model

index
/home/mpastor/soft/eTAM/src/model.py

```
# -*- coding: utf-8 -*-
#
#    Description    eTAM model class
#
#
#    Authors:       Manuel Pastor (manuel.pastor@upf.edu)
#
#    (c) PhI 2013
```

Modules

numpy	os	shutil	sys
openbabel	pybel	subprocess	

Classes

model

class **model**

Methods defined here:

__init__(self, vpath)

adjustPentacle(self, row, nprobes, Bcol)
Adjust the row of GRIND descriptors in "row" to Bcol size, asuming that we have nprobes
GRID probes, applying a procrustean transform for each block (correlogram)

Both the row and the returning array are NumPy float64 arrays

build(self, data)
Uses the data extracted from the training series to build a model, using the Rlearner object

This function also creates the "itrain.txt" file that describes the training series, including InChiKey of the compounds

checkIdentity(self, mol)
Checks if the compound "mol" is part of the training set

We used InChiKeys without the last three chars (ionization state) to make the comparison

This version uses OpenBabel (OB)

TODO: Migrate to RCDKit

computeAD(self, md, pr, detail)
Carries out a protocol for determining how fat is the query compound from the model

Provisionally, implements a temporary version of the ADAN method

Returns a tuple that contains
1) True or False, indicating the success of the computation
2) (if True) a number between 0 and 5 with the number of criteria broken
   (if False) an error message

computeMD(self, mol)
Computes the Molecular Descriptors for compound "mol"

In this implementation we run Pentacle with default settings
It returms a tuple that contains
1) True or False, indicating the success of the computation
2) A vector of floats (if True) with the GRIND descriptors
   An Error message (if False)

computePR(self, md, charge)
Computes the prediction for compound "mol"

This function makes use of the molecular descriptor vector (md) to project the compound in the model
The model has been loaded previously as an R object

computeRI(self, ad)
Calculates a Reliability Index for the given prediction

Provisionally it returns a tuple that contains
1) True or False, indicating the success of the computation
2) (if True) - the model SDEP when the #broken AD criteria is 0 or 1
              - twice the model SDEP when the #broken AD criteria is 2 or 3
              - 0.0 otherwyse (this must be interpreted as a "model NA")
   (if False) An error message

convert3D(self, moli)
Converts the 2D structure of the molecule "moli" to 3D

```
      In this implementation, it uses CORINA from Molecular Networks
      The result is a tuple containing:
      1) suucTrue/False: describes the success of the 3D conversion for this compound
      2) (if True ) The name of the 3D molecule
         (if False) The error mesage
```

extract(self, mol)

```
      Process the compound "mol" for obtaining
      1) InChiKey (string)
      2) Molecular Descriptors (NumPy float64 array)
      3) Biological Activity (float)

      Returns a Tuple as (True,('InChi',array[0.0,0.0, 0.0],4.56))
```

getBio(self, mol)

```
      Extracts the value of the experimental biological property from the compound "mol"

      Such value must be identified by the tag <activity>
```

getInChi(self, mol)

```
      Computes the InChiKey for the compound "mol"

      We used InChiKeys without the last three chars (ionization state) to make the comparison
```

normalize(self, mol)

```
      Preprocesses the molecule "mol" by running a workflow that:

      - Normalizes the 2D structure (DUMMY)
      - Adjusts the ionization state
      - Converts the structure to 3D

      The result is a tuple containing:
      1) True/False: describes the success of the normalization
      2) (if True ) The name of the normalized molecule and its formal charge
         (if False) The error mesage
```

predict(self, molN, detail)

```
      Runs the prediction protocol
```

protonate(self, moli, pH)

```
      Adjusts the ionization state of the molecule "moli"

      In this implementation, it ises blabber_sd from Molecular Discovery
      The result is a tuple containing:
      1) True/False: describes the success of the protonation for this compound
      2) (if True ) The name of the protonated molecules and its formal charge
         (if False) The error message
```

standardize(self, moli)

```
      Applies a structure normalization protocol

      DUMMY method. At present it does nothing

      The name of the output molecules is built as a+'original name'

      Returns a tuple containing:
      1) True/False: depending on the result of the method
      2) (if True ) The name of the output molecule
         (if False) The error message
```

Data

opt = '/home/mpastor/soft/eTAM/opt/'