–459.
27. Kapralov, A. A.; Feng, W. H.; Amoscato, A. A.; Yanamala, N.; Balasubramanian, K.; Winnica, D. E.; Kisin, E. R.; Kotchey, G. P.; Gou, P.; Sparvero, L. J.; et al. Adsorption of surfactant lipids by single-walled carbon nanotubes in mouse lung upon pharyngeal aspiration. *ACS Nano*, **2012**, *6*, 4147–4156 DOI: 10.1021/nn300626q.
28. Kaufman, L.; Rousseeuw, P. J. *Finding Groups in Data: an Introduction to Cluster Analysis*; John Wiley and Sons, **1990**; p. 342; DOI: 10.1002/9780470316801.
29. Kim, D.; Joung, J.-G.; Sohn, K.-A.; Shin, H.; Park, Y. R.; Ritchie, M. D.; Kim, J. H. Knowledge boosting: a graph-based integration approach with multi-omics data and genomic knowledge for cancer clinical outcome prediction. *J. Am. Med. Inform. Assoc.* **2014**, 1–10 DOI: 10.1136/amiajnl-2013-002481.
30. KNIME. KNIME and R, The best of two worlds. **2013** https://www.knime.org/files/kos-13/interactive_r_integration.pdf (Accessed Jun 30, 2015).
31. Kurczab *et al*, The influence of negative training set size on machine learning-based virtual screening, *J Cheminform* **6**, 32 (**2014**)
32. Lazzeroni, L., Owen, A. Plaid models for gene expression data http://statweb.stanford.edu/~owen/reports/plaid.pdf (accessed Nov 27, **2014**).
33. Lesniak, A.; Fenaroli, F.; Monopoli, M. P.; Åberg, C.; Dawson, K. A.; Salvati, A. Effects of the presence or absence of a protein corona on silica nanoparticle uptake and impact on cells. *ACS Nano* **2012**, *6*, 5845–5857 DOI: 10.1021/nn300223w.
34. Malkiewicz, K.; Pettitt, M.; Dawson, K. A.; Hansson, S. O.; Lynch, I.; Lead, J. Nanomaterials in reach. *Toxicol. Lett.*, **2011**, *205 Supplement*, S45 – DOI: http://dx.doi.org/10.1016/j.toxlet.2011.05.179.
35. Monopoli, M. P.; Walczyk, D.; Campbell, A.; Elia, G.; Lynch, I.; Baldelli Bombelli, F.; Dawson, K. A. Physical-Chemical aspects of protein corona: Relevance to in vitro and in vivo biological impacts of nanoparticles. *J. Am. Chem. Soc.,* **2011**, *133*, 2525–2534 DOI: 10.1021/ja107583h.
36. Netzeva, T. I.; Worth, A. P.; Aldenberg, T.; Benigni, R.; Cronin, M. T. D.; Gramatica, P.; Jaworska, J. S.; Kahn, S.; Klopman, G.; Marchant, C. A.; et al. Current status of methods for defining the applicability domain of (quantitative) structure-activity relationships. *ATLA Alternatives to Laboratory Animals,* **2005**, *33*, 155–173.
37. O'Boyle, N. M.; Morley, C.; Hutchison, G. R. Pybel: a Python wrapper for the OpenBabel cheminformatics toolkit. Chem. Cent. J. **2008**, *2*, 5 DOI: 10.1186/1752-153X-2-5.
38. Ooms, J. The OpenCPU System : Towards a Universal Interface for Scientific Computing through Separation of Concerns. *arXiv* **2014**, 1–23.
39. Ooms, J. The RAppArmor Package : Enforcing Security Policies in R Using Dynamic Sandboxing on Linux. *J. Stat. Softw.* **2013**, *55*.
40. Pechter, R. What's PMML and what's new in PMML 4.0? *ACM SIGKDD Explorations Newsletter*, **2009**, *11*, 19.
41. Periwal *et al*, Predictive models for anti-tubercular molecules using machine learning on high-throughput biological screening datasets, *BMC Res Notes* **4**, 204 (**2011**)
42. Piatetsky, G. R leads RapidMiner, Python catches up, Big Data tools grow, Spark ignites. 2015 http://www.kdnuggets.com/2015/05/poll-r-rapidminer-python-big-data-spark.html (Accessed Jun 30, 2015).
43. Puzyn, T.; Rasulev, B.; Gajewicz, A.; Hu, X.; Dasari, T. P.; Michalkova, A.; Hwang, H.-M.; Toropov, A.; Leszczynska, D.; Leszczynski, J. Using nano-QSAR to predict the cytotoxicity of metal oxide nanoparticles. *Nature nanotechnology*, **2011**, *6*, 175–178.
44. R Development Core Team. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/. R Found. Stat. Comput. Vienna, Austria. **2012**.
45. Roduner, E. Size matters: why nanomaterials are different. *Chem. Soc. Rev.* **2006**, *35*, 583–592 DOI: 10.1039/b502142c.
46. Rorabacher, D. B. Statistical Treatment for Rejection of Deviant Values : Critical Values of Dixon ' s " Q " Parameter and Related Subrange Ratios at the 95 % Confidence Level. Anal. Chem. **1991**, 139–146 DOI: 10.1021/ac00002a010.
47. RStudio. RStudio: Integrated development environment for R. *The Journal of Wildlife Management*, **2012**, *75*.

48. Saber Hussain, Christin Grabinski, Nicole Schaeublin, Elizabeth Maurer, Mohan Sankaran, Ravindra Pandey, Jerzy Leszczynski, W. T. *Toxicity Evaluation of Engineered Nanomaterials: Risk Evaluation Tools (Phase 3 Studies)*; **2012**; pp. 1–55.
49. Sarimveis, H.; Alexandridis, A.; Bafas, G. A fast training algorithm for RBF networks based on subtractive clustering. *Neurocomputing* **2003**, *51*, 501–505 DOI: 10.1016/S0925-2312(03)00342-4.
50. Sass, S.; Buettner, F.; Mueller, N. S.; Theis, F. J. A modular framework for gene set analysis integrating multilevel omics data. *Nucleic Acids Res.* **2013**, *41*, 9622–9633 DOI: 10.1093/nar/gkt752.
51. Scott, D.W. On optimal and data-based histograms, *Biometrika* **66**, 605-10 (1979)
52. Smith, D. R is Hot Revolution Analytics. **2014** http://www.revolutionanalytics.com/whitepaper/r-hot (Accessed Nov 27, 2014).
53. Smith, D. R Is Still Hot–and Getting Hotter. **2015** http://www.revolutionanalytics.com/sites/default/files/r-is-still-hot.pdf (Accessed Jun 30, 2015).
54. Steinbeck, C.; Hoppe, C.; Kuhn, S.; Floris, M.; Guha, R.; Wilighagen, E. Recent Developments of the Chemistry Development Kit (CDK) - An Open-Source Java Library for Chemo- and Bioinformatics. *Curr. Pharm. Des.* **2006**, *12*, 2111–2120 DOI: 10.2174/138161206777585274.
55. Stewart, J. J. P. MOPAC: A semiempirical molecular orbital program. *J. Comput. Aided. Mol. Des.* **1990**, *4*, 1–103 DOI: 10.1007/BF00128336.
56. Subramanian, A.; Subramanian, A.; Tamayo, P.; Tamayo, P.; Mootha, V. K.; Mootha, V. K.; Mukherjee, S.; Mukherjee, S.; Ebert, B. L.; Ebert, B. L.; et al. Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *Proc. Natl. Acad. Sci. U. S. A.* **2005**, *102*, 15545–15550 DOI: 10.1073/pnas.0506580102.
57. Svetnik, V.; Liaw, A.; Tong, C.; Wang, T. Application of Breiman's random forest to modelling structure-activity relationships of pharmaceutical molecules. *Mult. Classif. Syst.* **2004**, 334–343 DOI: 10.1007/978-3-540-25966-4_33.
58. Tcheremenskaia, O.; Benigni, R.; Nikolova, I.; Jeliazkova, N.; Escher, S. E.; Batke, M.; Baier, T.; Poroikov, V.; Lagunin, A.; Rautenberg, M.; et al. OpenTox predictive toxicology framework: toxicological ontology and semantic media wiki-based OpenToxipedia. *J. Biomed. Semantics* **2012**, 3 Suppl 1, S7 DOI: 10.1186/2041-1480-3-S1-S7.
59. Tibshirani, R. Regression Selection and Shrinkage via the Lasso. *Journal of the Royal Statistical Society B*, **1996**, *58*, 267–288.
60. Truszkowski *et al*, New developments on the cheminformatics open workflow environment CDK-Taverna, *J Cheminform* **3**, 54 (**2011**)
61. Tsiliki, G.; Munteanu, C.R.; Seoane, J.A.; Fernandez-Lozano, C.; Sarimveis, H.; Willighagen, E.L. RRegrs: An R package for Computer-aided Model Selection with Multiple Regression Models, Under review in *Journal of Cheminformatics*.
62. Walkey, C. D.; Olsen, J. B.; Song, F.; Liu, R.; Guo, H.; Olsen, D. W. H.; Cohen, Y.; Emili, A.; Chan, W. C. W. Protein corona fingerprinting predicts the cellular interaction of gold and silver nanoparticles. *ACS Nano* **2014**, *8*, 2439–2455 DOI: 10.1021/nn406018q.
63. Winkler, D. a; Mombelli, E.; Pietroiusti, A.; Tran, L.; Worth, A.; Fadeel, B.; McCall, M. J. Applying quantitative structure-activity relationship approaches to nanotoxicology: current status and future potential. *Toxicology*, **2013**, *313*, 15–23 DOI: 10.1016/j.tox.2012.11.005.
64. Yang, X.; Regan, K.; Huang, Y.; Zhang, Q.; Li, J.; Seiwert, T. Y.; Cohen, E. E. W.; Xing, H. R.; Lussier, Y. A. Single sample expression-anchored mechanisms predict survival in head and neck cancer. *PLoS Comput. Biol.* **2012**, *8* DOI: 10.1371/journal.pcbi.1002350.
65. Zou, H.; Hastie, T. Regularization and variable selection via the Elastic Net. *J. R. Stat. Soc. Ser. B* **2005**, *67*, 301–320.