

Flame

Flame is a flexible framework supporting predictive modeling within the eTRANSAFE (<http://etransafe.eu>) project.

Flame allows to:

- Easily develop machine-learning models, for example QSAR-like models, starting from annotated collections of chemical compounds stored in standard formats (i.e. SDFiles)
- Transfer new models into a production environment where they can be used by web services to predict the properties of new compounds.

Flame is in active development and **no stable release has been produced so far**. Even this README is under construction, so please excuse errors and inaccuracies.

Installation

Flame can be used in most Windows, Linux or macOS configurations, provided that a suitable execution environment is set up. We recommend, as a first step, installing the Conda package and environment manager. Download a suitable Anaconda anaconda distribution for your operative system from [here](#).

Download the repository:

```
git clone https://github.com/phi-grib/flame.git
```

Go to the repository directory

```
cd flame
```

and create the **conda environment** with all the dependencies and extra packages (numpy, RDKit...):

```
conda env create -f environment.yml
```

Once the environment is created type:

```
source activate flame
```

to activate the environment.

Conda environments can be easily updated using a new version of the environment definition

```
conda env update -f new_environment.yml
```

Main features

- Native support of most common machine-learning algorithms, including rich configuration options and facilitating the model optimization.
- Support for any standard formatted input: from a tsv table to a collection of compounds in SMILES or SDF file format.
- Multiple interfaces adapted to the needs of different users: as a web service, for end-user prediction, as a full featured GUI for model development, as command line, integration in Jupyter notebooks, etc.
- Support for parallel processing.
- Integration of models developed using other tools (e.g. R, KNIME).
- Support for inter-model communication: the output of a model can be used as input for other models.
- Integrated model version management.

Quickstarting

Flame provides a simple command-line interface `flame.py`, which is useful for accessing its functionality and getting acquainted with its use.

You can run the following commands from any terminal, in a computer where flame has been installed and the environment (flame) was activated (`source activate flame` in Linux, `activate flame` in Windows)

Let's start creating a new model:

```
python flame.py -c manage -a new -e MyModel
```

This creates a new entry in the model repository and the development version of the model, populating these entries with default options. The contents of the model repository are shown using the command.

```
python flame.py -c manage -a list
```

Building a model only requires entering an input file formatted for training one of the supported machine-learning methods. In the case of QSAR models, the input file can be an SDF file, where the biological property is annotated in one of the fields.

The details of how Flame normalizes the structures, obtains molecular descriptors and applies the machine-learning algorithm are defined in a parameters file (*parameter.yaml*) which now contains default options. These can be changed as we will describe later, but for now let's use the defaults to obtain a Random Forest model on a series of 100 compounds annotated with a biological property in the field <activity>:

```
python flame.py -c build -e MyModel -f series.sdf
```

After a few seconds, the model is built and a summary of the model quality is presented in the screen.

This model is immediately accessible for predicting the properties of new compounds. This can be done locally using the command:

```
python flame.py -c predict -e MyModel -v 0 -f query.sdf
```

And this will show the properties predicted for the compounds in the query SDFFile.

In the above command we specified the model version used for the prediction. So far we only have a model in the development folder (version 0). This version will be overwritten every time we develop a new model for this endpoint. Let's imagine that we are very satisfied with our model and want to store it for future use. We can obtain a persistent copy of it with the command

```
python flame.py -c manage -a publish -e MyModel
```

This will create model version 1. We can list existing versions for a given endpoint using the list command mentioned below

```
python flame.py -c manage -e MyModel -a list
```

Now, the output says we have a published version of model MyModel.

Imagine that the model is so good you want to send it elsewhere, for example a company that wants to obtain predictions for confidential compounds in their own computing facilities. The model can be exported using the command

```
python flame.py -c manage -a export -e MyModel
```

This creates a very compact file with the extension .tgz in the local directory. It can be sent by e-mail or uploaded to a repository in the cloud from where the company can download it. In order to use it, the company can easily install the new model using the command

```
python flame.py -c manage -a import -e MyModel
```

And then the model is immediately operative and able to produce exactly the same predictions we obtain in the development environment

Flame commands

Command	Description
-c/ --command	Action to be performed. Acceptable values are <i>build</i> , <i>predict</i> and <i>manage</i>
-e/ --endpoint	Name of the model which will be used by the command. This name is defined when the model is created for the first time with the command <i>-c manage -a new</i>
-v/ --version	Version of the model, typically an integer. Version 0 refers to the model development “sandbox” which is created automatically upon model creation
-a/ --action	Management action to be carried out. Acceptable values are <i>list</i> , <i>new</i> , <i>kill</i> , <i>publish</i> , <i>remove</i> , <i>export</i> and <i>import</i> . The meaning of these actions and examples of use are provided below
-f/ --infile	Name of the input file used by the command. This file can correspond to the training data (<i>build</i>) or the query compounds (<i>predict</i>)

-h/ --help	Shows a help message on the screen
------------	------------------------------------

Management commands deserve further description:

Management commands

Command	Example	Description
new	<i>python -c manage -a new -e NEWMODEL</i>	Creates a new entry in the model repository named NEWMODEL
kill	<i>python -c manage -a kill -e NEWMODEL</i>	Removes NEWMODEL from the model repository. Use with extreme care , since the program will not ask confirmation and the removal will be permanent and irreversible
publish	<i>python -c manage -a publish -e NEWMODEL</i>	Clones the development version, creating a new version in the model repository. Versions are assigned sequential numbers
remove	<i>python -c manage -a remove -e NEWMODEL -v 2</i>	Removes the version specified from the NEWMODEL model repository
list	<i>python -c manage -a list</i>	Lists the models present in the repository and the published version for each one. If the name of a model is provided, lists only the published versions for this model
export	<i>python -c manage -a</i>	Exports the model entry NEWMODE, creating a tar compressed file

	<i>export -e NEWMODEL</i>	<i>NEWMODEL.tgz</i> which contains all the versions. This file can be imported by another flame instance (installed in a different host or company) with the <i>-c manage import</i> command
import	<i>python -c manage -a import -e NEWMODEL</i>	Imports file <i>NEWMODEL.tgz</i> , typically generated using command <i>-c manage -a export</i> creating model <i>NEWMODEL</i> in the local model repository

Flame web-app

Flame includes a simple prediction web server.

```
python predict-ws.py
```

To access the web graphical interface, open a web browser and enter the address <http://localhost:8080>

Flame predict interface

Input

minicaco.sdf

Browse

model

CACO2

version

dev

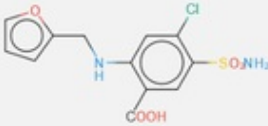
Predict

Export

Console

```
[{"obj_nam": ["furosemide", "guanabenz", "floxacin", "mibefradil", "verapamil", "guanoxan", "saquinavir", "lidocaine", "Enalapril", "theophylline"], "SMILES": ["NS(=O)(=O)c1cc(C(=O)O)c(NC2CCCC2)cc1Cl", "N=C(N)NN=Cc1c(Cl)cccc1Cl", "CN1CCN(c2c(F)cc3c(=O)c(C(=O)O)cn(CCF)c3c2F)CC1", "COCC(=O)OC1(CCN(C)CCCC2nc3cccc3[nH]2)CCc2cc(F)ccc2C1C(C)C", "COc1ccc(CCN(C)CCCC(C#N)(c2ccc(OC)c(O)c2)C(C)C)cc1OC", "N=C(N)NCC1COc2cccc2O1", "CC(C)(C)NC(=O)C1CC2CCCC2CN1CC(O)C(Cc1cccc1)NC(=O)C(CC(N)=O)NC(=O)c1ccc2cccc2n1", "CCN(CC)CC(=O)Nc1c(C)cccc1C", "CCOC(=O)C(CCC1CCCC1)NC(C)C(=O)N1CCCC1C(=O)O", "Cn1c(=O)c2nc[nH]c2n(C)c1=O"], "ymatrix": [-6.5, -4.5, -4.81, -4.87, -4.58, -4.71, -6.26, -4.21, -5.64, -4.35], "experim": [null, null, null, null, null, null, null, null, null, null], "origin": "apply", "values": [-5.2756625, -4.824337500000002, -4.988012499999999, -4.920112499999999, -4.683350000000012, -4.725950000000009, -5.505262500000001, -4.551875000000003, -5.223512499999993, -4.747500000000005], "lower_limit": [-6.5, -6.176125000000004, -6.415374999999999, -6.313900000000005, -6.075125000000009, -6.155750000000004, -6.900175000000001, -5.969675000000008, -6.523424999999994, -6.049400000000007], "upper_limit": [-4.051325, -3.472549999999999, -3.560649999999991, -3.526324999999994, -3.291575000000015, -3.296150000000014, -4.11035, -3.134074999999966, -3.923599999999915, -3.445599999999994], "meta": {"main": ["values"], "CI": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], "RI": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]}}
```

Results

#	obj nam	SMILES	values	ymatrix	experim	lower limit	upper limit	CI	RI
1	furosemide		-5.276	-6.500	-	-6.500	-4.051	0.000	0.000
2	guanabenz		-4.824	-4.500	-	-6.176	-3.473	0.000	0.000

“web GUI”)

Web API services available:

(in development)

URL	HTTP verb	Input data	Return data	HTTP status codes
/info	GET		application/json: info_message response	200
/dir	GET		application/json: available_services response	200

/predict	POST	multipart/form-data encoding: model and filename	application/json: predict_call response	200, 500 for malformed POST message
----------	------	---	---	-------------------------------------

The exact syntax of the JSON object returned by predict will be documented in detail elsewhere.

Technical details

Using Flame

Flame was designed to be used in different ways, using diverse interfaces.

For example:

- Using the web-GUI, starting the `predict-ws.py` web-service
- Using the `flame.py` command described above
- As a Python package, making direct calls to the high-level objects *predict*, *build* or *manage*
- As a Python package, making calls to the lower level objects *idata*, *apply*, *learn*, *odata*

The two main modeling tasks that must be supported by Flame are the *model development* and the use of the models for *prediction*. These are typically carried out by people with different expertise and in different environments. Flame was designed around this concept and allows to decouple both tasks completely. Somebody can develop a model in a research environment which can be easily exported to be installed in a

production environment to serve prediction services. Flame implements interfaces designed specifically for each task, even if they share exactly the same code, to guarantee compatibility and consistency.

Developing models

Typically, Flame models are developed by modeling engineers. This task requires importing an appropriate training series and defining the model building workflow.

Model building can be easily customized by editing the parameters defined in a command file (called *parameters.yaml*), either with a text editor or with the Flame modeling GUI (**in development**). Then, the model can be built using the `flame.py` build command, and its quality can be assessed in an iterative process which is repeated until optimum results are obtained. This task can also be carried out making calls to the objects mentioned above from an interactive Python environment, like a Jupyter notebook. A full documentation of the library can be obtained running Doxygen on the root directory.

Advanced users can customize the models by editing the objects *idata_child*, *appl_child*, *learn_child* and *odata_child* present at the *model/dev* folder. These empty objects are childs of the corresponding objects called by flame, and it is possible to override any of the parents' methods simply by copying and editing these within the childs' code files.

Models can be published to obtain persistent versions, usable for prediction in the same environment, or exported for using them in external

production environments, as described above.

Runnning models

Models built in Flame can be used for obtaining predictions using diverse methods. We can use the command mode interface with a simple call:

```
python flame.py -c predict -e MyModel -v 1 -f query.sdf
```

This allows to integrate the prediction in scripts, or workflow tools like KNIME and Pipeline Pilot.

Also, the models can run as prediction web-services, using the provided flame-ws interface. These services can be consumed by the stand-alone web GUI provided and described above or connected to a more complex platform, like the one currently in development in the eTRANSAFE project.

Licensing

Flame was produced at the Pharmacoinformatics lab (<http://phi.upf.edu>), in the framework of the eTRANSAFE project (<http://etransafe.eu>).

eTRANSAFE has received support from IMI2 Joint Undertaking under Grant Agreement No. 777365. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and the European Federation of Pharmaceutical Industries and Associations (EFPIA).



Copyright 2018 Manuel Pastor (manuel.pastor@upf.edu)

Flame 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**.

Flame 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 Flame. If not, see <http://www.gnu.org/licenses/>.