Building and using a QSAR model in eTOXlab using the command line interface

(version updated for eTOXlab 0.97)

Here we describe a workflow for building and using a QSAR model in eTOXlab based on a data set of CACO-2 provided by Hou et al. in *J. Chem. Inf. Comput. Sci.*, 2004, 44 (5): 1585–1600 (http://dx.doi.org/10.1021/ci049884m). The aim is to illustrate some eTOXlab functionalities using the command line interface.

First we download and unzip the datasets into the virtual machine.

```
wget  http://pubs.acs.org/doi/suppl/10.1021/ci049884m/suppl_file/ci049884msi20040403_083100.zip

unzip ci049884msi20040403_083100.zip

Archive:  ci049884msi20040403_083100.zip
  inflating: test_set.sdf
  inflating: training_set.sdf
```

Files training_set.sdf and test_set.sdf are available.

We create a new endpoint called “ABCD” with description “/abcd/1”

```
manage --new -e ABCD -t "/abcd/1"
```

In order to define the properties of the model we request the default model with:

```
manage -e ABCD --get=model -v 0

mv imodel.py mymodel.py
```

A file named "mymodel.py" is available in the current folder. This file contains the "imodel" class that is a child of the "model" class, so methods defined here will override corresponding method of the parent "model" class. For the CACO2 model we will just edit the "init" method. In particular, we will define that the activity value is coded in the field “caco2” of the input SDFile, and a PLS model with 3LV should be used. The changes can be done with the idle editor.

```
idle mymodel.py
```

After changing imodel.py the result is:

```
# -*- coding: utf-8 -*-

##    Description    eTOXlab model template
##
##    Authors:       Manuel Pastor (manuel.pastor@upf.edu)
##
##    Copyright 2013-2016 Manuel Pastor
##
##    This file is part of eTOXlab.
##
##    eTOXlab is free software: you can redistribute it and/or modify
##    it under the terms of the GNU General Public License as published by
##    the Free Software Foundation version 3.
##
##    eTOXlab is distributed in the hope that it will be useful,
##    but WITHOUT ANY WARRANTY; without even the implied warranty of
##    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
##    GNU General Public License for more details.
##
##    You should have received a copy of the GNU General Public License
##    along with eTOXlab.  If not, see <http://www.gnu.org/licenses/>.

from model import model
```

```
class imodel(model):
    def __init__ (self, vpath):
        model.__init__(self, vpath)

        ##############################################################################################################
        ##
        ## General settings
        ##
        ##    General characteristics of the model
        ##
        ##############################################################################################################
        self.buildable    = True          # True if the model needs to be trained (built) before it can be used
                                          # for prediction

        self.quantitative = True          # True for quantitative dependent variables, False for qualitative
                                          # (e.g. 1 or 0) endpoints

        self.confidential = False         # True for building the model without storing any information about
                                          # the training series stuctures

        self.identity     = False         # If set to True, the structure of any query compound is compared with
                                          # those in the training series, returning the annotated value

        self.experimental = False         # If set to True, and the input file contains a field with the label
                                          # described by 'SDFileExperimental' this value is returned

        ##############################################################################################################
        ##
        ## Input setting
        ##
        ##    Labels of SDFile fields recognized by the program in the input file. These have the following
        ##    synthax. For example, if the SDFilenName field is 'name', the input file will contain this string
        ##    as shown:
        ##
        ##      > <name>
        ##      aspirin
        ##
        ##############################################################################################################
        self.SDFileName = 'name'          # Label of the SDFile field in the input file containing the name of
                                          # the molecule

        self.SDFileActivity = 'caco2'     # Label of the SDFile field in the input file containing the value of
                                          # the dependent variable. Leave blank for non-supervided models (PCA)

        self.SDFileExperimental = ''      # Label of the SDFile field in the input file containing the value to
                                          # be returned, typically an experimentally determined value

        self.SDFileMetadata = []          # Label of any SDFile field that must be passed throug the normalization
                                          # filters. Typically they contain information that is used by models

        ##############################################################################################################
        ##
        ## Normalization settings
        ##
        ##    Define how the input structures are normalized and transformed prior to being processed
        ##
        ##    These settings apply both to the stucture of the training series and query structures, which are
        ##    submitted to exactly the same normalization workflow
        ##
        ##    * Licenses: the program moka is third party software and requires a license activation
        ##
        ##############################################################################################################
        self.norm = False                 # If True the structures are normalized. When set to False the values
                                          # of all the following settings are ignored

        self.normStand = True             # If True the structures are submitted to the structural normalization
                                          # program 'standardizer'

        self.normNeutr = True             # If True the ionization of the structures is adjusted to the pH
                                          # detailed by 'normNeutr_pH' using the program defined below

        self.normNeutrMethod = 'moka'     # Name of the program used to adjust the ionization of the structures
                                          # when the value of 'normNeutr' is set to True

        self.normNeutr_pH = 7.4           # Value of the pH used to adjust the ionization of the structures
                                          # when the value of 'normNeutr' is set to True

        self.norm3D = True                # If True the structures are converted to 3D using the program 'corina'
                                          # irrespectivelly if the input structure is 2D or already 3D

        ##############################################################################################################
        ##
        ## Molecular descriptor settings
        ##
        ##    Define the program used to calculate the molecular descriptors (MD) from the input structures and
        ##    diverse parameters defining how these programs will carry out the molecular descriptor computation
        ##
        ##    * Licenses: the program pentacle is third party software and requires a license activation
```

```
##
#############################################################################################################
self.MD = 'padel'                   # Program used to calculate the MD. Must be one of the following
                                    # values: 'padel','pentacle' or 'adriana'

# padel relevant settings

self.padelMD = ['-2d']              # Type of padel MD to be calculared. Must be a subset of the values
                                    # in the following list, separated by a comma: '-2d' or '-3d'

self.padelMaxRuntime = None         # Timeout (in miliseconds) used by padel to compute a single molecule.
                                    # If the computation time excedes this value the MD are undefined

self.padelDescriptor = None         # Name of a xml formated filed containing the names of the MD computed
                                    # by padel

# pentacle relevant settings

self.pentacleProbes = ['DRY','O',
                       'N1','TIP']  # Name of the molecular probes used by program pentacle to compute
                                    # the molecular descriptors. Must be a subset of the values in the
                                    # following list, separated by a comma: 'DRY', 'O', 'N1' or 'TIP'

self.pentacleOthers = []            # Include here any command line parameter to be sent to the program
                                    # pentacle

#############################################################################################################
##
## Modeling settings
##
##    Define the modeling method used to build the predictive model and diverse parameters defining
##    how this method will work
##
#############################################################################################################
self.model = 'pls'                  # Name of the modeling method used to build the model. Must be one of
                                    # the following values: 'pls', 'pca', 'RF'

# pls relevant settings

self.modelLV = 2                    # Number of latent variables used to build the model

self.modelAutoscaling = True        # If true, the molecular descriptors will be normalized to unit
                                    # variance by dividing each value by the variable variance

self.modelCutoff = 'auto'           # Applicable only for PLS-DA ('quantitative' is False). The cutoff
                                    # value used to asign 1 or 0 to the predictions.
                                    # When set to 'auto' the value is assigned automatically as the one
                                    # producing the best compromise between sensitivity and specificity

self.selVar = False                 # If True the program will run the variable selection method
                                    # defined by 'selVarMethod'. This setting can extend significantly
                                    # the time required to build a model (by many hours or even days)

# variable selection relevant settings

self.selVarMethod = 'golpe'         # Name of the variable selection method. Must be one the following
                                    # values: 'golpe' (to be extended in following versions)

self.selVarLV = 2                   # Number of latent variables used to build the reduced models used
                                    # by the variable selection. Could be diferent than 'modelLV'

self.selVarCV = 'LOO'               # Name of the cross-validation method used during the variable
                                    # selection. Must be one the following values: 'LOO' (to be extended
                                    # in following versions)

self.selVarRun = 3                  # Number of maximumm sequential runs of the GOLPE algorithm to run
                                    # before stop. The algorithm will run only until the model predictive
                                    # ability (assesed by cross-validation) will no longer improve

self.selVarMask = None              # Name of a file containing a previously computed mask of selected and
                                    # non selected variables
```

Now we build the model with the training data set with

```
build -e ABCD -f training_set.sdf -m mymodel.py

('training_set.sdf', 77)
Progress: [####################] 100.00% Done...
cross-validating...
LV 1 R2:0.439 Q2:0.291 SDEP:  0.628
LV 2 R2:0.538 Q2:0.305 SDEP:  0.623
LV 3 R2:0.718 Q2:0.218 SDEP:  0.660
Model OK
```

The next step is to use the model to predict the test dataset. The output is the predicted CACO2 value, an applicability domain index and the 95% CI for the predicted value.

```
predict -e ABCD -f test_set.sdf  -v 0

-5.83198 0  1.29419
-6.45577 2  2.58838
-4.05182 4  0.00000
-4.83560 0  1.29419
-5.53917 2  2.58838
-6.73768 2  2.58838
-6.42743 1  1.29419
-4.65175 0  1.29419
-5.06969 0  1.29419
-4.83893 0  1.29419
-6.17705 3  2.58838
-4.23142 0  1.29419
-5.19872 0  1.29419
-4.29936 0  1.29419
-5.09670 2  2.58838
-5.39153 0  1.29419
-5.13877 0  1.29419
-5.96280 3  2.58838
-4.67481 0  1.29419
-6.06141 2  2.58838
-5.58145 1  1.29419
-5.72423 2  2.58838
-4.30183 5  0.00000
```

Once the model is ready to use we can publish it to create a copy of this version on the model repository

```
manage -e ABCD  --publish

/home/modeler/soft/eTOXlab/src/ABCD/version0001
```

Stored versions can be exposed as web services and then be used from outside of the virtual machine. This only requires to type

```
manage -e ABCD -v 1 –expose=1

version exposed OK
```

Annex II. Building and using a QSAR model in eTOXlab using the GUI interface

Here we describe a workflow for building and using a QSAR model in eTOXlab based on a data set of CACO-2 provided by Hou et al. in *J. Chem. Inf. Comput. Sci.*, 2004, 44 (5): 1585–1600 (http://dx.doi.org/10.1021/ci049884m). The aim is to illustrate some eTOXlab functionalities using the graphic user interface (GUI).

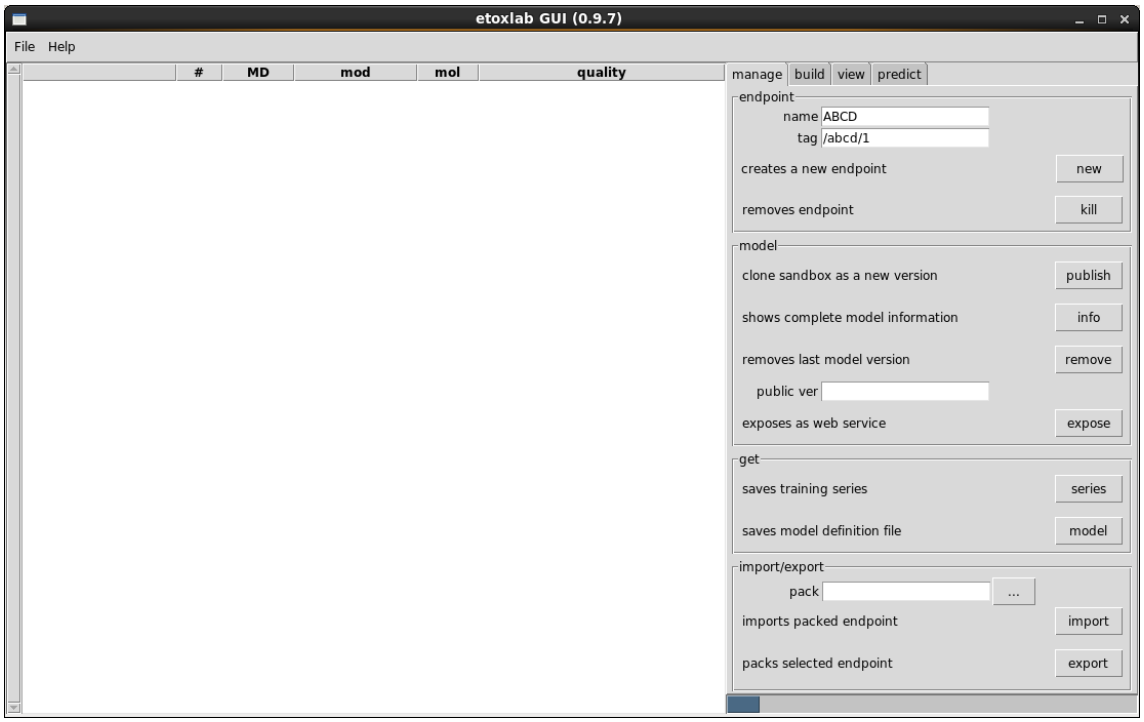
First we download and unzip the datasets into the virtual machine.

```
wget  http://pubs.acs.org/doi/suppl/10.1021/ci049884m/suppl_file/ci049884msi20040403_083100.zip

unzip ci049884msi20040403_083100.zip

Archive:  ci049884msi20040403_083100.zip
  inflating: test_set.sdf
  inflating: training_set.sdf
```

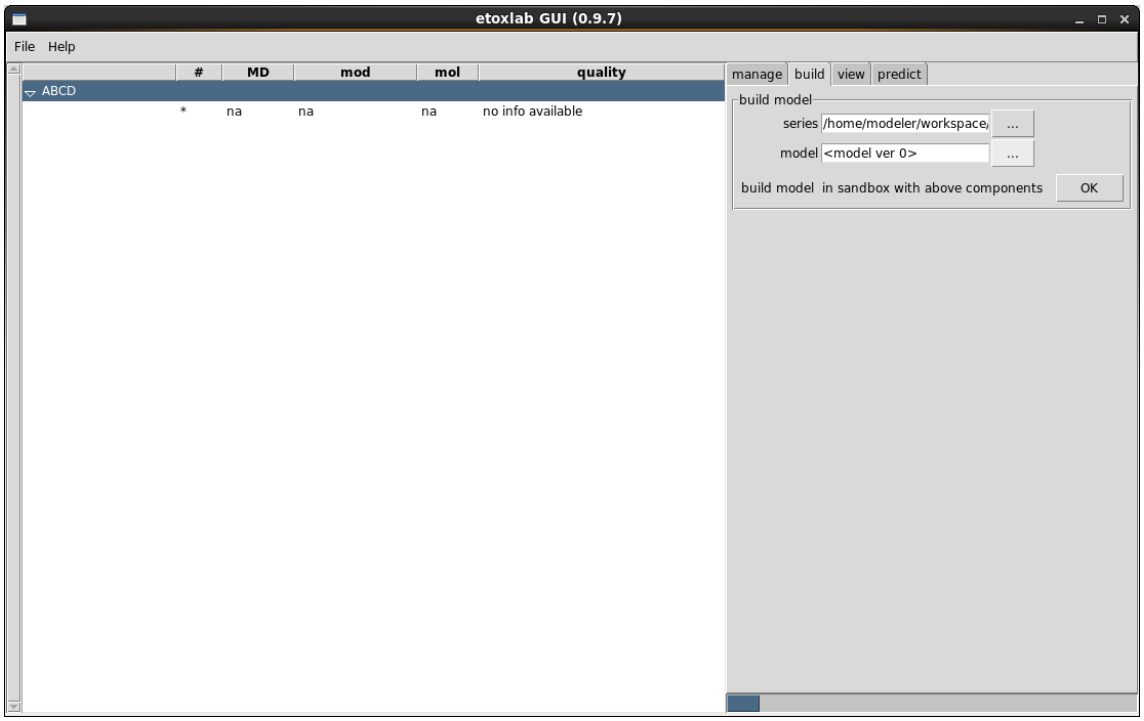
Files training_set.sdf and test_set.sdf are available.

We start the graphical interface by clicking on the "etoxlab" icon in the desktop.

For creating a new endpoint called ABCD with description "/abcd/1" we simply type these values in the "name" and "tag" input fields on the upper right corner and press the "new" button



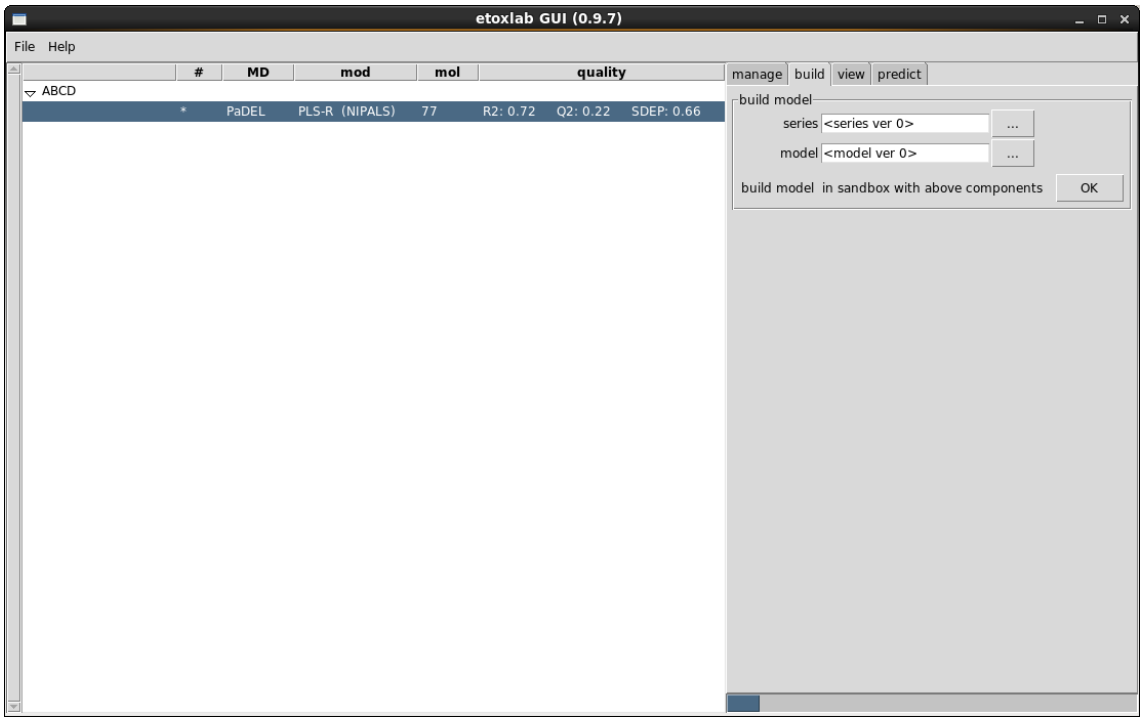
The new endpoint is created and shown in the list of existing endpoints and models



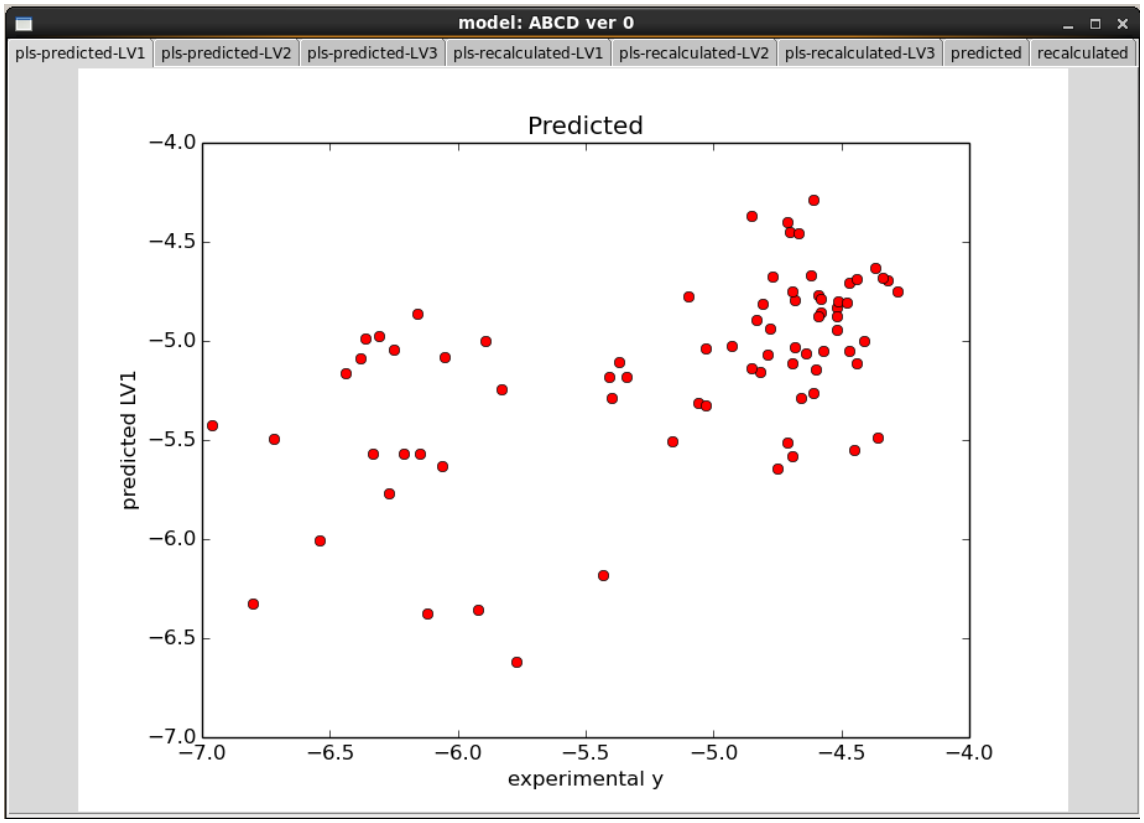
We build the model by selecting the "build" tab. Select the training series "training_set.sdf". Regarding the model, we must configure that the activity value is coded in the field “caco2” of the input SDFile, no normalization should be performed, PaDel descriptors should be used, and a PLS model with 3LV should be used. Pressing the "..." button besides the model will open an idle editor where we can introduce the changes mentioned in Annex I, that basically consist in:

- Setting "self.SDFileActivity" to "caco2"
- Setting "self.modelLV" to 3

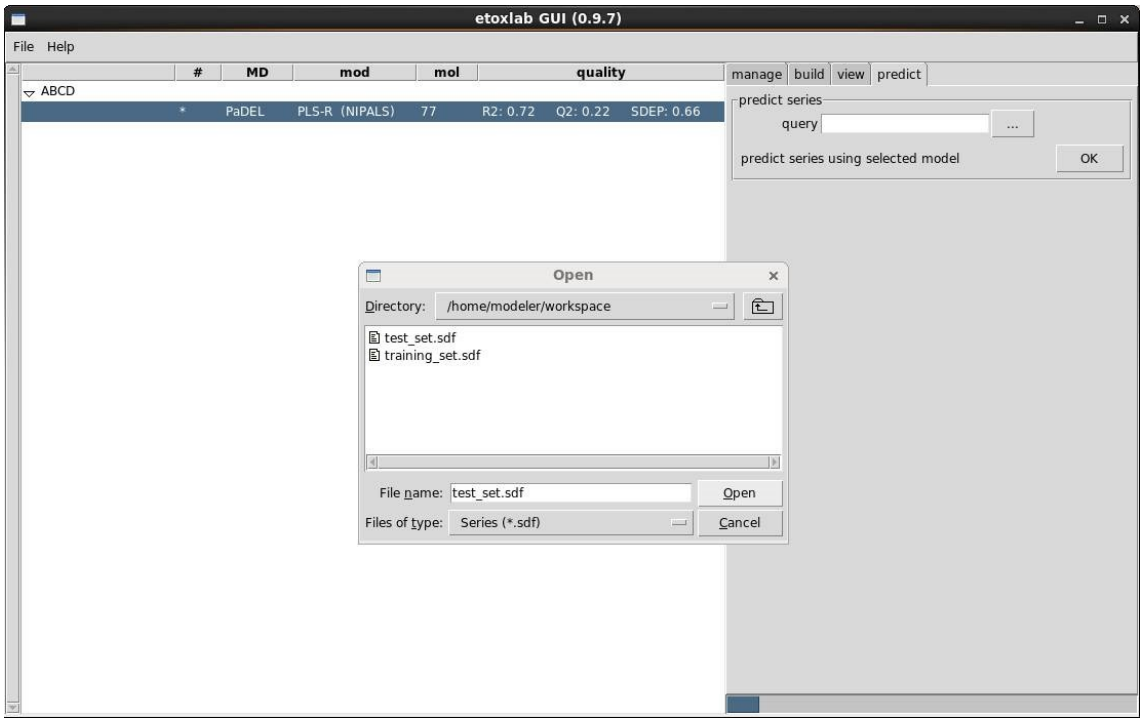
Then we press OK and wait for a few minutes.



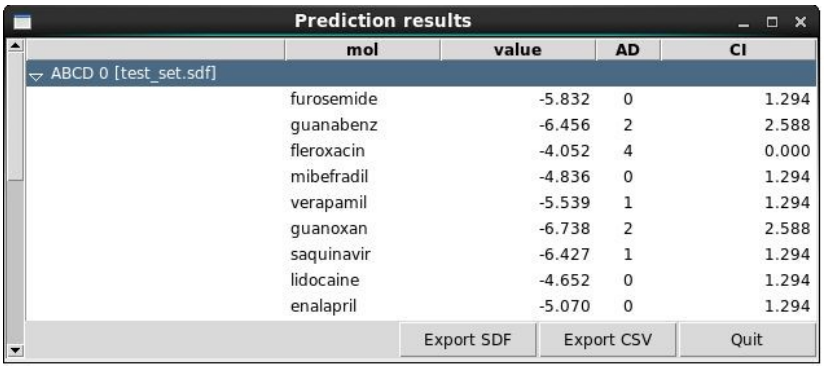
When the model building is finished, the values of R^2 and Q^2 are shown in the model tree. Additionally, the GUI generates and displays scatter-plots with the experimental vs recalculated and experimental vs predicted values for all model dimensionalities, which are useful for diagnosing the model quality.



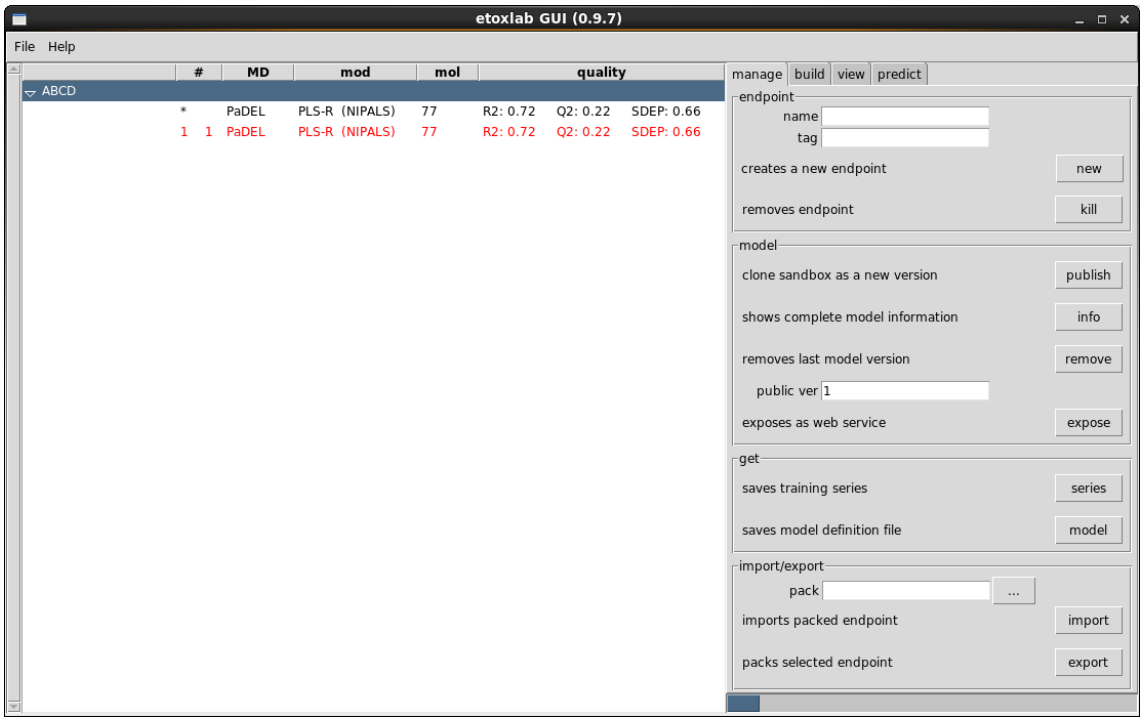
The new model can be used for prediction immediately. Select the "predict" tab, enter the name of the query series and press the OK button.



The prediction results are shown in a separate window from where the results can be exported in CSV format or as an annotated SDFile.



Once the model is ready to use we publish it to create a copy of this version on the model repository. This can be done from the manage tab simply selecting the version and pressing the "publish" button.



Stored versions can be exposed as web services and used from outside of the virtual machine simply selecting the model version, entering “1” as public version number and pressing the "expose" button. Exposed versions are highlighted in red with the public version number in front

Annex III. Example of method overriding in eTOXlab

In order to illustrate the method overriding technique we present here how to implement in eTOXlab a very simple rule-base prediction method. This method was extracted from Tomizawa K. et al. Physicochemical and cell-based approach for early screening of phospholipidosis-inducing potential. *J Toxicol Sci.* 2006, 31 (4):315-24. (http://www.ncbi.nlm.nih.gov/pubmed/17077586)

A rule-based method does not require building a model. Therefore, we only need to override methods of the prediction workflow in the *model* class.

The procedure starts exactly as described in Annex I and II, but the editing of the local mymodel.py requires inserting the definition of the new methods, as described below. Please note that in no case we need to edit the original source code, just add the following text at the bottom of the mymodel.py file.

We begin by adding a couple of lines at the top of the file, just after 'from model import model' to allow our model to access functions which will be called by our methods

```
from model import model

from logp import *
from utils import removefile
```

Then we edit the method "predict", which has been simplified to execute only two tasks: call "computeLogP" and use the results to call a new version of "computePrediction".

Then we write the code of "computePrediction". This latter method applies a simplified version of the rules described in the original article; if the compound is neutral or negatively charged, it is considered phospholipidosis negative. Compounds with charge +2 or compound with charge +1 and logP higher than 1.61 are considered phospholipidosis positive. Compounds with formal charges higher than +2 are considered out of the prediction range.

```
    def computePrediction (self, logP, charge):
        result = 'negative'

        if charge == 1:
            if logP[0] >= 1.61 :
                result = 'positive'
        elif charge == 2 :
            result = 'positive'
        elif charge > 2 :
            return (False, 'charge out of range')

        return (True, result)


    def predict (self, molFile, molName, molCharge, detail, clean=True):
        # default return values
        molPR=molCI=molAD=(False,0.0)

        success, molMD = computeLogP (molFile)

        if not success: return (molPR,molAD,molCI)

        success, pr  = self.computePrediction (molMD,molCharge)
        molPR = (success, pr)
        if not success: return (molPR,molAD,molCI)

        if clean: removefile (molFile)

        return (molPR,molAD,molCI)
```

There are other two methods that must be overridden in this example: "setSeries" and "log". These simply avoid storing information about the training series (non-existing in this case) and provide information about the methods used to generate the prediction ("Decision tree").

```
    def setSeries (self, molecules, numMol):
        self.infoSeries = []


    def log (self):
        self.infoModel = []
        self.infoModel.append( ('model','Decision tree') )

        result = model.log(self)
        return (result)
```

Once the editing has been completed, the model is ready for testing. In this particular case, there is no need to build the model and it can be used for prediction directly.

```
cp mymodel.py /opt/eTOXlab/src/ABCD/version0000/imodel.py

predict -e ABCD -f test_set.sdf  -v 0
```

Annex IV. Demo Application Programming Interface

In the demo VM, the web service is accessible at port 9001. From inside, it can be called by any browser calling to http://localhost:9001, followed by a valid URL

URL	HTTP verb	Input data	return data	HTTP status codes
/info	GET		application/json: info_message response	200
/available_services	GET		application/json: available_services response	200
/predictform	GET		text/html: web form	200
/calculate	POST	multipart/form-data encoding: - model tag - SDFile	application/json: calculate_call response	200 500 (in case of malformed POST message)

/info
Returns basic info about the provider of the models

Example of "info_message response" schema:

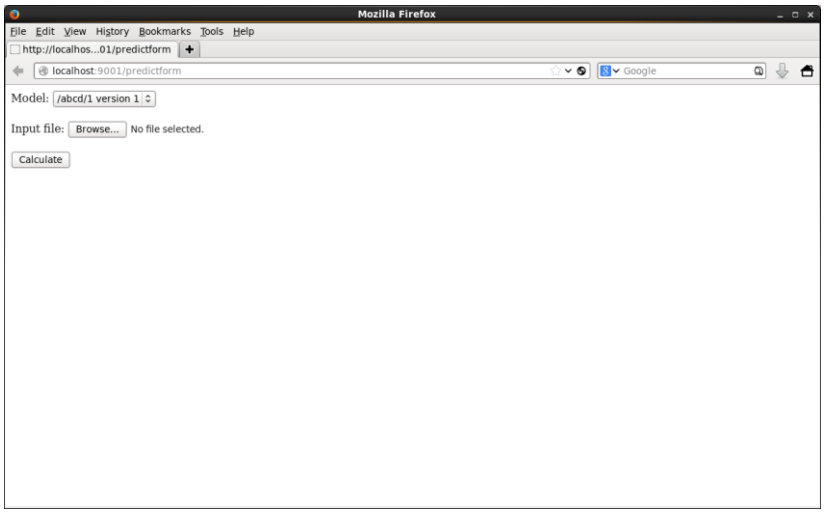
```
{"provider": "FIMIM",
"homepage": "http://phi.imim.es",
"admin": "Manuel Pastor",
"admin-email": "manuel.pastor@upf.edu" }
```

/available_services
Returns the list of all available prediction services.
The predictions field in will contain an array of the tags of the models available to make predictions.

Example of "available_services response" schema:

```
{"predictions" : ["ABCD"] }
```

/predictform
Shows a form allowing the user to make predictions. The user is asked to select the model and version to use and to upload the SDFile with the 2D structures of the query compounds



/calculate

Returns the prediction for a model and a SDFile provided by the user.
The model is specified by the tag provided by the "/available_services" call.
The model tag and the SDFile are encoded as multipart/form-data.
To encode the tag we use the “model” field and to encode the SDFile the “uploadfile” field.

Example of "calculate call response" schema:


```
[
        {"cmp_id": "0",
         "success": "True",
         "value": "-6.0743503303",
         "AD":   {
                 "success": "True",
                 "value": "0",
                 "message": ""
                 },
         "RI": {
                 "success": "True",
                 "value": "1.14595589767",
                 "message": ""
                 }
        },
        {"cmp_id": "1",
         "success": "True",
         "value": "-5.33778477437",
         "AD":   {
                 "success": "True",
                 "value": "1",
                 "message": ""
                 },
         "RI":   {
                 "success": "True",
                 "value": "1.14595589767",
                 "message": ""
                 }
        }
]
```


Therefore, for every compound we obtain:

- "compound_id": index of the compound in the SDFile.
- "value": the value of the prediction
- "AD": applicability domain of the prediction (in ADAN method format, see manuscript)
- "RI": reliability index of the prediction (95%CI)

For every "value", "AD" and "RI" there is a "success" value that indicate if the computation was correct ("True") or not ("False")

The aspect of the output in the web interface is as follows: