Getting started with eTOXlab version 0.90

Introduction

eTOXlab has a simple command mode interface. There are only three commands: build, predict and manage, corresponding to the main tasks involving in the building, use and maintenance of models (respectively). Typing the name of the command will show a brief reminder of the command syntax and the available arguments.

In the following sections we provide a guided tour by the main features of the program, describing step-by-step how it is used in practice.

The Annex contains a more formal reference manual describing the commands, the syntax of the arguments and all the options.

Modeling environment

Working at the place where the eTOXlab source code is located is not a good idea. Building and using models requires handling many files and if we work in this directory we risk removing important files.

For this reason we have created the directory workspace at the home of the modeler user. You can move to this directory:

```
cd workspace
```

You can easily access eTOXlab from here without the need of introducing complicated paths. Also, if you want to work in diverse projects, feel free to create your own subdirectories.

Listing existing models

Type:

```
manage --info=short
```

This will display a list of all models developed in this VM. Please note that models are organized by the biological property they represent (“endpoint”). We can build many different models for the same endpoint (“versions”).

We can obtain a more detailed list using:

```
manage --info=long
```

If we only want to obtain information for the versions of a given endpoint we can specify it with the -e argument followed by the endpoint label:

```
manage -e ABCD --info=long
```

Building your first eTOXlab model

We start by telling eTOXlab that we want to build a new model and assign a new endpoint name and label:

```
manage -e DEMO --new -t “demo model”
```

DEMO is the endpoint name and will be used to identify the model within eTOXlab. The public name of this endpoint is “demo”. We suggest using very short, uppercase text for the name and longer descriptions for the public name.

This command creates a directory tree in the filesystem where all the model versions for this endpoint will be permanently stored.

eTOXlab can be used to build any kind of predictive model. The whole definition of how the model is built is defined by the Python class “imodel”. This is a child of the generic class “model” and inherits all its methods and variables. Every eTOXlab model has its own private copy of “imodel” in a separate “imodel.py” file. When we created a new model (as done above), a default imodel.py version is created, with default options.

The simplest way to create a model is to accept all the default options. We only need to provide a training series in a single SDFile and eTOXlab will build a simple QSAR model using PLS as machine learning method:

```
build -e DEMO -v 0 -f training_series.sdf
```

Here

- -v 0 indicates that we will use the unmodified “imodel.py” created by default in version 0 directory.
- -f training_series, indicates that we will use this SDFile as the training series. It must be a regular 2D or 3D SDFile, where the biological property is included in a field tagged as “<activity>” (this can be changed, as described below).

This command will build and validate the model, showing a report of the model quality and generating a few plots representing the values of the experimental property versus the calculated and predicted (cross-validation, LOO) values.

Understanding versions

The version “0” is a kind of “modeling sandbox” and is assigned every time we build a model. Whenever we are satisfied with a version, we can publish it and eTOXlab will assign it a sequential number using:

```
manage -e DEMO --publish
```

The last version is assumed to be the best version and is the one exposed as a prediction web service. The version 0 is never exposed as web service, therefore, models must be published explicitly for being used by the web service.

Customizing your model

In most cases, defaults will not provide the optimum results. The simplest method for customizing the models is adjusting the tunable parameters of the embedded methods. First we need to get a copy of the imodel class:

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

This will copy the imodel.py file to the current directory. Then we can edit it using for example:

```
gedit imodel.py
```

The header contains a list of variables that control the options of diverse tasks involved in building the model, like the structure normalization, the generation of molecular descriptors or the machine learning method. The label used to identify the activity in the SDFiles is also defined here.

In this example, we will simply adjust the number of latent variables used in the PLS from the default (2) to 3. Simple edit the following line replacing the “2” by a “3”:

```
self.modelLV = 2
```

Then rebuild the model using your new settings

```
build -e DEMO -v 0 -m ./imodel.py
```

Notice that we don’t need to reintroduce the name of the SDFile; it is already stored. Also, since the changes do not involve any change in the molecular descriptors, the model is rebuilt in a few seconds using 3 LV instead of 2.

Feel free to experiment with the options. Every time the model can be rebuilt as explained. Once you were satisfied with the results, publish the model using:

```
manage -e DEMO --publish
```

Making your first prediction

Published models are automatically visible from the outside and can be used for predicting the properties of new compounds making use of the web services implemented in the VM.

However, for testing purposes, the predict command allows to run predictions from inside the VM:

```
predict -e DEMO -v last -f test.sdf
```

This command will use the last published version for endpoint DEMO and present the results in the screen.

```
-6.10154 0  1.08282
-5.22064 0  1.08282
```

For every compound in the SDFile, the model shows the predicted value of the biological property, followed by two numbers that represent how far is the compound from the model applicability domain (0-6, 0 means close to the AD, 6 far from the AD) and the estimated 95% CI of the prediction. These are the results of the ADAN analysis. Further details can be obtained from the original ADAN reference.

Retraining your eTOXlab model with new data

One of the most frequent model maintenance operations is rebuilding the models with new training series containing more compounds or better biological property results. Typically, this task requires no model adjustment, just re-run the model building command with a new training series:

```
build -e DEMO -v last -f new_training.sdf
```

If the model quality is satisfactory, the new model can be published, thus replacing the old one for external predictions, simply using:

```
manage -e DEMO --publish
```

Advanced model customization

The true power of eTOXlab resides in the possibility to customize completely how the model is built and used for prediction. The model classes can be overridden and replaced in the imodel.py model by others that, for example, apply a different normalization protocol to the compounds, compute molecular descriptors using commercial software or build models using R code. There are no limits to how much the model can be modified. Even so, provided that the changes do not alter the format of the classes input and output the whole system keeps the parent class functionality.

A deeper description of how this customization is performed is beyond this short introduction, but we have included in the distribution a few models that apply a heavy customization and that can be used by interested readers as templates for their own work.

Building a model in confidential mode

The eTOXlab models can be built using a special option that guarantees that the model keeps no trace of the training series structures. This option is useful for building models that are trained with confidential datasets (typically, confidential structures from a pharmaceutical company) by installing a VM behind the company firewalls. The resulting model can be exported and shared with other companies with the absolute guarantee that the structures of the training set are disclosed.

The procedure is very simple. We start for creating a new model using the option “conf”, instead of “new”:

```
manage -e SECRET --conf -t “secret model”
```

Then we build the model as usual:

```
build -e SECRET -v 0 -f training_series.sdf
```

The new model can be exported using the manage command:

```
manage -e SECRET --export
```

This creates a compressed file called SECRET.tgz, containing inside only the coefficients of the PLS model in human readable format. This file can be opened and audited to make sure it does not contain any sensitive information. Then it can be exported by e-mail or using a portable device and imported into another VM installed outside the company firewalls, copying it to the local directory and using:

```
manage -e SECRET --import
```

This option will create the endpoint tree and all the models, thus allowing running predictions immediately without any further operation.

One of the limitations of this approach is that the ADAN method cannot be used to estimate the prediction reliability and therefore this information is not provided for models generated in confidential mode.

---

manuel.pastor@upf.edu *Manuel Pastor, April 2014*

Annex. Reference manual of eTOXlab commands.

build.py -e TEST -v 0|last -f mytrain.sdf -m imodel.py

-e TEST Endpoint label. We recommend using a short label in uppercase.

-f mytrain.sdf Name of a SDFile with all the compounds in the training series. Must contain the activity in numeric format, in a field labeled as <activity>.

-m imodel.py Name of a model definition file in Python, based on the provided templates.

-v 0|last Missing elements required for building the model will be taken from corresponding version folder. The version is provided as a number or as the text "last" to make reference to the last model version.

This command is used to build a new predictive model, using the training set and the model definition file defined by the arguments, as described above.

New models are always created into the “version0000” directory, overwriting previously unpublished models for this endpoint. The user can refine the model and once satisfied the model can be published using the command “*manage -e TEST --publish”*

New versions are added incrementally as version 1, 2, …

When the command contains the name of a training series or the specified model definition changes the normalization of previous models, this command will run the steps of the workflow required to regenerate the matrix of descriptors. Otherwise, the command only rebuilds the mathematical model.

Examples of use:

build -e TEST -f new.sdf -m imodel.py

This will build a new model using the provided training (new.sdf) and model (imodel.py) files.

build -e TEST -v last -f new.sdf

This will build a new model using the imodel.py from last version.

build -e TEST -v 2 -f new.sdf

This will build a model using new.sdf as the training series and the imodel.py obtained from version0002 directory

build -e TEST -v last -m imodel.py

This builds a new model using the training.sdf from last version

predict.py -e TEST -v 0|last -f mols.sdf

-e TEST Endpoint label

-**v 0|last** Number of the model version used for the prediction or the label “last” for indicating the last version

-f mols.sdf Name of a SDFile containing one or more molecules

This command is used to carry out a prediction on the molecules provided as argument using the specified model.

Examples of use:

predict -e TEST -v last test.sdf

This predicts the modeled endpoint for all compounds in test.sdf using last model.

manage.py -e TEST -v 0|last --publish|new|conf|remove|import|export|version|info=[short|long] |get=[model|series][-t tag]

This command performs diverse eTOXlab administrative tasks, including creating new endpoint trees, publishing model versions and showing model information. It is strongly recommended that all the eTOXlab model manipulation is carried out using this command.

-e TEST Endpoint label. This argument is compulsory for “publish”, “new”, “remove” and “get”, and optional for “info”.

-v 0|last Number of the model version used or “last” to indicate the last model version. This argument is compulsory for “get” and optional for “info”.

-- publish Creates a new version starting from the model present in the “version0000” folder of a given endpoint. This procedure copies all the contents of the folder, including the training set and the model definition files. The number of the new version is created incrementally over previous versions (e.g. if the last version is the 2, this procedure will create version 3)

--new Creates a new endpoint (e.g. TEST in the example), with an empty “version0000” folder. This command requires the “-e” argument for assigning the endpoint name as well as the “-t” argument for defining the label of this endpoint in the eTOXsys web service

--conf Equivalent to “new” option, but the model is created in “confidential mode”

--remove Removes the last model version for a given endpoint. All the contents of the folder and the folder itself will be removed. The version 0 cannot be removed with this command. The actions of this command cannot be undone, please use with extreme care.

--import Reads from the current directory an export file created by eTOXlab with the name of the endpoint, provided by the -e option, and the extension “tgz” (e.g. TEST.tgz), creates the endpoint file tree and all the associated files.

--export Creates in the current directory an export file with the name of the endpoint, provided by the -e option, and the extension “tgz” (e.g. TEST.tgz) containing all the models for this endpoint.

--version Shows the current eTOXlab version

--info=short|long Presents information about the model or models defined by the endpoint and version arguments, using the specified format. If no endpoint is specified, the

command shows models for all the endpoints. If no version is specified, the command shows all the model versions.

--get=model|series Retrieves the required information from the model defined by the endpoint and version arguments. Both of these are required. If used with the “model” argument, the command copies the model definition file “imodel.py” to the current directory. If used with the “series” argument copies the training series “training.sdf” to the current directory. Please notice that this last file is a verbatim copy of the original file used to build the model, before the normalization.