DREAMTools

Release 1.3.0

Thomas Cokelaer and the DREAMTools developers

March 21, 2016

Contents

Current version: 1.3.0, March 21, 2016

Python version DREAMTools is supported for Python 2.7, 3.4 and 3.5. Pre-compiled versions are available for Linux and MAC platforms through Anaconda and the **bioconda** channel.

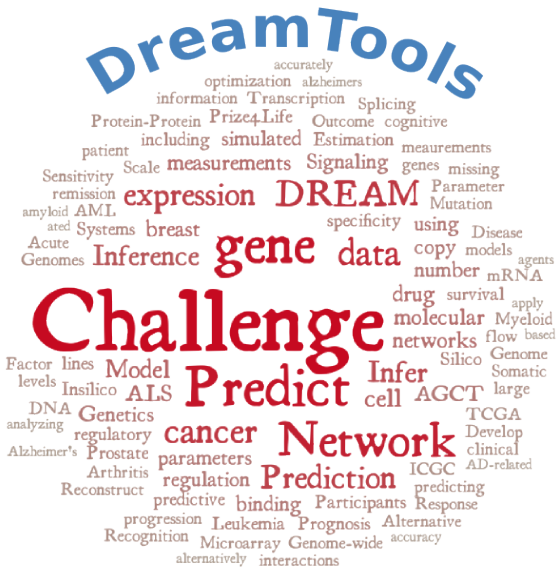
Note about coverage We do not run the entire test suite on Travis, which reports a 40% test coverage. Note however, that the actual test coverage is about 80%.

Contributions Please join https://github.com/dreamtools/dreamtools

Online documentation On readthedocs

Issues and bug reports On github

How to cite Cokelaer T, Bansal M, Bare C et al. DREAMTools: a Python package for scoring collaborative challenges [version 1; referees: awaiting peer review] F1000Research 2015, 4:1030 (doi: 10.12688/f1000research.7118.1) F1000 link



Contents

CHAPTER 1

Overview

1.1 Motivation

DREAMTools aims at sharing code used in the scoring of DREAM challenges that pose fundamental questions about system biology and translational medicine.

The main goals of **DREAMTools** are to provide:

1. Scoring functions equivalent to those used during past DREAM challenges for **end-users** via a standalone application (called **dreamtools**).
2. A common place for **developers** involved in the DREAM challenges to share code

DREAMTools does not provide code related to aggregation, leaderboards, or more complex analysis even though such code may be provided (e.g., in D8C1 challenge).

Note that many scoring functions requires data hosted on Synapse . We therefore strongly encourage you to **register to Synapse**. Depending on the challenge, you may be requested to accept terms of agreements to use the data.

1.2 Installation

For those familiar with Python, you may use the pip executable provided with Python. It will installed the latest release and the dependencies:

```
pip install cython
pip install dreamtools
```

If you are not familiar with compilation and/or Python, you may use conda since we have pre-compiled packages with a conda channel called **bioconda**:

```
conda config --add channels r
conda config --add channels bioconda
conda install dreamtools
```

See *Installation* section for details.

1.3 Usage

Every DREAM challenge is different. We will not explain here the scientific goal but show how one could score its own prediction. Developers can you **DREAMTools** as a Python package:

```
>>> from dreamtools import D6C3
>>> s = D6C3()
>>> s.score(s.download_template())
{'results': chi2            53.980741
R-square        34.733565
Spearman(Sp)     0.646917
Pearson(Cp)      0.647516
dtype: float64}
```

Besides, a standalone application can be used from a terminal. The executable is called **dreamtools**. Here is an example:

```
dreamtools --challenge D6C3 --submission path_to_a_file
```

See *User Guide* for more details about the usage of the standalone application.

CHAPTER 2

Available challenges, templates and gold standards

DREAMTools includes about 80% of DREAM challenges from DREAM2 to DREAM9.5 Please visit F1000 link (Table 1).

All gold standards and templates are retrieved automatically. Once downloaded, you can obtain the location of a gold standard or template as follows:

```
dreamtools --challenge D6C3 --download-gold-standard
dreamtools --challenge D6C3 --download-template
```

CHAPTER 3

Full documentation

Contents

3.1 Installation

3.1.1 Familiar with Python ecosystem ?

If you are familiar with Python and the **pip** application and your system is already configured (compilers, development libraries available), these two commands should install **DREAMTools** and its dependencies (in unix or windows terminal):

```
pip install cython
pip install dreamtools
```

If you do not have dependencies installed yet (e.g., pandas, numpy, scipy), this may take a while depending on your system (typically 10-15 minutes). If you are in a hurry or do not want to compile libraries, see the Anaconda solution here below.

3.1.2 If you are new to Python

If you are not familiar with Python, or have issues with the previous method (e.g., compilation failure), or do not have root access, we would recommend to use the Anaconda solution.

Anaconda is a free Python distribution. It includes most popular Python packages for science and data analysis and has dedicated channels. One such channel is called **'bioconda >https://bioconda.github.io/>'_** and complements the default channel (conda) with a set of packages dedicated to life science.

We have included **DREAMTools** in **bioconda** channel. So, once Anaconda is installed, you first need to add the **bioconda** channel to your environment (and R channel):

```
conda config --add channels r
conda config --add channels bioconda
```

This should be done only once. Then, install **DREAMTools** itself:

```
conda install dreamtools
```

This command should install **DREAMTools** in your default conda environment. If you wish to try **DREAMTools** in another (independent) environment (e.g., a different python version), you would need to create and activate the environment first:

```
conda create --name test_dreamtools python=3.5
source activate test_dreamtools
conda install dreamtools
```

3.1.3 Installation from source

The previous methods relies on released versions of **DREAMTools**. If a new feature is only available in the source code, then you will need to get the source code, which is available in the github repository:

```
git clone git@github.com:dreamtools/dreamtools.git
cd dreamtools
python setup.py install
```

Dependencies (e.g. Pandas) will need to be compiled or pre-installed (see above).

3.1.4 Note for Windows and Anaconda

We do not provide any DREAMTools package on **bioconda** for Windows.

However, if you use Anaconda and decide to compile the source yourself under Windows, then you will have to install a compiler that is compatible with Anaconda. In other words, you will have to use the same compiled as the one used by Anaconda.

For Python 2.7, compilation should work easily. You need to know that pre-compiled packages (e.g., Cython) used a specific version of a compiler (http://docs.continuum.io/anaconda/faq#how-did-you-compile-cpython), which is Visual Studio version 2008 and is provided by Microsoft (http://www.microsoft.com/en-us/download/details.aspx?id=44266) for free.

For Python3.4 and 3.5, this is a bit more difficult. You should get Visual C version 2010 (http://stackoverflow.com/questions/29909330/microsoft-visual-c-compiler-for-python-3-4) or for Python 3.5 another Visual C version 2015. This may change with time but this information was found on Anaconda documentation (March 2016). You may found useful information here as well for VS2015: http://www.microsoft.com/en-us/download/details.aspx?id=44266

3.1.5 Note for Python2.X and Python3.X

DREAMTools is compatible with Python2.7, Python3.4, Python3.5. The **bioconda** channel provide these 3 versions. If you still want to use Python2.6 or 3.3, **DREAMTools** may work as well but you would need to compile the dependencies yourself.

3.2 User Guide

3.2.1 Introduction

DREAMTools provides scoring functions that were used in past DREAM challenges. In order to use those scoring functions, users may use an executable called **dreamtools** (See *The dreamtools executable* section) while developers may use the Python library directly in their own pipelines.

The main idea behind **DREAMTools** is to provide to researchers the scoring functions that were used in past DREAM challenges. Usually, researchers would already know the topic / purposes of a challenge but information can also be retrieved with **DREAMTools** as shown here below. Then, one would need to design a new methodology to solve the challenge. The difficulties then may be to (i) retrieve a template, (ii) fill the template with a prediction and (iii) to score the prediction to evaluate the performance of the prediction.

DREAMTools will help researchers in retrieving information and templates about a challenge, and apply the relevant scoring function to evaluate their algoritm(s).

3.2.2 What is the data format, what is the challenge about ?

Each data format is different and each challenge is complex and specific to a biological problem so we will not explain each challenge or template format in this documentation. However, links and information provided within **DREAMTools** should give enough help to start with.

There were tens of challenges (see http://f1000research.com/articles/4-1030/v1) during the last years and we will refer to a given challenge by a nickname (e.g., D6C3 stands for challenge 3 in DREAM version 6). Finally, note that some challenges have sub-challenges whose names must be provided.

For example, to retrieve information about the D9C1 challenge (Gene essentiality), Python users can type these commands:

```
from dreamtools import D9C1
challenge = D9C1(download=False)
challenge.onweb()
```

Or, using the **dreamtools** standalone application, one can type in a shell the following command:

```
dreamtools --challenge D9C1 --info
```

This should open the Synapse web page of the challenge where description, template, leaderboards are stored altogether.

3.2.3 Synapse login

Before giving more details about **DREAMtools**, we would like to emphasize that the software is closely linked to Synapse where challenges are described and where data required for the scoring may be stored.

Note: Consequently, users will need to sign up to Synapse website: http://www.synapse.org.

Once you have a Synapse login, you can also create a local authentication by creating a file called **.synapseConfig** in your home directory and add this content:

```
[authentication]
username: email
password: password
```

where the email and password are those you have created/obtained from Synapse . This will avoid you to have to enter the login/password each time **DREAMTools** tries to connect to Synapse.

3.2.4 Notes about data restrictions

DREAMTools provides functions to obtain the template and gold standard(s) used in a given challenge. Some challenge have restrictions of data access and require the user to accept conditions of use. Such data are stored on Synapse and the first time you run a challenge within **DREAMTools**, files may be downloaded and you may be asked to accept some conditions of use.

3.2.5 The dreamtools executable

For users, **DREAMTools** package provides an executable called **dreamtools**, which should be installed automatically. To check that it is installed properly, type this command in a shell:

```
dreamtools --help
```

this will give you some basic help about the usage. Let us seee how it works. First, let us choose a challenge. Challenge are named DXCY where X starts from number 2 to indicate the DREAM session. Y indicates the challenge itself.

```
dreamtools --challenge D5C1
```

This will raise an error because there is no submission provided. A template/example can be retrieved as follows:

```
dreamtools --challenge D5C1 --download-template
```

This prints the path to a template, which can now be scored (even though the template contains dummy data in general):

```
dreamtools --challenge D5C1 --filename <path2template>
```

similarly one can download the gold standard. This is a good way to check the scoring function since scoring the gold standard with itself should give a perfect score:

```
dreamtools --challenge D5C1 --download-gold-standard
dreamtools --challenge D5C1 --filename <path2gold>
```

If there are sub challenges like in D9C1 challenge, a sub-challenge name must be provided. If one type:

```
dreamtools --challenge D9C1 --download-template
```

an error message will tell you that the sub-challenge name is missing together their names. Here, the names are shown to be sc1, sc2, sc3:

```
dreamtools --challenge D9C1 --download-template --sub-challenge sc1
```

3.2.6 Scripting

An alternative to the standalone application is to use **DREAMTools** inside a Python script. Similarly to what we have seen in the previous section, you can download templates, gold standards and scoring functions. All challenges are based upon a single `Challenge` class and use a very similar syntax:

```
# import a challenge
from dreamtools import D5C1
# create the challenge structure
c = D5C1()

# figure out the path to a template
filename = c.download_template()

# score that template
results = c.score(filename)

# print the results
print(results)
```

If you have sub challenges, they can be found in the attribute called *sub_challenges*:

```
from dreamtools import D9C1
c = D9C1()
subname = c.sub_challenges[0]   # get only the first sub challenge name
filename = c.download_template(subname)
results = c.score(filename, subname)
print(results)
```

3.2.7 Getting information about a challenge

From the Python command line, for a given challenge, you can get a brief summary and the Synapse page identifier:

```
from dreamtools import D9C1
s = D9C1(download=False)  # Needed if you do not have a Synapse account
print(s)
```

You can also open the Synapse web page corresponding to that challenge:

```
s.onweb()
```

Or use the **dreamtools** executable:

```
dreamtools --challenge D9C1 --info
dreamtools --challenge D9C1 --onweb
```

3.2.8 Where to get more help or examples ?

All dream challenges have their own Synapse page and should be used as the official references. Especially if you want to contact the organisers of a challenge.

However, you may also get brief help and information from other sources:

1. From the DREAMTools paper on F1000.

2. The *References* of DREAMTools itself
3. Notebooks provided in DREAMTools. There are only a few at the moment but contributions are welcome and will be added.
4. Notebooks

3.3 For developers

3.3.1 How to structure a new challenge:

If you wish to include a scoring function in **DREAMTools**, we provide an executable called **dreamtools-layout** (from `dreamtools.core.layout` module). It creates a minimalist layout automatically to help you to start. You first need to think about a challenge nickname. Let us assume a challenge for DREAM8 session, which is the fourth one. Its nickname would be D8C4.

First, move in the github tree structure to the **dream8** directory:

```
cd dreamtools
cd dream8
```

if the directory **dream8** does not exist, create it and add an empty file called **__init__.py**. Then, all you need to do is to go to *dreamtools/dream8* directory and type:

```
dreamtools-layout --challenge-name D8C4
```

Some sub directories and files are created including the **scoring.py** with a basic class where to code or wrap your scoring function.

If data file or templates are too large, we strongly recommend to store them in a project on Synapse. We have created a synapse project called dreamtools where for example the D5C2 data files have been stored. Other files can all be stored there. This may be duplicated with existing projects but would ease the maintenance of the 30-40 DREAM challenges already available in **DREAMTools**.

3.3.2 Naming conventions

There is no strict conventions but to help creating more uniformed code, try to name the template after the challenge nicknname for instance **D3C1_template**. Irrespective of the name, place it in the templates/ directory. Similarly for gold standards: start the filename with **D3C1_goldstandard** tag.

Again, if those files are too large, consider placing them in synapse and use tools inside DREAMTools to retrieve them automatically (see below).

3.3.3 Basic Structure of the Challenge class

```
import os
from dreamtools.core.challenge import Challenge

class D7C4(Challenge):
    """A class dedicated to D7C4 challenge

    ::

        from dreamtools import D7C4
        s = D7C4()
```

```
        filename = s.download_template()
        s.score(filename)

    Data and templates are downloaded from Synapse. You must have a login.

    """
    def __init__(self, verbose=True, download=True, **kargs):
        """.. rubric:: constructor"""
        super(D7C4, self).__init__('D7C4', verbose, download, **kargs)
        self._init()
        # if several sub-challenges, name them here
        self.sub_challenges = []

    def _init(self):
        # should download files from synapse if required.
        pass

    def score(self, prediction_file):
        raise NotImplementedError

    def download_template(self):
        # should return full path to a template file
        self.getpath_template('filename_in_templates_directory')

    def download_goldstandard(self):
        # should return full path to a template file
        self.getpath_goldstandard('filename_in_goldstandard_directory')
```

3.3.4 Storing large files

Templates and gold standards are either stored within the **DREAMTools** package or, if there are too large, on the Synapse web site. In the latter case, the files will be downloaded on request (only once). You should therefore not change those files, which are located in the **DREAMTools** directory (e.g., /home/user/.config/dreamtools). The downloaded files are stored in specific directories. For instance, files related to the D9C1 challenge are stored in /home/user/.config/dreamtools/dream9/D9C1.

So, as developers, you should also figure out if a file should be stored in Synapse or not. Large files are currently stored in this synapse page dreamtools

If your class inherits from *dreamtools.core.challenge.Challenge*, you can then just type this kind of command to (1) download the file and get its local location:

```
filename = self._download_data('DREAM5_GoldStandard_probes.zip',
           'syn2898469')
```

In this example, the file *DREAM5_GoldStandard_probes.zip* is stored in this directory: /home/user/.config/dreamtools/dream5/D5C2 for the example above.

3.3.5 License/header

Please add this `header` at the top of your Python files:

```
# -*- python -*-
# -*- coding: utf-8 -*-
#
#  This file is part of DREAMTools software
#
```

```
#  Copyright (c) 2015-2016, DREAMTools Development Team
#  All rights reserved
#
#  Distributed under the BSD 3-Clause License.
#  See accompanying file LICENSE distributed with this software
#
#  File author(s): Your name <youremail>
#
#  website: http://github.com/dreamtools
#
##############################################################################
```

3.4 References

3.4.1 CORE modules

Contents

Challenge

Common utility to all challenges

class **LocalData**

Used by *Challenge*

getpath_data (*filename*)

getpath_gs (*filename*)

getpath_lb (*filename*)

getpath_template (*filename*)

Return full path of the template location named *filename*

class **Challenge** (*challenge_name*, *verbose=False*, *download=True*, ***kargs*)

Common class to all challenges

If you have not setup a .synapseConfig in your HOME, you must provide a synapse client

```
from dreamtools import *
s = Challenge('D2C1')
client = Login(username=username, password=pwd).client
s.client = client
```

constructor

Parameters **challenge_name** (*str*) – Must be formatted as DXCY where X and Y are numbers. Intermediate challenges from e.g. D9.5 should be encoded as D9dot5CY

debug = None
alias of the challenge as DXCY form with X, Y being 2 numbers

directory
Gets directory where data will be stored.

download_template(*sub_challenge=None*)
Must be provided

get_pathname(*filename*)
Return pathname of a file to be found on ./config/dreamtools if available

import_scoring_class()
Dynamic import of a challenge class

```
c = Challenge('D7C1')
inst_class = c.import_scoring_class()
```

loadmat(*filename*)
Load a MATLAB matrix

mainpath = None
directory where is stored the configuration file and data files.

mkdir()
Create local dreamtools directory

onweb()

score(*filename*, *sub_challenge=None*)
Must be provided

test()

unzip(*filename*)
Simple method to extract all files contained in an archive

Rocs

Provides tools related to Receiver Operating Characteristic (ROC).

Those codes were directly translated from Perl or matlab codes. We should be using scikit-learn in the future.

class ROC(*scores=None*, *classes=None*)
A class to compute ROC, AUC and AUPRs for a binary problem

```
>>> r = ROC() # could provide scores and labels as arguments
>>> r.scores = [0.9,0.8,0.7,.6,.6,.3]
>>> r.classes = [1,0,1,0,1,1]
>>> r.compute_auc()
0.4375
```

Constructor

Parameters

- **scores** (*list*) – the scores
- **classes** (*list*) – binary class made of 1 or 0 numerical values. Also called labels in the literature.

classes
Read/Write the classes

get_roc()
See *get_statistics()*

get_statistics()
Compute the ROC curve X/Y vectors and some other metrics

Returns a dictionary with different metrics such as FPR (false positive rate), PTR (true positive rate).

plot_roc(*roc=None*)
Plot ROC curves

```
from dreamtools.core.rocs import ROC
r = ROC()
r.scores = [.9,.5,.6,.7,.1,.2,.6,.4,.7,.9, .2]
r.classes = [1,0,1,0,0,1,1,0,0,1,1]
r.plot_roc()
```

scores
Read/Write the scores

class **ROCDiscovery**(*discovery*)
A variant of ROC statistics

Used in D5C2 challenge.

Note: Does not work if any NA are found.

constructor

Parameters **discovery** – a list of 0/1 where 1 means positives

compute_aupr(*roc=None*)
Returns AUPR normalised by (1-1/P) P being number of positives

get_statistics()
Return dictionary with FPR, TPR and other metrics

class **D3D4ROC**

get_statistics(*gold_data*, *test_data*, *gold_index*)

plot()

MCC(*TP*, *TN*, *FP*, *FN*)
Matthews correlation coefficient

settings

Tools to handle a configuration file.

class **DREAMToolsConfig**(*verbose=False*)

Synapse utilities

A module dedicated to synapse

The class *SynapseClient* is a specialised class built upon synapseclient package, which source code is on GitHub:

```
git clone git://github.com/Sage-Bionetworks/synapsePythonClient.git
cd synapsePythonClient
python setup.py install
```

This class may be removed but for now it is used in D8C1 challenge.

```
>>> from dreamtools.core import sageutils
>>> s = sageutils.SynapseClient()
```

class SynapseClient (*username=None*, *password=None*)

This class inherits all methods from synapseClient.

Be aware that most of the functionalities are now available in synapseclient itself. So, most of the methods that were written are hidden (double underscore) and may be removed in the future.

The only remaining feature is the automatic login, and simple version of the downloadSubmission method. There is also a *json()* method used throughout the dream8hpn code.

Constructor

Parameters

- **username** – your synapse usename
- **password** – your synapse password

You can create create a file called **.synapseConfig** (note the dot) in your home directory and add something like:

```
[authentication]
username: yourlogin
password: yourpassword
```

downloadSubmissionAndFilename (*sub*, *downloadFile=True*, ***kargs*)

Return filename of a submission downloaded from synapse.

Parameters

- **sub** – A submission (as a dictionary).
- **version** – The specific version to get. Defaults to the most recent version.
- **downloadFile** – Whether associated files(s) should be downloaded. Defaults to True. If set to False, downloadLocation and ifcollision are ignored
- **downloadLocation** – Directory where to download the Synapse File Entity. Defaults to the local cache.
- **ifcollision** – Determines how to handle file collisions. May be "overwrite.local", "keep.local", or "keep.both". Defaults to "keep.both".

Warning: ifcollision does not seem to work (0.5.1)

getMyProfile()
Returns user profile

json(*data*)
Transform relevant object into json object

class Login(*client=None*, *username=None*, *password=None*)
A simple class to login to synapse

Parameters **client** – Connection to synapse takes a couple of seconds. This may be too much if in a debugging mode or accessing to synapse from different places. The login can be instantiate with an existing instance of SynapseClient, if which case, the instance creation is fast. Otherwise, the default behaviour is to create a new connection.

```
>>> from dreamtools.core.sageutils import Login
>>> l = Login()
This is a SynapseClient built on top of Synapse class.
Trying to login automatically.
Welcome, ****************
You're logged in Synapse
Welcome, XXX

In [10]: l = sageutils.Login(l)
```

Ziptools

class ZIP
Simple utility to load a ZIP file

Note: could be moved to easydev package

---

extractall(*path*)

loadZIPFile(*filename*)
Loads a ZIP file

This method uses the zipfile module and stores the data into `zip_data`. The filenames contained within this archive can be found in `zip_filenames`. To read the data contained in the first filename, type:

```
self.zip_data.open(self.filenames[0].read()
```

Parameters **filename** (*str*) – the ZIP filename to load

read(*filename*)

Downloader

Utility to download a synapse project in the dreamtools directory

class Downloader(*challenge*, *client=None*, *username=None*, *password=None*)
Factory to download gold standard files

Download a synpase file once for all in the dreamtools directory.

constructor

Parameters **challenge** (*str*) – alias of a challenge (e.g., D5C1)

To automatically connect to synapse, create a file called .synapseConfig with this content:

```
[authentication]
username: email
password: password
```

download(*synid*)
Download a file into the dreamtools directory

Parameters **synid** – a valid synapse id (e.g., syn123456)

You must have a login on synapse website.

Layout

Layout to create a new challenge from scratch

class Layout(*name*, *verbose=True*)
Class to create automatic layout for a given challenge

Usage

```
dreamtools-layout --name D8C2
```

Warning: for developers only.

create_layout()

layout(*args=None*)
This function is used by the standalone application called dreamscoring

```
dreamtools-layout --help
```

Concordance Index

Concordance index computation (exact version)

Based on R code provided by Ben Sauerwine and Erhan Bilal double checked with concordance.index from survcomp R package.

class ConcordanceIndex(*survtime=None*, *survevent=None*)
See *cindex()* function for details

cindex(*prediction*)
Returns concordance index

cindex(*prediction*, *survtime*, *survevent*)
Function to compute the concordance index for a risk prediction, i.e., the probability that, for a pair of randomly chosen comparable samples, the sample with the higher risk prediction will experience an event before the other sample or belongs to a higher binary class.

Parameters

- **prediction** – a vector of risk predictions.

- **survtime** – a vector of event times.
- **survevent** – a vector of event occurence indicators (True and False).

```
>>> from dreamtools.core.cindex import cindex
>>> print(cindex([0, 1, 3,4], [0, 4, 3, 1], [True]*4))
0.5
>>> print(cindex([0, 1, 3,4], [0, 1, 3, 4], [True]*4))
0.0
```

concordanceIndex(*prediction*, *survtime*, *survevent*)

3.4.2 DREAM challenges

Contents

DREAM2

D2C1

D2C2 scoring function

Class imlemented in Python based on original code in MATLAB from Gustavo A. Stolovitzky.

class `D2C1` (*verbose=True*, *download=True*, ***kargs*)

`download_goldstandard`()
Returns D2C1 gold standard file location

`download_template`()
Returns D2C1 template location

`load_leaderboard`()

`score` (*filename*)
Returns statistics (e.g. AUPR/AUROC)

Parameters **`filename`** (*str*) – a valid filename as returned by *download_template()*

`score_and_compare_with_lb` (*filename*)
Example of a comparative leaderboard that scores

D2C2

D2C2 scoring function.

Implementation in Python based on a MATLAB code from Gustavo A. Stolovitzky

class `D2C2` (*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D2C2 challenge

```
from dreamtools import D2C2
s = D2C2()
filename = s.download_template()
s.score(filename)
```

constructor

`download_goldstandard`()
Returns the gold standard

`download_template`()
Returns a valid template

`score` (*filename*)
Returns statistics (e.g. AUROC)

Parameters **`filename`** (*str*) – a valid filename as returned by *download_template()*

D2C3

D2C3 scoring functions

The original algorithm was developed in MATLAB by Gustavo Stolovitzky

class D2C3 (*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D2C3 challenge

```
from dreamtools import D2C3
s = D2C3()
subname = "DIRECTED-UNSIGNED_qPCR"
filename = s.download_template(subname)
s.score(filename, subname)
```

There are 12 gold standards and templates. There are scored independently (6 for the chip case and 6 for the qPCR).

Although there is no sub-challenge per se, there are 12 different templates so we use the template names as sub-challenge names

constructor

download_goldstandard (*subname=None*)
Returns one of the 12 gold standard files

Parameters **subname** – one of the sub challenge name. See *sub_challenges*

download_template (*subname=None*)

score (*filename*, *subname=None*)

sub_challenges = None
sub challenges (12 different values)

D2C4

D2C4 scoring function

original code in MATLAB by Gustavo Stolovitzky

class D2C4 (*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D2C4 challenge

```
from dreamtools import D2C4
s = D2C4()
subname = 'DIRECTED-UNSIGNED_InSilico1'
filename = s.download_template(subname)
s.score(filename, subname)
```

constructor

download_goldstandard (*subname=None*)

download_template (*subname=None*)

score (*filename*, *subname=None*, *goldstandard=None*)

sub_challenges = None
12 different sub challenges

D2C5

D2C5 scoring functions

Original code in MATLAB by Gustavo Stolovitzky

class D2C5 (*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D2C5 challenge

```
from dreamtools import D2C5
s = D2C5()
subname = "UNSIGNED"
filename = s.download_template(subname)
s.score(filename, subname)
```

constructor

download_goldstandard (*subname=None*)

download_template (*subname=None*)

score (*filename*, *subname=None*, *goldstandard=None*)

DREAM3

D3C1

D3C1 scoring function

Original matlab code from Gustavo A. Stolovitzky and Robert Prill.

class D3C1 (*verbose=True*, *download=True*, ***kargs*)
D3C1 scoring function to evaluate the accuracy of a prediction

```
from dreamtools import D3C1
s = D3C1()
filename = s.download_template()
s.score(filename)
```

download_goldstandard ()

download_template ()
Return filename of a template to be used for testing

probability (*x*)

score (*filename*)
Scoring function

Returns tuple with first element being the number of correct predictions and second element being the pvalue

D3C2

D3C2 scoring function

Implemented after an original MATLAB code from Gustavo Stolovitzky and Robert Prill.

class D3C2 (*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D3C2 challenge

```
from dreamtools import D3C2
s = D3C2()
filename = s.download_template('cytokine')
s.score(filename, 'cytokine')

filename = s.download_template('phospho')
s.score(filename, 'phospho')
```

Data and templates are downloaded from Synapse. You must have a login.

constructor

download_goldstandard (*subname*)

download_template (*name*)

score (*filename*, *subname*)
Returns score of a prediction

D3C3

D3C3 scoring function

Original matlab version (Gustavo A. Stolovitzky, Ph.D. Robert Prill) translated into Python by Thomas Cokelaer.

class D3C3 (*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D3C3 challenge

```
from dreamtools import D3C3
s = D3C3()
filename = s.download_template()
s.score(filename)
```

Data and templates are downloaded from Synapse. You must have a login.

Note: the spearman pvalues are computed using R and are slightly different from the official code that used matlab. The reason being that the 2 implementations are different. Pleasee see cor.test in R and corr() function in matlab for details. The scipy.stats.stats.spearman has a very different implementation for small size cases.

---

constructor

download_goldstandard()

download_template()

score (*filename*)

D3C4

Implementation in Python from Thomas Cokelaer. Original code in matlab (Gustavo Stolovitzky and Robert Prill).

class D3C4 (*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D3C4 challenge

```
from dreamtools import D3C4
s = D3C4()
filename = s.download_template(10)
s.score(filename)
```

Note: AUROC/AUPR and p-values are returned for a given sub-challenge. In the DREAM LB, the 5 networks are combined together. We should have same implemntatin as in D4C2

constructor

download_goldstandard (*subname*)

download_template (*subname*)

plot (*filename*, *size*, *batch*)

score (*filename*, *subname*)

score_prediction (*filename*, *subname*)

Parameters

- **filename** –
- **size** –
- **name** –

Returns

DREAM4

D4C1

D4C1 scoring function

Based on an original matlab code from Gustavo A. Stolovitzky, and Robert Prill.

class D4C1 (*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D4C1 challenge

```
from dreamtools import D4C1
s = D4C1()
filename = s.download_template()
s.score(filename)
```

Data and templates are downloaded from Synapse. You must have a login.

constructor

download_goldstandard ()

download_template ()

score (*filename*)

score_kinases()

score_pdz()

score_sh3()

D4C2

D4C2 scoring function

From an original code in matlab (Gustavo Stolovitzky and Robert Prill).

class D4C2(*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D4C2 challenge

```
from dreamtools import D4C2
s = D4C2()
filename = s.download_template(10, )
s.score(filename)
```

Data and templates are downloaded from Synapse. You must have a login.

constructor

directed_to_undirected()

download_goldstandard(*subname*)

download_template(*subname*)

load_prob(*filename*)

plot(*filename*, *subname*)

score(*filename*, *subname=None*)

score_prediction(*filename=None*, *subname=None*)
This is a longish scoring function translated from the matlab original code of D4C2

Parameters

- **filename** –
- **tag** –
- **batch** –

Returns

Todo

merge this function with the one from D4C2

D4C3

D4C3 scoring function

Based on Matlab script available on https://www.synapse.org/#!Synapse:syn2825304, which is an original code from Gustavo A. Stolovitzky and Robert Prill.

class D4C3(*verbose=True*, *download=True*, *edge_count=None*, *cost_per_link=0.0827*, ***kargs*)
A class dedicated to D4C3 challenge

```
from dreamtools import D4C3
s = D4C3()
filename = s.download_template()
s.edge_count = 20
s.score(filename)
```

Data and templates are inside Dreamtools.

Note: A parameter called cost_per_link is hardcoded for the challenge. It was compute as min {Prediction Score / Edge Count} amongst all submissions. For this scoring function, `cost_per_link` is set to 0.0827 and may be changed by the user.

---

constructor

Parameters

- **edge_count** (*int*) – if not provided, a prompt will ask for its value.
- **cost_per_link** (*float*) – a cost

download_goldstandard()

download_template()

plot()
Plots prediction versus gold standard for each species

```
from dreamtools import D4C3
s = D4C3()
filename = s.download_template()
s.edge_count = 20
s.score(filename)
s.plot()
```

score(*filename*)
Compute the score

See synapse page for details about the scoring function.

DREAM5

D5C1

D5C1 scoring function

From an original matlab code from Gustavo A. Stolovitzky, Robert Prill

class D5C1(*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D5C1 challenge

```
from dreamtools import D5C1
s = D5C1()
filename = s.download_template()
s.score(filename)
```

constructor

download_goldstandard()

download_template()

score(*filename*)

Returns dictionay with AUC/AUPR metrics and score.

D5C2

D5C2 challenge scoring functions

Based on TF_web.pl (perl version) provided by Raquel Norel (Columbia University/IBM) also used by the web server http://www.ebi.ac.uk/saezrodriguez-srv/d5c2/cgi-bin/TF_web.pl

This implementation is independent of the web server.

class D5C2(*verbose=True*, *download=True*, *tmpdir=None*, *Ntf=66*, ***kargs*)
A class dedicated to D5C2 challenge

```
from dreamtools import D5C2
s = D5C2()

# You can get a template from www.synapse.org page (you need
# to register)
filename = s.download_template()
s.score(filename) # takes about 5 minutes
s.get_table()
s.plot()
```

Data and templates are downloaded from Synapse. You must have a login.

constructor

Parameters

- **Ntf** – not to be used. Used for fast testing and debugging
- **tmpdir** – a local temporary file if provided.

cleanup()
Remove the temporary directory

compute_statistics()
Returns final results of the user predcition

Returns a dataframe with various metrics for each transcription factor.

Must call *score()* before.

download_all_data()
Download all large data sets from Synapse

download_goldstandard()

download_template()
Download a template from synapse into ~/config/dreamtools/dream5/D5C2

Returns filename and its full path

get_table()
Return table with user results from the user and participants

There are 14 participants as in the Leaderboard found here https://www.synapse.org/#!Synapse:syn2887863/wiki/72188

Returns a dataframe with different metrics showing performance of the submission with respect to other participants.

```
table = s.get_table()
with open('test.html', 'w') as fh:
    fh.write(table.to_html(index=False))
```

init()
Creates the temporary directory and the sub directories.

Behaviour differs whether the directory was provided in the constructor or not.

plot(*fontsize=16*)
Show the user prediction compare to 20 other participants

score(*prediction_file*)
Compute all results and compare prediction with official participants

This scoring function can take a long time (about 5-10 minutes).

D5C3

D5C3 scoring function

Original matlab code from Gustavo A. Stolovitzky, Robert Prill, Ph.D. sub challenge B original code in R from A. de la Fuente

class D5C3(*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D5C3 challenge

```
from dreamtools import D5C3
s = D5C3()
filename = s.download_template()
s.score(filename)
```

Data and templates are downloaded from Synapse. You must have a login.

3 subchallenges (A100, A300, A999) but also 3 others simpler with B1, B2, B3

For A series, 5 networks are required. For B, 3 are needed.

constructor

download_goldstandard(*subname*)

download_template(*subname*)

score(*filename*, *subname*)

score_challengeA(*filename*, *subname*)

score_challengeB(*filenames*)

D5C4

D5C4 scoring function

Based on original matlab code from Gustavo A. Stolovitzky and Robert Prill

class `D5C4`(*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D5C4 challenge

```
from dreamtools import D5C4
s = D5C4()
filename = s.download_template()
s.score(filename)
```

Data and templates are downloaded from Synapse. You must have a login.

constructor

download_goldstandard()

download_template()

score(*filenames*)

score_challengeA(*filename*, *tag*)

Parameters

- **filename** –
- **tag** –

Returns

DREAM6

D6C1

D6C1 scoring function

scoring author: bobby prill

class `D6C1`(*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D6C1 challenge

```
from dreamtools import D6C1
s = D6C1()
filename = s.download_template()
s.score(filename)
```

Todo

not yet implemented. Requires code to compute the recall and precision from the GS and submission.

constructor

download_goldstandard()

download_template()

score(*filename*)

D6C2

D6C2 scoring function

See D7C1

class D6C2(*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D6C2 challenge

```
from dreamtools import D6C2
s = D6C2()
filename = s.download_template()
s.score(filename)
```

Data and templates are downloaded from Synapse. You must have a login.

constructor

download_goldstandard()

download_template()

score(*prediction_file*)

D6C3

D6C3 scoring function

Based on Pablo's Meyer Matlab code.

class D6C3(*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D6C3 challenge

```
from dreamtools import D6C3
s = D6C3()
filename = s.download_template()
s.score(filename)
```

Absolute score in the Pearson coeff but other scores such as chi-square and rank are based on the 21st participants.

Pearson and spearman gives same values as in final LB but X2 and R2 are slightly different. Same results as in the original matlab scripts so the different with the LB is probably coming fron a different set of predictions files, which is stored in ./data/predictions and was found in http://genome.cshlp.org/content/23/11/1928/suppl/DC1

The final score in the official leaderboard computed the p-values for each score (chi-square, r-square, spearman and pearson correlation coefficient) and took -0.25 log (product of p-values) as the final score.

constructor

download_goldstandard()

download_template()

read_all_participants()

score(*filename*)

D6C4

Original scoring function: Kelly Norel

class D6C4(*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D6C4 challenge

```
from dreamtools import D6C4
s = D6C4()
filename = s.download_template()
s.score(filename)
```

Data and templates are downloaded from Synapse. You must have a login.

constructor

download_goldstandard()

download_template()

score(*filename*)

DREAM7

D7C1

DREAM7 Challenge 1 (Parameter estimation and network topology prediction)

References

- http://dreamchallenges.org/project-list/dream7-2012/
- https://www.synapse.org/#!Synapse:syn2821735/wiki/

Publications http://www.biomedcentral.com/1752-0509/8/13/abstract

class D7C1(*verbose=True*, *download=True*, *path='submissions'*, ***kargs*)
DREAM 7 - Network Topology and Parameter Inference Challenge

Here is a quick example on calling the scoring methods:

```
from dreamtools import D7C1
s = D7C1()
s.score_model1_timecourse(filename)
s.score_model1_parameters(filename)
s.score_topology(filename)
```

This class provides 3 main scoring functions:

1. *score_topology()*
2. *score_model1_timecourse()*
3. *score_model1_parameters()*

Each takes as an input a valid submission as described in the official synapse page.

Templates are also provided within the source code on github dreamtools in the directory dreamtools/dream7/D7C1/templates.

D7C1 scoring function are also included in the standalone code **dreamtools-scoring**.

For the details of the scoring functions, please refer to the paper (see module documentation) Some details are provided in the methods themselves as well.

There are other methods (starting with leaderboard) that should not be used. Those are draft version used to compute pvalues and report scores as in the final leaderboard.

Note: the scoring functions were implemented following Pablo Meyer's matlab **code-score_dream7_c1s1.m**

For admin only: put the submissions in ./submissions/ directory and call the :meth:

constructor

Parameters **path** – path to a directory containing submissions

Returns

download_goldstandard(*name*)

download_template(*name*)
Return filename of a template

Parameters **name** (*str*) – one in 'topology', 'parameter', 'timecourse'

get_null_parameters_model1(*N=10000*)
Returns score distribution (parameter model1)

get_null_timecourse_model1(*N=10000*)

get_null_topology(*N=10000*)
Return null distribution of the topology score

get_pvalues_parameter(*score*)

get_pvalues_timecourse(*score*)

get_pvalues_topology(*x*)
Return pvalues of a given score (topology challenge)

get_random_topology()

leaderboard()
Computes all scores for all submissions and returns dataframe

Returns dataframe with scores for each submissions for the model1 (parameter and timecourse) and model2 (topology)

leaderboard_compute_score_parameters_model1()
Computes all scores (parameters model1)

Returns Nothing but fills `scores`.

For the metric, see *score_model1_parameters()*.

See also:

load_submissions()

leaderboard_compute_score_timecourse_model1(*startindex=10*, *endindex=39*)
Computes all scores (timecourse model1)

Returns Nothing but fills `scores`

For the metric, see *score_model1_parameters()*.

Note that *endindex* is set to 39 so it does not take into account last value at time=20 This is to be in agreement with the implemenation used in the final leaderboard

https://www.synapse.org/#!Synapse:syn2821735/wiki/71062

If you want to take into account final point, set endindex to 40

leaderboard_compute_score_topology()
Computes all scores (topology) for loaded submissions

For the metric, see *score_topology()*.

Returns fills `scores`.

See also:

load_submissions()

load_submissions()
Load a bunch of submissions to be found in the submissions directory

The directory name is defined in `path`

Returns nothing. Populates `data` attribute and `team_names`.

score(*filename*, *subname=None*)
Return score for a given sub challenge

Parameters **filename** (*str*) – input filemame.

Returns name of a sub_challenge. See `sub_challenges` attribute.

score_model1_parameters(*filename*)
Return distance between submission and gold standard for parameters challenge (model1)

Parameters **filename** – must be valid templates

Returns score (distance)

```
>>> from dreamtools import D7C1
>>> s = D7C1()
>>> filename = s.download_template('parameter')
>>> s.score(filename, 'parameter')
0.022867555017785129
```

The score is computed using the square of the ratio of the user prediction and the gold standard. Taking the mean of the log10 :

$$S = \overline{\log 10 \left(\left(\frac{X}{X_{\text{gold standard}}} \right)^2 \right)}$$

score_model1_timecourse(*filename*)
Returns distance between prediction and gold standard (model1)

Parameters **filename** – must be valid templates

Returns score (distance)

```
>>> from dreamtools import D7C1
>>> s = D7C1()
>>> filename = s.download_template('timecourse')
>>> s.score_model1_timecourse(filename)
0.0024383612676804048
```

There are 3 time courses to be predicted. The score for each time course is

$$S_i = \frac{(X_i - \hat{X}_i)^2}{0.01 + 0.04 * X_i^2}$$

where X is the gold standard and $\hat{X}$ the prediction. and final score is just the average across the 3 time courses.

score_topology (*filename*)
Standalone version of the network topology scoring

Parameters **filename** (*str*) –

```
>>> from dreamtools import D7C1
>>> s = D7C1()
>>> filename = s.download_template('topology')
>>> s.score(filename, 'topology')
12
```

Scoring details The challenge requests predictions for 3 missing links, knowing that a gene can regulate up to two genes when they are in the same operon, 6 gene interactions have to be indicated by the participants (3 links*2 genes) and whether these interactions are activating (+) or repressing (-).

For each of the predicted links i=1,2,3, we define a score:

$S_i^{link} = L_i + N_i$

where L_j is 6 if the nature of the regulation iscorrect (that is, the source gene, the sign of the connection, and the destination gene are all correct) and $L_i = 12$ if the link regulates an operon composed of two genes and both connections are correct. If $L_i > 0$ then $N_i = 0$.

In case a link is NOT correctly predicted ($L_i = 0$) N_i is defined as follows. It is increased by 1 for each correctly regulated gene, 2 if the regulated gene and the nature of the regulation (i.e +/-) are correct and 1 if the regulator gene is correct

The gold standard contains 3 lines similar to

```
5 + 7 + 11
```

It means gene 5 positively regulates gene 7 and gene 11. If a prediction is

```
5 + 7 + 2
```

Then L =6. If the prediction is

```
2 + 7 + 2
```

L = 0 so N may be updated. Here the regulon (2) is not correct, However, one gene (7) is correctly predicted with the good sign so N = 2.

D7C2

class `D7C2`(*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D7C2 challenge

```
from dreamtools import D7C2
s = D7C2()
filename = s.download_template()
s.score(filename)
```

Data and templates are downloaded from Synapse. You must have a login.

as R objects implementing a function called customPredict() that returns a vector of risk predictors when given a set of feature data as input. The customPredict()

constructor

`download_goldstandard`(*subname=None*)

`download_template`(*subname=None*)

`score`(*filename*)

D7C3

class `D7C3`(*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D7C3 challenge

```
from dreamtools import D7C3
s = D7C3()
filename = s.download_template()
s.score(filename)
```

Data and templates are downloaded from Synapse. You must have a login.

constructor

`download_goldstandard`(*subname=None*)

`download_template`(*subname=None*)

`score`(*filename*, *subname=None*, *goldstandard=None*)

D7C4

Original code for challenge B translted from Mukesh Bansal Sub challenge A is currently a wrapping of a perl code provided by Jim Costello

class `D7C4`(*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D7C4 challenge

```
from dreamtools import D7C4
s = D7C4()
filename = s.download_template()
s.score(filename)
```

Data and templates are downloaded from Synapse. You must have a login.

```
    # columns represent the probabilistic c-index of the given team for
      each drug.
    # following the columns of teams are 5 columns which are used for
      calculating the overall team score
    # |-> Test_data = the probabilistic c-index for the experimentally
      determined test data scored against itself
    # |-> Mean Null Distribution = a set of 10,000 random predictions
      were scored to create the null distribution, of which this column
      represents the mean
    # |-> SD Null Distribution = a set of 10,000 random predictions
      were scored to create the null distribution, of which this column
      represents the standard deviation
    # |-> z-score of test data to null = score of the test data minus
      the mean of the null distribution divided by the standard deviation
      of the null distribution
    # |-> weight of drug (normalized z-score) = the z-score normalized
      by the largest z-score across all 31 drugs.
    # to calculate your team overall score, simply mulitple the score
      of all drugs by the corresponding weight.  Divide the sum of these
      weighted scores by the sum of the weights
```

constructor

This challenge uses PERL script that requires specific packages.

First, you need cpanm tools (http://search.cpan.org/dist/App-cpanminus/)

Under Fedora 23:

> sudo dnf install perl-App-cpanminus

Then, install the dependencies that will be required

> sudo cpanm install Math::Libm sudo cpanm install Algorithm::Pair::Best2 sudo cpanm install Digest::SHA1 sudo cpanm install Tk sudo cpanm install Games::Go::AGATourn

finally install the Games-go-GoPair.tar.gz package stored in dreamtools github repositotry in dreamtools/dreamt7/D7C4/misc:

```
    cd dreamtools/dream7/D7C4/misc
    tar xvfz Games-Go-GoPair-1.001.tar.gz
    cd Games-Go-GoPair-1.001
    perl Makefile.PL
    make
    sudo make install
```

download_goldstandard(*subname*)

download_template(*subname*)

score(*filename*, *subname*)

score_A(*filename*)

score_B(*filename*)

DREAM8

D8C1

This module provides utilities to compute scores related to HPN-Dream8

It can be used and should be used indepently of Synapse altough for testing, data sets may be downloading from synapse if you don't have any local files to play with.

Here is an example related to the Network subchallenge:

```
>>> from dreamtools.dream8.D8C1 import scoring
>>> s = scoring.HPNScoringNetwork()
>>> s.compute_all_descendant_matrices()
>>> s.compute_all_rocs()
>>> s.compute_all_aucs()
```

https://www.synapse.org/#!Synapse:syn1720047/wiki/60530

class **HPNScoringNetwork** (*filename=None*, *verbose=False*, *true_descendants=None*)

Class to compute the score of a Network submission

A user will provide a ZIP file that contains 65 files: 32 EDA, The 32 files should be tagged with the 32 combos of cell lines and ligands. To create an instance of HPNScoringNetwork, type:

```
s = HPNScoringNetwork("TeamName-Network.zip")
# or later
s = HPNScoringNetwork()
s.load_submission("TeamName-Network.zip")

s.get_auc_final_scoring() # as in the challenge ignoring some regimes
```

You then need to specifically load the EDA files. This may be done with *load_all_eda_files_from_zip()*:

```
s.load_all_eda_files_from_zip()
```

The content of the ZIP file can be validated using the *validation()* method.:

```
s.validation()
```

Each EDA and SIF file must be a complete graph where all species correspond to the CSV files provided on the synapse web page. The size of the network varies depending on the cell line.

Each EDA file that contains score on each edge and first needs to be transformed into a descendancy matrix. This is achieve via *compute_descendant_matrix()* and/or *compute_descendant_matrix()* methods.:

```
s.compute_all_descendant_matrices()
```

From each matrix, we'd like to compare a specific row (corresponding to mTOR) to the true scores that are expected. The true descendant for each combinaison of cell line and ligand are provided and loaded in the constructor via `load_true_descendants_from_zip()`, which can be called at any time.

Parameters **filename** (*str*) –

compute_all_aucs ()

Computes all AUC

This function can be called once EDA files are loaded and all descendant matrices have been computed as well.

In theory, one should compute ROC and then AUC but this function recomputes ROC since it is fast to compute.

See also:

load_all_eda_files_from_zip(), *compute_all_descendant_matrices()*

compute_all_auprs()

compute_all_descendant_matrices()
Compute all descendancy matrices

For each cell line and ligand, the matrix is stored in the `edge_scores` dictionary.

See also:

compute_descendant_matrix()

compute_all_metrics()

compute_all_rocs()
Computes all ROC curves

This function can be called once EDA files are loaded and all descendant matrices have been computed as well.

See also:

load_all_eda_files_from_zip(), *compute_all_descendant_matrices()*

compute_auc(*roc*)
Compute AUC given a ROC data set

Parameters **roc** (*str*) – The roc data structure must be a dictionary with "tpr" key. Could be an variable returned by *compute_roc()*.

compute_aupr(*roc*)

compute_descendant_matrix(*cellLine*, *ligand*)
Computes the descendancy matrix for a given cell line and ligand

Parameters

- **cellLine** (*str*) – a valid cell line
- **ligand** (*str*) – a valid ligand

Note: we use a cython module to conmpute the matrix. This function is the bottle neck of the entire procedure to compute the score. This is especially important to estimate te null distribution of the AUCs. Using Cython does not improve much the performance (80%) but it improves it...

See also:

compute_all_descendant_matrices()

compute_metrics(*cellLine*, *ligand*)

compute_other_metrics(*roc*)

compute_roc(*cellLine*, *ligand*)
Compute the ROC curve

Parameters

- **scores** – list of scores (probabilities)
- **classes** – list of classes (true values)

See also:

compute_roc()

compute_score(*validation=True*)
Computes the final score that is the average over the 32 AUCs

This function compute the final score. First, il loads all EDA files provided by the participany (from the ZIP file). Then, it computes the 32 descendant matrices. Finally, it computes the 32 ROCS and AUCS. The scores is for now based on the z-score. Since scores must be between 0 and 1, where 0 is the best, we will need to normalise.

Parameters **validation** (*bool*) – perform validation of the input ZIP file

edge_score_to_eda_files(*teamname*)

get_auc_final_scoring()
This function returns the mean AUC using only official ligands as used in final scoring and collaborative rounds.

The individual AUCs must be computed first with *compute_all_aucs()* or .:

```
>>> s = scoring.HPNScoringNetwork(filename)
>>> auc = s.get_auc_final_scoring()
```

get_aucs()
Returns all AUCs

get_average_auc()
Returns mean of all AUCs

get_mean_and_sigma_null_parameters()
Retrieve mean and sigma for 32 combi from a null AUC distribution

get_mean_zscores(*aucs=None*)

get_null_distribution(*sample=100*, *cellLine='BT20'*, *ligand='EGF'*, *store_rocs=False*, *distr='uniform'*)
Computes the null distribution for a given combinaison

•Creates a uniformly distribution of a EDA file and stores it in the edge_score attribute.

•recompute the corresponding descendancy matrix

•Get the corresponding true prediction

•compute the ROC and AUC

Parameters

- **sample** (*int*) – number of distribution to compute
- **cellLine** –
- **ligand** –
- **store_rocs** (*bool*) – if set to True, save the rocs as well

Returns rocs and aucs (rocs is set to [] for debugging)

```
from dreamtools.dream8.D8C1 import scoring
from pylab import clf, plot, hist, grid, pi, exp, sqrt, mean, std
s = scoring.HPNScoringNetwork()
rocs, aucs, auprs = s.get_null_distribution(100)
mu = mean(aucs)
sigma = std(aucs)
```

```
clf()
res = hist(aucs,50, normed=True)
plot(res[1], 1/(sigma * sqrt(2 * pi)) * exp( - (res[1] - mu)**2 / (2 * sigma**2) ), lin
grid()
```

get_zscores(*aucs=None*)

load_all_eda_files_from_zip()
Loads all EDA file from a participant into `edge_scores`

load_eda_file(*filename*, *local=False*)
Loads scores from one EDA file

Parameters **filename** – here filename should be one of the filename to be found within the ZIP file! This is not a standard file system (See note).

Input data is EDA format that is:

```
A 1 B = 0.4
A 1 C = 0.5
```

It containts edges such that the final graph is complete and a matrix can be built with column1 as the rows and column2 as the columns. The values being tmade from the fifth column. Second and fourth are ignored.

loaded data is stored in `data` as a numpy matrix.

Note: to overwrite the input ZIP file, use `loadZIPFile()`

load_submission(*filename*)

plot_all_rocs(*cellLines=None*)
Plots all 32 ROC once scores/rocs have been computed

```
from pylab import clf, plot, hist, grid
from dreamtools.dream8.D8C1 import scoring
import os
s = scoring.HPNScoringNetwork()
from dreamtools import D8C1
filename = D8C1().download_template('SC1A')
s.load_submission(filename)
s.compute_score()
s.plot_all_rocs()
```

plot_roc(*cellLine*, *ligand*, *hold=False*)
Plots a psecific ROC curve

print_aucs()

test_synapse_id = 'syn1971273'

true_synapse_id = 'syn1971278'

validation()
General validation

- Check that there are 32 EDA files
- **For each EDA file, calls further check**
  - format of the filename (correct cell line and ligand names)

- format of the dataa = character with a RHS and LHS
- LHS is made of 3 elements
- skip the header

class **HPNScoring** (*verbose=True*)

Base class common to all scoring classes

The HPN challenges use data from 32 types of combinaison of cell lines (4) and ligands (8). This class provides aliases to:

- valid cell lines (*valid_cellLines*)
- valid ligands (*valid_ligands*)
- expected length of the vectors for each cell line (*valid_length*)
- indices of the row vectors containing the mTOR species within the descendancy matrices (*mTor_index*)

Note: all matrices and vectors are sorted according to a hard-coded list of species for a combinaison of cell line and ligand. The species are indeed sorted alphabetically following hhe same order as in the original CSV files containing the data sets.

In addition, it the *score* attributes can be used to store the score computed by `compute_score()` .

All classes that need to compute scores require a data file submitted by a participant. We enforce the usage of ZIP file, which can be loaded by using `loadZIPFile()`.

error (*message*)

If you want to raise an error, use this method.

It raises a ScoringException and set the `exception` attribute. The message is stored in exception.value If called, the :attr:‘ ‘ is set to “INVALID” and the score is set to 1 (worst score).

load_species ()

Loads names of the expected phospho names for each cell line from the synapse files provided to the users

mTor_index = None

indices of the mTOR species in the different cell lines within

score

R/W attribute to store the score (in [0,1] only)

valid_cellLines = None

List of valid cell lines (e.g, BT20)

valid_length = None

length of the vectors to be found within each cell line

valid_ligands = None

List of valid ligands (e.g, EGF)

class **HPNScoringNetworkInsilico** (*filename=None*, *verbose=False*)

Scoring class for HPN DREAM8 Network Insilico sub challenge

This class retrieves the true graph and a test example from synapse.

```
from dreamtools.dream8.D8C1 import HPNScoringNetworkInsilico
s = HPNScoringNetworkInsilico()
import os
filename = s.download_template("SC1B")
s.read_file(filename)
```

Note: If you want to test your own local file, provide a filename.

auc

compute_score()
The official score for the SC1B challenge

Returns zscore

get_auc()

get_null_auc_aupr(*N*)
Get null distribution of the AUCs and AUPRs

Parameters **N** (*int*) – number of samples

Returns tuple made of 2 lists: the AUCc ad AUPRs

get_roc()
Gets a ROC instance using thegiven the user and true graphs as inputs

get_zscore()
Returns scores for the current submission

Returns

a single value based on the assumption that the distribution of the NULL AUC follows a gaussian distribution with parameters that are hardcoded as mu=0.497404 and std=0.037436.

aucs2, auprs2 = s.get_null_auc_aupr(500000) scipy.stats.gamma.fit([x for x in auprs if numpy.isnan(x)==False]) scipy.stats.norm.fit(aucs)

plot_null_distribution(*aucs=None*, *auprs=None*, *N=10000*)
Plots the null distribution of the AUCs

```
from dreamtools.dream8.D8C1 import HPNScoringNetworkInsilico
from dreamtools import D8C1
import os

s = HPNScoringNetworkInsilico()
filename = D8C1().download_template('SC1B')
s.read_file(filename)
aucs, auprs = s.get_null_auc_aupr(1000)
s.plot_null_distribution(aucs)
from pylab import xlim
xlim([0.35,0.65])
```

read_file(*filename*)

test_synapse_id = 'syn1973430'

to_eda(*filename*)
EXports the user EDA file

true_synapse_id = 'syn1976597'

class **HPNScoringNetwork_ranking**
This class is used to compute the ranks of the different participants based on an average rank over the 32 combinaisons of cell line and ligands.

```
s = HPNScoringNetwork(filename="file1zip")
s.compute_all_aucs()
```

```
sall = HPNScoringNetwork_all()
# s.aucs is a list where each element is a dictionary of
sall.add_auc(s1.auc, "team1")

# let us build
aucs2 = copy.deepcopy(s.auc)
for c in s.valid_cellLines:
    for l in s.valid_ligands:
        auc2[c][l] = numpy.random.uniform(0.5,0.7)

sall.add_auc(s2.auc, "team2")
sall.get_ranking()
{'team1': 1.96875, 'team2': 1.03125}
```

This class is independent of HPNSCoringNetwork. However, it takes as input the returned values of HPNScoringNetwork.compute_all_auc()

add_auc(*auc*, *participant_id*)

get_empty_auc()

get_integer_ranks()

get_mean_ranks()

get_mean_zscores()

get_rank_participant(*participant*)

get_ranking()

class HPNScoringPrediction(*filename=None*, *version=2*, *verbose=False*)

compute_all_rmse()

Some species were removed on purpose during the analysis

Those are hardcoded. To compute null distribution, we can keep all the species, in which case, _version parameter must be 0 must be set to False.

create_random_data()

Here, we don't want the true prediction that contains only what is requested (AZD8055) but the orignal training data with 2 or 3 inhibitors such as GSK and PD17 so that we can shuffle them.

We want to select for a given cell line and phosphos a data set to fill at a given time. The datum is selected accross the 8 stimuli, inhibitors +DMSO, and time points.

TAZ and FOX were asked to be excluded so this cause some trouble now but some user preidction still include them. Should add a if statement to ignore them. Does not matter to compute the null distribution

get_mean_rmse()

get_null(*N=100*, *tag='sc2a'*)

```
s = HPNScoringPrediction()
nulls = s.get_null(1000)
# the nulls contains the 4 cell lines
# let us save the first one
for name in ['UACC812', 'BT549', 'MCF7', 'BT20']:
    data = [x[name] for x in nulls]
    fh = open('%s.json' % name, 'w')
    import json
    json.dump(data, fh)
    fh.close()
```

get_rmse(*cellLine*, *phospho*)

> **Warning:** x in converted into a log2 scale

get_training_data()

get_true_prediction()
Reads true predcition from the 4 CSV files that contain the true prediction

data is stored as follows in the tru_prediction attribute:

get_user_prediction()
should be MIDAS files as in https://www.synapse.org/#!Synapse:syn1973835

test_synapse_id = 'syn2000886'

true_synapse_id = 'syn2009136'

class HPNScoringPrediction_ranking
This class is used to compute the ranks of the different participants based on an average rank over the 4 cell lines times phosphos

```
s = HPNScoringPrediction(filename="file1zip")
s.compute_all_rmes()

r = HPNScoringPrediction_ranking()
# s.aucs is a list where each element is a dictionary of
r.add_rme(s1.rmse, "team1")

rmse1 = r.get_randomised_rmse(r.rmse[0], sigma=1)
rmse2 = r.get_randomised_rmse(r.rmse[0], sigma=2)
rmse3 = r.get_randomised_rmse(r.rmse[0], sigma=3)

r.add_rmse(rmse1, "team2")
r.add_rmse(rmse2, "team3")
r.add_rmse(rmse3, "team4")

sall.add_rmse(s2.rmse, "team2")
sall.get_ranking()
{'team1': 1.96875, 'team2': 1.03125}
```

This class is independent of HPNSCoringPrediction. However, it takes as input the returned values of HPNScoringPrediction.compute_all_rmse()

add_rmse(*data*, *participant_id*)

get_integer_ranks()

get_mean_ranks()

get_mean_zscores()

get_randomised_rmse(*rmse*, *sigma=1*)
THis is useful for testing. See class documentaton

get_rank_participant(*participant*)

get_ranking()

species

valid_phosphos

exception `ScoringError`(*value*)
An exception class for scoring classes

class `HPNScoringPredictionInsilico_ranking`
This class is used to compute the ranks of the different participants based on an average rank over the 4 cell lines times phosphos

```
s = HPNScoringPredictionInsilico(filename="file1zip")
s.compute_all_rmes()

r = HPNScoringPrediction_ranking()
# s.aucs is a list where each element is a dictionary of
r.add_rme(s1.rmse, "team1")

rmse1 = r.get_randomised_rmse(r.rmse[0], sigma=1)
rmse2 = r.get_randomised_rmse(r.rmse[0], sigma=2)
rmse3 = r.get_randomised_rmse(r.rmse[0], sigma=3)

r.add_rmse(rmse1, "team2")
r.add_rmse(rmse2, "team3")
r.add_rmse(rmse3, "team4")

sall.add_rmse(s2.rmse, "team2")
sall.get_ranking()
{'team1': 1.96875, 'team2': 1.03125}
```

This class is independent of HPNSCoringPrediction. However, it takes as input the returned values of HPNScoringPrediction.compute_all_rmse()

`add_rmse`(*data*, *participant_id*)

`get_integer_ranks`()

`get_mean_ranks`()

`get_mean_zscores`()

`get_randomised_rmse`(*rmse*, *sigma=1*)
THis is useful for testing. See class documentaton

`get_rank_participant`(*participant*)

`get_ranking`()

class `HPNScoringPredictionInsilico`(*filename=None*, *verbose=False*, *version=2*)
dimension1 :inhibitor dimenssion2: phosp dimensson3 stimulus dimnesion4 : time

SC2B sub challenge (prediction in silico)

Parameters

- `filename` – file to score
- `version` (*str*) – default to 'official' (see note below). Set to anything else to use correct network

---

Note: This code use the official gold standard used in https://www.synapse.org/#!Synapse:syn1720047/wiki/60532 . Note, however, that a new corrected version is now provided and may be used. Differences with the official version should be small and have no effect on the ranking shown in the synapse page.

`compute_all_rmse`(*null=False*)

`create_random_data`()
Here, we don't want the true prediction that contains only what is requested (AZD8055) but the orignal training data with 2 or 3 inhibitors such as GSK and PD17 so that we can shuffle them.

We want to select for a given cell line and phosphos a data set to fill at a given time. The datum is slected accross the 8 stimuli, inhibitors +DMSO, and time points.

get_mean_rmse()

get_mean_zscores()

get_null(*N=100*, *tag='sc2b'*)

get_rmse(*inhibitor*, *phospho*)

> **Warning:** x in converted into a log2 scale

get_training_data()

get_true_prediction()

get_user_prediction()

get_zscores(*rmses=None*)

read_prediction_insilico(*filename*)
Reads true predcition from the 20 CSV files

read_true_prediction_michael(*filename*)

test_synapse_id = 'syn2009175'

true_synapse_id = 'syn2143242'

class D8C1(*version=2*, *verbose=True*, *download=True*, ***kargs*)
Factory for the D8C1 (HPN-Breast challenge)

download_goldstandard(*subname*)

download_template(*subname*)

score(*filename*, *subname=None*)

D8C2

class D8C2(*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D8C2 challenge

```
from dreamtools import D8C2
s = D8C2()
filename = s.download_template()
s.score(filename)
```

Data and templates are downloaded from Synapse. You must have a login.

constructor

download_goldstandard(*subname*)

download_template(*sub_challenge*)
Download template

Parameters **sub_challenge** – sc1 or sc2 string

score(*filename*, *subname*)
Scoring functions for the 2 sub challenges

score_sc1(*filename*)
See *D8C2_sc1* class for details

score_sc2 (*filename*)
See D8C2_sc2 class for details

class D8C2_sc1 (*filename*, *verboseR=True*)
Scoring class for D8C2 sub challenge 1

```
from dreamtools impoty D8C2
s = D8C2_sc1(filename)
s.run()
s.df
```

see github README for details

run()
Compute the score and populates df attribute with results

class D8C2_sc2 (*filename*, *verboseR=True*)
D8C2 Tox challenge scoring (sub challenge 2)

```
from dreamtools import D8C2_s2
s = D8C2_sc2(filename)
s.run()
s.df
```

see github README for details

run()
Compute the score and populates df attribute with results

DREAM9

D9C1

Based on https://github.com/Sage-Bionetworks/DREAM9_Broad_Challenge_Scoring/ and instructions and communications from Mehmet Gonen.

Original code in R. Translated to Python by Thomas Cokelaer

class D9C1 (*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D9C1 challenge

```
from dreamtools import D9C1
s = D9C1()
filename = s.download_template()
s.score(filename)
```

For consistency, all gene essentiality and genomic data files will be given in the same gct file format.

Briefly, this means:

The first and second lines contains the version string and numbers indicating the size of the data table that is contained in the remainder of the file:

```
#1.2
(# of data rows) (tab) (# of data columns)
```

The third line contains a list of identifiers for the samples associated with each of the columns in the remainder of the file:

```
        Name (tab) Description (tab) (sample 1 name) (tab) (sample 2 name) (tab) ... (sample N n

    And the remainder of the data file contains data for each of the genes.
    There is one line for each gene and one column for each of the samples.
    The first two fields in the line contain name and descriptions for the
    genes (names and descriptions can contain spaces since fields are
    separated by tabs). The number of lines should agree with the number of
    data rows specified on line 2.:

        (gene name) (tab) (gene description) (tab) (col 1 data) (tab) (col 2 data) (tab) ... (co
```

constructor

download_goldstandard(*subname=None*)

download_template(*subname=None*)

score(*filename*, *subname=None*)

D9C2

D9C3 scoring function

Based on original source code from Mette Peters found at https://www.synapse.org/#!Synapse:syn4308980

class D9C3(*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D9C3 challenge

```
    from dreamtools import D9C3
    s = D9C3()
    filename = s.download_template()
    s.score(filename)
```

Data and templates are downloaded from Synapse. You must have a login.

constructor

download_goldstandard(*subname=None*)

download_template(*subname=None*)

score(*filename*, *subname=None*)

DREAM9.5

D9dot5C1

D9dot5C1 challenge scoring functions

class D9dot5C1(*verbose=True*, *download=True*, ***kargs*)
A class dedicated to D9dot5C1 challenge

```
    from dreamtools import D9dot5C1
    s = D9dot5C1()

    s.download_templates()
    s.score('templates.txt.gz') # takes about 5 minutes
```

constructor

download_goldstandard(*subname*)

download_gs()

download_template(*name*)

download_templates()
Download a template from synapse into ~/config/dreamtools/dream5/D5C2

Returns filename and its full path

score(*filename*, *sub_challenge_name*)

score_sc1(*prediction_file*)
Compute all results and compare user prediction with all official participants

This scoring function can take a long time (about 5-10 minutes).

score_sc2(*prediction_file*)

DREAM10

3.5 Credits

Contributions to **DREAMTools** are direct (e.g. by contributing to the code in the github repository http://github.com/dreamtools/dreamtools) or indirectly (e.g., by developing an original scoring functions in DREAM challenges). Since **DREAMTools** framework is dynamic by nature, we won't keep a list of contributors, which would be outdated quickly. Instead, we recommend to check the scoring module themselves (e.g., dreamtools/dream5/D5C2/scoring.py) or the github log, the DREAM website (http://dreamchallenges.org), or the Synapse web of the challenge itself.

3.6 ChangeLog

3.6.1 1.3.0 16/03/2016

- stable version synchronised with F1000 v2 paper.
- add a new notebook example (D9C1)
- all Challenge constructor have now the following parameters: verbose, download and **kargs so that if another parameter is added, no code will need to be changed.
- documentation added for D7C4 to install Perl dependencies

3.6.2 1.2.6 02/03/2016

- **CHANGES:**
  - since dreamtools is now in bioconda, we removed the installation scripts conda_install.bat and conda_install.sh.
  - Doc updates.
  - conda recipes in ./conda removed (now in bioconda-recipes project)

3.6.3 1.2.5

- More cleanup for conda (removing tests from the distribution)

3.6.4 1.2.4

- cleanup for bioconda

3.6.5 1.2.3

- CHANGES: improved documentation
- NEWS: 2 installer for linux/mac and for windows

3.6.6 1.2.2

- NEWS: add –version in dreamtools standalone
- NEWS: add install.sh

3.6.7 1.2.1

- Some synapse data requires the synapse's user to accept the conditions of use on the synapse web page (by clicking on some widgets. In dreamtools, if the terms have not been accepted, there was just an error message from synapse. We now catch the error, print a more friendly message and open the synapse project so that the user can browse through the project directly to accept the conditions.

3.6.8 1.2.0

- Better handling of errors and more friendly messages in the dreamtools standalone app.

3.6.9 1.1.1

- Fixes a couple of Python3 issues
- Finalise travis integration. Coverage and testing simplified but allows Travis to finish on time. May add back tests little by little in the future.
- Update README and doc.

3.6.10 1.1.0

- Portage to Python3
- Add missing data files in D7C1
- Discard tests related to D9C3, which is not yet included.

3.6.11 1.0.0

first official release synchronized with submission to F1000 F1000 link

3.6.12 0.11

- adding license
- adding onweb option in the executable.
- settings now uses CustomConfig class from easydev rather than a local implementation.
- fixing the distribution (MANIFEST)
- adding a docker example
- fixing Login to be interactive not jsut an error
- Fix MANIFEST to add missing cython file and README.rst
- All challenges from DREAM2 to DREAM8 are included except for D6C2, D7C2 and

D8C3. D6C2 and D7C2 may be included soon and D8C3 is available on an external site. D8.5 and D9.5 and D9C1 also available.

3.6.13 0.9.2

- CHANGES: some changes in dream2/dream3 to finalise all those challenges.

3.6.14 0.9.1

- NEWS: add `dreamtools.dream9.scoring.D9C1` challenge
- NEWS: add `dreamtools.dream6.scoring.D6C3` challenge

3.6.15 0.9.0

- Upgrade version to higher number to reflect the fact that the package is now more robust

3.6.16 0.1.6

- Add a bunch of other challenges mostly D2/D3/D4/D5 and fixes + tests

3.6.17 0.1.5

- NEWS: some new classes `dreamtools.dream2` related to DREAM2
- NEWS: add `dreamtools.dream5.scoring.D5C1` challenge in Dreamtools
- NEWS: add `dreamtools.dream3.scoring.D3C2` challenge in Dreamtools
- NEWS: add `dreamtools.dream3.scoring.D3C3` challenge in Dreamtools
- NEWS: add `dreamtools.dream3.scoring.D3C4` challenge in Dreamtools
- Changes: fix `dreamtools.dream4.scoring.D4C2` challenge in Dreamtools

3.6.18 0.1.4

- NEWS: add `dreamtools.dream4.scoring.D4C2` challenge in Dreamtools
- NEWS: add `dreamtools.dream4.scoring.D4C1` challenge in Dreamtools
- CHANGES: move a download_data method from D5C2 into the `Challenge` main class to factorise some code.

3.6.19 0.1.3

- NEWS: add D4C3 challenge in Dreamtools

3.6.20 0.1.2

- NEWS: added dreamtools-layout for the developer to automatically create a challenge layout
- CHANGES: dreamtools-scoring now handles automatically new challenges providing the Challenge class has the mehod score() and download_template() available.

3.6.21 0.1.1

- NEWS: add D9dot5C1 challenge

3.6.22 0.1.0

- NEWS: Challenge D8C1, D8C2, D5C2, D7C1 (D6C1) available
- NEWS: dreamtools-scoring standalone provided

3.7 FAQS

3.7.1 Installation:: compilation issues

DREAMTools depends on packages (e.g., numpy, cython) that requires a C compilator. When using the **pip** commands dependencies will be compiled. This takes time but more importantly may fail (e.g., missing library). In this situation, we would recommend you to use Anaconda solution. This will also speed up the installation. Visit Anaconda.org and install the software. Once done, open a terminal and type:

```
pip install cython
pip install dreamtools
```

3.7.2 What are the challenges available in DREAMTools ?

You can either check this reference http://f1000research.com/articles/4-1030/v1 (Table1), or type:

```
dreamtools --help
```

3.7.3 I cannot find the gold standard or template in the source code, why ?

To keep DREAMTools package light-weight, we have moved some of the data files on Synapse website. However, data (templates and gold standard) are downloaded on request and stored locally in a common directory. For instance under Linux, the files are stored in /home/user/.config/dreamtools/. On Windows's platforms, the location looks like but may depend on the system: C:UsersuserAppDataLocaldreamtoolsdreamtools.

Once the DREAMTools package is installed, you can retrieve the location of a template of gold standard using the following methods (e.g., for D5C1 challenge):

```
from dreamtools import D5C1
s = D5C1()
s.download_template()
s.download_goldstandard()
```

3.7.4 Is DREAMTools available for Python 3 ?

Yes, it is Python3 compatible. Note, however, that some dependencies (e.g., gevent, synapseclient) were not Python3 compatible when we started the DREAMTools project. The gevent version available on the Python repository (pip command) should now be compatible. The synapseclient will be soon. Meanwhile, you should install this version:

```
pip install git+https://git@github.com/cokelaer/synapsePythonClient.git@v1.4.0_py3_dreamtools#eg
```

3.7.5 Do I need a Synapse account ?

It depends on the challenge you are interested in. In general, files will be downloaded from synapse so you may need to have an account. Besides, you may be requested to accept conditions of use of some data sets.

c

d

A

C

D

E

G

H

I

J

L

M

O

P

R

S

T

U

V

Z