GDSCtools

Release 0.19.0

Thomas Cokelaer

May 29, 2017

Contents

Current version: 0.19.0, May 29, 2017

Note developed and tested for Python 2.7, 3.4, 3.5

Note The GDSCTools libary works for Python 2.7 and 3.5 but the standalone pipeline to be ran on cluster works on Python 3.5 only

Source https://github.com/CancerRxGene/gdsctools

GDSCTools is a free open-source Python library dedicated to the study of drug responses in the context of the GDSC (Genomics of Drug Sensitivity in Cancer) project. The main developer is Thomas Cokelaer (Institut Pasteur), and it is a joint effort of the groups of Mathew Garnett (Sanger Institute) and Julio Saez-Rodriguez (RWTH Aachen & EMBL-EBI).

It contains utilities to find significant associations between drugs and genomic features (e.g., gene mutation) based on an ANOVA analysis. Other methods, such as multi-factorial linear models based on Elastic Net are also available. Besides, the library should also be useful for manipulating dedicated data sets such as IC50 (drug response) or MoBEM (genomic features) data structures. Hence, we hope that GDSCTools serves as basis for other scientists to develop further methods.

- **|association|**
- **|volcano|**
- **|histo|**
- **|elastic|**
- **|elastic_weights|**

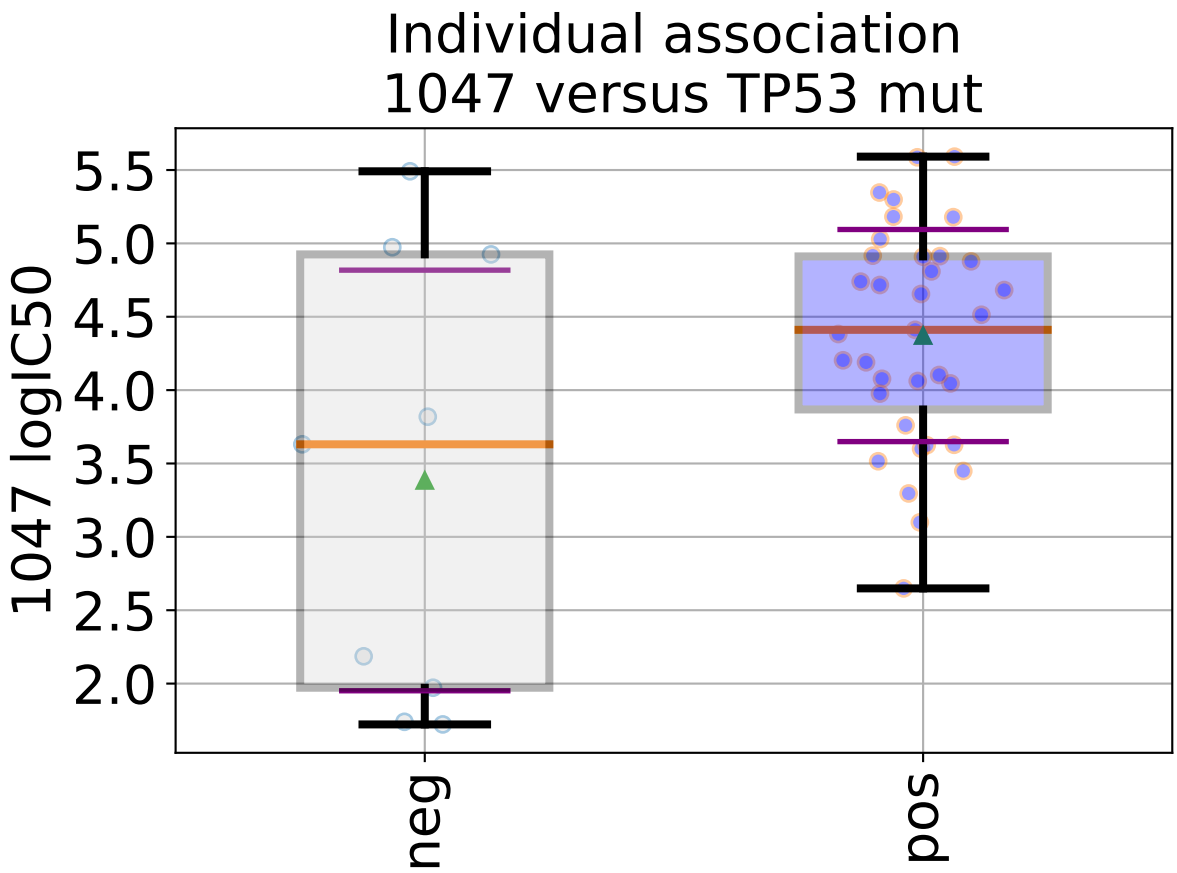
GDSCTools is written in Python. If you are a developer and/or knows already about the Python ecosystem and the **pip** command, just type the following command in a *Terminal* to install **GDSCTools**:

```
pip install gdsctools
```

add the option `--upgrade` to get the latest release. Conversely, if you are not familiar with Python or the command above, please see the *Installation* section for further details. Note also that we strongly recommend to use Anaconda to install dependencies (e.g., numpy, matplotlib).

In the following example, we provide a short Python snippet that uses the **GDSCTools** library. You can either copy and paste the code in a file, and execute it or type the commands in an *IPython* shell. With this example we investigate the associations between the *IC50* of a given drug (across 52 breast cancer cell lines) and a genomic feature (here, TP53 mutation). Drugs are refer to by a unique identifier (here 1047):

```
from gdsctools import ANOVA, ic50_test
gdsc = ANOVA(ic50_test)
gdsc.set_cancer_type('breast')
df = gdsc.anova_one_drug_one_feature(1047, 'TP53_mut', show=True)
```



The `df` object returned in the last statement is a dataframe that contains information explained in *Regression analysis* section.

See also:

For more examples and explanations, please visit the *ANOVA analysis (introduction)* section.

The previous example may be verbose with comments and warnings. You may set the verbose option to False and ignore warnings as follows:

```
import warnings
warnings.simplefilter("ignore","exceptions.Warning")
gdsc = ANOVA(ic50_test, verbose=False)
```

We will see more examples on how to use **GDSCTools** to perform more systematic studies. However, let us note that **GDSCTools** also provide a standalone application called **gdsctools_anova**, which can be used within a standard *Terminal* (same output as in the previous example):

```
gdsctools_anova --input-ic50 <ic50 filename> --drug 1047
    --feature TP53_mut --onweb
```

If you want to have a go, please download this `IC50 example`, which is required as an input.

Another data set is required for this analysis, which is a genomic feature file (see *Data Format and Readers*) but it can be replaced by yours. The default data set contains only a small set of genomic features and can be downloaded: `GenomicFeature example`, and adapted to your needs.

See also:

See *Standalone application* section for more details about the standalone application and the *Data Format and Readers* section to learn more about the expected input data formats.

CHAPTER 1

Contents

Installation

GDSCTools is written in Python and depends on a set of established scientific libraries such as Matplotlib (visualisation) and Pandas (data manipulation) to cite just a few. We post releases on the Python repository and make use of the Python ecosystem to provide a robust software. Would you have any trouble, please see the *FAQS* or fill an issue/ticket on github.

Python users and developers

Releases of **GDSCTools** are available on Pypi so **GDSCTools** can be installed in a *Terminal* using the **pip** command:

```
pip install gdsctools
```

Dependencies (e.g., Pandas, Matplotlib) should be taken care of automatically.

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. Anaconda will install the software required by **GDSCTools**. Since it does not require root access, it should not interfere with your system.

Please, visit the Anaconda website and follow the instructions. You may need to choose between 2 versions of Python (2.X or 3.5). **GDSCTools** is tested under Python 2.7 and 3.5 (as well as 3.3 and 3.4) so the version should not matter.

Once anaconda is installed, open a new *Terminal* (under Windows, open the Anaconda prompt) and type:

```
conda install numpy pandas numexpr matplotlib pandas scipy jinja2 statsmodels scikit-learn
```

Other dependencies and **GDSCTools** itself can be installed as follows:

```
pip install gdsctools
```

alternatively, if you prefer to get the source code (latest version), first obtain the source code and install manually the latest code:

```
# go in a working directory and type:
git clone https://github.com/CancerRxGene/gdsctools
cd gdsctools
python setup.py install
```

Install IPython

This is not strictly speaking required to use **GDSCTools**, but we strongly recommend to install IPython that provides a specialised shell for Python users. If you have installed Anaconda, just type:

```
conda install ipython
```

otherwise:

```
pip install ipython
```

You have already installed GDSCTools

If you have already installed **GDSCTools** and want to get the latest release, use the **pip** as follows:

```
pip install gdsctools --upgrade
```

If you used the source code from github, then, get the lastest source code and install again. Assuming you have installed the source code from github in **github_gdsctools** directory, then type:

```
cd github_gdsctools
git pull
python setup.py install
```

Testing your installation

You should now be ready to use **GDSCTools**. A good test is to check that the following executable is available. In a shell, type:

```
gdsctools_anova --test
```

or

```
gdsctools_anova --help
```

or for developers, start an IPython shell, and type for example:

```
from gdsctools import *
an = ANOVA(ic50_test)
```

Please, go to the next section for a *Quick Start* session.

Open an IPython shell

Under Windows, got to All Programs–>Anaconda –>Anaconda Prompt.

A shell will be opened where you can type **ipython** command.

Or alternatively, under Windows, got to All Programs–>Anaconda –>IPython

Notes for windows/mac/linux

The Anaconda method was tested successfully on the following systems: MAC, Windows 7 Pack1, Fedora 19 (Nov 2015) with version 0.16.5 of GDSCTools.

Under Windows, an error was raised due to scipy. This was fixed by typing:

```
conda remove scipy scikit-learn -y
conda install scipy scikit-learn -y
```

https://github.com/scikit-learn/scikit-learn/issues/4830

Quick Start

ANOVA analysis

If you already know what you can do with **GDSCTools**, we assume you have a well formatted *IC50* matrix and a genomic features binary matrix. Then, you can run the entire ANOVA analysis as follows:

```
from gdsctools import ANOVA
# For example, use these test files
# from gdsctools import ic50_test as ic50_filename
# from gdsctools import gf_v17 as genomic_feature_filename
gdsc = ANOVA(IC50_filename, genomic_feature_filename)
results = gdsc.anova_all()
```

And create an HTML report as follows:

```
from gdsctools import ANOVAReport
report = ANOVAReport(gdsc, results)
report.create_html_pages()
```

The **results** variable contains all tested associations within a single dataframe. The report will focus on significant associations and create boxplots or volcano plots accordingly.

More details about the ANOVA analysis itself can be found in the *ANOVA analysis (introduction)* and *The ANOVA analysis in details* sections. The data structure can be found in the next section (*Data Format and Readers*).

Regression analysis

Similarly, for the regression analysis, one can write a script as above:

```
from gdsctools import GDSCLasso
lasso = GDSCLasso(IC50_filename, genomic_feature_filename)
for drugid in lasso.drugIds:
    res = lasso.runCV(drugid, kfolds=8)
    best_model = lasso.get_model(alpha=res.alpha)
    weights = lasso.plot_weight(drugid, best_model)
    boxplots = lasso.boxplot(drugid, model=best_model, n=10, bx_vert=False)
```

However, we could recommend to use a worflow designed for this analysis. In a shell, type:

```
gdsctools_regression -I IC50_filename -F genomic_feature_filename
    --method lasso -o analysis
cd analysis
```

Edit the config.yaml file to change any parameters. Then, execute the pipeline:

```
snakemake -s regression.rules
```

See *Regression analysis* section for details.

Data Format and Readers

CSV and TSV formats

The main formats used in **GDSCTools** are *CSV*-based but *TSV*-based formatted files may be accepted although not encouraged. The data files may be compressed using bz2, xz, gzip of zip methods. See Pandas documentation to know the exact list of authorised compression method. Note that files saved using **GDSCTools** will be saved in CSV format only.

GDSCTools provides tools to read different kind of structured CSV files. For instance in the ANOVA analysis, these 3 types of CSV-input files defined in `readers` module) are used:

- *IC50*
- *GenomicFeatures*
- *DrugDecode*

There are all based on the same *Reader* class. Not a number values are encoded with the string **NA** or **NaN**. Besides, empty strings or fields made of spaces or tabs are also replaced by spaces.

Quote characters are also removed.

> **Warning:** Some readers will use the name of the extensions to infer the separator so it is important that the extension reflects the content of the file. A compressed file named as e.g. *ic50.csv.gz* will therefore be interpreted as a CSV file.

Note: With CSV files, if a column's name is ambiguous in the header (i.e, contains already a comma), then it should be enclosed within double quotes to avoid ambiguities (e.g *A,B* should be *"A,B"*). See the *Drug Decode* section for an example.

IC50

The specific format **IC50** is CSV file where the header must contain a column named **COSMIC_ID**. Other column should correspond to a given drug identifier (an integer). The order of the columns does not matter. So, each row contains the IC50s for a given COSMIC identifier.

Note: The IC50 matrix can be populated with other data (e.g., AUCs).

Although the name in the header does not matter note that if one column's name starts with **Drug** then all columns that do not start with **Drug** are ignored (except the special column COSMIC_ID). This feature was implemented to account for old data files that stores all Drug identifiers and also all genomic features within a single file.

Here is an example of IC50 input:

```
COSMIC_ID, 1,           20,           40
111111,    0.5,         0.8,          10
222222,    1,           2,            10
```

The following old-style IC50 would be equivalent since (i) the last column is dropped (does not start with Drug) and (ii) the other column's names are transformed into integer keeping only the middle part:

```
COSMIC_ID, Drug_1_IC50, Drug_20_IC50, Drug_40_IC50, other
111111,    0.5,         0.8,          10,           10
222222,    1,           2,            10,           20
```

If you save that example in a file, you can read it with the *IC50* class as follows:

```
>>> from gdsctools import IC50
>>> r = IC50('_static/ic50_tiny.csv')
>>> r.drugIds
[1, 20, 40]
```

Note: the columns' names should be identifiers (not drug names). There are two main reasons. The first one is that it allows us to keep anonymous all drug names and targets. The second reason is that many characteristics such as plate number and drug concentration may be associated with a drug identifier. This should be stored in a different table rather than in the name. It can then be handled and interpreted using the DrugDecode file (see below).

Note: column without a name are ignored.

See also:

developers should look at the references for more functionalities of the *IC50* class (e.g., filter by tissues, removing drugs, visualisation of IC50s).

Genomic Features

The **ANOVA** analysis computes the associations between the *IC50* and genomic features. This is the second input data set required for instance in the ANOVA analysis. Be aware that in the ANOVA analysis, the intersection between the IC50 and GenomicFeatures is made on the **COSMIC_ID**: cell lines not found in both CSV files will be dropped.

In addition to the COSMIC identifiers, the genomic feature file **should** contain the following columns:

```
- TISSUE_FACTOR
- MSI_FACTOR
- MEDIA_FACTOR
```

If not provided, the tissue, *MSI* and *MEDIA* factors will not be taken into account in the regression analysis. If the *TCGA* tissue is not provided, it is created and set to *unidentified*.

Note: Changed in version 0.9.11: A column called 'Sample Name' was interpreted if found. This is not the case anymore. It is actually removed now.

All remaining columns are assumed to be genomic features.

Warning: In the current version, all columns starting with *Drug_* are removed without warning.

Here is a simple example:

```
COSMIC_ID, TISSUE_FACTOR, MSI_FACTOR, BRAF_mut, gain_cna
111111, lung_NSCLC,  1, 1, 0
222222, prostate,    1, 0, 1
```

It can be saved and read as follows with the `GenomicFeatures`

```
>>> from gdsctools import GenomicFeatures
>>> gf = GenomicFeatures('_static/gf_tiny.csv')
>>> gf
GenomicFeatures <Nc=2, Nf=2, Nt=2>
```

In **GDSCTools**, we provide a `zipped Genomic Features file`. It contains about 1000 cell lines and 47 genomic features (gene mutations). A more complex file tagged v17 is also provided with about 600 features `v17 genomic feature`.

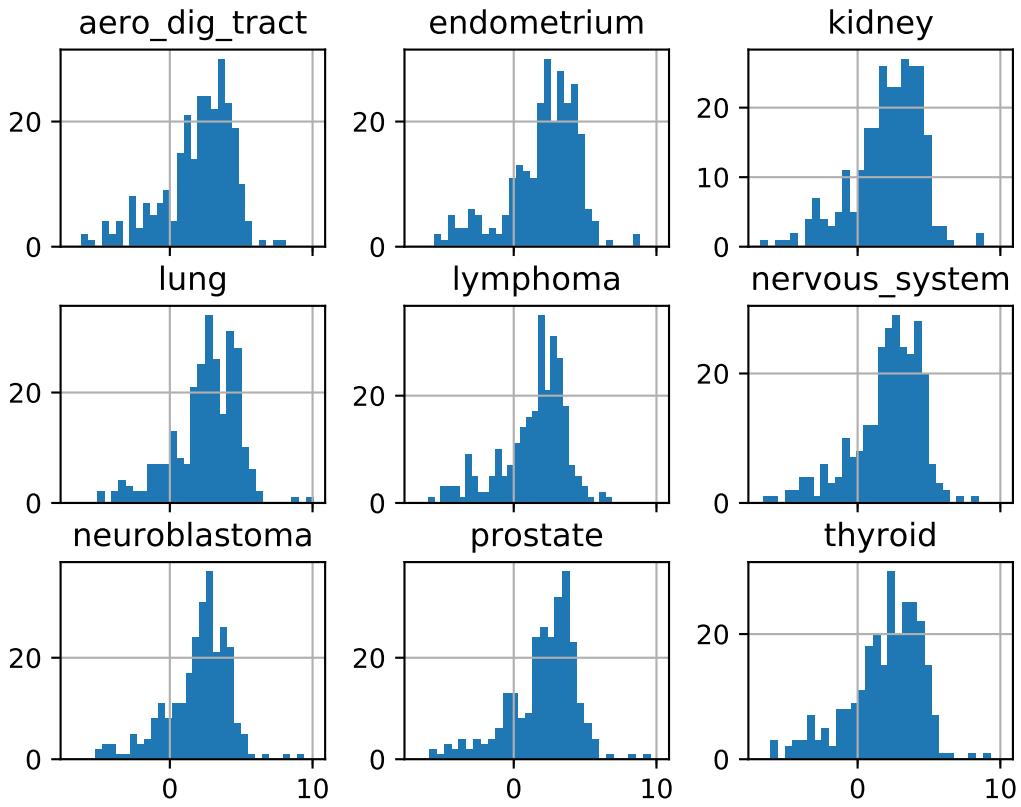
Note that you may create instance of GenomicFeatures without input but a default data set is loaded (the subset aforementionned):

```
>>> from gdsctools import GenomicFeatures
>>> gf = GenomicFeatures()
>>> print(gf)
Genomic features distribution
Number of unique tissues 27
Here are the first 10 tissues: myeloma, nervous_system, soft_tissue, bone, lung_NSCLC, skin, Bladder,
MSI column: yes
MEDIA column: no

There are 47 unique features distributed as
- Mutation: 47
- CNA (gain): 0
- CNA (loss): 0
```

Combine IC50 and Genomic Features

Here is an example on how to plot histograms of IC50s grouped by tissues. For convenience, we keep only 9 tissues.

```
from gdsctools import *
from numpy import mean
ic50 = IC50(ic50_v17)
gf = GenomicFeatures(gf_v17)
# select tissue column in same order as those stored in IC50 dataframe
tissues = gf.df.loc[ic50.df.index]['TISSUE_FACTOR']
ic50.df['tissue'] = tissues

# keep only 9 tissues
tokeep = list(set(tissues))[0:9]
ic50.df = ic50.df.query("tissue in @tokeep")

# Group by tissues
tt = ic50.df.groupby("tissue").aggregate(mean).transpose()
#plot histogram of IC50 group by tissues
tt.hist(bins=30, sharex=True)
```



Drug Decode

DrugDecode files are not required to perform the analysis. You may skip that section.

Drugs used in **GDSCTools** analysis may be public or not. In order to guarantee that drugs are kept anonymised (if not public), we enforce the CSV files that contains the IC50s to used drug identifiers instead of drug names.

When creating reports, the *Data Packages* producer or owner or the drugs may want to decode the drug identifier. The information to perform that task is provided within the **DrugDecode** CSV file.

The `DrugDecode` class reads a CSV file that contains information about a drug and its target(s). It must contain 3 columns named as follows:

```
DRUG_ID,     DRUG_NAME,  DRUG_TARGET
999,         Erlotinib,  EGFR
1039,        SL 0101-1,  "RSK, AURKB, PIM3"
```

Note the usage of quotes in the last row/last columns to avoid conflicts with the CSV format itself.

These columns will be used if provided:

```
- WEBRELEASE
- OWNED_BY
```

In addition, these columns may be populated for later use:

```
- CHEMSPIDER_ID
- PUBCHEM_ID
- CHEMBL_ID
```

An example can be read as follows:

```
>>> from gdsctools import DrugDecode, datasets
>>> drug_filename = datasets.testing.drug_test_csv.location
>>> dd = DrugDecode(drug_filename)
>>> dd.get_name(1047)
'Nutlin-3a'
>>> dd.df.loc[999]
CHEMBL_ID               NaN
CHEMSPIDER_ID           NaN
DRUG_NAME         Erlotinib
DRUG_TARGET            EGFR
OWNED_BY                NaN
PUBCHEM_ID              NaN
WEBRELEASE              NaN
Name: 999, dtype: object
```

DrugDecode files are not required for the analysis but are used by *gdsctools.anova_report.ANOVAReport* to fill the HTML reports.

You can also run the analysis and set the drug names and target later on as follows using the `drug_annotations` method:

```
from gdsctools import *
an = ANOVA(ic50_test)
an.anova_all()
results = an.anova_all()
dd = DrugDecode("v19_drug_decode.csv")
newdf = dd.drug_annotations(results.df)
```

ANOVA analysis (introduction)

Although we provide a *Standalone application* called **gdsctools_anova**, the most up-to-date and flexible way of using **GDSCTools** is to use the library from an *IPython* shell. This method (using IPython shell) is also the only way to produce *Data Packages*.

In this section we will exclusively use Python commands, which should also be of interest if you want to look at the *Notebooks* section later.

We assume now that (i) you have **GDSCtools** installed together with *IPython*. If not, please go back to the *Installation* section. (ii) you are familiar with the INPUT data sets that will be used hereafter.

Note: Beginners usually enter in an Python shell (typing python). Instead, we strongly recommend to use IPython, which is a more flexible and interactive shell. To start IPython, just type this command in a terminal:

```
ipython
```

You should now see something like:

```
Python 2.7.5 (default, Nov  3 2014, 14:33:39)
Type "copyright", "credits" or "license" for more information.

IPython 4.0.0 -- An enhanced Interactive Python.
```

```
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

Note: All snippets in this documentation are typed within IPython shell. You may see >>> signs. They indicate a python statement typed in a shell. Lines without those signs indicate the output of the previous statement. For instance:

```
>>> a = 3
>>> 2 + a
5
```

means the code **2 + a** should print the value 5

The IC50 input data

Before starting, we first need to get an IC50 data set example. Let us use this `IC50 example` test file.

See also:

More details about the data format can be found in the *Data Format and Readers* section as well as links to retrieve IC50 data sets.

In Python, one can easily import all functionalities available in **GDSCTools** as follows:

```
from gdsctools import *
```

Although this syntax is correct, in the following we will try to be more explicit. So, we would rather use:

```
from gdsctools import IC50
```

This is better coding practice and has also the advantage of telling beginners which functions are going to be used.

Here above, we imported the *IC50* class that is required to read the example file aforementioned. Note that this IC50 example is installed with **GDSCTools** and its location can be obtained using:

```
from gdsctools import ic50_test
print(ic50_test.filename)
```

The *IC50* class is flexible enough that you can provide the filename location or just the name **ic50_test** as in the example below, and of course the filename of a local file would work as well:

```
>>> from gdsctools import IC50, ic50_test
>>> ic = IC50(ic50_test)
>>> print(ic)
Number of drugs: 11
Number of cell lines: 988
Percentage of NA 0.206569746043
```

As you can see you can get some information about the IC50 content (e.g., number of drugs, percentage of NaNs) using the The print statement function. See *gdsctools.readers.IC50* and *Data Format and Readers* for more details.

Getting help

At any time, you can get help about a **GDSCTools** functionality or a python function by adding question tag after a function's name:

```
IC50?
```

With **GDSCTools**, we also provide a convenience function called `gsdctools_help()`:

```
gdsctools_help(IC50)
```

that should open a new tab in a browser redirecting you to the HTML help version (on ReadTheDoc website) of a function or class (here the `IC50` class).

The ANOVA class

One of the main application of **GDSCTools** is based on an ANOVA analysis that is used to identify significant associations between drug and genomic features. As mentionned above, a first file that contains the IC50s is required. That file contains experimentall measured IC50s for a set of N_d drugs across N_c cell lines. The second data file is a binary file that contains various features across the same cell lines. Those N_f features are usually of genomic types (e.g., mutation, CNA, Methylation). A default set of about 50 genomic features is provided and automatically fetched in the following examples. You may also provide your own data set as an input.

The default `genomic feature file` is downloadable and its location can be found using:

```
from gdsctools import datasets
gf =  datasets.genomic_features
```

See also:

More details about the genomic features data format can be found in the *Data Format and Readers* section.

The class of interest *ANOVA* takes as input a compulsary IC50 filename (or data) and possibly a genomic features filename (or data). Using the previous IC50 test example, we can create an ANOVA instance as follows:

```
from gdsctools import ANOVA, ic50_test
an = ANOVA(ic50_test)
```

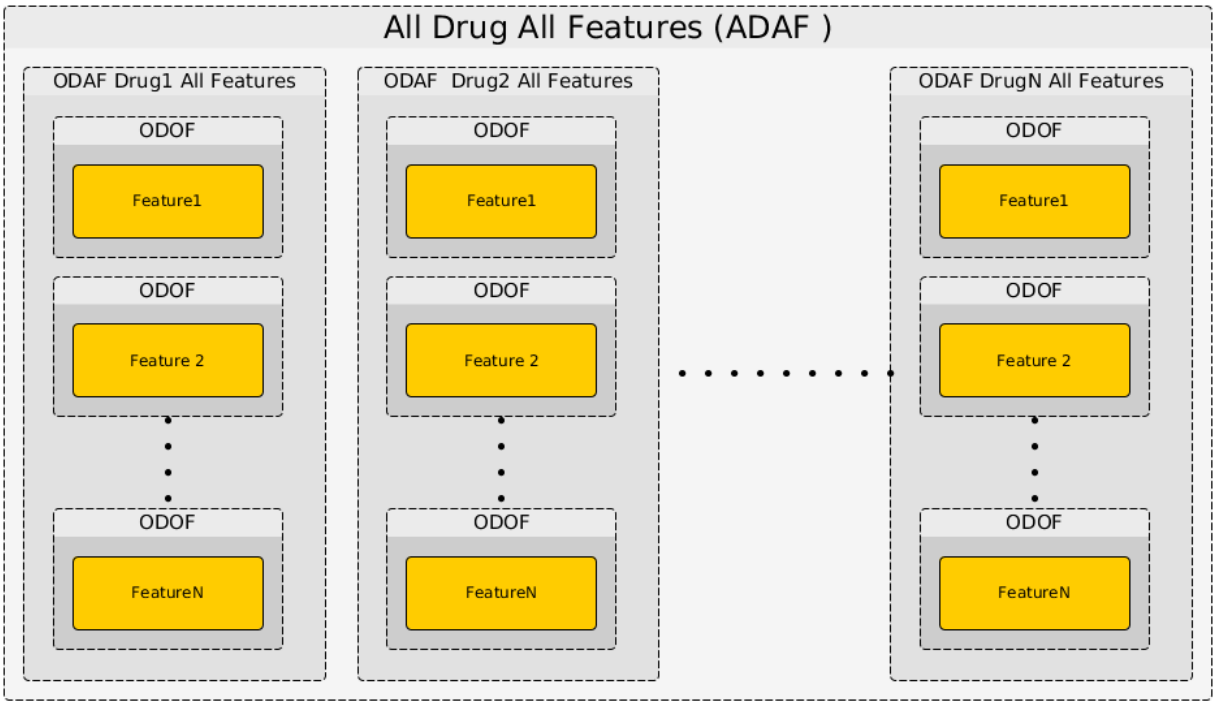
Again, note that the genomic features is not provided, so the default file aforementionned will be used except if you provide a specific genomic features file as the second argument:

```
an = ANOVA(ic50_test, "your_genomic_features.csv")
```

There are now several possible analysis but the core of the analysis consists in taking One Drug and One Feature (*ODOF* hereafter) and to compute the association using a regression analysis (see *Regression analysis* for details). The IC50 across the cell lines being the dependent variable Y and the explanatory variables denoted X are made of tissues, *MSI* and one genomic feature. Following the regression analysis, we compute the ANOVA summary leading to a p-value for the significance of the association between the drug's IC50s and the genomic feature considered. This calculation is performed with the *anova_one_drug_one_feature()* method.

We will see a concrete example in a minute. Once an *ODOF* is computed, one can actually repeat the *ODOF* analysis for a given drug across all features using the *anova_one_drug()* method. This is also named One Drug All Feature case (*ODAF*). Finally we can even extend the analysis to All Drugs All Features (*ADAF*) using *anova_all()*.

The following image illustrates how those 3 methods interweave together like Russian dolls.

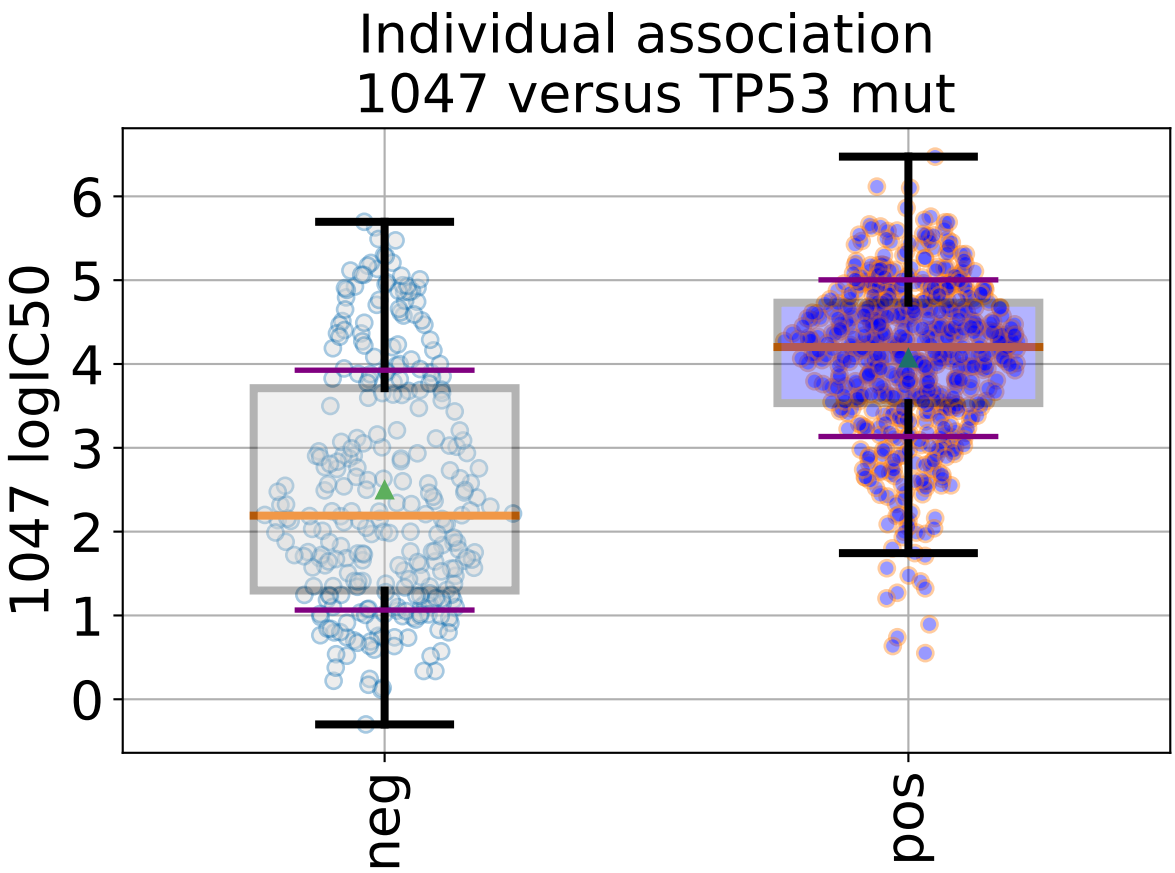


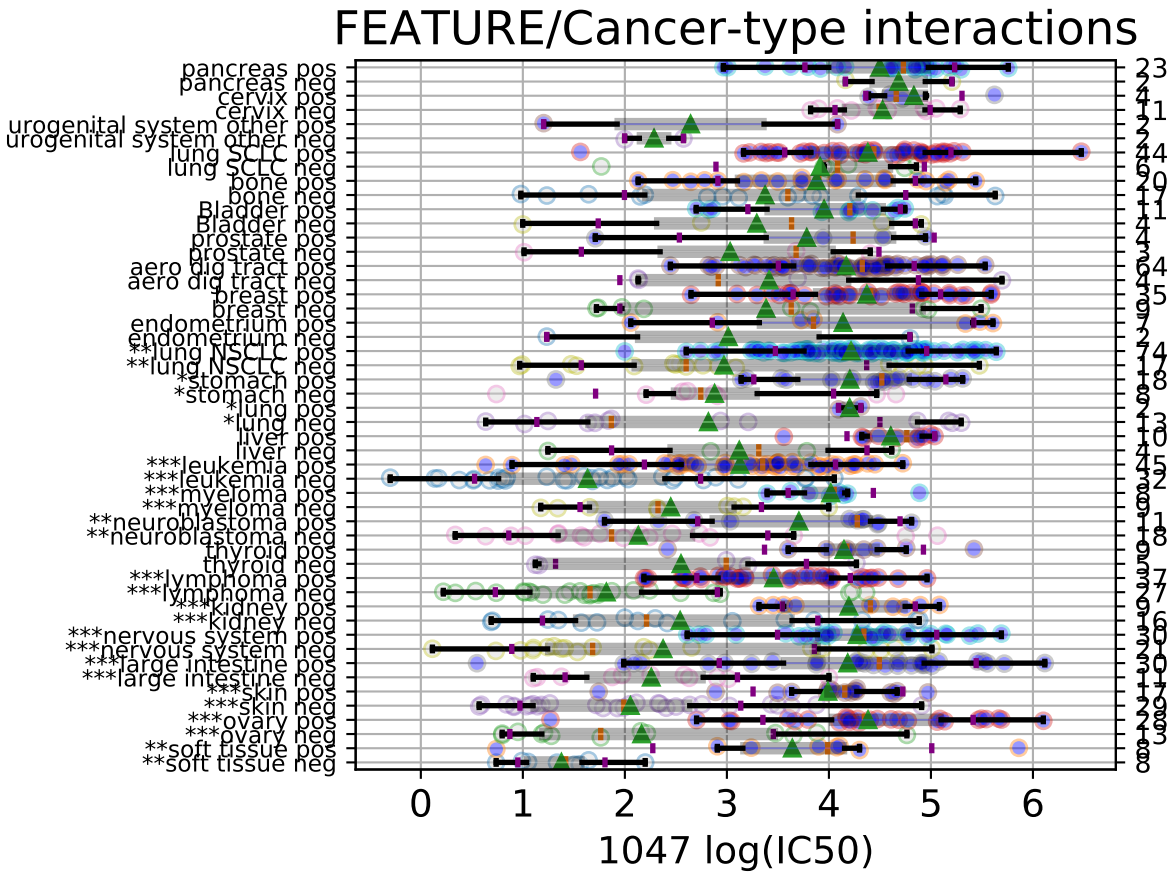
The computational time is therefore increasing with the number of drugs and features. Let us now perform the analysis for the 3 different cases.

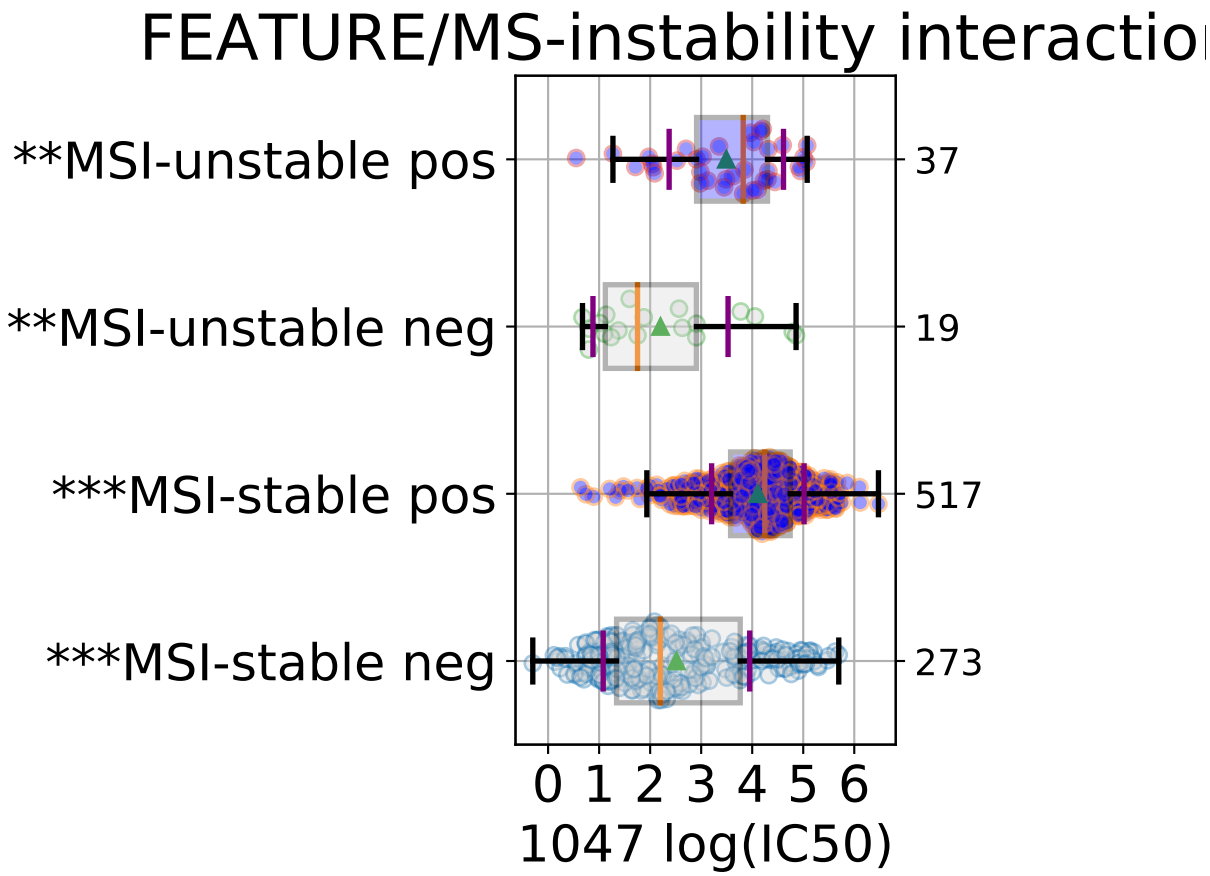
One Drug One Feature (ODOF)

Let us start with the first case (*ODOF*). User needs to provide a drug and a feature name and to call the *anova_one_drug_one_feature()* method. Here is an example:

```
from gdsctools import ANOVA, ic50_test
gdsc = ANOVA(ic50_test)
gdsc.anova_one_drug_one_feature(1047, 'TP53_mut', show=True)
```


Individual association
1047 versus TP53 mut
1047 logIC50
0
1
2
3
4
5
6
neg
pos


FEATURE/Cancer-type interactions
pancreas pos
pancreas neg
cervix pos
cervix neg
urogenital system other pos
urogenital system other neg
lung SCLC pos
lung SCLC neg
bone pos
bone neg
Bladder pos
Bladder neg
prostate pos
prostate neg
aero dig tract pos
aero dig tract neg
breast pos
breast neg
endometrium pos
endometrium neg
**lung NSCLC pos
**lung NSCLC neg
*stomach pos
*stomach neg
*lung pos
*lung neg
liver pos
liver neg
***leukemia pos
***leukemia neg
***myeloma pos
***myeloma neg
**neuroblastoma pos
**neuroblastoma neg
thyroid pos
thyroid neg
***lymphoma pos
***lymphoma neg
***kidney pos
***kidney neg
***nervous system pos
***nervous system neg
***large intestine pos
***large intestine neg
***skin pos
***skin neg
***ovary pos
***ovary neg
**soft tissue pos
**soft tissue neg
23
4
11
2
44
6
20
11
4
3
64
4
35
9
2
74
17
18
8
2
13
10
4
45
32
8
9
11
18
9
5
37
27
9
16
30
21
30
11
17
29
28
13
8
8
0
1
2
3
4
5
6
1047 log(IC50)



Setting the `show` parameter to True, we created a set of 3 boxplots that is one for each explanatory feature considered: tissue, *MSI* and genomic feature.

If there is only one tissue, this factor is included in the explanatory variable is not used (and the corresponding boxplot not produced). Similarly, the *MSI* factor may be ignored if irrelevant.

In the first boxplot, the feature factor is considered; we see the IC50s being divided in two populations (negative and positive features) where all tissues are mixed.

In the second boxplot, the tissue variable is explored; this is a decomposition of the first boxplot across tissues.

Finally, the third boxplot shows the impact of the *MSI* factor. Here again, all tissues are mixed. In the MSI column, zeros and ones correspond to MSI unstable and stab le, respetively. The **pos** and **neg** labels correspond to the feature being true or not, respetively.

The output of an *ODOF* analysis is a time series that contains statistical information about the association found between the drug and the feature. See for *gdsctools.anova_results.ANOVAResults* for more details.

If you want to repeat this analysis for all features for the drug **1047**, you will need to know the feature names. This is stored in the following attribute:

```
gdsc.feature_names
```

The best is to do it in one go though since it will also fill the FDR correction column based on all associationa computed.

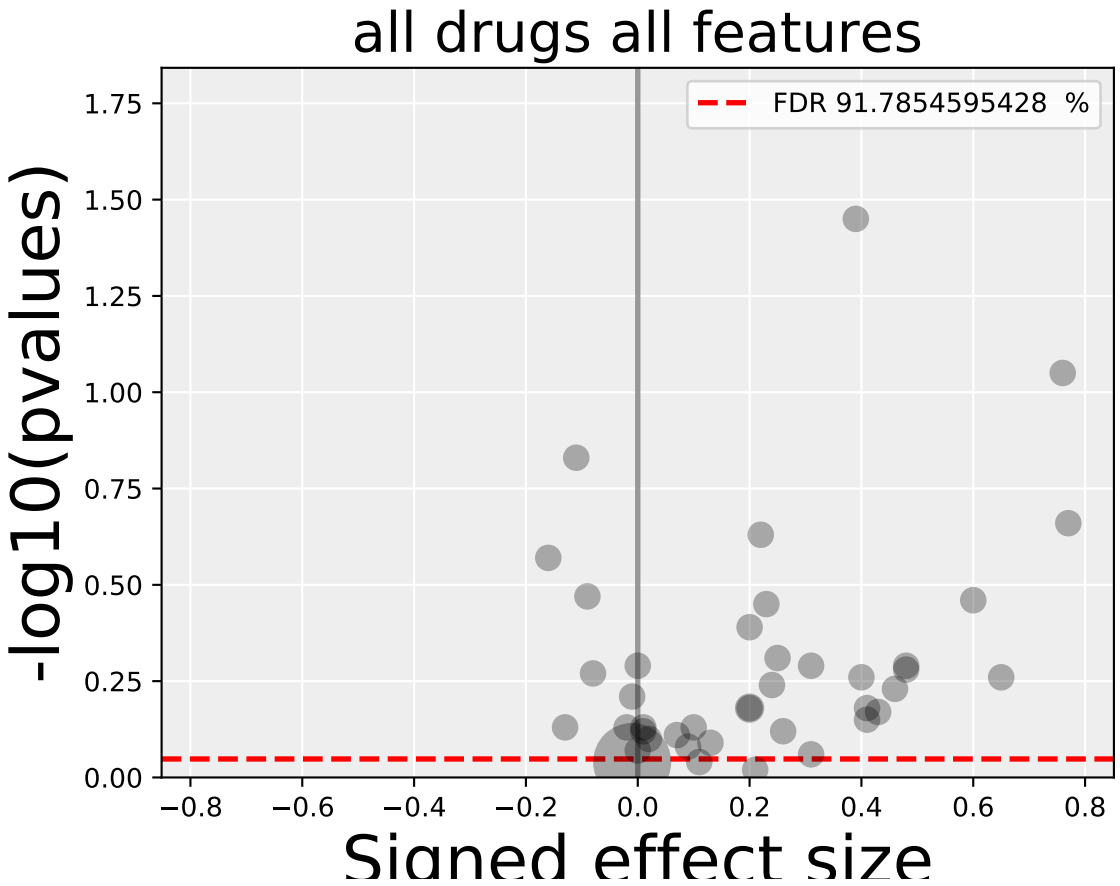
See also:

gdsctools.anova and *Data Packages*.

One Drug All Features (ODAF)

Now that we have analysed one drug for one feature, we could repeat the analysis for all features. However, we provide a method that does exactly that for us (`anova_one_drug()`):

```
from gdsctools import ANOVA, ic50_test
gdsc = ANOVA(ic50_test)
results = gdsc.anova_one_drug(999)
results.volcano()
```



In a python shell, you can click on a dot to get more information.

Here, we have a different plot called a volcano plot provided in the *gdsctools.volcano* module. Before explaining it, let us understand the x and y-axis labels.

Each row in the dataframe produced by anova_one_drug() is made of a set of statistical metrics (look at the header **results.df.columns**). It includes a p-value (coming from the ANOVA analysis) and a signed effect size can also be computed as follows.

In the ANOVA analysis, the population of IC50s is split into positive and negative sets (based on the genomic feature). The two sets are denoted $IC50_{pos}$ and $IC50_{neg}$. Then, the signed effect size η is computed as follows:

$$\eta = \text{sgn}(\Delta) * \text{Es}(\text{IC50}_{\text{pos}}, \text{IC50}_{\text{neg}})$$

where

$$\Delta = \overline{IC50_{pos}} - \overline{IC50_{neg}}$$

and Es is the effect size function based on the Cohens metric (see *gdsctools.stats.cohens()*).

In the volcano plot, each drug vs genomic feature has a p-value. Due to the increasing number of possible tests, we have more chance to pick a significant hit by pure chance. Therefore, p-values are corrected using a multiple testing correction method (e.g., BH method). The column is labelled *FDR*. Significance of associations should therefore be based on the FDR rather than p-values. In the volcano plot, horizontal dashed lines (red) shows several FDR values and the values are shown in the right y-axis. Note, however that in this example there is no horizontal lines. Indeed, the default value of 25% is well above the limits of the figure telling us that there is no significant hits.

Note that the right y-axis (FDR) is inversed, so small FDRs are in the bottow and the max value of 100% should appear in the top.

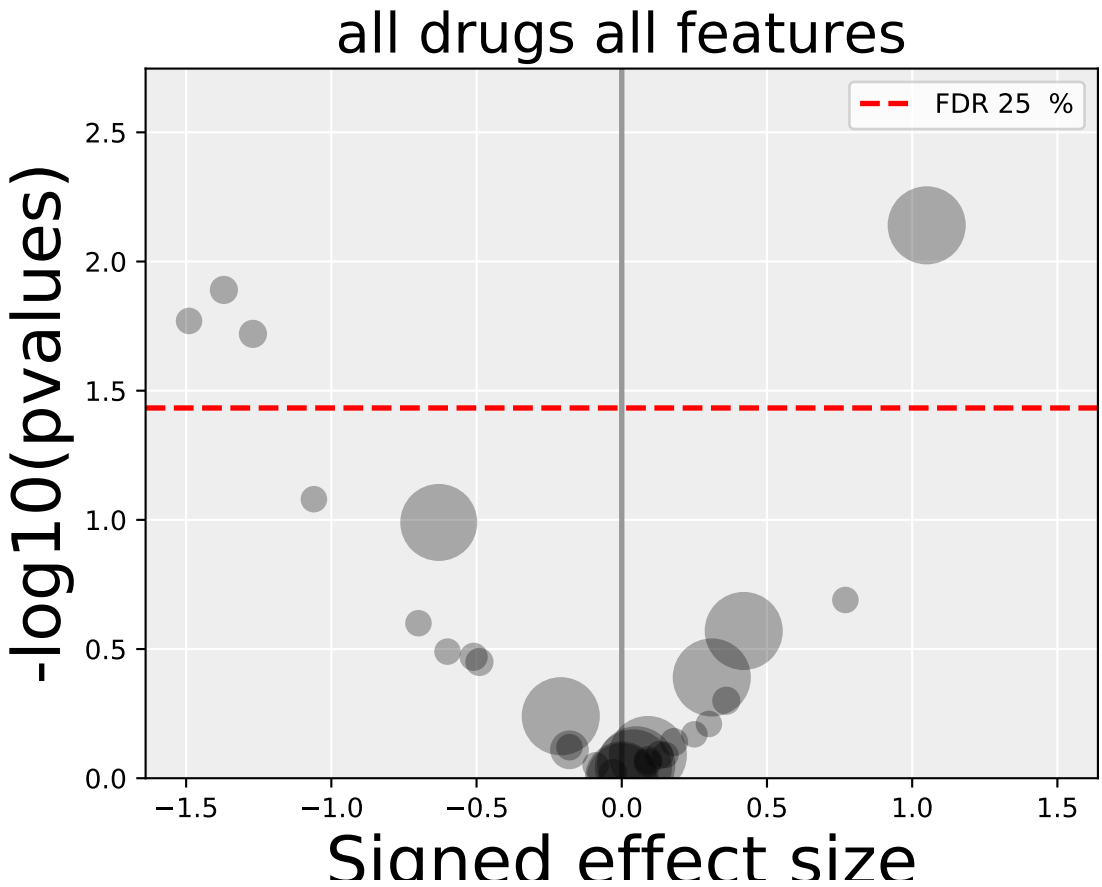
Note: P-values reported by the *ODOF* method need to be corrected using multiple testing correction. This is done in the the *ODAF* and *ADAF* cases. For more information, please see the *gdsctools.stats.MultipleTesting()* description.

All Drug All Features (ADAF)

Here we compute the associations across all drugs and all features. In essence, it is the same analysis as the *ODAF* case but with more tests.

In order to reduce the computational time, in the following example, we restrict the analysis to the breast tissue using `set_cancer_type()` method. This would therefore be a **cancer-specific analysis**. If all cell lines are kept, this is a *PANCAN* analysis. The information about tissue is stored in the genomic feature matrix in the column named **TISSUE_FACTOR**.

```
from gdsctools import ANOVA, ic50_test
gdsc = ANOVA(ic50_test)
gdsc.set_cancer_type('breast')
results = gdsc.anova_all()

results.volcano()
```



Warning: `anova_all()` may take a long time to run (e.g., 10 minutes, 30 minutes) depending on the number of drugs and features. We have a buffering in place. If you stop the analysis in the middle, you can call again `anova_all()` method and previous *ODAF* analysis will be retrieved starting the analysis where you previously stoped. If this is not what you want, you need to call *`reset_buffer()`* method.

The volcano plot here is the same as in the previous section but with more data points. The output is the same as in the previous section with more associations.

Learn more

If you want to learn more, please follow one of those links:

- *About the settings* also covers how to set some parameters yourself.
- Creating HTML reports from the analysis: *HTML report*.
- Learn more about the input *Data Format and Readers* .
- How to reproduce these analysis presented here above using the *Standalone application*.
- Get more examples from IPython *Notebooks*.
- How to produce *Data Packages* and learn about their contents.

The ANOVA analysis in details

Contents

About the settings

When using the *ANOVA* instance or the `ANOVAReport` to create an HTML report (see *HTML report*), all tunable settings are accessible from an attribute called `settings`:

```
from gdsctools import ANOVA, ic50_test
gdsc = ANOVA(ic50_test)
gdsc.settings
```

This `settings` attribute is an instance of *gdsctools.settings.ANOVASettings*, which is fully documented in the reference. For example in the *HTML report* section, we will change the default output **directory** where HTML pages are saved to a user-defined value.

It is also important to note that when calling ANOVAReport, the first argument is an ANOVA instance that already contains the settings. So, ANOVAReport will use that settings automatically (it may still be changed later). Consider this example:

```
>>> from gdsctools import ANOVA, ic50_test, ANOVAReport
>>> gdsc = ANOVA(ic50_test)
>>> gdsc.settings.FDR_threshold = 15
>>> results = gdsc.anova_all()

>>> ar = ANOVAReport(gdsc, results)
>>> ar.settings.FDR_threshold
15
```

We will see more settings here below but first let us come back on the ANOVA analysis.

Regression analysis

By default, the regression uses an *OLS* method. 4 Factors may be taken into account depending on the content of the `GenomicFeature` data set.

Here below, we use the R syntax where C() stands for categorical data and Y is the variable to be explained. Depending on the data and the `gdsctools.anova.settings` one of the following formula will be used:

$$Y \sim C(tissue) + C(media) + C(MSI) + Feature$$

$$Y \sim C(tissue) + C(MSI) + Feature$$

$$Y \sim C(MSI) + Feature$$

$$Y \sim Feature$$

Here are some rules applied:

- Feature factor is always included by definition and is in last position
- MSI and Media are included by default if found in the genomic feature data set. Note, however than one can exclude these factors using the `settings`.
- Tissue is included if there are more than 2 tissues. Again, one can change the `settings.analysis_type` to the name of the tissue (instead of PANCAN, the default value).

Note: The order of the different features in the equations may have an impact on the analysis.

Since analysis may be time-consuming, we have hard-coded the regression formula. Note, however, that in version 0.16, we have added the *anova_one_drug_one_feature_custom()* method, which can be use for any type of regression based on a user formula. This is slower than the 4 hardcoded versions mentionned above but is more flexible. One can for instance set the formula to specify the treatement to be used as a reference:

Changed in version 0.16: The regression method is the *OLS* method. Other methods will be used in an independent module (`gdsctools.regression`)

The ANOVA analysis itself uses a **type I** error. The summary can be obtained for a specific combination of drug and feature as follows:

```
from gdsctools import *
an = ANOVA(ic50_test, gf_v17)
drugid = 1047
feature = an.feature_names[0]
odof = an._get_one_drug_one_feature_data(drugid, feature)
res = an.anova_one_drug_one_feature(drugid, feature)
an._get_anova_summary(an.data_lm, output="dataframe", odof=odof)
```

and should show the following summary:

```
               Df        Sum Sq     Mean Sq  F value        PR(>F)
tissue       26.0    352.345257   13.551741  9.26853  1.63864e-31
msi           1.0      5.309389    5.309389  3.63129    0.0570537
feature       1.0      3.186109    3.186109   2.1791     0.140282
Residuals   817.0   1194.554709    1.462123     None         None
```

An alternative (simpler but slower) way since version 0.16 is to use:

```
an.anova_one_drug_one_feature_custom(drugid, feature,
    formula='Y ~ C(tissue) + C(msi) + feature')
```

Type I error

The ANOVA analysis is based on a **Type I** error, also called *sequential* sum of squares. Consider 2 effects A and B, it tests the main effect of factor A, followed by the main effect of factor B after the main effect of A, followed by the interaction effect AB after the main effects. So, this type of sums of squares gives different results for unbalanced data depending on the sequence.

MSI factor

MSI is always included by default. However, you may exclude it by setting its value to False:

```
settings.include_MSI_factor
```

If **MSI_FACTOR** column is not found in the Genomic Feature data set, the MSI factor will be excluded automatically and the parameter above set to False.

Warning: If you force the MSI factor to True whereas there is not enough data in the binary sets of the MSI factor, error will be raised.

MEDIA factor

If included in the genomic feature data set, MEDIA are included by default. However, you may exclude it by setting its value to False:

```
settings.include_MEDIA_factor
```

If **MEDIA_FACTOR** column is not found in the Genomic Feature data set, the MEDIA factor will be set automatically to False.

Tissue factor

Another factor used in the regression (tissue) will be automatically excluded if there is only one tissue (or none). If several tissues are available, you can still exclude it from the regression analysis by settings this parameter to anything different from the default value (PANCAN):

```
settings.analysis_type = PANCAN
```

Filtering

When performing the analysis for a given drug and feature, the regression may not be performed if there is not enough statistics.

These parameters will influence the number of tests being performed (number of associations of drug vs feature in *anova_all()*):

```
- minimum_nonna_ic50
- MSI_feature_threshold
- feature_factor_threshold
```

The first parameter indicates the minimum number of valid IC50 required for a given drug to be analysed. The current default value is 6.

The second parameter indicates the minimum size of the positive and negative population when IC50 are filtered by MSI factor (defaults to 2).

The third parameter indicates the minimum size of the positive and negative population when IC50 are filtered by Feature factor (defaults to 3).

This table summarizes the effect of these parameters:

	IC50 data							Genomic Features			
Drug name:	D_1	D_NA	D_pMSI=0	D_pMSI=1	D_2	D_3		MSI	feature1	feature2	feature3
IC50s	0.1	0.1	NA	NA	0.1	0.1		1	1	1	1
	0.2	0.2	NA	0.2	0.2	0.2		1	1	1	1
	0.3	NA	0.3	0.3	NA	0.3		0	1	0	1
	0.4	0.4	0.4	0.4	0.4	0.4		0	1	0	0
	0.5	0.5	0.5	0.5	0.5	0.5		0	0	0	0
	0.6	NA	0.6	0.6	0.6	NA		0	0	0	0
	0.7	NA	0.7	0.7	0.7	NA		0	0	0	0
	0.8	0.8	0.8	0.8	0.8	0.8		0	0	0	0
valid NA?	YES	NO	YES	YES	YES	YES					
MSI valid	YES		NO	NO	YES	YES					
valid feature	1 and 3				1	3					

The left hand side table mimics the IC50 data. The first column should and last 3 rows are not to be included in an IC50 matrix (see *Data Format and Readers*) but are added here as annotations for the following discussions.

When the regression analysis is performed for a given drug and a given feature, 3 filters are applied. First, a minimum number of values is required (minimum_nonna_ic50 setting). Therefore, the drug is not analysed. The second check is performed with respect to the MSI values. A drug can be analysed only if (once NA have been discarded) the number of IC50s corresponding to positive and negative MSIs is greater or equal to **MSI_feature_threshold**. In our example, the drugs in column **D_pMSI=0** and **D_pMSI=1** are therefore discarded since they have zero and only one positive MSI, respectively.

Finally, similarly to the MSI check, a drug/feature association is analysed if the number of IC50s corresponding to positive and negative feature is or equal to **feature_factor_threshold**.

Multiple testing corrections

By default, the multiple testing correction is based on Benjamini–Hochberg (BH) method but it can be set to other methods using

```
settings.pval_correction_method
```

See also:

MultipleTesting for details.

The multiple testing is performed globally across all drugs and all cell lines.This parameter is stored in

```
settings.pvalue_correction_level
```

By default it is set to *global*. Set it to *local* to keep the multiple correction at the drug level (ODAF).

When you perform an ANOVA analysis, the multiple correction method is used to populate the results column named ANOVA_FEATURE_FDR.

If you change your mind and wish to run the analysis with another method, you do not need to re-run the entire analysis. Instead, simply change the method's name and call `anova_all()` again. Only the multiple testing computation is performed, skipping ANOVA testing, which have already been done.

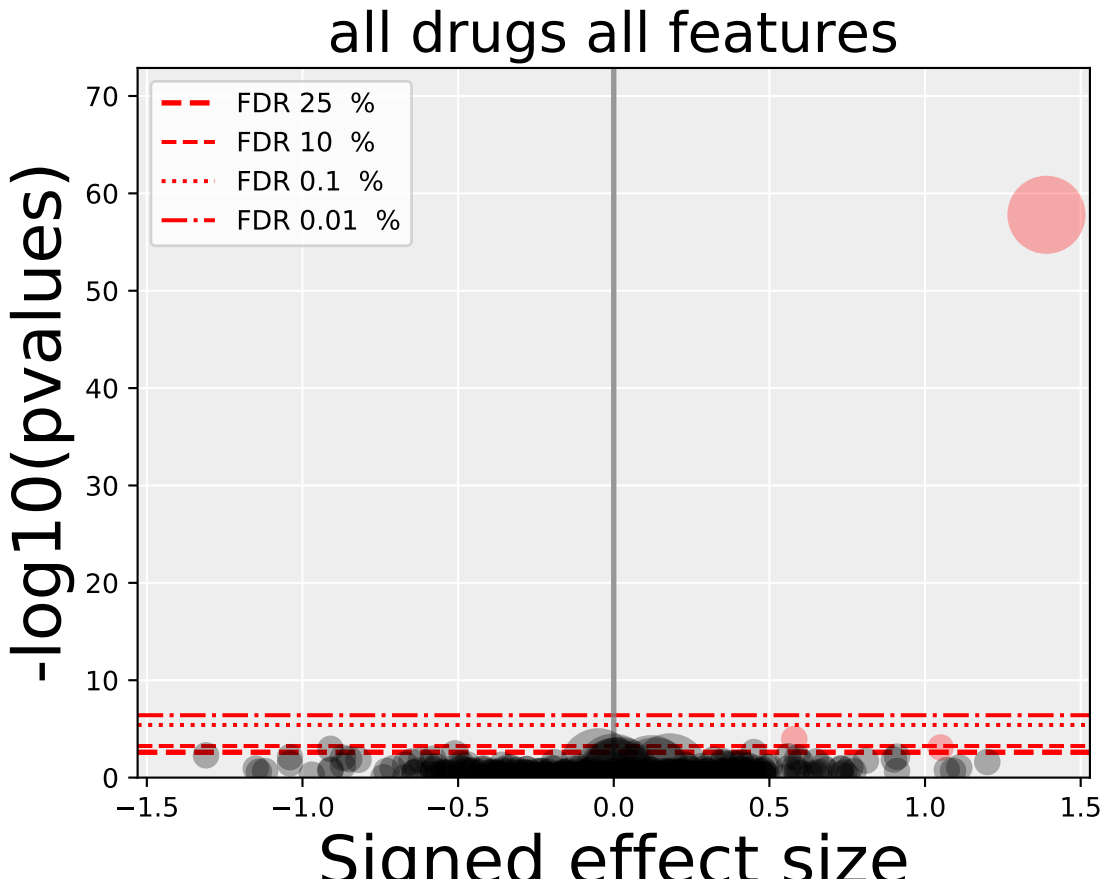
```
results = an.anova_all()
an.settings.pvalue_correction_method = 'qvalue'
results = an.anova_all()
```

volcano plots

The volcano plots are one of the main results of the analysis and summarizes visually the significance of the different associations.

It is part of the `AnovaResults` class and is returned either by an ODAF or ADAF analysis:

```
from gdsctools import ANOVA, ic50_test
gdsc = ANOVA(ic50_test)
res = gdsc.anova_all()
res.volcano()
```



Here are some parameters used to tune the plots and selection of significant events:

- **pvalue_threshold** is used to select significant hits. See *ANOVAReport*.
- **effect_threshold** is used to select significant hits as well.
- **FDR_threshold** is used in *gdsctools.volcano.VolcanoANOVA* (horizontal lines)
- **volcano_FDR_interpolation** uses interpolation to plot the FDR lines in the volcano plot.
- **volcano_additional_FDR_lines** : [0.01, 0.1, 10]

See also:

VolcanoANOVA.

others

See *ANOVASettings* for the full listing.

Note: Some settings will be set automatically when calling some functions. For instance, if you call `anova.ANOVA.set_cancer_type()` to a single tissue, then the analysis_type will be set to the tissue's name. If there are not enough positive or negative MSI, the MSI factor will ignored.

HTML report

The output of an ANOVA analysis can be used to create an HTML report.

First, let us generate the results again (see *Quick Start*).

```
>>> from gdsctools import ANOVA, ic50_test
>>> gdsc = ANOVA(ic50_test)
>>> gdsc.set_cancer_type('breast')
>>> results = gdsc.anova_all()
```

The `results` variable contains a dataframe `df`. This dataframe contains as many rows as associations of drugs and features. See *ANOVAResults* for the contents. The HTML report extracts the significant associations, and then create figures and HTML pages for each of the associations that are significant. You can easily create HTML report as follows:

```
>>> from gdsctools import ANOVAReport
>>> report = ANOVAReport(gdsc, results)
>>> report.create_html_pages()
```

The report creates a **Data Package**, which details can be found in the *Data Packages* section.

Images are created for each significant associations and may take a while.

Some tunable settings are available in the `settings` (see *About the settings*). For instance, you can set the output directory to a user value instead of (html_gdsc_anova):

```
>>> report.settings.directory = 'BLCA'
```

Data Packages

Warning: Only one IC50 files should be provided. It is filtered out according to the genomic feature file.

Definition

A **Data Package** is a terminology used to speak about a directory that contains the results of an analysis. For example, the data package called **BLCA** has a tree directory that looks like:

```
BLCA/
|-- code
|-- css
|-- images
|-- INPUT
|-- js
`-- OUTPUT
```

The main directory contains a file named **index.html**. Many other HTML files may be present but the **index** is your entry point to browse the content of data package.

The directory **css** and **js** contains resources required by the HTML documents.

The **INPUT** directory contains 3 data files and the settings used during the analysis:

1. ANOVA_input.csv
2. DRUG_DECODE.csv
3. genomic_features.csv
4. settings.json

See section on *Data Format and Readers* for details about the data format and the *About the settings* section.

Finally, the **OUTPUT** directory contains:

```
- drugs_summary.csv
- features_summary.csv
- results.csv
```

The **images** directory contains a mix of images and HTML for each significant associations.

Create your own package

In fact, we have already seen how to create a package. This is covered in *HTML report* when we used the `ANOVAReport` class but let us look at the code again:

```
from gdsctools import ANOVA, ic50_test, ANOVAReport
gdsc = ANOVA(ic50_test)
results = gdsc.anova_all()

report = ANOVAReport(gdsc, results)
report.create_html_pages()
```

Here, we have not yet mentionned the type of cancers or tissues since we used a simple genomic feature file but one we need to repeat this analysis across man y different genomic features files. The *gdsctools.gdsc.GDSC* class will help us for that purpose.

Create data packages across TCGA

When we do a full GDSC analysis, the cell lines span a set of TCGA tissues (e.g., COREAD, BLCA) and generally we want to perform the analysis not on all cellines at the same time but each type of tissues independently.

Besides, you may then wish to have data packages not only for a given TCGA tissue but also for a given company (if your DrugDecode file is filled properly; see later).

The recommended way is to used the *gdsctools.gdsc.GDSC* class that will help you in this task.

First, you need to prepare the input data. Create a directory and add these files:

```
- a unique IC50 file
- The genomic features files for each type of tissues.
- The DrugDecode file
```

The genomic feature must be named as follows:

```
<prefix>_BLCA.csv
<prefix>_COREAD.csv
...
```

The name of the TCGA can include **ALL**, **PANCAN** and will be used later to create the directories for each data paakage.

The important point being that there must be an underscore only and followed by the TCGA tag.

The GDSC class will then loop over the TCGA cases and create data packages.

```
from gdsctools import GDSC
gg = GDSC("IC50.csv", "DrugDecode.csv", "GF_*.csv")
gg.anaalyse()
```

This may take hours to finalise: the ANOVA and creation of all images will be done for each TCGA.

This may be parallelised since each input Genomic Feature analysis is independent:

```
gg_blca = GDSC("IC50.csv", "DrugDecode.csv", "GF_BLCA.csv")
gg_blca.analyse()

gg_coread = GDSC("IC50.csv", "DrugDecode.csv", "GF_COREAD.csv")
gg_coread.analyse()
```

In an error occurs for one Genomic Feature file, the analysis we jump to the next file. You may need to check re-run the specific TCGA tissue analysis your self when an error occured (meaning you do not need to re-run everything).

Once done, you should have all data packages locally in the directory where you ran the scripts.

The next step is to read back all those results and create data pacakges dedicated to a company. Based on the DRUG_DECODE file:

```
gg = GDSC("IC50.csv", "DrugDecode.csv", "GF_*.csv")
gg.create_data_packages_for_companies()
```

For each companies, which names can be checked with:

```
gg.companies
```

a new directory (data package) is created locally

For now, it is important to run this in the same directory where previous pacakges were created.

Again thiis may be parallelised:

```
for each company in gg.companies:
    single = GDSC("IC50.csv", "DrugDecode.csv", "GF_*.csv")
    single.create_data_packages_for_companies([company])
```

Create summary pages

Following the creating of the "all" TCGA packages and the dedicated packages for all companies, you end up with quite a few directories. This command will create summary HTML page to ease your life:

```
gg.create_summary_pages()
```

This must be called after `analyse()` and `create_data_packages_for_companies()`.

OmniBEM Builder

OmniBEM Builder is an optional tool within **GDSCTools** designed to give the user more flexibility on the levels of genomic annotation probed by **GDSCTools**.

By default, **GDSCTools** is based on the genomic annotation of 1001 cell lines represented in COSMIC and published by *Iorio et al* (Cell Resource). The annotation includes genetic variants as identified by exome sequencing, copy number alterations and differentially methylated CPG islands.

OmniBEM Builder allows the user to merge the different levels of annotations into a single gene-level view that queries whether a given gene has been altered at any level of annotations.

The user can additionally specify which sets of genomic annotations to integrate as well as upload and integrate their own sets of genomic annotations.

```
from gdsctools import OmniBEM
```

Input Data Structure

At its most basic, OmniBEM Builder requires a cell line ID (e.g. COSMIC ID), a gene name and an alteration type (e.g. Point mutation, Amplification or Copy Number Alteration). A simple example table is given below:

COSMIC ID	GENE	TYPE	SAMPLE	TISSUE_TYPE	IDENTIFIER
111	ZAP	Methylation	NM	breast	1

Example

We provide a data set available in **GDSCTools** that can be loaded as follows

```
from gdsctools import gdsctools_data, OmniBEMBuilder

input_data = gdsctools_data("test_omnibem_genomic_alterations.csv.gz")
bem = OmniBEMBuilder(input_data)
```

The data is stored in the `df` attribute:

```
bem.df
```

From this data frame, one can filter, group and perform various data mangling operations. For instance these commands group the data by tissue type, count the number of row per tissue and return the 5 most representative tissues:

```
>>> count = bem.df.groupby("TISSUE_TYPE").count()
>>> list(count.sort_values("COSMIC_ID").index[-5:])
['SKCM', 'BRCA', 'HNSC', 'LUAD', 'COAD/READ']
```

In *OmniBEMBuilder*, we provide convenient methods to filter the data by genes, cosmic IDs, sample names, tissues and types.

For instance:

```
>>> # You may filter the data for instance to keep only a set of genes.
>>> # Here, we keep the 100 most present genes:
>>> bem = OmniBEMBuilder(input_data)
>>> len(bem)
56943
>>> gene_list = bem.get_significant_genes(100)
>>> bem.filter_by_gene_list(gene_list.index)  # gene_list is a dataframe
```

```
>>> len(bem)
6141
```

Once you BEM data is filtered as expected, save it:

```
gf = bem.get_genomic_features()
gf.to_csv("Your_genomic_features.csv")
```

This file can now be used with tha ANOVA or Regression analysis.

See also:

gdsctools.omnibem.OmniBEMBuilder

Regression analysis

Since version 0.15, we have also the ability to perform a regression analysis at the drug level (*ODAF* mode).

The are currently 3 types of regression implemented:

1. Ridge
2. Lasso
3. Elastic Net

Information about functionalities implemented in GDSCTools are available in the reference `gdsctools.regression.` and in a notebook (see https://github.com/CancerRxGene/gdsctools/tree/master/notebooks) called Regression.

We first give some examples following information contained in the reference and the notebook. In the example here below, the data sets are small. In practice, you may have many features and would require a cluster to perform the computation in a reasonable amount of time. We therefore develop a pipeline that wraps up the analysis. The pipeline uses the Snakemake framework (see below).

Doing the analysis using GDSCTools library

As usual, we will use some data provided in GDSCTools in the form of ain IC50 data set and a genomic features data set:

```
from gdsctools import *
ic50 = IC50(ic50_v17)
gf = GenomicFeatures(gf_v17)
```

To decrease the computational time, let us select only genomic features that are factors or mutations:

```
gf.df = gf.df[[x for x in gf.df.columns if "FACTOR" in x or "mut" in x]]
```

Here, we will use the Lasso regression:

```
lasso = GDSCLasso(ic50, gf)
```

For each drug, all features are taken and a regression is performed. For each drug, we tune the **alpha** parameter using a cross validation (a K-fold CV).

Note: the regression implementation is based on scikit-learn. The **alpha** parameter is called **lambda** in the R glmnnet package. In the ElasticNet case, and additional parameter is called **l1_ratio** (0.5 by default) and correspond to the glmnet parameter **alpha**.

Before, doing the tuning, let us choose one drug. Let us pick up an interesting one:

```
drugid = 1047
```

Tuning

Users are able to choose the number of K-folds, which is set to 10 by default. Here is the method to call:

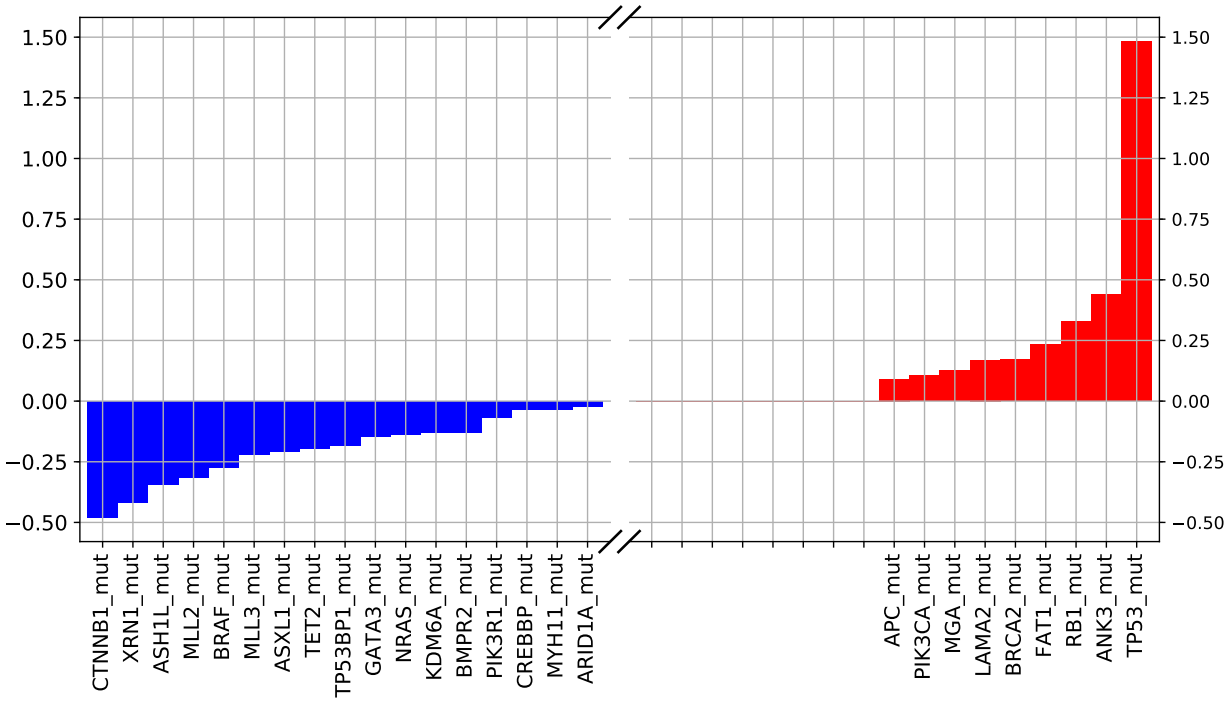
```
res = lasso.runCV(drugid, kfolds=8)
```

The returned object contains the best alpha but also the pearson coefficient:

```
res.alpha
```

From this parameter, the best model can be created and used for further analysis, in particular to see the important features:

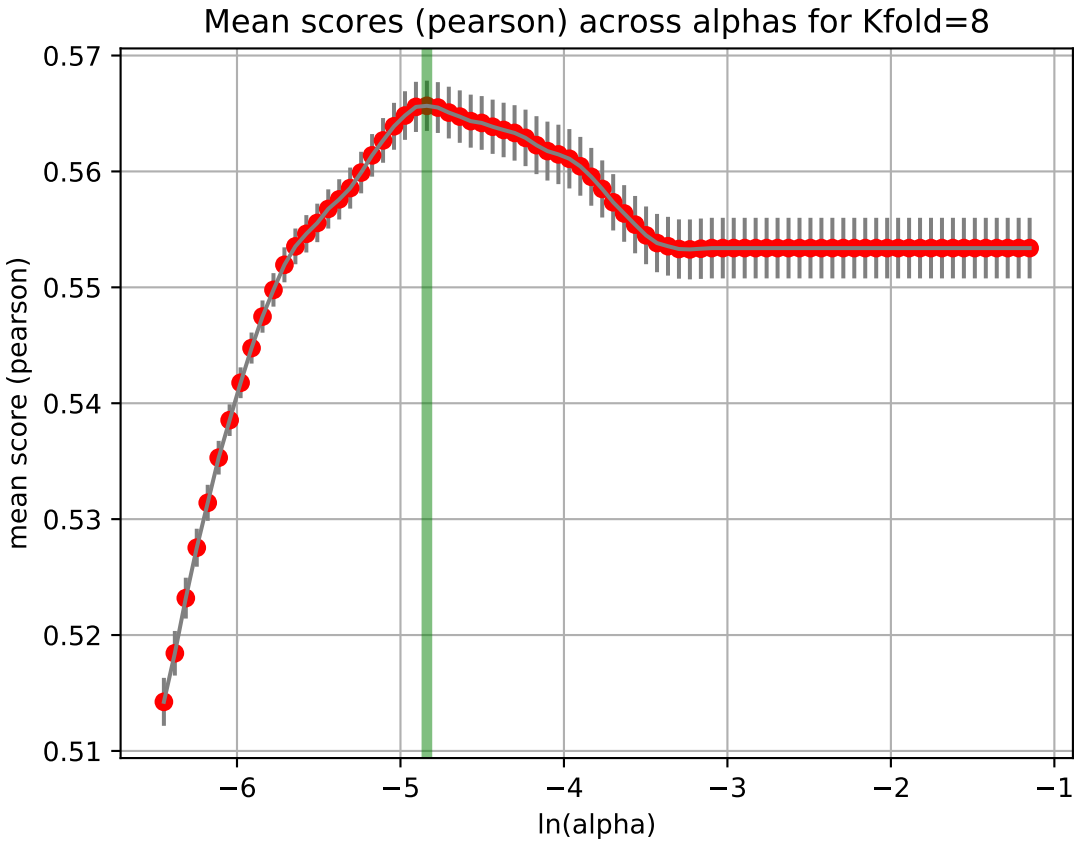
```
best_model = lasso.get_model(alpha=res.alpha)
weights = lasso.plot_weight(drugid, best_model)
```



Validation

The runCV function mentionned above does not plot any figures for optimisation reasons. We implemented another function called **tune_alpha**, which has a visual representation. This is however 20 time slower than **runCV** and is not used in production.:

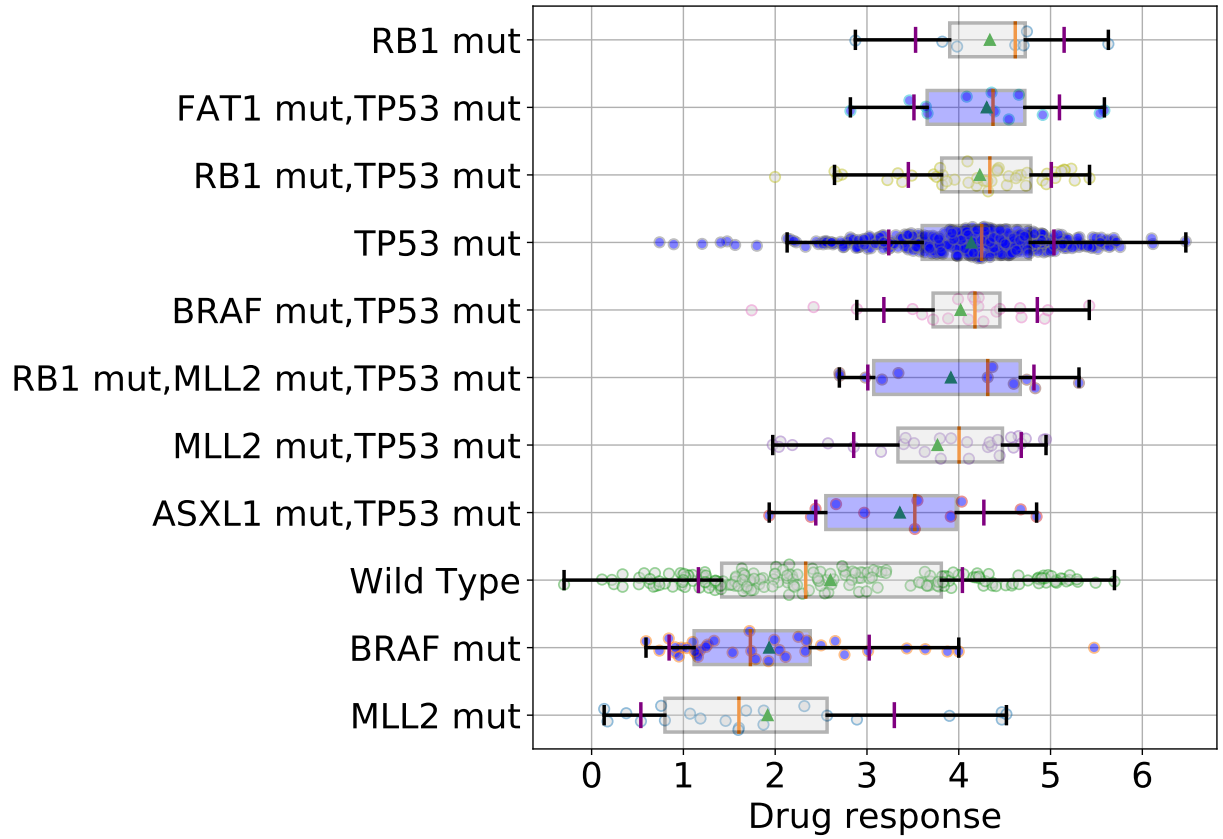
```
lasso.tune_alpha(drugid, alpha_range=[-2.8,-0.5])
```



Another important function is the **check_randomness** method. It runs N times the `gdsctools.regression.GDSCLAsso.runCV()` function, and N times the same analysis shuffling the Y data. This creates a NULL model. The Pearson correlation values between the NULL model and the real data is then compared using a Bayes factor metric (independent of N).

boxplots

```
boxplots = lasso.boxplot(drugid, model=best_model, n=10, bx_vert=False)
```



ADAF analysis

We have now a good picture of what the regression tools can do. If one wants to play with ElasticNet or Ridge methods, just replaced GDSCLasso by GDSCElasticNet or GDSCRidge.

We now want to run the regression on all drugs. This can be done manually of course using a loop over each drug identifiers:

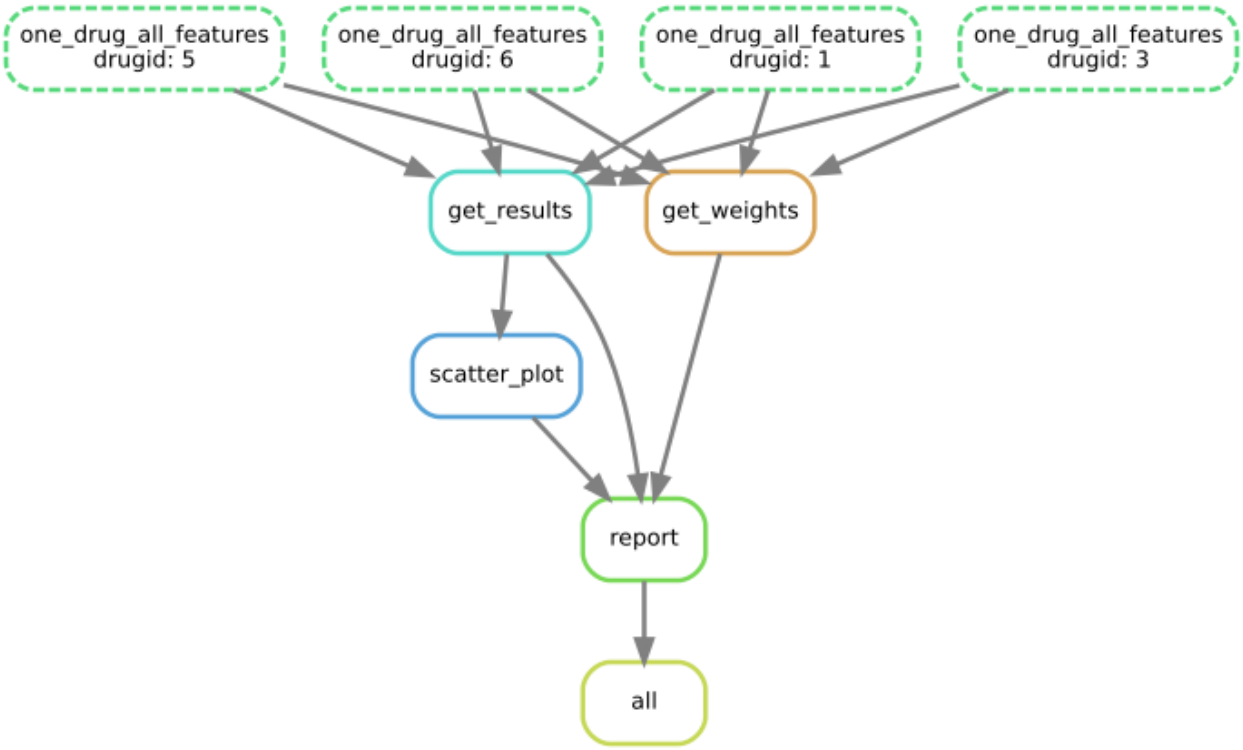
```
for drugID in lasso.drugIds:
    res = lasso.runCV(drugid, kfolds=8)
    best_model = lasso.get_model(alpha=res.alpha)
    weights = lasso.plot_weight(drugid, best_model)
    boxplots = lasso.boxplot(drugid, model=best_model, n=10, bx_vert=False)
    # Save images here
```

The snakemake pipeline

> **Warning:** This is only available for Python 3.5 users since the snakemake utility is only available for Python 3.

We provide a pipeline in a form of a snakemake file. The pipeline is called **regression.rules** and a config file named **regression.yaml** is also provided.

The workflow looks like:



Imagine the case where you have 4 drugs, then results and weights are computed for each drug. This is parallelised on a distributed-computer. Once the computation is performed, a report is created.

The path of those files can be obtained using

```
from gdsctools import gdsctools_data
gdsctools_data("regression.rules", "../pipelines")
gdsctools_data("regression.yaml", "../pipelines")
```

Those two files must be copied in a local directory.

Then, edit the config file that looks like:

```
regression:
    method: lasso
    kfold: 10
    randomness: 50

input:
    ic50:
    genomic_features:
```

so as to set the input IC50 and genomic_features files. Once done, you can run the analysis. Just type:

```
snakemake -s regression.rules -j 4
```

Or a cluster, you may add the following information (for instance on a slurm system):

```
snakemake -s regression.rules -j 40 --cluster "sbatch --qos normal"
```

where -j 40 means uses 40 cores. Wait until it is finished. You should have an index.html file at the end.

Note: There is a standalone that fetches the pipeline and its config file, autofilled with user's argument ready to run.

The standalone is called **gdsctools_regression**. Please see *Standalone application* section

Notebooks

Within **GDSCTools**, we also provide a set of IPython notebooks. Those notebooks are like recipes and tutorials and reproduce some of the examples available in this documentation. We encourage you to have a look at the IPython and IPython notebook online documentation if you want to create your own notebooks.

Note: IPython is a growing project, with increasingly language-agnostic components. As of IPython 4.0, the language-agnostic parts of the project (e.g., notebook) have moved to new projects under the name Jupyter.

The original notebooks that we provide in **GDSCTools** are available in the GitHub repository and can visualised via the nbviewer.

Warning: if you installed GDSCTools with pip, it may not be obvious to find the notebooks on your system, in which case we would recommend to download the notebooks form the source code (link above).

Note, however, that to get the best of the those notebooks (interactivity), you should try them locally:

- install GDSCTools and IPython if not already done.
- If you installed from source, the notebooks are in ./gdsctools/notebooks, otherwise, download the notebooks from the online repository.
- Go in the directory where are located the notebooks and type (in a shell):

```
ipython notebook
```

  This should start the interactive notebook in a browser and you should see the notebooks. Click on one of them and go through the notebooks.

The notebooks cover many different aspects of the **GDSCTools** library including the quickstart discussed in this documentation. We will not give more details about their contents since they may evolve and change we time. We encourage you to create new ones and to share them.

Standalone application

Although we would encourage you to use the Python shell to have as much flexibility as possible, we also provide a standalone applications.

Currently, there are two standalones. The **gdsctools_anova** and **gdsctools_regression**. The first one is a pure Python implementation while the second one is snakemake-based.

gdsctools_anova application

called **gdsctools_anova**. This standalone application should be installed with **GDSCTools** automatically. It focuses on the ANOVA analysis only, and can be used to analysis one set of IC50 and genomic feature at a time.

You can obtain the help by typing:

```
gdsctools_anova --help
```

The main goal is to provide an interface to the python library and consequently, one be able to redo the analysis as shown in the quickstart:

```
* One drug One Feature with figure(s) and HTMLs
* One Drug All Feature with figure and HTMLs
* All Drug All Feature with figures and HTMLs
```

We suppose the input data file is called IC50_10drugs.tsv

ODOF

```
gdsctools_anova --input-ic50 IC50_10drugs.tsv --drug
    Drug_999_IC50 --feature TP53_mut --onweb
```

ODAF

```
gdsctools_anova --input-ic50 IC50_10drugs.tsv --drug
    Drug_999_IC50 --onweb
```

ADAF

```
gdsctools_anova --input-ic50 IC50_10drugs.tsv --onweb
```

Some other settings

Again, you can use the `--help` to get up-to-date information about the available arguments. However, let us give a couple of interesting ones.

If you are interesting in a specific association of drug and feature, it is convenient to get the valid drug names:

```
--print-drug-names
```

or feature names:

```
--print-feature-names
```

By default the analysis is *PANCAN* (includes all tissues) but you can restrict the analysis to a set of tissues (or just one):

```
--tissues breast, cervix
```

To know the names of the tissues, use:

```
--print-tissue-names
```

gdsctools_regression application

Let us consider the case where you have an IC50 file and a genomic file. The first step consists in preparing the working directory:

```
gdsctools_regression -I IC50_v17.csv.gz -F genomic_features_v17.csv.gz
    --method lasso -O lasso_analysis
```

```
cd lasso_analysis
```

On a local computer:

```
snakemake -s regression.rules -j 4
```

On a distributed-computing system using e.g SLURM framework, use:

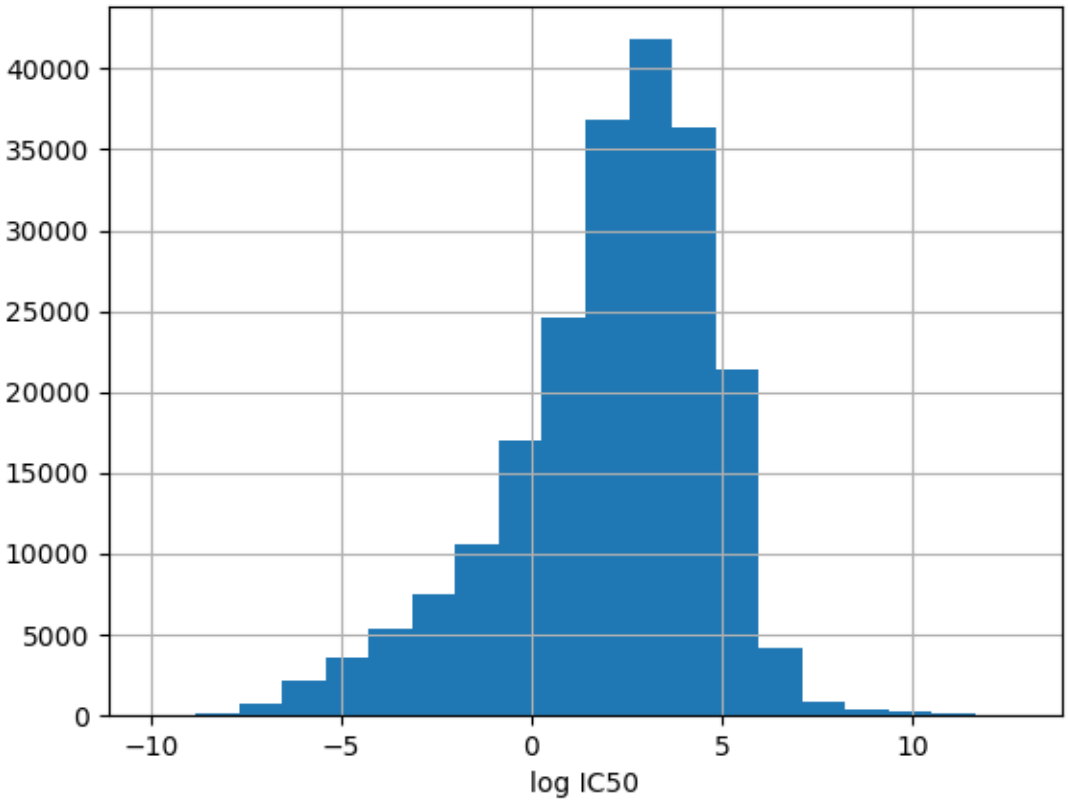
```
srun --qos normal snakemake -s regression.rules -j 40 --cluster "sbatch --qos normal"
```

Gallery / Examples

General-purpose examples for GDSCTools library.

Histogram of the IC50 from v17 data set

histogram of the IC50



```
from gdsctools import IC50, ic50_v17
r = IC50(ic50_v17)
r.hist()
```

Total running time of the script: (0 minutes 0.695 seconds)

Download Python source code: plot_ic50_hist.py

Download Jupyter notebook: plot_ic50_hist.ipynb

Generated by Sphinx-Gallery

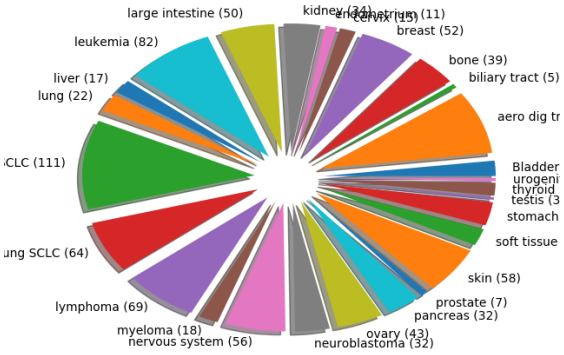
Visualise genomic features

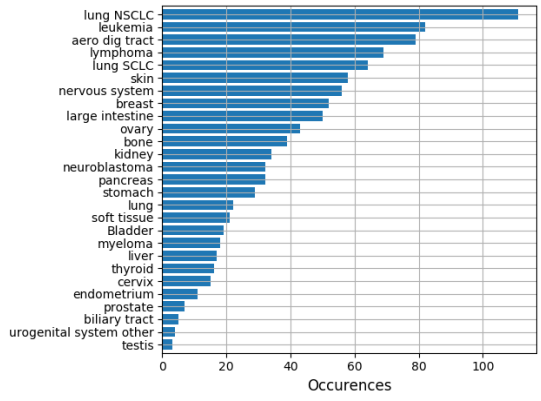
Here, we get a quick overview of the cancer cell types

```
from gdsctools import GenomicFeatures, gf_v17
```

Read the genomic featues (here version 17 of GDSC) and visualise the distribution of the different cancer types as a pie chart or bar plot

```
gf = GenomicFeatures(gf_v17)
gf.plot()
```

- 

- 

Total running time of the script: (0 minutes 0.654 seconds)

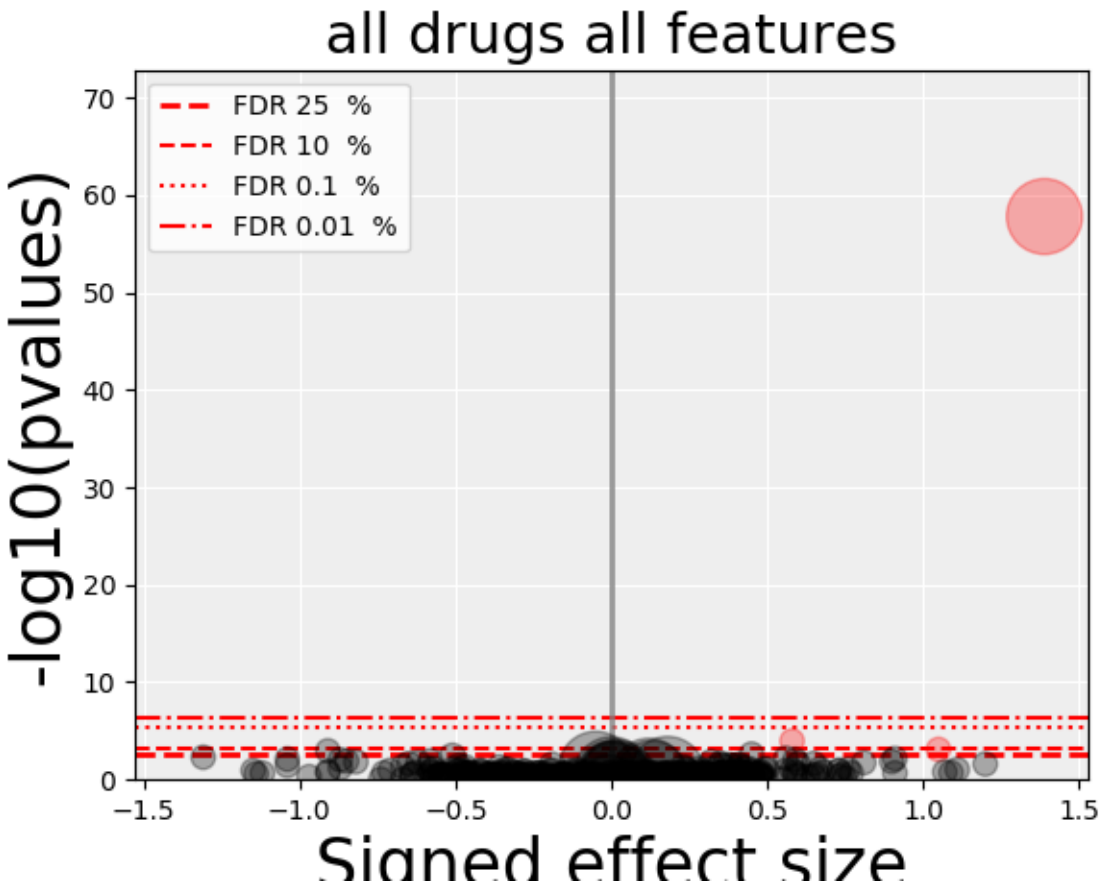
Download Python source code: plot_ic50.py

Download Jupyter notebook: plot_ic50.ipynb

Generated by Sphinx-Gallery

Analyse all associations (drug/feature)

Volcano plot (all associations)



Out:

```
[                  0%                  ] 0 of 11 complete in 0.0 sec
 [---               9%                   ] 1 of 11 complete in 0.4 sec
 [------           18%                   ] 2 of 11 complete in 0.9 sec
 [----------       27%                   ] 3 of 11 complete in 1.4 sec
 [-------------    36%                   ] 4 of 11 complete in 1.8 sec
 [-----------------45%                   ] 5 of 11 complete in 2.3 sec
 [-----------------54%                   ] 6 of 11 complete in 2.7 sec
 [-----------------63%----               ] 7 of 11 complete in 3.2 sec
 [-----------------72%-------            ] 8 of 11 complete in 3.6 sec
 [-----------------81%-----------        ] 9 of 11 complete in 4.1 sec
 [-----------------90%--------------     ] 10 of 11 complete in 4.5 sec
 [-----------------100%-----------------] 11 of 11 complete in 5.0 sec
```

```
from gdsctools import ANOVA, ic50_test
gdsc = ANOVA(ic50_test)
results = gdsc.anova_all()
results.volcano()
```

Total running time of the script: (0 minutes 5.895 seconds)

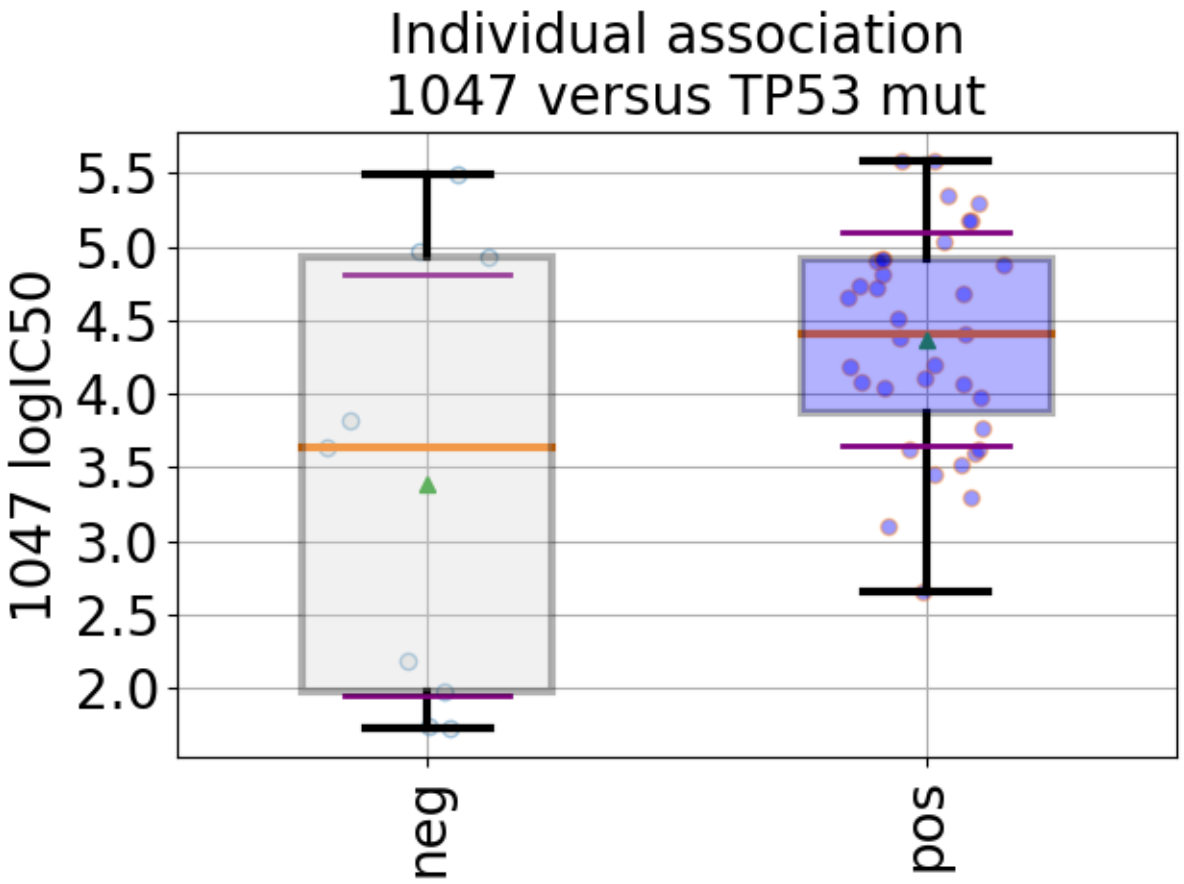
Download Python source code: plot_volcano.py

Download Jupyter notebook: plot_volcano.ipynb

Generated by Sphinx-Gallery

Association between a Drug and a Feature

Boxplot association




```
from gdsctools import ANOVA, ic50_test
gdsc = ANOVA(ic50_test)
gdsc.set_cancer_type('breast')
df = gdsc.anova_one_drug_one_feature(1047, 'TP53_mut', show=True)
```

Total running time of the script: (0 minutes 3.556 seconds)

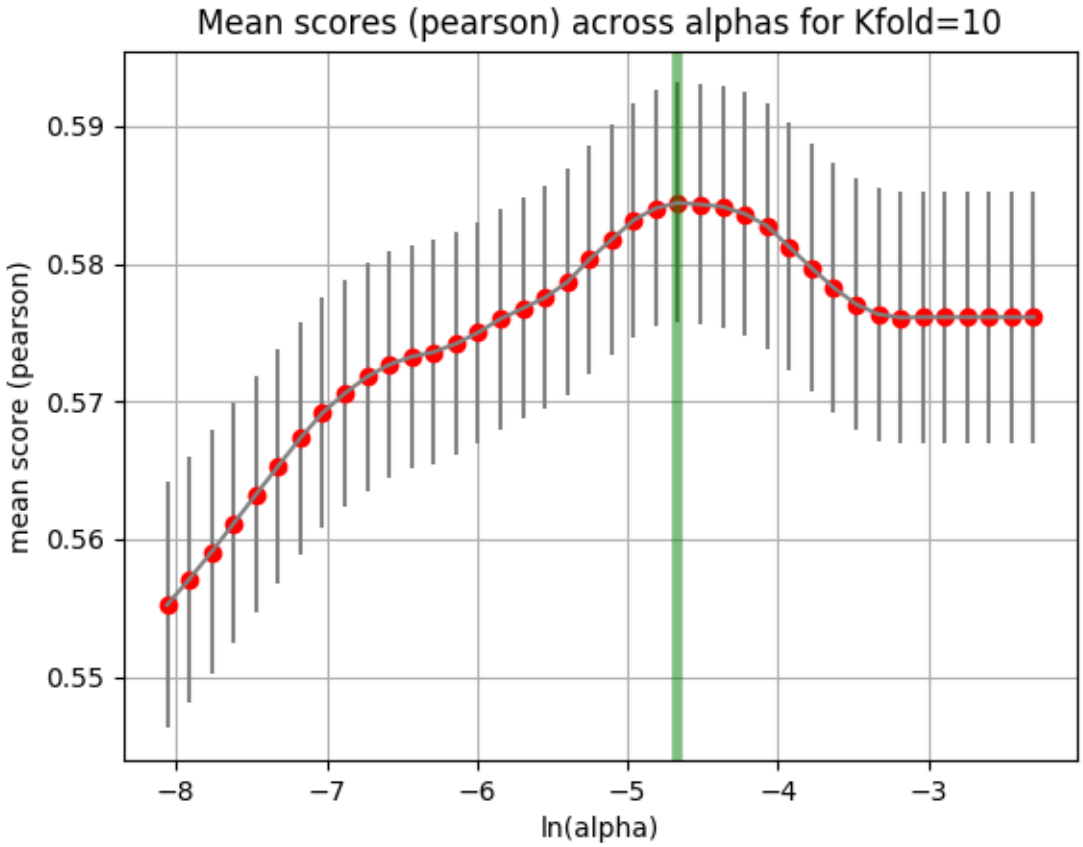
Download Python source code: plot_association.py

Download Jupyter notebook: plot_association.ipynb

Generated by Sphinx-Gallery

Tuning alpha (elastic net case)

Elastic net requires tune an alpha parameter.




```
from gdsctools import *
ic = IC50(gdsctools_data("IC50_v5.csv.gz"))
gf = GenomicFeatures(gdsctools_data("genomic_features_v5.csv.gz"))
en = GDSCElasticNet(ic, gf)
en.tune_alpha(1047, alpha_range=(-3.5,-1), N=40, l1_ratio=0.1)
```

Total running time of the script: (0 minutes 9.466 seconds)

Download Python source code: plot_elastic_tuning.py

Download Jupyter notebook: plot_elastic_tuning.ipynb

Generated by Sphinx-Gallery

Plot weights resulting from an Elastic Net analysis

Note that only the 50 most important weigths are shown

We look at the" effect of the alpha parameter on the weights returned by the elastic net analysis

```
from gdsctools import *
```

First we alpha=0.01

```
gd = GDSCElasticNet(ic50_v17, gf_v17)
drugid = 1047
```

Find best model and corresponding alpha

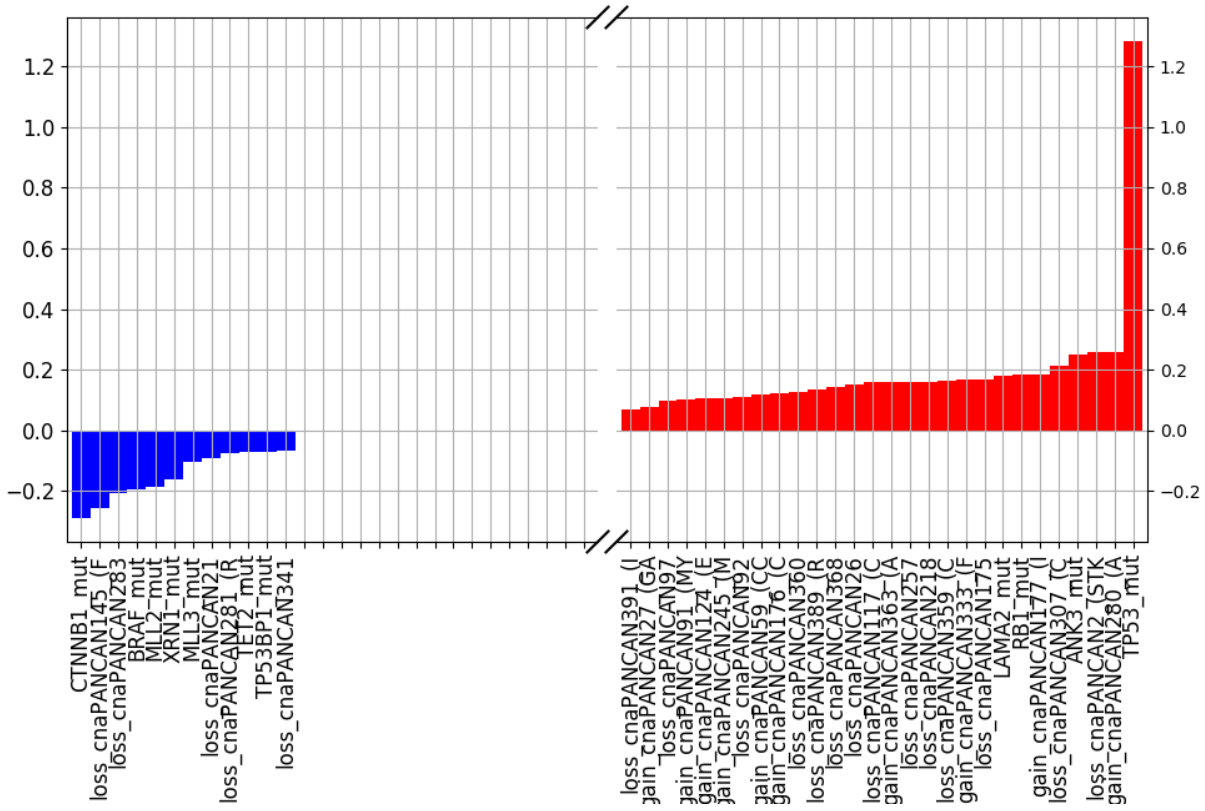
```
res = gd.runCV(drugid, kfolds=10)
best_alpha = res.alpha
```

Out:

```
Best alpha on 10 folds: 0.0176272087611 (-4.04 in log scale); Rp=0.67082409065
```

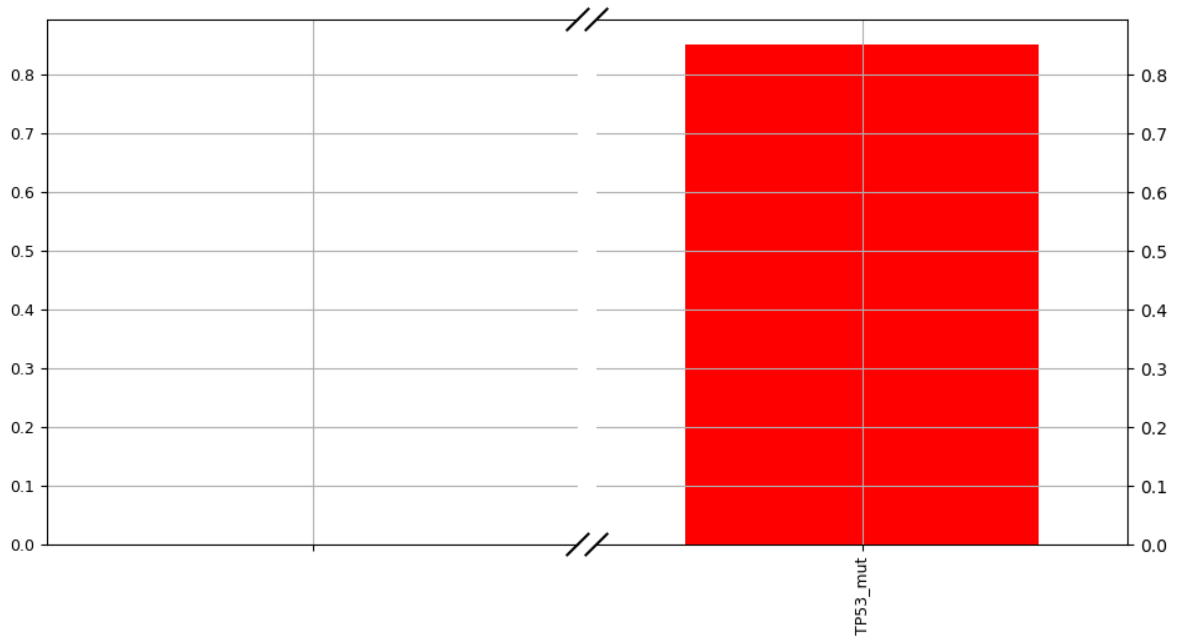
Plot weights of best model

```
best_model = gd.get_model(alpha=best_alpha)
gd.plot_weight(drugid, model=best_model)
```



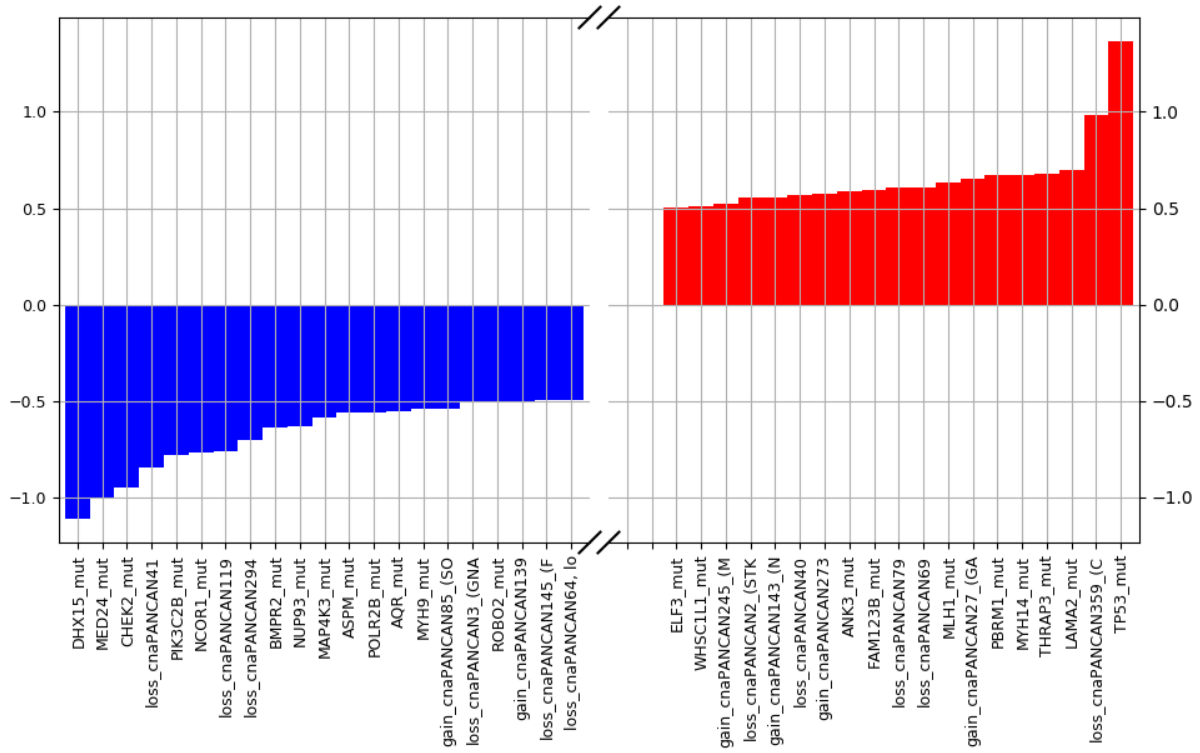
increasing alpha

```
model1 = gd.get_model(alpha=best_alpha*10.)
gd.plot_weight(drugid, model=model1, fontsize=9)
```



decreasing alpha

```
model2 = gd.get_model(alpha=best_alpha/10.)
gd.plot_weight(drugid, model=model2, fontsize=9)
```



Total running time of the script: (0 minutes 21.438 seconds)

`Download Python source code: plot_elastic_weights.py`

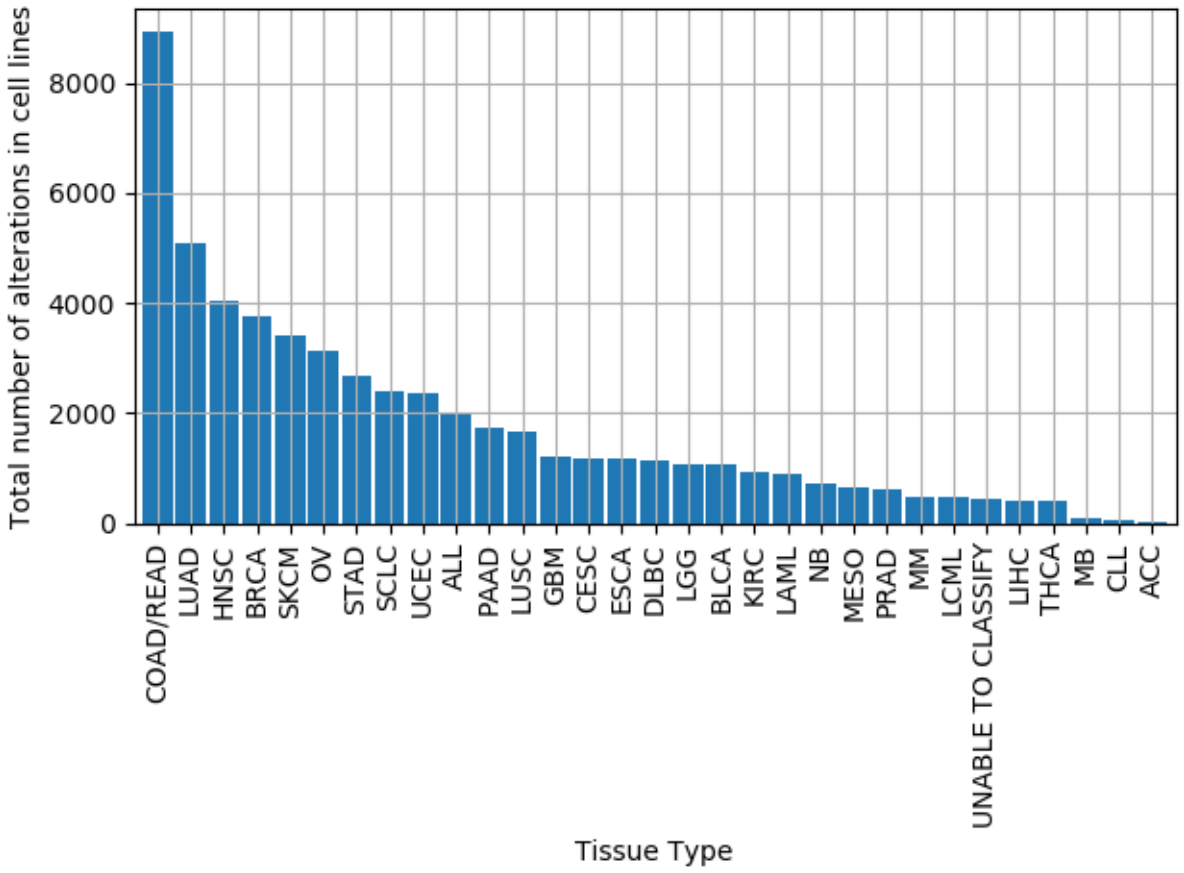
`Download Jupyter notebook: plot_elastic_weights.ipynb`

Generated by Sphinx-Gallery

Use Omnibem to filter annotation data

This example shows how to create a new genomic feature data set from an annotation file.

```
from gdsctools import *
data = gdsctools_data("test_omnibem_genomic_alterations.csv.gz")
bem = OmniBEMBuilder(data)
bem.filter_by_gene_list(gdsctools_data("test_omnibem_genes.txt"))
bem.plot_number_alteration_by_tissue()
```

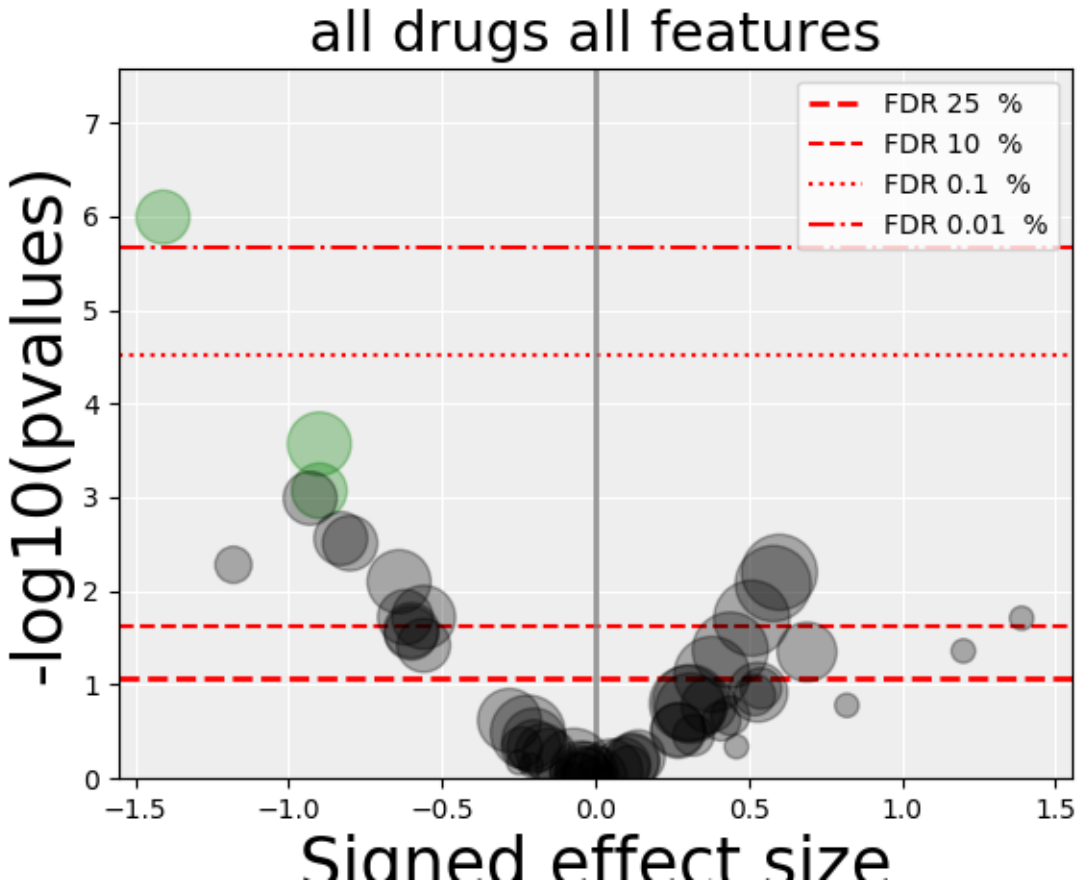


Finally, create a MoBEM dataframe

```
mobem = bem.get_mobem()

# features
bem.filter_by_type_list(["Methylation"])
mobem = bem.get_mobem()

# Then, let us create a dataframe that is compatible with
# GenomicFeature. We just need to make sure the columns are correct
mobem[[x for x in mobem.columns if x!="SAMPLE"]]
gf = GenomicFeatures(mobem[[x for x in mobem.columns if x!="SAMPLE"]])
```

The final volcano plot

```
an = ANOVA(ic50_test, gf, verbose=False)
results = an.anova_all(animate=False)
results.volcano()
```



Total running time of the script: (0 minutes 3.113 seconds)

`Download Python source code: plot_omnibem.py`

`Download Jupyter notebook: plot_omnibem.ipynb`

Generated by Sphinx-Gallery

`Download all examples in Python source code: auto_examples_python.zip`

`Download all examples in Jupyter notebooks: auto_examples_jupyter.zip`

Generated by Sphinx-Gallery

Reproducibility

In this section we show how to reproduce some specific results such as those published in the **GDSC1000 paper** (see v17a section).

The motivation for this section is to show that results can be reproduced using an official release and the proper set of parameters used in the analysis.

Notes to reproduce v17a results (GDSC1000 Cell paper)

Reference Iorio et al 2016

We refer to the data used in the reference above as the **v17a** data sets. It can be found on the CancerRxGene page. We will use this data to reproduce the results published in the same web page.

First, we need to download the data. For example, let us retrieve the BRCA tissue specific data set using **wget** command:

```
wget http://www.cancerrxgene.org/gdsc1000/GDSC1000_WebResources//Data/anova/BRCA/DATA/INPUT/ANOVA_inp
```

Warning: in v17a data, the input file is named **ANOVA_input.txt** It is a tabulated format and contains all IC50s and genomic features altogether. This is not recommended and we now expect the IC50s and genomic_features to be in two different files. However, for back compatibility, GDSCTools is able to extract the IC50 alone, or the genomic features alone from that kind of format. See hereafter and *Data Format and Readers* section for details.

Once downloaded, create an ANOVA instance as follows:

```
from gdsctools import ANOVA
an = ANOVA("ANOVA_input.txt", "ANOVA_input.txt")
```

Here, we provide the filename twice; this is not a mistake, please see the warning box above. Internally, the first argument extracts the IC50s and the second one extracts the genomic features.

In v17a, there are two parameters that are not the current default values that must be set to reproduce the results:

```
an.setttings.pvalue_correction_method = "qvalue"
an.settings.equal_var_ttest = False
```

The later parameter is used in some plots for annotation but is not essential. The first parameter is important since it sets the method used for multiple testing correction. It will also define the number of associations that are significant. The FDR threshold that defines the significant associations was set to 25.

In the case of a tissue specific analysis (here BRCA), the name of the tissue is unknown (not specified anywhere inside the file) and so one should provide the information. This is not important for the analysis itself but is used for instance to name the output of the directory where HTML reports are stored:

```
report = ANOVAReport(an)
report.settings.directory = "BLCA"
report.create_html_pages()
```

Note that the multiple corrected values reported by GDSCTools and found on the website are different by a systematic bais of 2-3 %. This is known and due to a different implementation of the qvalue method (smoothing function). However, the number of tests and the ANOVA_FEATURE_pval column (pvalues of the FEATURE factor) should agree perfectly. Finally, note that because the value of the FDR (corrected values) differ, the number of significant associations below that threshold may also slightly differ. However, results are consitent for FDR not close to the threshold.

As for the PANCAN case, results are currently different between GDSCTools and what is posted within the link at the top of the page because there is currently a mismatch between the ANOVA_input file provided and the results provided (one has the MEDIA factor while there other has not)

For information, on an intel i7 core, the analysis of the PANCAN data set (265 drugs and about 1000 features) takes about 20 minutes to finish. Tissue specific data files takes a few minutes or less in general.

Notes about v18 onwards

In V18 onwards (May 2016), the IC50 may have duplicated columns for a given drug. So some drugs are clustered together. The algorithm was implemented in GDSCTools in IC50Cluster class. It should be used as follow.

IC50 must be read with the IC50Cluster class as follows:

```
from gdsctools import *
ic50 = IC50Cluster("v18_data")
```

This will ensure also that the drug identifiers are unique. Indeed, in v18 data sets, columns for a given DRUG ID may be duplicated (for different drug concentration).

Then, as usual:

```
an = ANOVA(ic50, "GF.csv", "DRUG_DECODE.csv")
```

All default parameters were used except for the FDR threshold:

```
report.settings.FDR_threshold = 35
```

References

Contents

ANOVA related

The ANOVA analysis

Code related to the ANOVA analysis to find associations between drug IC50s and genomic features

class ANOVA(*ic50*, *genomic_features=None*, *drug_decode=None*, *verbose=True*, *set_media_factor=False*)

ANOVA analysis of the *IC50* vs Feature matrices

This class is the core of the analysis. It can be used to compute

1.One association between a drug and a feature

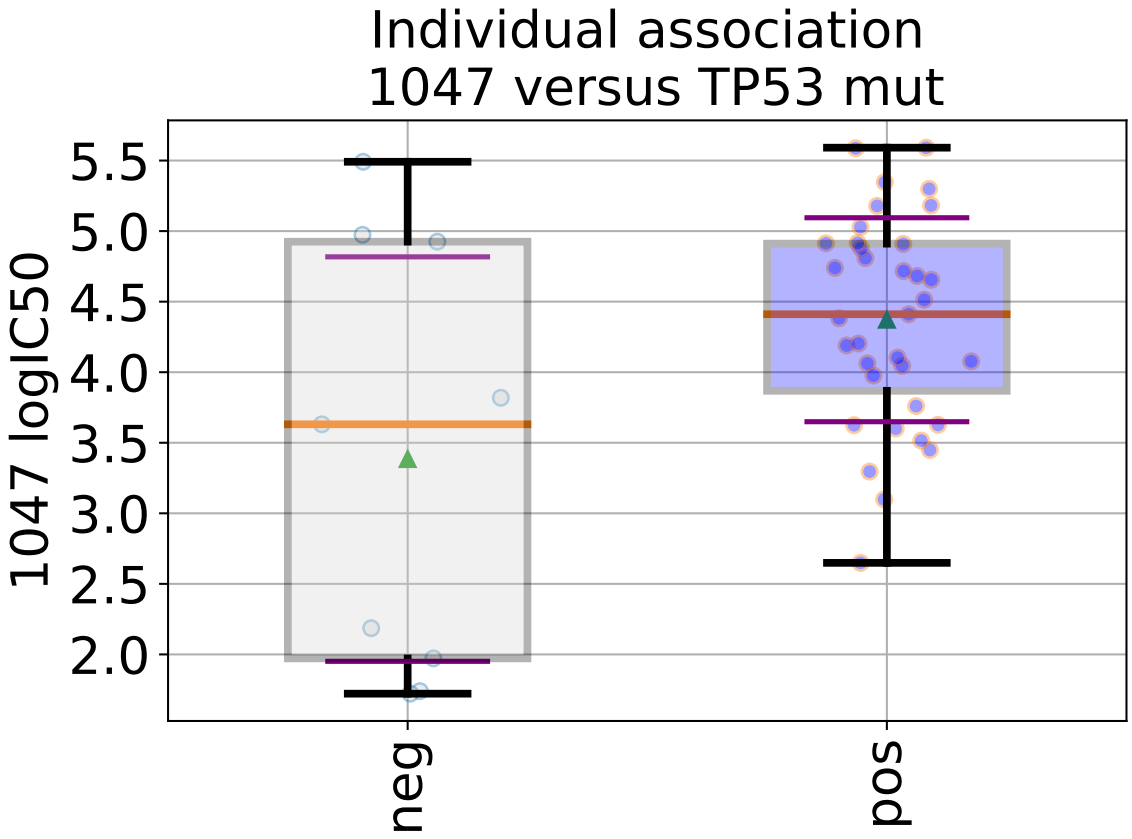
2.The association**S** between a drug and a set of features

3.All assocations between a set of drugs and a set of features.

For instance here below, we read an IC50 matrix and compute the association for a given drug with a specific feature.

Note that genomic features are not provided as input but a default file is provided with this package that contains 49 genomic features for 988 cell lines. If your IC50 contains unknown cell lines, you can provide your own file or use the *gdsctools.datasets* v17.

```
from gdsctools import IC50, ANOVA, ic50_test
ic = IC50(ic50_test)
an = ANOVA(ic)
# This is to select a specific tissue
an.set_cancer_type('breast')
df = an.anova_one_drug_one_feature(1047, 'TP53_mut', show=True)
```



Details about the anova analysis In the example above, we perform a regression/anova test based on OLS regression. This is done for one feature one drug across all cell lines (tissue) in the method *anova_one_drug()*. The regression takes into account the following factors: tissue, MSI, MEDIA and features. If there is only one tissue, this factor is dropped. If the number of MSI values is less than a pre-defined parameter (see *ANOVASettings*), it is dropped. MEDIA, MSI columns are optional. The other methods *anova_one_drug()* and *anova_all()* are wrappers around *anova_one_drug_one_feature()* to loop over all drugs, and loop over all drugs and all features, respectively.

Please see the online documentation (ANOVA sections) for more help on gdsctools.readthedocs.io

Specific notes about the parameters. Default settings are used except for those releases:

V17

```
gdsc.volcano_FDR_interpolation = False
gdsc.settings.pvalue_correction_method = 'qvalue'
```

V18

```
gdsc.settings.FDR_threshold = 35
```

Constructor

Parameters

- **IC50** (*DataFrame*) – a dataframe with the IC50. Rows should be the COSMIC identifiers and columns should be the Drug names (or identifiers)
- **features** – another dataframe with rows as in the IC50 matrix and columns as features. The first 3 columns must be named specifically to hold tissues, MSI (see format).
- **drug_decode** – a 3 column CSV file with drug's name and targets see `readers` for more information.
- **verbose** – verbosity in "WARNING", "ERROR", "DEBUG", "INFO"

Please see *readers* module for details about the input formats.

The `settings` attribute contains specific settings related to the ANOVA analysis or visualisation.

add_pvalues_correction(*df*, *colname='ANOVA_FEATURE_pval'*)
Compute and add corrected pvalues in column ANOVA_FEATURE_FDR

Parameters

- **df** – a dataframe with a column named after colname (defaults to ANOVA_FEATURE_pval). The output of *anova_all()* contains such a dataframe.
- **colname** (*str*) – name of the column that contains the pvalues to be corrected.

The default multiple testing correction (FDR correction) is stored in `settings.pvalue_correction_method` and can be changed to other methods (e.g., **qvalue**).

The results in stored in a column named **ANOVA_FEATURE_FDR** inside the input dataframe **df**.

Values are in the range 0 to 1.

See also:

anova_all(), *MultipleTesting*

anova_all(*animate=True*, *drugs=None*, *multicore=None*)
Run all ANOVA tests for all drugs and all features.

Parameters

- **drugs** – you may select a subset of drugs
- **animate** – shows the progress bar

Returns an *ANOVAResults* instance with the dataframe stored in an attribute called **df**

Calls *anova_one_drug()* for each drug and concatenate all results together. Note that once all data are gathered, *add_pvalues_correction()* is called to fill a new column with FDR corrections.

An extra column named "ASSOC_ID" is also added with a unique identifer sorted by ascending FDR.

Note: A thorough comparison with version v17 gives the same FDR results (difference ~1e-6); Note however that the qvalue results differ by about 0.3% due to different smoothing in R and Python.

---

anova_one_drug(*drug_id*, *animate=True*, *output='object'*)

Computes ANOVA for a given drug across all features

Parameters

- **drug_id** (*str*) – a valid drug identifier.
- **animate** – shows the progress bar

Returns a dataframe

Calls *anova_one_drug_one_feature()* for each feature.

anova_one_drug_one_feature(*drug_id*, *feature_name*, *show=False*, *production=False*, *directory='.'*)

Compute ABOVA one drug and one feature level

Parameters

- **drug_id** – a valid drug identifier
- **feature_name** – a valid feature name
- **show** (*bool*) – show boxplots with the different factor used
- **directory** (*str*) – where to save the figure.
- **production** (*bool*) – if False, returns a dataframe otherwise a dictionary. This is to speed up analysis when scanning the drug across all features.

Note: **for developer** this is the core of the analysis and should be kept as fast as possible. 95% of the time is spent here.

Note: **for developer** Data used in this function comes from _get_one_drug_one_feature_data method, which should also be kept as fast as possible.

---

anova_one_drug_one_feature_custom(*drug_id*, *feature_name*, *formula*, *odof=None*)

Same as *anova_one_drug_one_feature()* but allows any formula

Returns full ANOVA table but also populate interal attribute anova_pvalues that is a dictionary with pvalues for feature, media, msi and tissue

Formula must be set in the settings attribute as settings.regression_formula:

```
an = ANOVA(...)
an.settings.formula = "Y ~  C(tissue) + feature"
```

Note: This function is convenient but 3 times slower than *anova_one_drug_one_feature()*. So if your formula are one of:

```
"Y ~  C(tissue) + C(media) + C(msi) + feature"
"Y ~  C(tissue) + C(msi) + feature"
"Y ~  C(msi) + feature"
"Y ~  feature"
```

you should use *anova_one_drug_one_feature()* instead.

By default, in categories, the first treatment (e.g tissue) is used a reference and is not shown in the results. You may set the reference as follows:

```
"Y ~ C(tissue, Treatment(reference='breast'))"
```

ANOVA pvalues returned are of type I

New in version 0.15.0.

reset_buffer()
Reset the buffer used to store the results

When calling *anova_all()*, the method *anova_one_drug()* is called for each drug and the results saved in the individual_anova attribute. If called again, the results are simply returned from the buffer and not recomputed.

If you change a settings that affects the analysis and therefore the results, then you should call this method.

The ANOVA results

ANOVAResults data structure to store the output of the ANOVA analysis

class ANOVAResults(*filename=None*, *settings=None*)
Class to handle results of the ANOVA analysis

The ANOVA class and in particular its method *anova_all()* returns the results of the ANOVA analysis for each drug and genomic feature. The results are stored in a data structure defined in this class, which is just a dataframe stored in *df* attribute with the following header:

Column name	Description
ASSOC_ID	Alphanumeric identifier of the interaction
FEATURE	The CFE involved in the interaction, it can be a mutated cancer driver gene (CG) [suffix _mut], an abberrantly fused protein [suffix fusion], a copy number altered chromosomal region (RACS) [prefix gain for amplifications or loss for deletions];
DRUG_ID	Numerical id of the drug involved in the interaction;
DRUG_TARGET	Putative target of the drug involved in the interaction;
N_FEATURE_pos	Number of cell lines harbouring the CFE indicated in column E and that have been screened with the drug indicated in columns F and G, therefore have been included in the test;
N_FEATURE_neg	Number of cell lines not harbouring the CFE indicated in column E and that have been screened with the drug indicated in columns F and G, therefore have been included in the test;
FEATURE_pos_logIC50_MEAN	Average log IC50 of the population of cell lines accounted in colum i;
FEATURE_neg_logIC50_MEAN	Average log IC50 of the population of cell lines accounted in colum j;
FEATURE_delta_MEAN_IC50	Difference between the two average natural log IC50 values in the previous two columns (j - i). A negative value indicates an interaction for sensitivity, whereas a positive value indicates an interaction for resistance;
FEATURE_pos_IC50_sd	Log IC50 Standard deviation for the population of cell lines accounted in column i;
FEATURE_neg_IC50_sd	Log IC50 Standard deviation for the population of cell lines accounted in column j;
FEATURE_IC50_effect_size	Cohen's d, quantifying the effect size of the interaction. A value >=0.5 indicates a moderate effect size. A value >=1 indicates a large effect size (i.e. difference in mean log IC50 values greater than their pooled standard deviations). A value >= 2 indicates a very large effect size (i.e. difference in mean log IC50 is at least two times their pooled standard deviation);
FEATURE_pos_Glass_delta	Glass delta, quantifying the effect size of the interaction as the ratio between the difference of the mean log IC50 values and the standard deviation of the log IC50 values of the population of cell lines accounted in column i;
FEATURE_neg_Glass_delta	Glass delta Same as above for the negative set.
ANOVA_FEATURE_pval	ANOVA test p-value quantyfing the interaction significance;
ANOVA_TISSUE_pval	ANOVA test p-value quantifying the significance of the interaction between drug response and the tissue of origin of the cell lines; for the cancer-specific interactions this value is NA;
ANOVA_MEDIA_pval	ANOVA test p-value quantifying the significance of the interaction between drug response and the screening medium of the cell lines; for the cancer-specific interactions this value is NA;
ANOVA_MSI_pval	ANOVA test p-value quantifying the significance of the interaction between drug response and the micro-satellite instability status of the cell lines; for the cancer type with no micro-satellite instable cell line samples this value is NA;
ANOVA_FEATURE_FDR	False discovery rate obtained by correcting the p-values in column u, on an individual analysis basis, for multiple hypothesis testing with the q-value correction method (Storey & TIbshirani, 2003)

Note that those column names are renamed internally (and if the data is saved in a new file):

assoc_id	ASSOC_ID
Drug id	DRUG_ID
Owned_by	OWNED_BY
FEATUREpos_IC50_sd	FEATURE_pos_IC50_sd
FEATUREneg_IC50_sd	FEATURE_neg_IC50_sd
FEATUREpos_Glass_delta	FEATURE_pos_Glass_delta
FEATUREneg_Glass_delta	FEATURE_neg_Glass_delta
FEATUREpos_logIC50_MEAN	FEATURE_pos_logIC50_MEAN
FEATUREneg_logIC50_MEAN	FEATURE_neg_logIC50_MEAN
Drug Target	DRUG_TARGET
FEATURE_deltaMEAN_IC50	FEATURE_delta_MEAN_IC50
FEATURE_ANOVA_pval	ANOVA_FEATURE_pval
ANOVA FEATURE FDR %	ANOVA_FEATURE_FDR
MSI_ANOVA_pval	ANOVA_MSI_pval
Tissue_ANOVA_pval	ANOVA_TISSUE_pval
MEDIA_ANOVA_pval	ANOVA_MEDIA_pval
TISSUE_ANOVA_pval	ANOVA_TISSUE_pval
Drug name	DRUG_NAME

Constructor

Parameters **filename** (*str*) – Another ANOVAResults instance or a compatible CSV file with the correct header. The filename may also be set to None (default) and populated later.

astype(*df*)

barplot_effect_size()
Dev not for production

copy()
Returns a copy

df
dataframe with all results

drugIds
Returns the list of drug identifiers

get_html_table(*collapse_table=False*, *clip_threshold=2*, *index=False*, *header=True*, *escape=False*)
Return an HTML table for the reports

mapping = None
dictionary with the relevant column names and their expected types

read_csv(*filename*)
Read an ANOVAResults file from a CSV file

Todo

check validity of the header

---

to_csv(*filename*)
Save the ANOVAResults dataframe into a CSV file

volcano(*settings=None*)
Calls `VolcanoANOVA` on the results

x-value is sign(FEATURE_delta_MEAN_IC50) times FEATURE_IC50_effect_size y-value is the FDR correction

See the online documentation for details on gdsctools.readthedocs.io.

The ANOVA report

Code related to the ANOVA analysis to find associations between drug IC50s and genomic features

class ANOVAReport(*gdsc*, *results=None*, *sep='t'*, *drug_decode=None*, *verbose=True*)
Class used to interpret the results and create final HTML report

Results is a data structure returned by `ANOVA.anova_all()`.

```
from gdsctools import *

# Perform the analysis itself to get a set of results (dataframe)
an = ANOVA(ic50_test)
results = an.anova_all()

# now, we can create the report.
r = ANOVAReport(gdsc=an, results=results)

# we can tune some settings
r.settings.pvalue_threshold = 0.001
r.settings.FDR_threshold = 28
r.settings.directory = 'testing'
r.create_html_pages()
```

Significant association

An association is significant if

- The field *ANOVA_FEATURE_FDR* must be < FDR_threshold
- The field *ANOVA_FEATURE_pval* must be < pvalue_threshold

It is then labelled **sensible** if *FEATURE_delta_MEAN_IC50* is below 0, otherwise it is **resistant**.

Constructor

Parameters

- **gdsc** – the instance with which you created the results to report
- **results** – the results returned by `ANOVA.anova_all()`. If not provided, the ANOVA is run on the fly.

create_html_associations()
Create an HTML page for each significant association

The name of the output HTML file is **<association id>.html** where association id is stored in `df`.

create_html_drugs()
Create an HTML page for each drug

create_html_features()
Create an HTML page for each significant feature

create_html_main(*onweb=False*)
Create HTML main document (summary)

create_html_manova(*onweb=True*)
Create summary table with all significant hits

Parameters **onweb** – open browser with the created HTML page.

create_html_pages(*onweb=True*)
Create all HTML pages

diagnostics()
Return summary of the analysis (dataframe)

drug_summary(*top=50*, *fontsize=15*, *filename=None*)
Return dataframe with significant drugs and plot figure

Parameters

- **fontsize** –
- **top** – max number of significant associations to show
- **filename** – if provided, save the file in the directory

feature_summary(*filename=None*, *top=50*, *fontsize=15*)
Return dataframe with significant features and plot figure

Parameters

- **fontsize** –
- **top** – max number of significant associations to show
- **filename** – if provided, save the file in the directory

get_drug_summary_data()
Return dataframe with drug summary

get_feature_summary_data()
Return dataframe with feature summary

get_significant_hits(*show=True*)
Return a summary of significant hits

Parameters **show** – show a plot with the distribution of significant hits

---

Todo

to finalise

---

get_significant_set()
Return significant hits (resistant and sensible)

n_celllines
return number of cell lines

n_drugs
return number of drugs

n_tests

onweb()

Statistical Tools

Code related to the ANOVA analysis to find associations between drug IC50s and genomic features

class MultipleTesting(*method=None*)

This class eases the computation of multiple testing corrections

The method implemented so far are based on statsmodels or a local implementation of **qvalue** method.

method name	Description
bonferroni	one-step correction
sidak	one-step correction
holm-sidak	step down method using Sidak adjustments
holm	step down method using Bonferroni adjustments
simes-hochberg	step up method (independent)
hommel	close method based on Simes tests (non negative)
fdr_bh	FDR Benjamini-Hochberg (non-negative)
fdr_by	FDR Benjamini-Yekutieli (negative)
fdr_tsbky	FDR 2-stage Benjamini-Krieger-Yekutieli non negative
frd_tsbh	FDR 2-stage Benjamini-Hochberg' non-negative
fdr	same as fdr_bh
qvalue	see *QValue* class

See also:

gdsctools.qvalue.

See also:

http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2907892/

Constructor

Parameters **method** – default to **fdr** that is the FDR Benjamini-Hochberg correction.

get_corrected_pvalues(*pvalues*, *method=None*)

Return corrected pvalues

Parameters

- **pvalues** (*list*) – list or array of pvalues to correct.
- **method** – use the one defined in the constructor by default but can be overwritten here

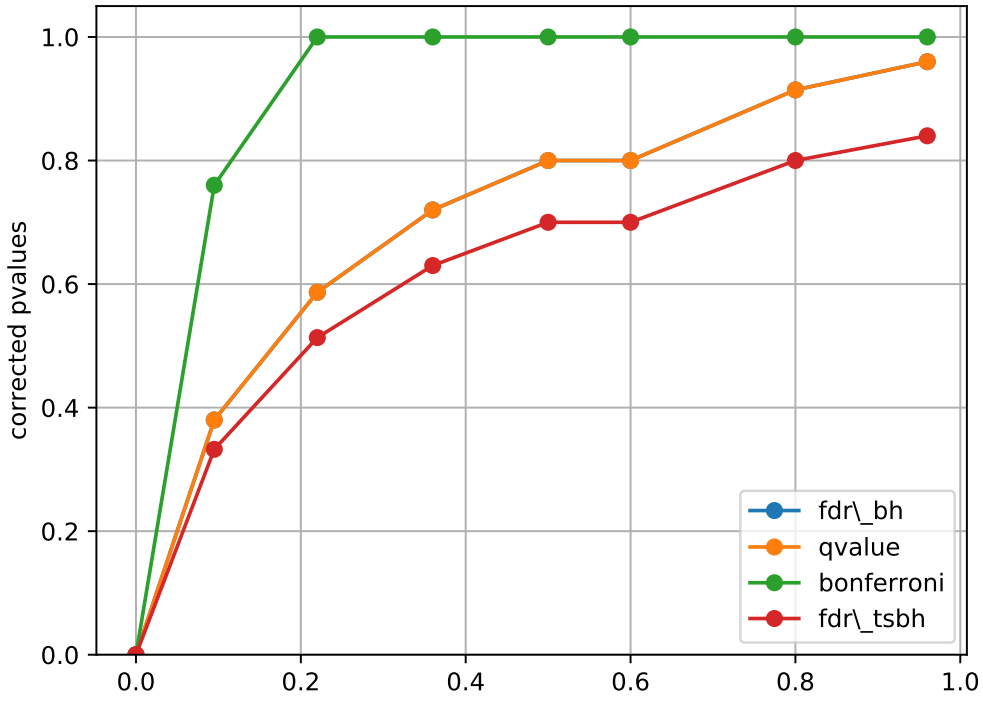
method

get/set method

plot_comparison(*pvalues*, *methods=None*)

Simple plot to compare the pvalues correction methods

```
from gdsctools.stats import MultipleTesting
mt = MultipleTesting()
pvalues = [1e-10, 9.5e-2, 2.2e-1, 3.6e-1, 5e-1, 6e-1,8e-1,9.6e-1]
mt.plot_comparison(pvalues,
    methods=['fdr_bh', 'qvalue', 'bonferroni', 'fdr_tsbh'])
```



Note: in that example, the qvalue and FDR are identical, but this is not true in general.

valid_methods = None
set of valid methods

cohens (x, y)
Effect size metric through Cohen's *d* metric

Parameters

- **x** – first vector
- **y** – second vector

Returns absolute effect size value

The Cohen's effect size *d* is defined as the difference between two means divided by a standard deviation of the data.

$$d = \frac{\bar{x}_1 - \bar{x}_2}{s}$$

For two independent samples, the *pooled standard deviation* is used instead, which is defined as:

$$s = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}}$$

A Cohen's *d* is frequently used in estimating sample sizes for statistical testing: a lower *d* value indicates the necessity of larger sample sizes, and vice versa.

Note: we return the absolute value

References https://en.wikipedia.org/wiki/Effect_size

signed_effects(*df*)

Implementation of qvalue estimate

Author: Thomas Cokelaer

class QValue(*pv*, *lambdas=None*, *pi0=None*, *df=3*, *method='smoother'*, *smooth_log_pi0=False*, *verbose=True*)

Compute Q-value for a given set of P-values

Constructor

The q-value of a test measures the proportion of false positives incurred (called the false discovery rate or FDR) when that particular test is called significant.

Parameters

- **pv** – A vector of p-values (only necessary input)
- **lambdas** – The value of the tuning parameter to estimate pi_0. Must be in [0,1). Can be a single value or a range of values. If none, the default is a range from 0 to 0.9 with a step size of 0.05 (inluding 0 and 0.9)
- **method** – Either "smoother" or "bootstrap"; the method for automatically choosing tuning parameter in the estimation of pi_0, the proportion of true null hypotheses. Only smoother implemented for now.
- **df** – Number of degrees-of-freedom to use when estimating pi_0 with a smoother (default to 3 i.e., cubic interpolation.)
- **pi0** (*float*) – if None, it's estimated as suggested in Storey and Tibshirani, 2003. May be provided, which is convenient for testing.
- **smooth_log_pi0** – If True and 'pi0_method' = "smoother", pi_0 will be estimated by applying a smoother to a scatterplot of log pi_0 rather than just pi_0

Note: Estimation of pi0 differs slightly from the one given in R (about 0.3%) due to smoothing.spline function differences between R and SciPy.

If no options are selected, then the method used to estimate pi_0 is the smoother method described in Storey and Tibshirani (2003). The bootstrap method is described in Storey, Taylor & Siegmund (2004) but not implemented yet.

See also:

gdsctools.stats.MultipleTesting

estimate_pi0(*pi0*)

Estimate pi0 based on the pvalues

qvalue()

Return the qvalues using pvalues stored in `pv` attribute

Readers

IC50, Genomic Features, Drug Decode

IO functionalities

Provides readers to read the following formats

- Matrix of IC50 data set *IC50*
- Matrix of Genomic features with *GenomicFeatures*
- Drug Decoder table with *DrugDecode*

class `IC50` (*filename*, *v18=False*)

Reader of IC50 data set

This input matrix must be a comman-separated value (CSV) or tab-separated value file (TSV).

The matrix must have a header and at least 2 columns. If the number of rows is not sufficient, analysis may not be possible.

The header must have a column called "COSMIC_ID" or "COSMIC ID". This column will be used as indices (row names). All other columns will be considered as input data.

The column "COSMIC_ID" contains the cosmic identifiers (cell line). The other columns should be filled with the IC50s corresponding to a pair of COSMIC identifiers and Drug. Nothing prevents you to fill the file with data that have other meaning (e.g. AUC).

If at least one column starts with `Drug_`, all other columns will be ignored. This was implemented for back compatibility.

The order of the columns is not important.

Here is a simple example of a valid TSV file:

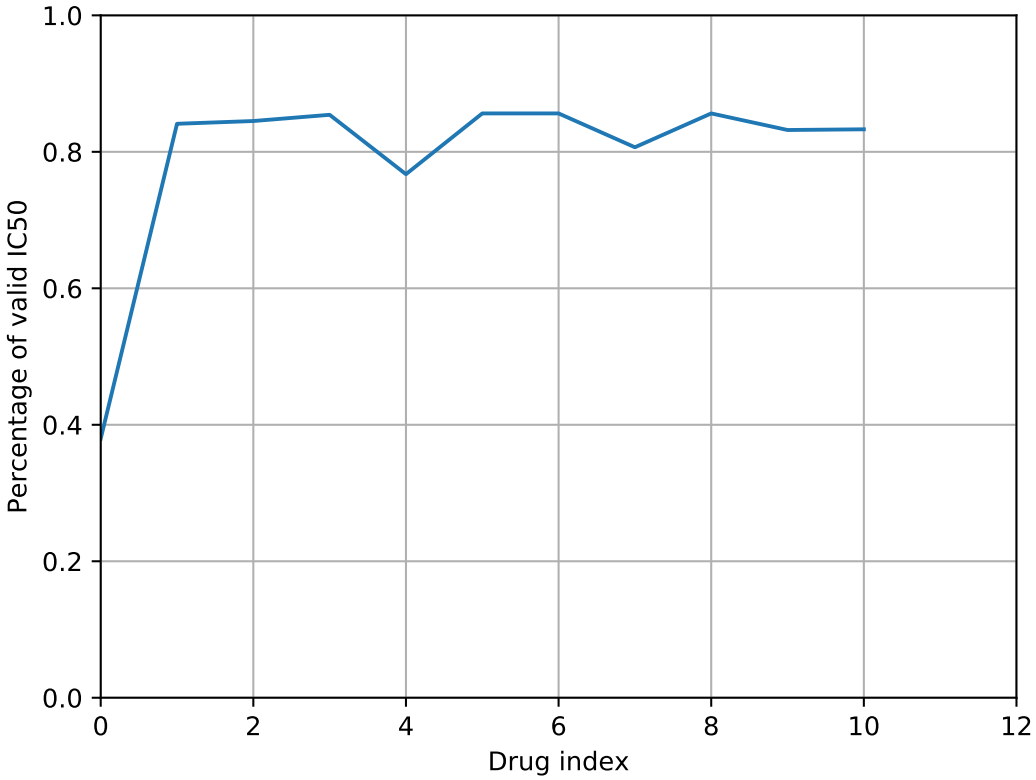
```
COSMIC_ID   Drug_1_IC50 Drug_20_IC50
111111      0.5         0.8
222222      1           2
```

A test file is provided in the gdsctools package:

```
from gdsctools import ic50_test
```

You can read it using this class and plot information as follows:

```
from gdsctools import IC50, ic50_test
r = IC50(ic50_test)
r.plot_ic50_count()
```



You can get basic information using the print function:

```
>>> from gdsctools import IC50, ic50_test
>>> r = IC50(ic50_test)
>>> print(r)
Number of drugs: 11
Number of cell lines: 988
Percentage of NA 0.206569746043
```

You can get the drug identifiers as follows:

```
r.drugIds
```

and set the drugs, which means other will be removed:

```
r.drugsIds = [1, 1000]
```

Changed in version 0.9.10: The column **COSMIC ID** should now be **COSMIC_ID**. Previous name is deprecated but still accepted.

Constructor

Parameters **filename** – input filename of IC50s. May also be an instance of *IC50* or a valid dataframe. The data is stored as a dataframe in the attribute called `df`. Input file may be gzipped

copy()

cosmic_name = 'COSMIC_ID'

drop_drugs (*drugs*)
drop a drug or a list of drugs

drugIds
list the drug identifier name or select sub set

drug_name_to_int (*name*)

get_ic50 ()
Return all ic50 as a list

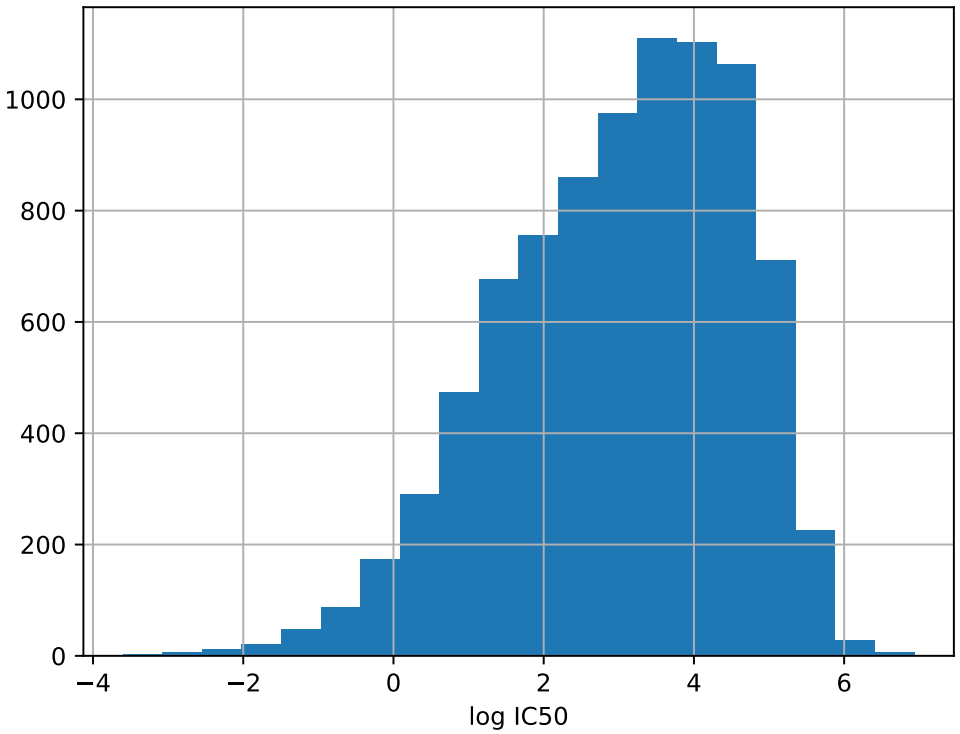
hist (*bins=20*, ***kargs*)
Histogram of the measured IC50

Parameters

- **bins** – binning of the histogram
- **kargs** – any argument accepted by pylab.hist function.

Returns all measured IC50

```
from gdsctools import IC50, ic50_test
r = IC50(ic50_test)
r.hist()
```



plot_ic50_count (***kargs*)
Plots the fraction of valid/measured IC50 per drug

Parameters **kargs** – any valid parameters accepted by pylab.plot function.

Returns the fraction of valid/measured IC50 per drug

class **GenomicFeatures** (*filename=None*, *empty_tissue_name='UNDEFINED'*)

Read Matrix with Genomic Features

These are the compulsary column names required (note the spaces):

- 'COSMIC_ID'
- 'TISSUE_FACTOR'
- 'MSI_FACTOR'

If one of the following column is found, it is removed (deprecated):

```
- 'SAMPLE_NAME'
- 'Sample Name'
- 'CELL_LINE'
```

and features can be also encoded with the following convention:

- columns ending in "_mut" to encode a gene mutation (e.g., BRAF_mut)
- columns starting with "gain_cna"
- columns starting with "loss_cna"

Those columns will be removed:

- starting with *Drug_*, which are supposibly from the IC50 matrix

```
>>> from gdsctools import GenomicFeatures
>>> gf = GenomicFeatures()
>>> print(gf)
Genomic features distribution
Number of unique tissues 27
Number of unique features 677 with
- Mutation: 270
- CNA (gain): 116
- CNA (loss): 291
```

Changed in version 0.9.10: The header's columns' names have changed to be more consistant. Previous names are deprecated but still accepted.

Changed in version 0.9.15: If a tissue is empty, it is replaced by UNDEFINED. We also strip the spaces to make sure there is "THIS" and "THIS " are the same.

Constructor

If no file is provided, using the default file provided in the package that is made of 1001 cell lines times 680 features.

Parameters **empty_tissue_name** (*str*) – if a tissue name is let empty, replace it with this string.

colnames = {'cosmic': 'COSMIC_ID', 'tissue': 'TISSUE_FACTOR', 'media': 'MEDIA_FACTOR', 'msi': 'MSI_FACT

compress_identical_features()

Merge duplicated columns/features

Columns duplicated are merged as follows. Fhe first column is kept, others are dropped but to keep track of those dropped, the column name is renamed by concatenating the columns's names. The separator is a double underscore.

```
    gf = GenomicFeatures()
    gf.compress_identical_features()
    # You can now access to the column as follows (arbitrary example)
    gf.df['ARHGAP26_mut__G3BP2_mut']
```

drop_tissue_in(*tissues*)
Drop tissues from the list

Parameters **tissues** (*list*) – a list of tissues to drop. If you have only one tissue, can be provided as a string. Since rows are removed some features (columns) may now be empty (all zeros). If so, those columns are dropped (except for the special columns (e.g, MSI).

features
return list of features

fill_media_factor()
Given the COSMIC identifiers, fills the MEDIA_FACTOR column

If already populated, replaced by new content.

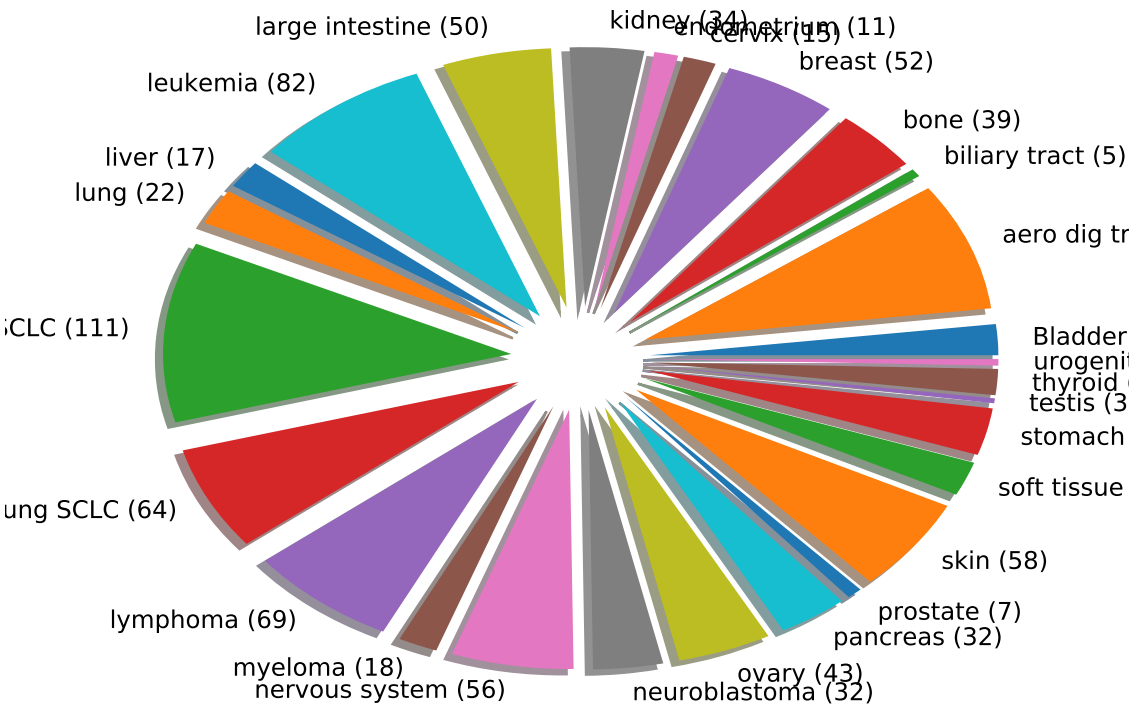
get_TCGA()

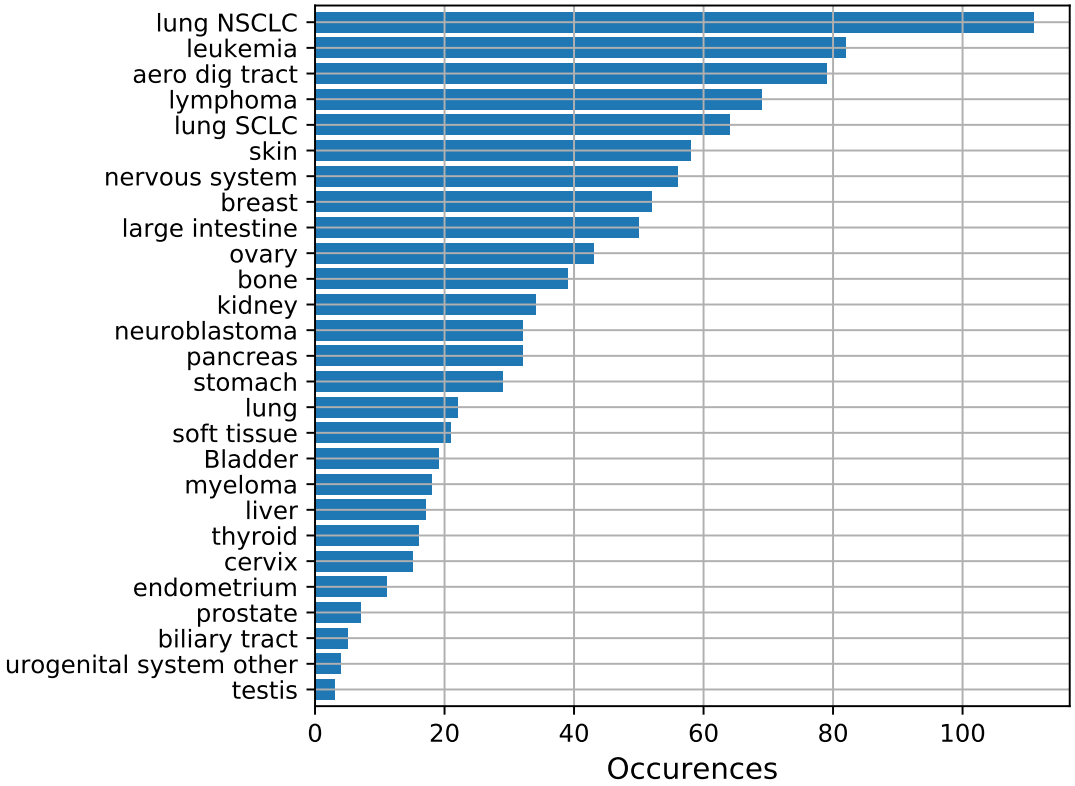
keep_tissue_in(*tissues*)
Drop tissues not in the list

Parameters **tissues** (*list*) – a list of tissues to keep. If you have only one tissue, can be provided as a string. Since rows are removed some features (columns) may now be empty (all zeros). If so, those columns are dropped (except for the special columns (e.g, MSI).

plot(*shadow=True*, *explode=True*, *fontsize=12*)
Histogram of the tissues found

```
    from gdsctools import GenomicFeatures
    gf = GenomicFeatures() # use the default file
    gf.plot()
```


large intestine (50)
kidney (34)
endometrium (11)
cervix (15)
breast (52)
leukemia (82)
bone (39)
liver (17)
biliary tract (5)
lung (22)
aero dig tr
SCLC (111)
Bladder
urogeni
thyroid
testis (3
stomach
soft tissue
ung SCLC (64)
skin (58)
prostate (7)
lymphoma (69)
pancreas (32)
myeloma (18)
ovary (43)
nervous system (56)
neuroblastoma (32)


lung NSCLC
leukemia
aero dig tract
lymphoma
lung SCLC
skin
nervous system
breast
large intestine
ovary
bone
kidney
neuroblastoma
pancreas
stomach
lung
soft tissue
Bladder
myeloma
liver
thyroid
cervix
endometrium
prostate
biliary tract
urogenital system other
testis
0
20
40
60
80
100
Occurences

shift

tissues
return list of tissues

unique_tissues
return set of tissues

class Reader(*data=None*)
Convenience base class to read CSV or TSV files (using extension)

Constructor

This class takes only one input parameter, however, it may be a filename, or a dataframe or an instance of *Reader* itself. This means than children classes such as *IC50* can also be used as input as long as a dataframe named `df` can be found.

Parameters **data** – a filename in CSV or TSV format with format specified by child class (see e.g. *IC50*), or a valid dataframe, or an instance of *Reader*.

The input can be a filename either in CSV (comma separated values) or TSV (tabular separated values). The extension will be used to interpret the content, so please be consistent in the naming of the file extensions.

```
>>> from gdsctools import Reader, ic50_test
>>> r = Reader(ic50_test.filename) # this is a CSV file
>>> len(r.df)    # number of rows
988
>>> len(r)       # number of elements
11856
```

Note that *Reader* is a base class and more sophisticated readers are available. for example, the *IC50* would be better to read this IC50 data set.

The data has been stored in a data frame in the `df` attribute.

The dataframe of the object itself can be used as an input to create an new instance:

```
>>> from gdsctools import Reader, ic50_test
>>> r = Reader(ic50_test.filename, sep="\t")
>>> r2 = Reader(r) # here r.df is simply copied into r2
>>> r == r2
True
```

It is sometimes convenient to create an empty Reader that will be populated later on:

```
>>> r = Reader()
>>> len(r)
0
```

More advanced readers (e.g. *IC50*) can also be used as input as long as they have a `df` attribute:

```
>>> from gdsctools import Reader, ic50_test
>>> ic = IC50(ic50_test)
>>> r = Reader(ic)
```

check()
Checking the format of the matrix

Currently, only checks that there is no duplicated column names

header = None
if populated, can be used to check validity of a header

read_data(*filename*)

to_csv(*filename*, *sep=', '*, *index=False*, *reset_index=True*)
Save data into a CSV file without indices

class DrugDecode(*filename=None*)
Reads a "drug decode" file

The format must be comma-separated file. There are 3 compulsary columns called DRUG_ID, DRUG_NAME and DRUG_TARGET. Here is an example:

```
DRUG_ID     ,DRUG_NAME   ,DRUG_TARGET
999         ,Erlotinib   ,EGFR
1039        ,SL 0101-1   ,"RSK, AURKB, PIM3"
```

TSV file may also work out of the box. If a column name called 'PUTATIVE_TARGET' is found, it is renamed 'DRUG_TARGET' to be compatible with earlier formats.

In addition, 3 extra columns may be provided:

```
- PUBCHEM_ID
- WEBRELEASE
- OWNED_BY
```

The OWNED_BY and WEBRELEASE may be required to create packages for each company. If those columns are not provided, the internal dataframe is filled with None.

Note that older version of identifiers such as:

```
Drug_950_IC50
```

are transformed as proper ID that is (in this case), just the number:

```
950
```

Then, the data is accessible as a dataframe, the index being the DRUG_ID column:

```
data = DrugDecode('DRUG_DECODE.csv')
data.df.iloc[999]
```

Note: the DRUG_ID column must be made of integer

Constructor

check()

companies

drugIds
return list of drug identifiers

drug_annotations(*df*)
Populate the drug_name and drug_target field if possible

Parameters **df** – input dataframe as given by e.g., `anova_one_drug()`

Return df same as input but with the FDR column populated

drug_names

drug_targets

get_info()

get_name(*drug_id*)

get_public_and_one_company(*company*)
Return drugs that belong to a specific company and public drugs

get_target(*drug_id*)

is_public(*drug_id*)

OmniBEM related

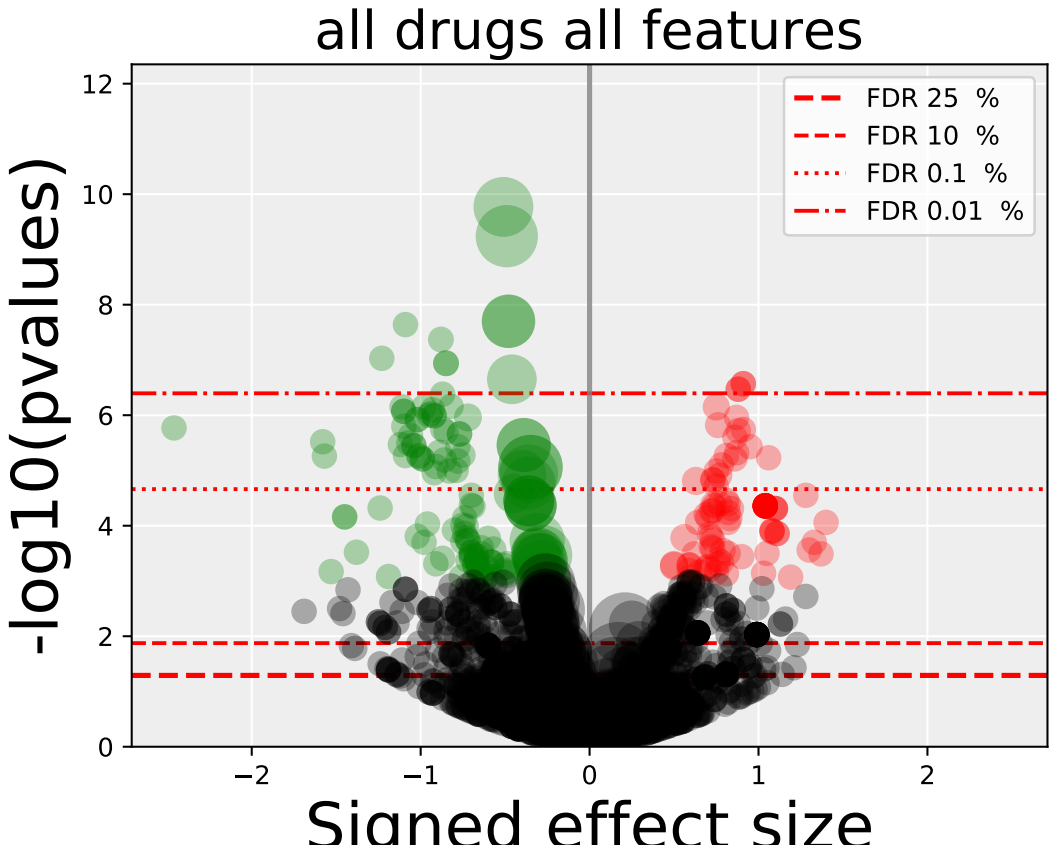
OmniBEM functions

class OmniBEMBuilder(*genomic_alteration*)
Utility to create *GenomicFeatures* instance from BEM data

Starting from an example provided in GDSCTools (test_omnibem_genomic_alteration.csv.gz), here is the code to get a data structure compatible with the `GenomicFeature`, which can then be used as input to the *ANOVA* class.

See the constructor for the header format.

```
from gdsctools import gdsctools_data, OmniBEMBuilder, GenomicFeatures

input_data = gdsctools_data("test_omnibem_genomic_alterations.csv.gz")
bem = OmniBEMBuilder(input_data)

# You may filter the data for instance to keep only a set of genes.
# Here, we keep everything
gene_list = bem.df.GENE.unique()
bem.filter_by_gene_list(gene_list)

# Finally, create a MoBEM dataframe
mobem = bem.get_mobem()

# You may filter with other functions(e.g., to keep only Methylation
# features
bem.filter_by_type_list(["Methylation"])
mobem = bem.get_mobem()

# Then, let us create a dataframe that is compatible with
# GenomicFeature. We just need to make sure the columns are correct
mobem[[x for x in mobem.columns if x!="SAMPLE"]]
gf = GenomicFeatures(mobem[[x for x in mobem.columns if x!="SAMPLE"]])

# Now, we can perform an ANOVA analysis:
from gdsctools import ANOVA, ic50_test
an = ANOVA(ic50_test, gf)
results = an.anova_all()
results.volcano()
```



Note: The underlying data is stored in the attribute `df`.

Constructor

Parameters **`genomic_alteration`** (*`str`*) – a filename in CSV format (gzipped or not). The format is explained here below.

The input must be a 5-columns CSV file with the following columns:

```
COSMIC_ID: an integer
TISSUE_TYPE: e.g. Methylation,
SAMPLE: this should correspond to the COSMID_ID
TYPE: Methylation,
GENE: gene name
IDENTIFIER: required for now but may be removed (rows can be
    used as identifier indeed
```

Warning: If GENE is set to NA, we drop it. In the resources shown in the example here above, this corresponds to 380 rows (out of 60703). Similarly, if TISSUE is NA, we also drop these rows that is about 3000 rows.

`filter_by_cosmic_list`(*cosmic_list*)
Filter the data by cosmic identifers

Parameters **cosmic** (*list*) – the cosmic identifiers to be kept. The data is updated in place.

filter_by_gene_list(*genes*, *minimum_gene=3*)
Keeps only required genes

Parameters

- **genes** – a list of genes or a filename (a CSV file with a column named GENE).
- **minimum_gene** – genes with not enough entries are removed (defaults to 3)

filter_by_sample_list(*sample_list*)
Filter the data by sample name

Parameters **tissue** (*list*) – the samples to be kept. The data is updated in place.

filter_by_tissue_list(*tissue_list*)
Filter the data by tissue

Parameters **tissue** (*list*) – the tissues to be kept. The data is update in place.

filter_by_type_list(*type_list*)
Filter the data by type

Parameters **tissue** (*list*) – the types to be kept. The data is update in place. Here are some examples of types: Point.mutation, Amplification, Deletion, Methylation.

get_genomic_features()

get_mobem()
Return a dataframe compatible with ANOVA analysis

The returned object can be read by *GenomicFeatures*.

get_significant_genes(*N=20*)
Return most present genes

Parameters **N** (*int*) – the maximum number of genes to return

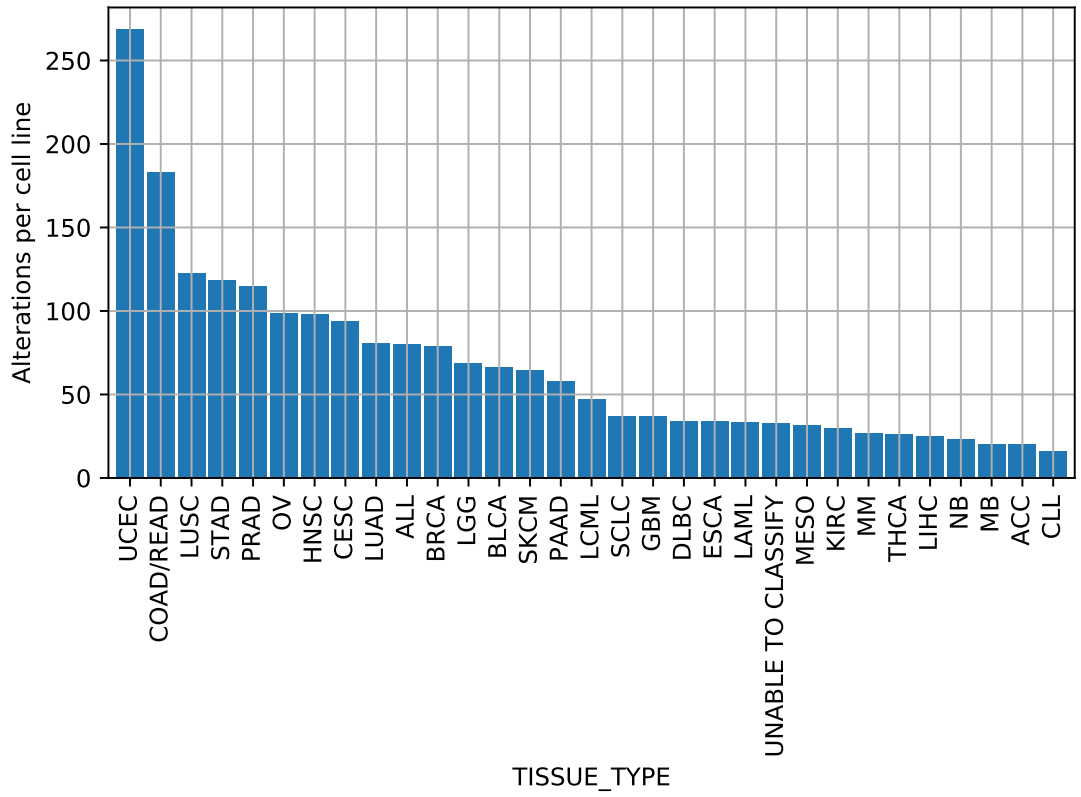
Returns list of genes with the number of occurences

The genes returned by be used to filter the data:

```
genes = bem.get_significant_genes(N=20)
bem.filter_by_gene_list(genes.index)
```

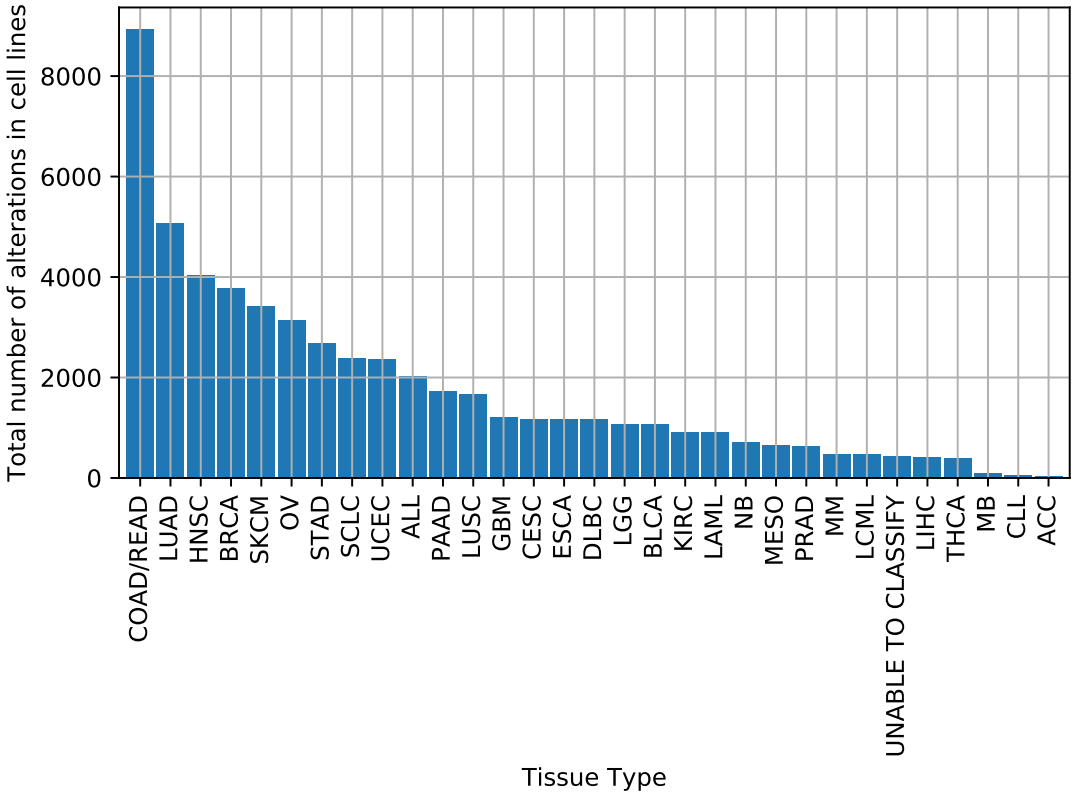
plot_alterations_per_cellline(*fontsize=10*, *width=0.9*)
Plot number of alterations

```
from gdsctools import *
data = gdsctools_data("test_omnibem_genomic_alterations.csv.gz")
bem = OmniBEMBuilder(data)
bem.filter_by_gene_list(gdsctools_data("test_omnibem_genes.txt"))
bem.plot_alterations_per_cellline()
```



plot_number_alteration_by_tissue (*fontsize=10*, *width=0.9*)
Plot number of alterations

```
from gdsctools import *
data = gdsctools_data("test_omnibem_genomic_alterations.csv.gz")
bem = OmniBEMBuilder(data)
bem.filter_by_gene_list(gdsctools_data("test_omnibem_genes.txt"))
bem.plot_number_alteration_by_tissue()
```



This module provides an interface to download and filter GDSC1000 data, as published by Iorio et al (2016).

Author: Elisabeth D. Chen Date: 2016-06-17

class `GDSC1000` (*annotate=False*, *data_folder_name='./data/gdsc_1000_data/'*)

Help data retrieval from GDSC website (version v17)

```
from gdsctools import GDSC1000
data = GDSC1000()
data.download_data()
```

Next time, starting GDSCTools in the same directory, just type:

```
data = GDSCTools()
data.load_data()
```

CNA (e.g., `cna_df`), Methylation and gene variants are downloaded. IC50 as well and a set of annotations. There are combined in `genomic_df`.

The `genomic_df` contains the `cna_df`, `methylation_df` and `variant_df` data. The three latter contains in common the CELL_LINE, ALTERATION_TYPE, IDENTIFIER and TISSUE_FACTOR.

ALTERATION_TYPE can be GENETIC_VARIATION, METHYLATION, DELETION / AMPLIFICATION variant_df also has the COSMIC_ID. The `annotation_df` gives mapping between identifiers and gene names.

download_data()	Download and load data in memory
load_data([annotation])	If CSV files are already downloaded, just load them
make_matrix([min_recurrence])	Return a dataframe compatible with ANOVA analysis

With this class, you may (1) download the data, (2) annotate or filter the data and (3) create a genomic matrix.

To load the data, either download it from the website using *download_data()* (downloads data from cancerrxgene.org loading methylation, cna, variant and cell lines datasets). It extract relevant columns, re-names column names and saves as csv files in the `data_folder_name` directory.

If you have already downloaded the data, you may just load it using *load_data()*. This function also annotates the data with gene information.

Then, you filter the data with one of the filter methods:

```
filter_by_cell_line([ "AsPC-1", "U-2-OS", "MDA-MB-231"... ] )
filter_by_cosmic_id([ 948121, 2839818, ...] )
filter_by_gene(["ATM", "TP53", ...])
filter_by_recurrence(min_recurrence = 3 )
filter_by_tissue_type([ "BRCA", "COAD/READ", ... ] )
filter_by_alteration_type(["METHYLATION", "AMPLIFICATION",
                                "DELETION", "GENE_VARIANT" ] )
```

Finally, you can create the genomic matrix using *make_matrix()*.

You can also look at the unique regions for agiven data set. For instance, methylation dataframe has about 2,000 regions but only 338 are unique:

```
len(data.methylation_df.groupby('IDENTIFIER').groups)
```

constructor

Parameters

- **annotate** (*bool*) –
- **data_folder_name** (*str*) –

annotate_all()
Annotates dataframes after read_annotation

download_data()
Download and load data in memory

filter_by_alteration_type(*type_list=None*)

filter_by_cell_line(*cell_line_list=None*)

filter_by_cosmic_id(*cosmic_list=None*)

filter_by_gene(*gene_list=None*)

filter_by_recurrence(*min_recurrence=None*)

filter_by_tissue_type(*tissue_list=None*)

get_cna_info()
Return dataframe with number of genes per unique CNA identifier

get_genomic_info()
Return information about the genomic dataframe

The returned dataframes contains two columns: the number of unique cosmic identifiers and features for each type of alterations.

get_methylation_info()
Return dataframe with number of genes per unique methylation identifier

load_data(*annotation=True*)
If CSV files are already downloaded, just load them

make_matrix(*min_recurrence=None*)
Return a dataframe compatible with ANOVA analysis

reset_genomic_data()

Visualisation

Volcano plot

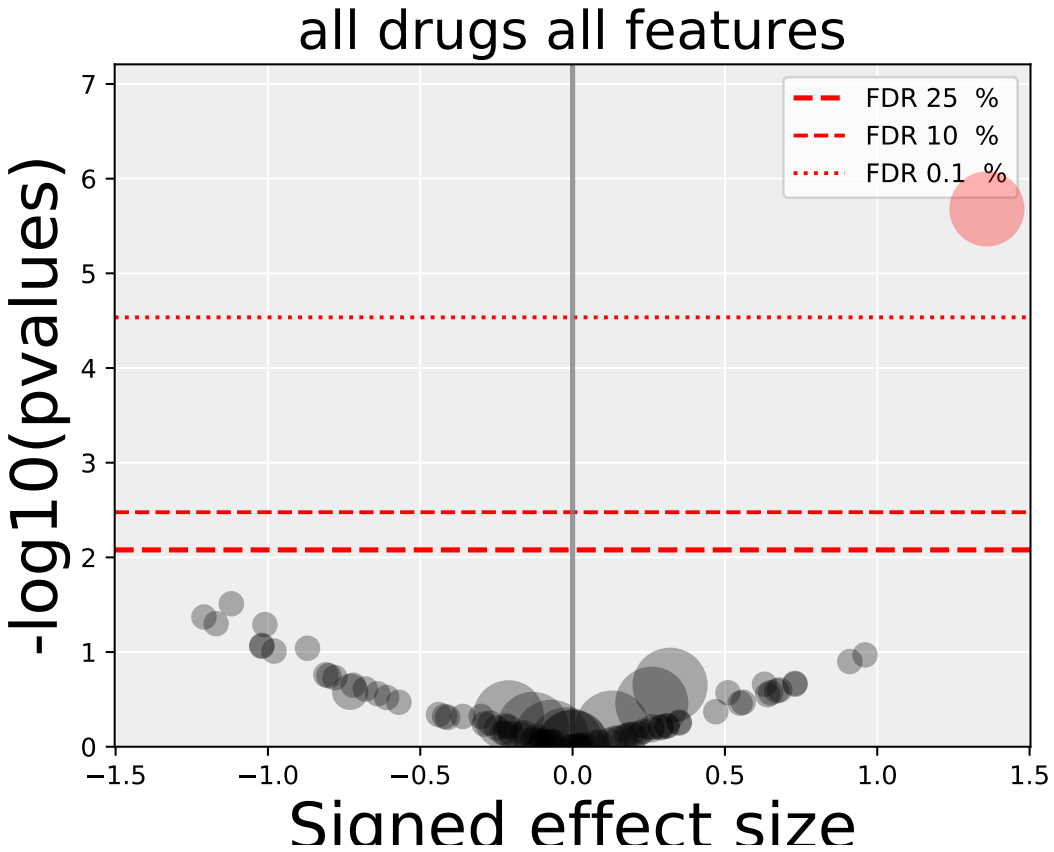
Volcano plot utilities

The *VolcanoANOVA* is used in the creation of the report but may be used by a usr in a Python shell.

class VolcanoANOVA(*data*, *sep='t'*, *settings=None*)
Utilities related to volcano plots

This class is used in *gdsctools.anova* but can also be used independently as in the example below.

```
from gdsctools import ANOVA, ic50_test, VolcanoANOVA
an = ANOVA(ic50_test)

# retrict analysis to a tissue to speed up computation
an.set_cancer_type('lung_NSCLC')

# Perform the entire analysis
results = an.anova_all()

# Plot volcano plot of pvalues versus signed effect size
v = VolcanoANOVA(results)
v.volcano_plot_all()
```



Note: Within an IPython shell, you should be able to click on a circle and the title will be updated with the name of the drug/feature and FDR value.

Legend and color conventions The green circles indicate significant hits that are resistant while reds show sensitive hits. Circles are colored if there are below the FDR_threshold AND below the pvalue_threshold AND if the signed effect size is above the effect_threshold.

Constructor

Parameters

- **data** – an `ANOVAResults` instance or a dataframe with the proper columns names (see below)
- **settings** – an instance of *ANOVASettings*

Expected column names to be found if a filename is provided:

```
ANOVA_FEATURE_pval
ANOVA_FEATURE_FDR
FEATURE_delta_MEAN_IC50
FEATURE_IC50_effect_size
N_FEATURE_pos
N_FEATURE_pos
```

```
    FEATURE
    DRUG_ID
```

If the plotting is too slow, you can use the *selector()* to prune the results (most of the data are noise and overlap on the middle bottom area of the plot with little information.

savefig(*filename*, *size_inches=(10*, *10)*)

selector(*df*, *Nbest=1000*, *Nrandom=1000*, *inplace=False*)
Select only the first N best rows and N random ones

Sometimes, there are tens of thousands of associations and future analysis will include more features and drugs. Plotting volcano plots should therefore be fast and scalable. Here, we provide a naive way of speeding up the plotting by selecting only a subset of the data made of Nbest+Nrandom associations.

Parameters

- **df** – the input dataframe with ANOVAResults
- **Nbest** (*int*) – how many of the most significant association should be kept
- **Nrandom** (*int*) – on top of the Nbest significant association, set how many other randomly chosen associations are to be kept.

Returns pruned dataframe

volcano_plot_all()
Create an overall volcano plot for all associations

This method saves the picture in a PNG file named **volcano_all.png**.

volcano_plot_all_drugs()
Create a volcano plot for each drug and save in PNG files

Each filename is set to **volcano_<drug identifier>.png**

volcano_plot_all_features()
Create a volcano plot for each feature and save in PNG files

Each filename is set to **volcano_<feature name>.png**

volcano_plot_one_drug(*drug_id*)
Volcano plot for one drug (all genomic features)

Parameters **drug_id** – a valid drug identifier to be found in the results

volcano_plot_one_feature(*feature*)
Volcano plot for one feature (all drugs)

Parameters **feature** – a valid feature name to be found in the results

class VolcanoANOVAJS(*data*, *sep='t'*, *settings=None*)

get_fdr_ypos()

render_all()

render_drug(*name*)

render_feature(*name*)

Boxplot and beeswarm

boxswarm(*data*, *names=None*, *vert=True*, *widths=0.5*, ***kwargs*)
Plot boxplot with all points as circles.

This function is a wrapper of *BoxSwarm*

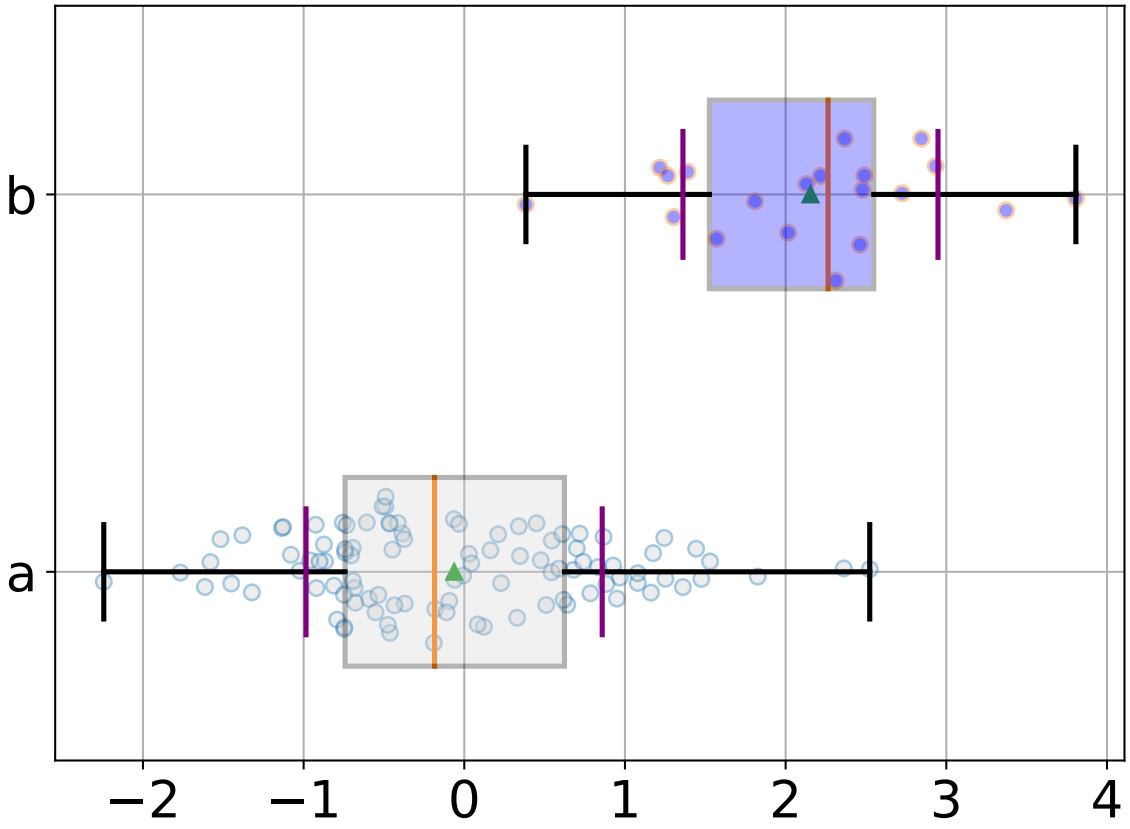
Parameters

- **data** – a dataframe. Each column is a data set from which a boxplot is created.
- **names** –
- **vert** – orientation of the boxplots
- **widths** – widths of the boxes
- **kargs** – any argument accepted by *BoxSwarm*

See *BoxSwarm* documentation for details

class BoxSwarm(*data, names=None, fontsize=20, hold=False, title='', lw=2, colors=['lightgrey', 'blue']*)
Simple beeswarm plot (boxplot + dots for each data point)

```
from pylab import randn
from gdsctools.boxswarm import BoxSwarm
b = BoxSwarm({'a':randn(100), 'b':randn(20)+2})
b.plot(vert=False)
```



Note: could use pybeeswarm, which is a proper implementation of beeswarm.

Constructor

Param a list of list (not same size) or a dictionary of lists

Parameters

- **data** –
- **names** –
- **fontsize** –
- **hold** –
- **title** –
- **lw** – width of lines
- **colors** – loop over the list of colors provided to fill boxplots
- ****kargs** –

beeswarm(*data*, *position*, *ratio=2.0*)
Naive plotting of the data points

We assume gaussian distribution so we expect fewers dots far from the mean/median. We'd like those dots to be close to the axes. conversely, we expect lots of dots centered around the mean, in which case, we'd like them to be spread in the box. We uniformly distribute position using

$$X = X + \frac{U() - 0.5}{ratio} \times factor$$

but the factor is based on an arctan function:

$$factor = 1 - \arctan(\frac{X - \mu}{\pi/2})$$

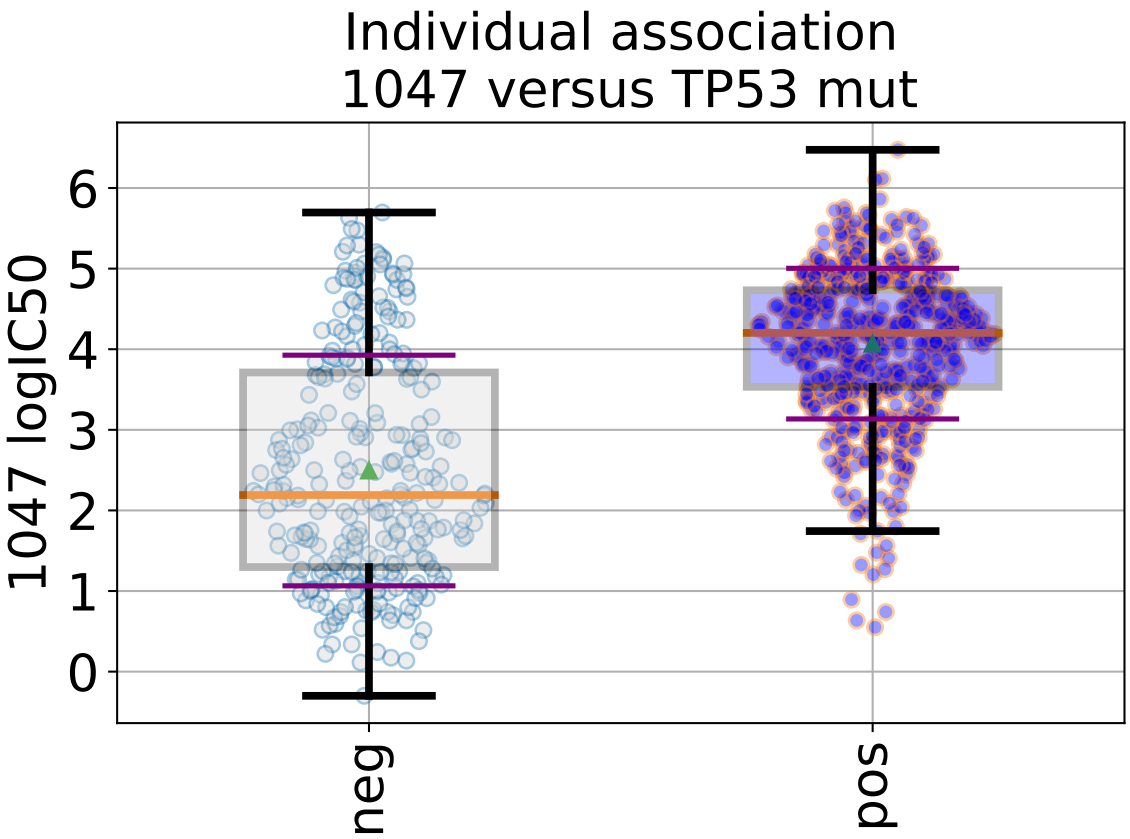
The farther the data is from the mean μ, the closest it is to the axes that goes through the box.

plot(*vert=True*, *alpha=0.4*, *widths=0.5*, ***kwargs*)
Plot the boxplots and dots

Code related to the ANOVA analysis to find associations between drug IC50s and genomic features

class BoxPlots(*odof*, *fontsize=20*, *savefig=False*, *directory='.'*)
Box plot for a given association of drug versus genomic feature

```
from gdsctools import ANOVA, ic50_test
from gdsctools.boxplots import BoxPlots

gdsc = ANOVA(ic50_test)

# Perform the entire analysis
odof = gdsc._get_one_drug_one_feature_data(1047, 'TP53_mut')

# Plot volcano plot of pvalues versus signed effect size
bx = BoxPlots(odof)
bx.boxplot_association()
```



If the **gdsc** analysis has the MSI factor and tissue factor on, then additional plots can be created using *boxplot_pancan()*.

Note that *odof* in the example above is a dictionary with the following keys:

- drug_name
- feature_name
- masked_tissue: a dataframe with cosmic ids as index and 1 column of tissues names.
- Y: list with the IC50s
- masked_features: a dataframe with cosmic ids as index and 1 column of masked feature (1/0)
- masked_msi: same as masked_features
- negatives: subset of the IC50s corresponding to positive feature
- positives: subst of the IC50s corresponding to negative feature

See also:

gdsctools.boxswarm.BoxSwarm

Constructor

boxplot_association(*fignum=1*)

Boxplot of the association (negative versus positive)

Parameters **fignum** – number of the figure

boxplot_pancan(*mode*, *fignum=1*, *title_prefix=''*)
Create boxplot related to the MSI factor or Tissue factor

Parameters **mode** – either set to **msi** or **tissue**

directory = None
directory where to save the figure.

fontsize = None
fontsize for the plots

lw = None
linewidth used in the plots

odof = None
dictionary as returned by ANOVA._get_one_drug_one_feature_data

savefig = None
boolean to save figure

class BoxPlotsJS(*odof*, *fontsize=20*, *savefig=False*, *directory='.'*)

get_html_association()

get_html_media()

get_html_msi()

get_html_tissue()

reports

Base classes to create HTML reports easily

class ReportMain(*filename='index.html'*, *directory='report'*, *overwrite=True*, *verbose=True*, *template_filename='index.html'*, *mode=None*, *init_report=True*)
A base class to create HTML pages

This `Report` class holds the CSS and HTML layout and will ease the creation of new reports and HTML pages. For instance, it will add a footer and header automatically, save files in the proper directory, create the directory if it is missing, copy CSS and JS files in the directory automatically.

```
from gdsctools import Report
r = Report()
r.add_section('Example with some text', 'Example' )
r.report(onweb=True)
```

Note: **For developers** the original CSS and JS files are stored in the share/data directory.

The idea is that you create sections (text + title) that you add little by little in your HTML documents. Then, you create the report. The report will add a footer, header, table of contents before the sections. The **text** of a section can contain any HTML document.

Constructor

Parameters

- **filename** – default to **index.html**

- **directory** – defaults to **report**
- **overwrite** – default to True
- **verbose** – default to True
- **dependencies** – add the dependencies table at the end of the document if True.
- **mode** (*str*) – if none, report have a structure that contains the following directories: OUTPUT, INPUT, js, css, images, code. Otherwise, if mode is set to 'summary', only the following directories are created: js, css, images

create_report(*onweb=True*)

show()
Opens a tab in a browser to see the document

Data-related

class TCGA
A dictionary-like container to access to TCGA cancer types keywords

```
>>> from gdsctools import TCGA
>>> tt = TCGA()
>>> tt.ACC
'Adrenocortical Carcinoma'
```

TCGA_GDSC1000 = ['BLCA', 'BRCA', 'COREAD', 'DLBC', 'ESCA', 'GBM', 'HNSC', 'KIRC', 'LAML', 'LGG', 'LIHC', 'LU
TCGA keys used in GDSC1000

class Tissues
List of tissues included in various analysis

Contains tissues included e.g in v17,v18

Data sets provided with GDSCTools

The datasets may be for testing purpose:

- ic50_test
- drug_test
- cosmic_builder_test

or informative:

- cancer_cell_lines
- cosmic_info

or used in analysis:

- genomic_features_test
- ic50_v17: IC50s for 1001 cell lines
- gf_v17: dataset with genomic features for 1001 cell lines and 680 features (mutation, CNA)
- ic50_v5
- gf_v5

class **Data** (*filename=None*, *description='No description'*, *authors='GDSC consortium'*)
A convenience class to hold information about a dataset

Each *Data* instance contains information about :

1.the file location (*filename*)

2.the data description (*description*)

3.the authors (*authors*)

But the data is not stored and users must read the data set using their own tools.

authors = None
list of authors (string)

description = None
a short description (string)

filename = None
where is located the data set (full path)

location

class **COSMICFetcher** (*filename=None*)
Utility to download a flat file about cosmic identier and build a small dataframe with cosmic identifiers and their diseases

The original flat file is downloaded from ftp.expasy.org/databases and contains records as follows:

```
ID         Identifier (cell line name)     Once; starts an entry
AC         Accession (CVCL_xxxx)           Once
SY         Synonyms                        Optional; once
DR         Cross-references                Optional; once or more
RX         References identifiers          Optional: once or more
WW         Web pages                       Optional; once or more
CC         Comments                        Optional; once or more
DI         Diseases                        Optional; once or more
OX         Species of origin               Once or more
HI         Hierarchy                       Optional; once or more
OI         Originate from same individual  Optional; once or more
SX         Sex (gender) of cell            Optional; once
CA         Category                        Optional; once
```

We keep only records with COSMIC cross references. From those records, we keep ID, AC, CA, DI (Disease) and the cosmic identifier.

The resulting dataframe can then be accessed in the `df` attribute.

```
>>> from gdsctools.cosmictools import COSMICFetcher
>>> cf = COSMICFetcher() # this may take a while to download the file
>>> cf.df.loc[0]
ID                                      OS-A
AC                                  CVCL_0C23
CA                             Cancer cell line
COSMIC_ID                               2239090
Disease      C4917; Small cell lung carcinoma
Name: 0, dtype: object
```

Constructor

Parameters **filename** (*str*) – If not provided, download file from expasy.org and store it in `data`. Otherwise, if filename is provided, reads a local file. Format should be the same as the file downloaded from expasy

class COSMICInfo

Retrieve information about cell line included in GDSC1000

This file reads a GDSCTools dataset `gdsctools.datasets.cosmic_info`. Its content is stored in *df*.

In corresponds to Table S1E (List cell line samples with data availability and annotations across the different omics

The method *get()* retrieves information contained in the dataframe *df*. Provide a known cosmic identifier as follows:

```
>>> from gdsctools import COSMICInfo
>>> c = COSMICInfo()
>>> c.get(909907, 'SAMPLE_NAME')
'ZR-75-30'
```

or get all available field as follows:

```
>>> c.get(909907)
SAMPLE_NAME            ZR-75-30
SEQ                           1
CNA                           1
EXP                           1
MET                           1
DRUG_SCR                      1
GDSC_description_1       breast
GDSC_description_2       breast
Study_Abbreviation         BRCA
MMR                       MSI-L
SCREEN_MEDIUM                 R
GROWTH_PROPERTIES      Adherent
Name: 909907, dtype: object
```

Note: there are only 1000 cell lines in the *df*. Additional cell lines may be retrieve using *COSMICFetcher*

If a cosmic identifier is not found, the returned object has the same structure as above but with all fields set to False.

See also:

http://www.cancerrxgene.org/translation/CellLine

constructor

df = None
dataframe with all information

get(*identifier*, *colname=None*)

Parameters

- **identifier** (*int*) – a cosmic identifiers. Possible values are stored in `df.index` attribute
- **colname** – specific field.

Returns if colname is not provided, returns a time series for the **identifier** with all available fields. Otherwise, returns a specific field.

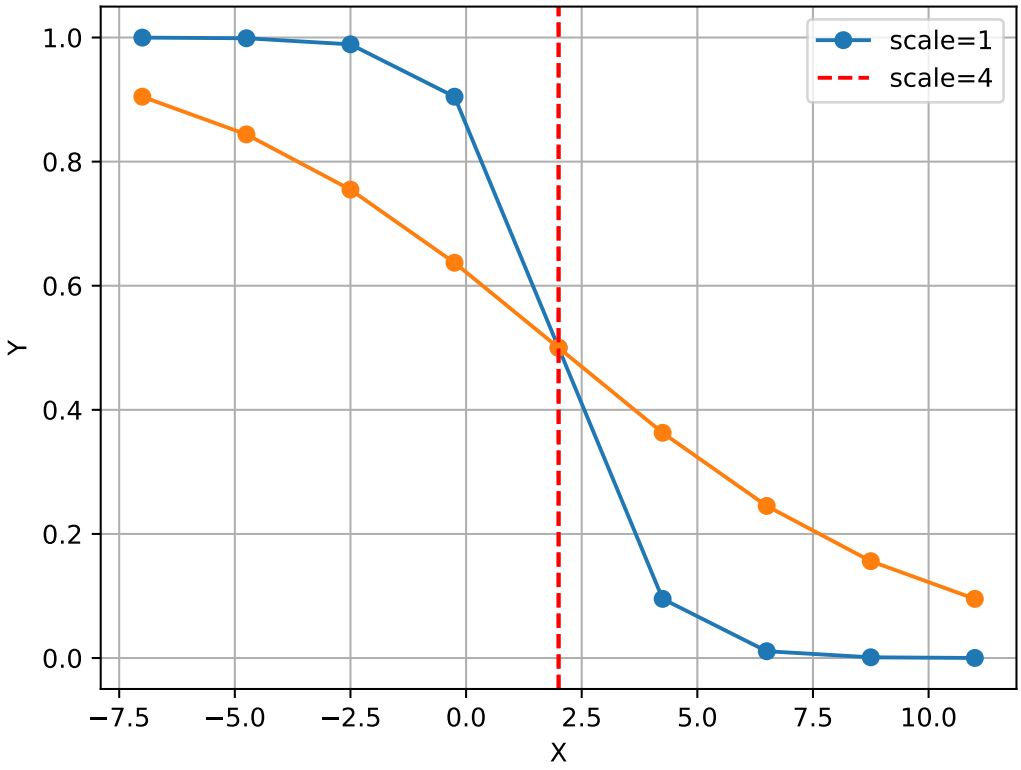
on_web (*identifier*)
Open a tab related to the COSMIC identifier (in your browser)

Logistics

Sets of miscellaneous tools

class Logistic (*xmid*, *scale*, *Asym=1*, *N=9*, *increase=False*)
Simple logistic class to see the curve implied by xmid/scale parameters

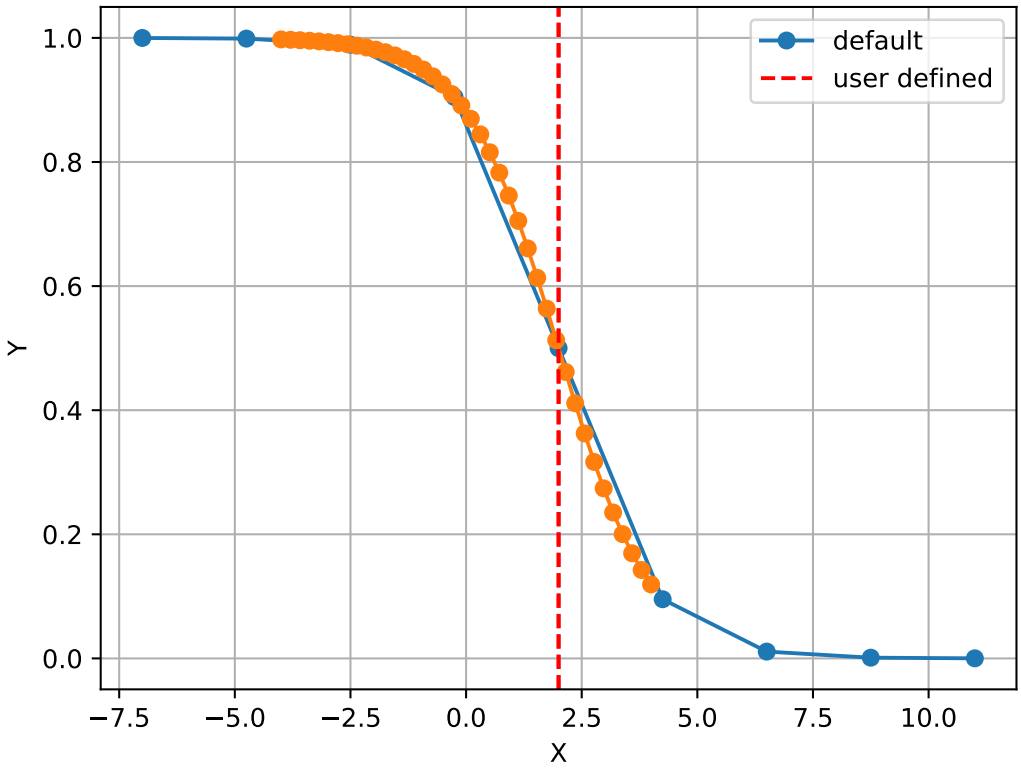
```
from gdsctools.logistics import Logistic
from pylab import legend
tl = Logistic(2, 1)
tl.plot()
tl.scale = 4
tl.plot(hold=True)
legend(['scale=1', 'scale=4'])
```



The X values are set automatically given the number of data points *N* and the minimum and maximum X values. By default a sensible values for the minimun and maximum x values are guessed based on *scale* parameter

but one can set the *xmin* and *xmax*

```
from gdsctools.logistics import Logistic
from pylab import legend
tl = Logistic(2,1)
tl.plot()
tl.xmin= -4
tl.xmax = 4
tl.N = 40
tl.plot(hold=True)
legend(['default', 'user defined'])
```



Constructor

Parameters

- **xmid** – the first logistic function parameter
- **scale** – the second logistic function parameter
- **Asym** – Amplitude of the function
- **N** – number of data point
- **increase** – increasing or decreasing function.
- **xmin** – starting range of x-values
- **xmax** – ending range of x-values

if `increase` is False:

$$L(x) = \frac{Asym}{1 + exp((xmid - X)/scale)}$$

if `increase` is True

$$L(x) = 1 - \frac{Asym}{1 + exp((xmid - X)/scale)}$$

Changing *xmin*, *xmax* or *N* does change the content of X. You can change *X* directly.

N
set/get the number points

X
get/set of the x-values

Y
Getter for Y-values

plot (*hold=False*)
Plot the logistic function

scale

xmax
set/get the maximum x range

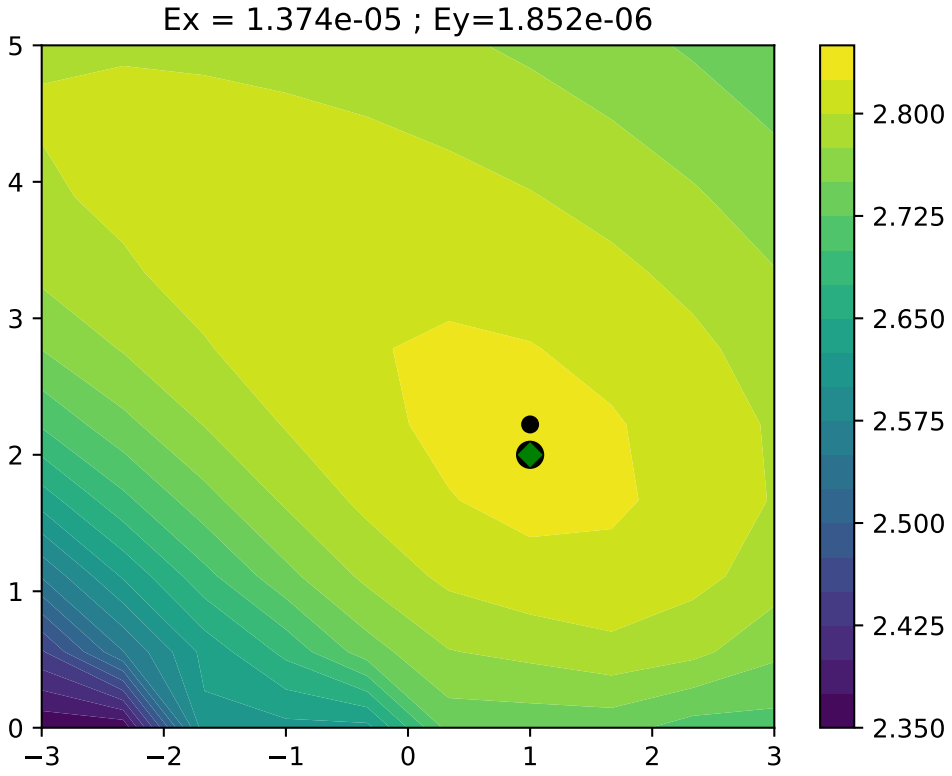
xmin
set/get the minimum x range

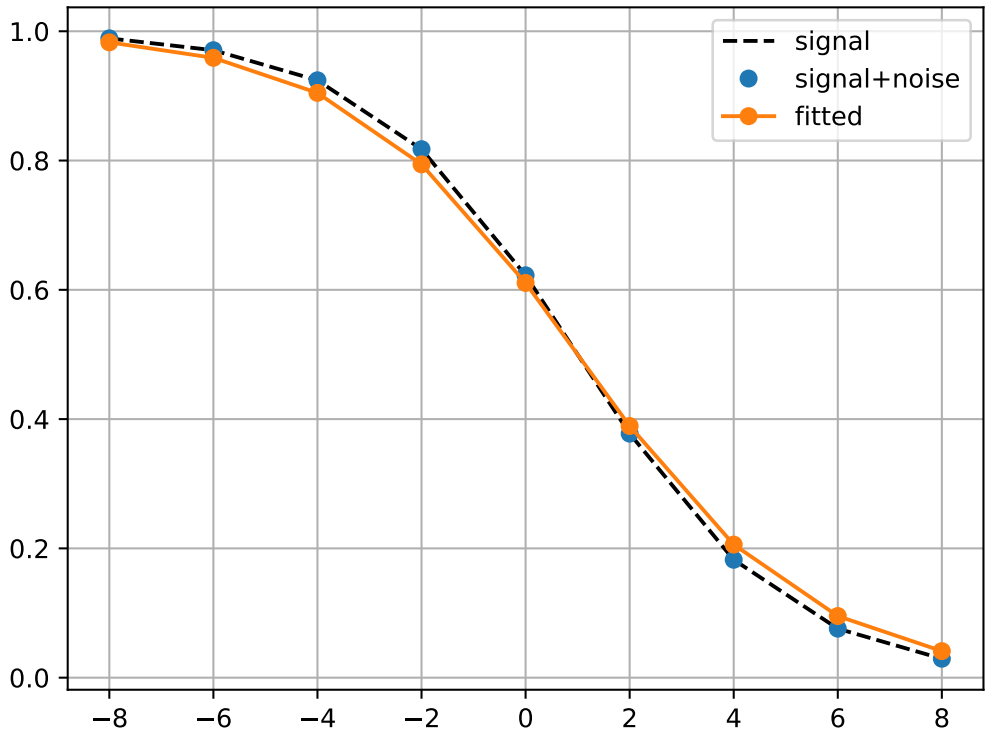
class LogisticMatchedFiltering (*xmid*, *scale*, *N=9*)
Experimental class to identify parameters of a noisy Logistic function

This class implements two methods to identify the parameters (xmid and scale) of a logistic function.

Note One constraint is to define the x-values.

```
from gdsctools import logistics
import pylab

mf = logistics.LogisticMatchedFiltering(1,2)
mf.scan(pylab.linspace(-3,3, 10), pylab.linspace(0,5,10))
```


Ex = 1.374e-05 ; Ey=1.852e-06
5
4
3
2
1
0
−3
−2
−1
0
1
2
3
2.800
2.725
2.650
2.575
2.500
2.425
2.350



constructor

Parameters

- **xmid** –
- **scale** –
- **N** –

N

X

get/set the X values the logistic signal

get_snr (*template*, *show=True*)

objective_function (*xmid*, *scale*)

optimise (*guess=[1, 1]*)

plot (*results*, *coords*, *best*, *xmid_range*, *scale_range*)

scale

get/set the scale value of the logistic signal

scan (*xmid_range*, *scale_range*, *method='mf'*, *show=True*)

set_noisy_data (*sigma=0.1*)

xmid
get/set the xmid value of the logistic signal

Regression Analysis

Common Regression class

class Regression(*ic50*, *genomic_features=None*, *verbose=False*)
Base class for all Regression analysis

In the *gdsctools.anova.ANOVA* case, the regression is based on the OLS method and is computed for a given drug and a given feature (*ODOF*). Then, the analysis is repeated across all features for a given drug (*ODAF*) and finally extended to all drugs (*ADAF*). So, there is one test for each combination of drug and feature.

Here, all features for a given drug are taken together to perform a Regression analysis (*ODAF*). The regression algorithm implemented so far are:

•Ridge

•Lasso

•ElasticNet

•LassoLars

Based on tools from the scikit-learn library.

Constructor

Parameters

- **ic50** – an IC50 file
- **genomic_features** – a genomic feature file

see *Data Format and Readers* for help on the input data formats.

boxplot(*drug_name*, *model*, *n=5*, *minimum_match_per_combo=10*, *bx_vert=True*, *bx_alpha=0.5*, *verbose=False*, *max_label_length=40*)
Boxplot plot of the most important features.

Parameters

- **n** (*int*) – maximum most important features to be shown (in term of coefficient/weight used in the model)
- **minimum_match_per_combo** (*int*) – minimum number of alterations required for a feature to be shown

```
# assuming there is a drug ID = 29
r = GDSCLasso()
model = r.get_best_model(29)
r.boxplot(29, model)
```

In addition, we also show the wild type case where the total number of mutations is 0.

check_randomness(*drug_name*, *kfolds=10*, *N=10*, *progress=False*, *nbins=40*, *show=True*, ***kargs*)
Compute Bayes factor between NULL model and best model fitted N times

Parameters

- **drug_name** –
- **kfolds** –
- **N** (*int*) – optimise NULL models and real model N times
- **progress** –
- **nbins** –
- **show** –

Bayes factor:

```
S = sum([s>r for s,r in zip(scores, random_scores)])
proba = S / len(scores)
bayes_factor = 1. / (1-proba)
```

Interpretation for values of the Bayes factor according to Kass and Raftery (1995).

Interpretation	B(1,2)
Very strong support for 1	< 0.0067
Strong support 1	0.0067 to 0.05
Positive support for 1	0.05 to .33
Weak support for 1	0.33 to 1
No support for either model	1
Weak support for 2	1 to 3
Positive support for 2	3 to 20
Strong support for 2	20 to 150
Very strong support for 2	> 150

references: **http://www.socsci.uci.edu/~mdlee/LodewyckxEtAl2009.pdf**
http://www.aarondefazio.com/adefazio-bayesfactor-guide.pdf

dendogram_coefficients (*stacked=False*, *show=True*, *cmap='terrain'*)
shows the coefficient of each optimised model for each drug

This works for demonstration and small data sets.

fit (*drug_name*, *alpha=1*, *l1_ratio=0.5*, *kfolds=10*, *show=True*, *tol=0.001*, *normalize=False*, *shuffle=False*, *perturbation=0.01*, *randomize_Y=False*)
Run Elastic Net with a cross validation for one value of alpha

Parameters

- **drug_name** – the drug to analyse
- **alpha** (*float*) – note that theis alpha parameter corresponds to the lambda parameter in glmnet R package.
- **l1_ratio** (*float*) – This is the lasso penalty parameter. Note that in scikit-learn, the l1_ratio correspond to the alpha parameter in glmnet R package. l1_ratio set to 0.5 means that there is a weight equivalent for the Lasso and Ridge effects.
- **kfolds** (*int*) – defaults to 10
- **shuffle** – shuffle the indices in the KFold

Returns kfolds scores for each fold. The score is the pearson correlation.

Note: l1_ratio < 0.01 is not reliable unless sequence of alpha is provided.

Note: alpha = 0 correspond to an OLS analysis

get_best_model(*drug_name*, *kfolds=10*, *alphas=None*, *l1_ratio=0.5*)
Return best model fitted using a CV

Parameters

- **drug_name** –
- **kfolds** –
- **alphas** –
- **l1_ratio** –

plot_importance(*drug_name*, *model=None*, *fontsize=11*, *max_label_length=35*, *orientation='vertical'*)
Plot the absolute weights found by a fittd model.

Parameters

- **drug_name** (*str*) –
- **model** – a model
- **fontsize** (*int*) – (defaults to 11)
- **max_label_length** – 35 by default
- **orientation** – orientation of the plot (vertical or horizontal)

Returns the dataframe with the weights (may be empty)

Note: if no weights are different from zeros, no plots are created.

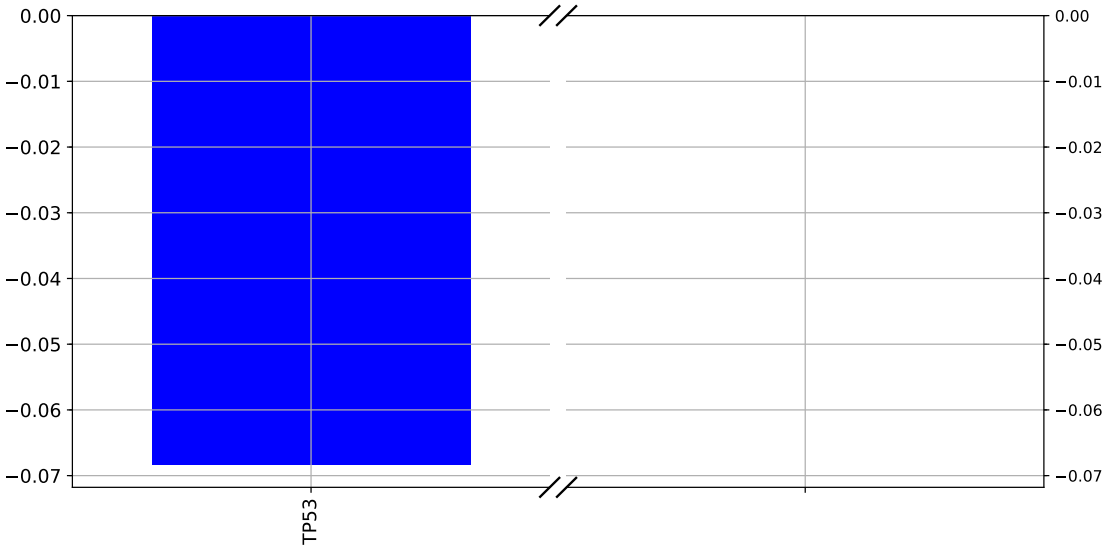
plot_weight(*drug_name*, *model=None*, *fontsize=12*, *figsize=(10, 7)*, *max_label_length=20*, *Nmax=40*)
Plot the elastic net weights

Parameters

- **drug_name** – the drug identifier
- **alpha** –
- **l1_ratio** –

Large alpha values will have a more stringent effects on the weigths and select only some of them or maybe none. Conversely, setting alphas to zero will keep all weights.

```
from gdsctools import *
ic = IC50(gdsctools_data("IC50_v5.csv.gz"))
gf = GenomicFeatures(gdsctools_data("genomic_features_v5.csv.gz"))
en = GDSCElasticNet(ic, gf)
model = en.get_model(alpha=0.01)
en.plot_weight(1047, model=model)
```



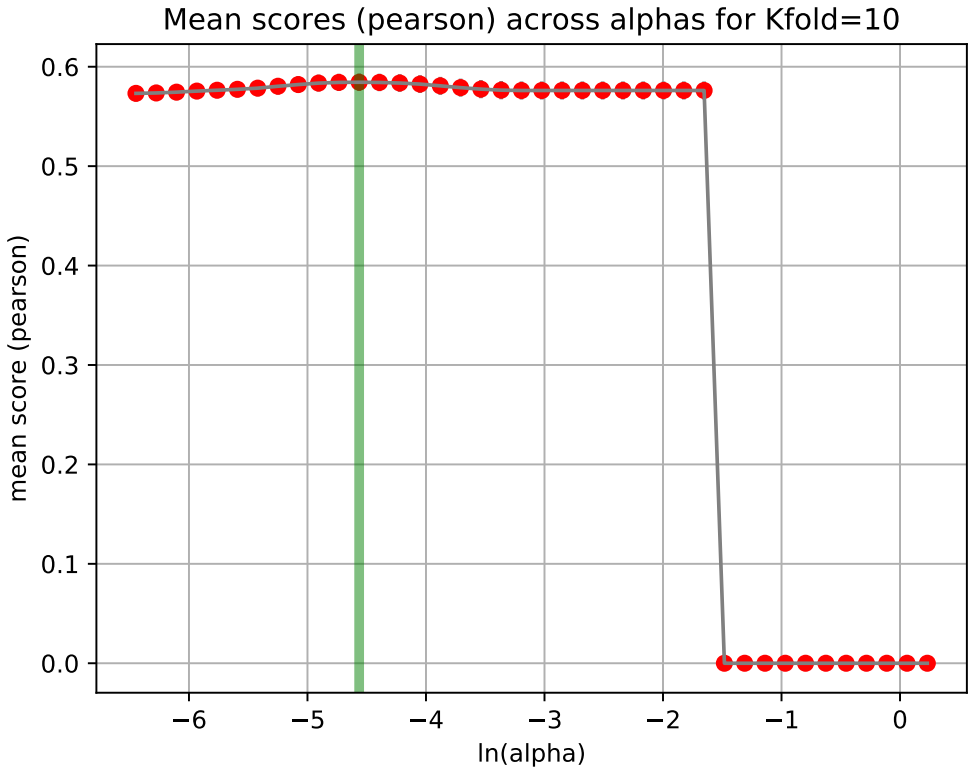
runCV(*drug_name*, *l1_ratio=0.5*, *alphas=None*, *kfolds=10*, *verbose=True*, *shuffle=True*, *randomize_Y=False*, ***kargs*)
Perform the Cross validation to get the best alpha parameter.

Returns an instance of `RegressionCVResults` that contains alpha parameter and Pearson correlation value.

tune_alpha(*drug_name, alphas=None, N=80, l1_ratio=0.5, kfolds=10, show=True, shuffle=False, alpha_range=[-2.8, 0.1], randomize_Y=False*)
Interactive tuning of the model (alpha).

This is much faster than `plot_cindex()` but much slower than `runCV()`.

```
from gdsctools import *
ic = IC50(gdsctools_data("IC50_v5.csv.gz"))
gf = GenomicFeatures(gdsctools_data("genomic_features_v5.csv.gz"))

en = GDSCElasticNet(ic, gf)

en.tune_alpha(1047, N=40, l1_ratio=0.1)
```



Lasso

class **GDSCLasso**(*ic50*, *genomic_features=None*, *verbose=False*)

See as `Regression`

get_model(*alpha=1*, *l1_ratio=None*, ***kargs*)

ElasticNet

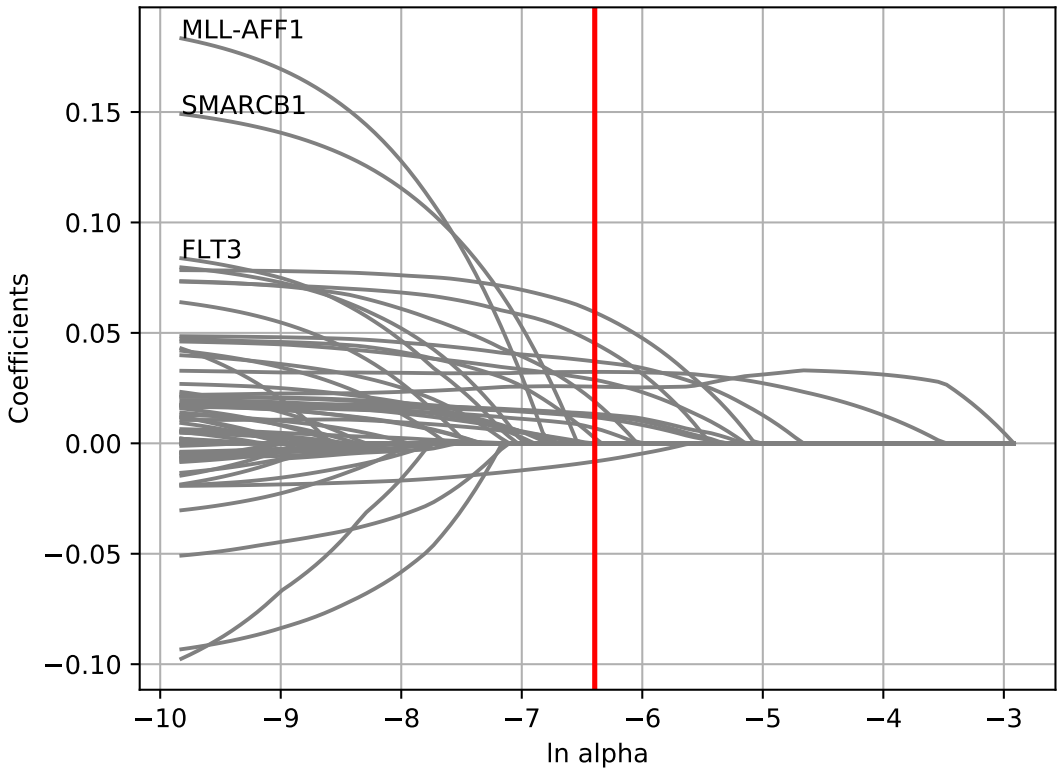
class **GDSCElasticNet**(*ic50*, *genomic_features=None*, *verbose=False*, *set_media_factor=False*)

Variant of `ANOVA` that handle the association at the drug level using an ElasticNet analysis of the IC50 vs Feature matrices.

As compared to the `GDSCRidge` and `GDSCLasso`

Here is an example on how to perform the analysis, which is similar to the ANOVA API:

```
from gdsctools import GDSCElasticNet, gdsctools_data, IC50, GenomicFeatures
ic50 = IC50(gdsctools_data("IC50_v5.csv.gz"))
gf = GenomicFeatures(gdsctools_data("genomic_features_v5.csv.gz"))

en = GDSCElasticNet(ic50, gf)
en.enetpath_vs_enet(1047)
```



For more information about the input data sets please see *ANOVA*, *readers*

Constructor

Parameters

- **IC50** (*DataFrame*) – a dataframe with the IC50. Rows should be the COSMIC identifiers and columns should be the Drug names (or identifiers)
- **features** – another dataframe with rows as in the IC50 matrix and columns as features. The first 3 columns must be named specifically to hold tissues, MSI (see format).
- **verbose** – verbosity in "WARNING", "ERROR", "DEBUG", "INFO"

enetpath_vs_enet (*drug_name*, *alphas=None*, *l1_ratio=0.5*, *nfeat=5*, *max_iter=1000*, *tol=0.0001*, *selection='cyclic'*, *fit_intercept=False*)

#if X is not scaled, the enetpath and ElasticNet will give slightly # different results #if X is scale using:

```
from sklearn import preprocessing
xscaled = preprocessing.scale(X)
xscaled = pd.DataFrame(xscaled, columns=X.columns)
```

get_model (*alpha=1*, *l1_ratio=0.5*, *tol=0.001*, *normalize=False*, ***kargs*)

plot_cindex (*drug_name*, *alphas*, *l1_ratio=0.5*, *kfolds=10*, *hold=False*)

Tune alpha parameter using concordance index

This is longish and performs the following task. For a set of alpha (list), run the elastic net analysis for a given **l1_ratio** with **kfolds**. For each alpha, get the CIndex and find the CINdex for which the errors are minimum.

Warning: this is a bit longish (300 seconds for 10 folds and 80 alphas) on GDSCv5 data set.

Ridge

class `GDSCRidge`(*ic50*, *genomic_features=None*, *verbose=False*)
Same as `Regression`

`get_model`(*alpha=1*, *l1_ratio=None*, ***kargs*)

Report

Code related to the Regression analysis to find associations between drug IC50s and genomic features

class `RegressionReport`(*method*, *directory='.'*, *verbose=True*, *image_dir='images'*, *data_dir='data'*, *config={'boxplot_n': '?'}*)
Class used to interpret the results and create final HTML report

Constructor

Parameters

- **method** – Method used in the regression analysis (lasso, elasticnet, ridge)
- **results** –

`create_html_drug`()
report for each individual drug

`create_html_main`(*onweb=False*)
Create HTML main document (summary)

Data Packages

class `IC50Cluster`(*ic50*, *ratio_threshold=10*, *verbose=True*, *cluster=True*)
Utility to cluster columns that correspond to the same drug ID

From GDSC v18 data sets onwards, DRUG identifiers may be duplicated to account for different drug concentration. This is not recommended since we'd rather use unique identifier for different experiments but to account for this feature, the IC50Cluster will rename them columns and transforming the data as follows.

Consider the case of the DRUG 1211. It appears 3 times in the original data:

```
Drug_1211_0.15625_IC50
Drug_1211_1_IC50
Drug_1211_10_IC50
```

Actually, there are about 15 such cases even though in general there are only 2 duplicates:

```
      DRUG_ID   1   2    3  total  icommon       ratio
21      1782  47  47  NaN     47       47  100.000000
20      1510  45  45  NaN     45       45  100.000000
19      1211  48  47  4.0     50       46   92.000000
18      1208  48  47  NaN     50       45   90.000000
16      1032  43  47  NaN     50       40   80.000000
17      1207  38  47  NaN     50       35   70.000000
13       231   2  39  NaN     39        2    5.128205
14       232  45   2  NaN     45        2    4.444444
10       226   2  45  NaN     45        2    4.444444
12       230   2  46  NaN     46        2    4.347826
15       238   2  46  NaN     46        2    4.347826
0        206   2  46  NaN     46        2    4.347826
1        211  46   2  NaN     46        2    4.347826
9        224   2  46  NaN     46        2    4.347826
8        223   2  46  NaN     46        2    4.347826
7        221   2  46  NaN     46        2    4.347826
6        217   2  46  NaN     46        2    4.347826
5        216   2  46  NaN     46        2    4.347826
4        215   2  46  NaN     46        2    4.347826
3        214   2  46  NaN     46        2    4.347826
2        213   2  46  NaN     46        2    4.347826
11       229   2  46  NaN     46        2    4.347826
```

The clustering works as follows. If the ratio of drugs in common between several concentrations is large, then they are studied independently. Otherwise they are merged.

In the final dataframe, the columns names are transformed into unique identifiers like in the IC50 class by removing the `Drug_` prefix and `''_conc_IC50` suffix.

The `mapping` contains the mapping between new and old identifiers.

See also:

`cleanup()` method.

constructor

Parameters

- **ic50** –
- **ratio_threshold** (*int*) –
- **verbose** (*bool*) –
- **cluster** (*bool*) – may be useful to not cluster the data for testing or debugging

cleanup(*offset=10000*)

Rename the columns into unique identifiers

Parameters **offset** (*int*) – if duplicated, add the offset

The `mapping` contains the mapping, which should be used to update the decoder file.

cluster()

duplicated

to_cluster

class GDSC(*ic50*, *drug_decode*, *genomic_feature_pattern='GF_*csv'*, *main_directory='tissue_packages'*, *verbose=True*)

Wrapper of the ANOVA class and reports to analyse all TCGA Tissues and companies automatically while creating summary HTML pages.

First, one need to provide an unique IC50 file. Second, the DrugDecode file (see *DrugDecode*) must be provided with the DRUG identifiers and their corresponding names. Third, a set of genomic feature files must be provided for each *TCGA* tissue.

You then create a GDSC instance:

```
from gdsctools import GDSC
gg = GDSC('IC50_v18.csv', 'DRUG_DECODE.txt',
    genomic_feature_pattern='GF*csv')
```

At that stage you may want to change the settings, e.g:

```
gg.settings.FDR_threshold = 20
```

Then run the analysis:

```
gg.analysis()
```

This will launch an ANOVA analysis for each TCGA tissue + PANCAN case if provided. This will also create a data package for each tissue. The data packages are stored in ./tissue_packages directory.

Since all private and public drugs are stored together, the next step is to create data packages for each company:

```
gg.create_data_packages_for_companies()
```

you may select a specific one if you wish:

```
gg.create_data_packages_for_companies(['AZ'])
```

Finally, create some summary pages:

```
gg.create_summary_pages()
```

You entry point is an HTML file called **index.html**

Constructor

Parameters

- **ic50** – an *IC50* file.
- **drug_decode** – an *DrugDecode* file.
- **genomic_feature_pattern** – a glob to a set of GenomicFeature files.

analyse(*multicore=None*)

Launch ANOVA analysis and creating data package for each tissue.

Parameters

- **onweb** (*bool*) – By default, reports are created but HTML pages not shown. Set to True if you wish to open the HTML pages.
- **multicore** – number of cpu to use (1 by default)

companies

create_data_packages_for_companies(*companies=None*)
Creates a data package for each company found in the DrugDecode file

create_summary_pages()
Create summary pages

Once the main analyis is done (*analyse()*), and the company packages have been created (*create_data_packages_for_companies()*), you can run this method that will creade a summary HTML page (index.html) for the tissue, and a similar summary HTML page for the tissues of each company. Finally, an HTML summary page for the companies is also created.

The final tree direcorty looks like:

```
|-- index.html
|-- company_packages
|   |-- index.html
|   |-- Company1
|   |   |-- Tissue1
|   |   |-- Tissue2
|   |   |-- index.html
|   |-- Company2
|   |   |-- Tissue1
|   |   |-- Tissue2
|   |   |-- index.html
|-- tissue_packages
|   |-- index.html
|   |-- Tissue1
|   |-- Tissue2
```

tcga

MISC

Sets of miscellaneous tools

class Savefig(*verbose=False*)
A simple class to save matploltib figures in the proper place

Note: For developers only

directory
directory where to save figures

savefig(*name*, *size_inches=None*, ***kargs*)
Save a matplotlib figure

Parameters

- **filename** (*str*) – where to save the figure.
- ****kargs** – accepts all parameters known by pylab.savefig

get_drug_id(*drug_lists*)
Deprecated since version 0.12: Used in IC50Cluster (version 18) for now but should be removed

Code related to the ANOVA analysis to find associations between drug IC50s and genomic features

class ANOVASettings(***kargs*)
All settings used in *gdsctools.anova.ANOVA* analysis

This class behaves as a dictionary but values for a given key (setting) can be accessed and changed easily like an attribute:

```
>>> from gdsctools import ANOVASettings
>>> s = ANOVASettings()
>>> s.FDR_threshold
25
>>> s.FDR_threshold = 20
```

It is the responsability of the users or developers to check the validity of the settings by calling the *check()* method. Note, however, that this method does not perform exhaustive checks.

Finally, the method *to_html()* creates an HTML table that can be added to an HTML report.

Note: **for developers** a key can be changed or accessed to as if it was an attribute. This prevents some functionalities (such as copy() or property) to be used normaly hence the creation of the *check()* method to check validity of the values rather than using properties.

Here are the current values available.

Name	Default	Description
include_MSI_factor	True	Include MSI in the regression
feature_factor_threshold	3	Discard association where a genomic feature has less than 3 positives or 3 negatives values (e.g., 0, 1 or 2)
MSI_factor_threshold	2	Discard association where a MSI count has less than 2 positives or 2 negatives values (e.g., 0, or 1).
analysis_type	PANCAN	Type of analysis. PANCAN means use all data. Otherwise, you must provide a valid tissue name to be found in the Genomic Feature data set.
pvalue_correction_method	fdr	Type of p-values correction method used. Could be *fdr*, *qvalue* or one accepted by *MultipleTesting*
pvalue_correction_level	global	Apply pvalue correction globally. Can also be set to 'drug_level' to apply corrections at drug level only.
equal_var_ttest	True	Assume equal variance in the t-test
minimum_nonna_ic50	6	Minimum number of IC50 required to perform an analysis for a given drug.
fontsize	25	Used in some plots for labels
FDR_threshold	25	FDR threshold used in volcano plot and significant hits
pvalue_threshold	0.001	Used to select significant hits see ANOVAReport
directory	html_gdsc_anova	Directory where images and HTML documents are saved.
savefig	False	Save the figure or not (PNG format)
effect_threshold	0	Used in the volcano plot. See VolcanoPlot

There are parameters dedicated to the regression method. Note that only regression_formula is effective right now.

Name	Default	Description
regression_method	OLS	Regression method amongst OLS. NOT USED YET.
regression_alpha	0.01	Fraction of penalty included. If 0, equivalent to OLS. NOT USED YET.
regression_L1_wt	0.5	Fraction of the penalty given to L1 penalty term. Must be between 0 and 1. If 0, equivalent to Ridge. If 1 equivalent to Lasso. NOT USED YET.
regression_formula	auto	if auto, use standard regression from GDSCTools (see link_formula) otherwise any valid regression formula can be used.

See also:

About the settings or gdsctools.readthedocs.org/en/master/settings.html#filtering decrease the number of significant hits.

check()
Checks the values of the parameters

This may not be exhaustive. Right now, checks

- MSI factor is boolean.
- Regression.method in OLS/Ridge/Lasso/ElasticNet
- FDR thresohld in [0,1]
- pvalues_threshold in [0,inf[
- effect_threshold in [0,inf[
- pvalue_correction_method
- etc

to_html()
Convert the sets of parameters into a nice HTML table

Small functionalities to retrive chembl/chemspider identifiers based on a drug name

class ChemSpiderSearch(*drug_decode*)
This class uses ChemSpider and ChEMBL to identify drug name

Warning: this is a draft version in dev mode

```
c = ChemSpiderSearch()
c.search_in_chemspider()
c.search_from_smile_inchembl()
df = c.find_chembl_ids()
```

It happens that most of public names can be found and almost none of non-public are found. As expected...

If chemspider, chembl and pubchem are empty, search for the drug name in chemspider.

CHEMSPIDER search: if no identifier found, the search if DROPPED if 1 identifier found, we keep going using the SMILE identifier If more than 1 identifier found, this is AMBIGUOUS.

If chembl and pubchem, check with unichem If chembl, check smiles If chembl and chemspider, check smiles ?

SMILES are not unique

filling_chembl_pubchem_using_unichem()

find_chembl_ids()

get_chemspider_ids(*drug_name*)

search_from_smile_inchembl()

search_in_chemspider()

Code related to the ANOVA analysis to find associations between drug IC50s and genomic features

class BaseModels(*ic50*, *genomic_features=None*, *drug_decode=None*, *verbose=True*, *set_media_factor=False*)
A Base class for ANOVA / ElaticNet models

Constructor

Parameters

- **IC50** (*DataFrame*) – a dataframe with the IC50. Rows should be the COSMIC identifiers and columns should be the Drug names (or identifiers)
- **features** – another dataframe with rows as in the IC50 matrix and columns as features. The first 3 columns must be named specifically to hold tissues, MSI (see format).
- **drug_decode** – a 3 column CSV file with drug's name and targets see `readers` for more information.
- **verbose** – verbosity in "WARNING", "ERROR", "DEBUG", "INFO"

The attribute *settings* contains specific settings related to the analysis or visulation.

cosmicIds
get/set the cosmic identifiers in the IC50 and feature matrices

diagnostics(*details=False*)
Return dataframe with information about the analysis

drugIds
Get/Set drug identifers

feature_names
Get/Set feature names

init()

read_drug_decode(*filename=None*)
Read file with the DRUG information

See also:

gdsctools.readers.DrugDecode

read_settings(*settings*)
Read settings and update cancer type if set

set_cancer_type(*ctype=None*)
Select only a set of tissues.

Input IC50 may be PANCAN (several cancer tissues). This function can be used to select a subset of tissues. This function changes the `ic50` dataframe and possibly the feature as well if some are not relevant anymore (sum of the column is zero for instance).

settings = None
an instance of *ANOVASettings*

exception GDSCToolsDuplicatedDrugError(*drug_id*)

exception `GDSCToolsError`(*value*)

For developers

Contents

The GDSCTools test suite

The GDSCTools package has a large set of tests in the ./test/gdsctools directory. In version 0.10, about 80% of the functionalities are covered. If you add a new module or function, please add a corresponding test file in ./test/gdsctools directory.

The test suite uses the **nosetests** utility and is integrated within the Travis continuous integration (see below).

To run the tests locally, you will need nose and coverage packages:

```
pip install nose coverage
```

Then, in the directory where is the **setup.py** file, type

```
python setup.py nosetests
```

You may also go directly in the ./test/gdsctools directory but some tests may required extra files or permission.

Documentation

The documentation is based on Sphinx. This means that all code is documented using the REST syntax. Docstring are added in classes and functions as much as possible with code examples.

See for example the file in gdsctools/anova.py and in particular the class ANOVA.

In addition, in the ./doc directory there are a set of files encoded using the REST syntax. If you go to that directory and type:

```
make html
```

then the entire documentation included Tutorial and Developer guide will be parsed and interpreted. The resulting html documentation can then be found in doc/build/html. For the Sphinx documentation to be generated, the **GDSCTools** package must be installed.

Note: the command above will work only if GDSCTools source code is available and installed in the shell where the command is executed.

The documentation can then be uploaded on pypi. Go to the directory were is the file **setup.py** and type:

```
python setup.py upload_docs
```

Continuous Integration

This is based on Travis. The reports should be available here: https://travis-ci.org/CancerRxGene/gdsctools and the status is also reported in the main github page (https://github.com/CancerRxGene/gdsctools) as an icon (**build**) that will be green or red depending on the status of the build within Travis.

The required files is in the main directory and called **.travis.yml**

ReadTheDocs

The doc is built on read the docs. The required files are **readthedocs.yml** and **requirements.txt**.

CHAPTER 2

Issues

Please fill bug report in https://github.com/CancerRxGene/gdsctools/issues

CHAPTER 3

Contributions

Please join https://github.com/CancerRxGene/gdsctools

ChangeLog

Contents

Version 0.19.0

- CHANGES:
  - Update GDSCTools to use latest Pandas version (0.20); in particular the .ix getter is deprecated and should be replaced by loc/ilc
  - pin version of pandas to 0.20 and colormap to 1.0.1 in the setup

Version 0.18.0

- BUG:
  - FDR not taken into account in the volcano as shown in https://github.com/CancerRxGene/gdsctools/issues/168
- CHANGES:
  - move from nosetests to pytest and fix tests accordingly
  - Fix gdsc1000 module with new methods and documentation

Version 0.17.1

- NEWS:
  - add test for module gdsc1000, which has been refactored and cleanup
- BUGS:
  - Fix GDSC links to fetch drug/genomic feature data sets
  - Fix bug in regression.rules pipeline (create missing directory)

Version 0.17

summary Fixing bunch of deprecated warnings, working regression pipeline based on snakemake

- CHANGES and BUG Fixes:
  - regression (snakemake pipeline)
  - add missing init and regression modules
  - starting to use colorlog
- NEWS:
  - add regression script to create the snakefile workflow easily

Version 0.16.4

- NEWS:
  - a snakemake pipeline for the regression analysis
  - Report for the regression analysis

Version 0.16.3

- BUG Fix: #158 infinity are removed in ANOVAReport in the constructor

Version 0.16.2

- BUG Fix:
  - Fix volcano plot and anova results (avoid infinite values)

Version 0.16.1

- BUG Fixes
  - Fix regression dendogram plot
  - Fix bug leading to NA in effect size reported by Carlos P. (private communication)"
- CHANGES:
  - Omnibem: drops the NA instead of replacing by ""

Version 0.16 (Nov 2016)

- BUG Fixes
  - Fix pending issue related to #153 ic_input.csv", "test_gf_input.csv" for the case where some media factor are missing. This is now handle properly.
  - Fix #151 : large integer are not cast properly with consequence that indices are strings, not integers leading to further issues in the HTML pages.
- NEWS:
  - anova_one_drug_one_feature_custom allows to perform any regression using formula like in R. This is not for production but should be useful to perform custom analysis
  - Include the MEDIA factor boxplot in the library and reports

Version 0.15 (Nov 2016)

- BUG Fixes
  - Fix issue #156 (GDSC failures in some cases). This was due to special MOBEM input files for which no tests are performed. In such cases, some codes were failing in ANOVAReport and ANOVAResults, which have been fixed in this release.
  - Fix buggy volcano plot (no plot if no data); if FDR threshold below minimum value, set fdr_threshold to minimum value so that it scaled the plots properly. Now, users can add as many lines as desired using settings.additional_fdr. pvalues found to be NAN are set to 0 to prevent plotting issues.
  - **anova module: regression in the case Y ~ C(TISSUE) + C(MSI) + feature,** the tissue sum of squares was using N-1 tissues (one missing).
  - **anova module: regression in the case Y ~ C(TISSUE) + C(MEDIA) + C(MSI) + feature,** the media sum of squares was not normalised properly.
- CHANGES:
  - elastic_net module renamed into regression
  - ANOVASetting prints keys in alphabetical order (instead of randomly)
  - ANOVASetting: regression_formula is added; other regression_XX settings are not removed but not used anymore.
  - GenomicFeature reader: if a tissue is empty, it is replaced by UNDEFINED.
  - standalone: the drug option must be an integer. THis is now caught are the option level, not later in the code.

  - anova module: remove code related to elastic net, ridge, and lasso. This won't be used in production with ANOVA. EN, Ridge and Lasso are used in the regression module and will be part of an independent type of analysis. See NEWS

- NEWS:
  - Add Ridge + Lasso + LassoLars classes in addition to ElasticNet regression method into the regression module.
  - Add more features in regression module (boxplot, dendogram)
  - New regression notebook in the notebooks directory
  - anova module: We can now use any combo of regression formula using statsmodels. This is slower but one can do use any formula accepted by statsmodels. The previous faster code is still used for the standard analysis.

Version 0.14 (20th June 2016)

- NEWS:
  - ElasticNet: new method elastic_all()
  - plot_elastic_weight in the gallery
- CHANGES:
  - ElasticNet plot_weights is now split into plot_weights and plot_importance.
- BUGS:
  - Fixes missing files in the pypi distributino (MANIFEST changed)

Version 0.13 (27th May 2016)

0.13.1

- CHANGES:
  - **DrugDecode**: In brief, the DRUG ID in the IC50 input file and the DrugDecode files should be integers. Some old data sets use the following convention to refer to a drug Drug_<ID>_IC50 and so DrugDecode was using the same convention. However, we now convert this type of identifier into integers. This is done internally for the IC50 file, however, was not done inside the DrugDecoder file. This is now effective.
  - **HTML reports when using the GDSC class:**
    - Company names now appear systematically in the top of the company data packages.
    - Drug Names were missing and do now appear in top of the relevant HTML pages.
  - Boxplots: If a DrugDecode file is provided Boxplots show the DRUG ID and the real drug name in the matplotlib and JS boxplots

0.13.0

- CHANGES:
  - Reader class simplification and improvments: files can now be compressed using gzip but also xz, zip and bz2 formats. The NA can be encoded as NA or NaN strings. Spaces are interpreted as NA.

  - Sort DrugDecode's dataframe columns
  - Updated all documentation
- BUG:
  - Fix scaling of the data with newest version of scikit-learn
  - fix typo in the setup.py file. Passed travis + all tests before main release.

Version 0.12 (9th May 2016)

0.12.1

- BUG:
  - add missing CSS in the distribution

0.12

- CHANGES:
  - **SPEEDUP:**
    - tissue specific analysis computational time decreased by 50% by dropping the creation of dataframe and using a simple numpy array inside ANOVA.anova_one_drug_one_feature
    - Creation of volcano plots uses pure javascript for the data packages and the creation of the volcano plots was dramatically sped up by a factor between 10 and 100e. One can still create volcano plot manually in pure matplotlib.
    - Similarly, boxplots for tissue, MSI and all associations are now created using JS.
  - Data packages have been refactored. The major difference concerns the HTML layout (most HTML files are now in the sub-directory called associations) so that is it cleaner at the top level. The volcano plots are not in PNG format anymore but pure HTML/JS, which can be exported manually. The consequences is that the creation of data packages is 10 times faster.
  - The standalone application had 2 options removed: –feature (alone) and –fast options
  - Drug Identifier are now handled as pure integer. For back compatibility, old files that mix up IC50 and Genomic Features (e.g. v17 data) are still interpreted; the DRUG ID in that case are written as Drug_ID_IC50 and are transformed as just <ID> everywhere.
  - associations output were named 1.html, 2.html... and are now named a1.html, a2.html...
  - Because DRUG_ID are now integer and all HTML stored in the same directory the naming of the HTML files have been altered (e.g., associations starts
  - Report now accepts only one argument (the anova isntance). Second argument (results) is now optional. If not provided, ANOVA are computed on the fly
  - Multicore module removed but ANOVA.anova_all has multicore option. This seems to work on Linux systems. Not tested on windows or MacOsX
  - IC50 may have duplicated drug ids (at different concentrations). Not good practice but that the format of e.g. v18, v19 IC50 files. A class IC50Cluster was created to interepret those files. ANOVA will switch to IC50Cluster automatically if there are duplicated files.
  - Settings: low_memory option has been removed

Version 0.11 (April-May 2016)

0.11.3

- CHANGES:
  - The parameter **pvalue_threshold** in the general settings was changed from infinite to 10e-3. This has an effect on the numlber of significant hits reported in the HTML reports and volvano plots. This should not have a strong impact on the number of hits but guarantees a reasonably low pvalue before multiple testing
  - If an input file named with .csv extension but the content is tabulated, there was no immediate error but lead to errors later (e.g. in ANOVA), which is difficult to debug. Now, in such cases, an error will occur immediately when reading the file.
  - The warnings about MEDIA factor is removed since most of the files do not contain that column.
- BUG
  - The data packages were stored in the "ALL" directory, which may be a TCGA tissue by itself. This has been renamed into "tissue_packages".

0.11.2

- BUG:
  - add missing file in the setup.py

0.11.1

- BUG:
  - Fixes the missing data package in the setup for pip installation

0.11.0

- NEWS:
  - Elastic notebook and module implemented
  - GenomicFeatures has now a compression method
- CHANGES:
  - anova module was split into modules + anova so that elastic_net module can inherit from module
  - all share/data moved to gdsctools data
  - add scikit-learn dependencies
- BUGS:
  - Fix onevent picking in the volcano plot and use 4 digit for the FDR plot

Version 0.10

0.10.2

- BUG:
  - Fixes issue #127 (If MSI factor missing, the anova still tries to use it)
  - Fixes issue #126 (–out-directory ignored in gdsctools-anova pipeline)
  - Fixes issue #125 and #124 (HTML report links broken)

0.10.1

- BUG:
  - Fix set_cancer_type to accept lists of tissues again
- CHANGES:
  - Fixes #119 by adding more tests.
  - reactivate get_significant hits functions.
  - rename ANOVAResults.get_significant_hits into get_html_table

0.10

Lots of changes in this version but for users the API should be very similar.

- NEWS:
  - Add a new factor called MEDIA_FACTOR. If not provided, genomic feature matrix can populated the MEDIA_FACTOR column automatically.
  - add a class COSMICInfo and a related data file called cosmic_info.csv.gz to get information about COSMIC ids. Replaces COSMIC class, which was removed.
  - add new class GDSC to perform the entire analysis splitting data across companies found in DrugDecode and across cancer types.
- CHANGES:
  - COSMIC class removed and replaced by COSMICInfo class
  - **Column name convention:**
    - FEATURE_ANOVA_pval –> ANOVA_FEATURE_pval
    - MSI_ANOVA_pval –> ANOVA_MSI_pval
    - TISSUE_ANOVA_pval –> ANOVA_TISSUE_pval
    - FEATURE_ANOVA_FDR_% –> ANOVA_FEATURE_FDR
    - new column named ANOVA_MEDIA_pval
    - to be constistent, names such as FEATURE_pos have now underscores to separate words e.g., (FEATUREpos –> FEATURE_pos, FEATUREneg –> FEATURE_neg, deltaMEAN –> delta_MEAN).
  - refactor *gdsctools.volcano* module to use new naming convention.

  - SAMPLE_NAME is not required anymore in the genomic features. This is indeed just an annotation and is now encoded in the flat file cosmic_info.csv.gz (see above)
  - ***anova*, anova_results modules:**
    * Implement new factor (MEDIA) in the regression
    * Uses new naming convention for the columns as described above
    * When initialising a ANOVA instance, prints the factor that will be included.
    * add new option (set_media_factor) to populate the MEDIA column automatically
  - ***readers* module:**
    * 'Sample Name' or SAMPLE_NAME are deprecated. There are removed from the genomic_feature matrix if found.
  - Uses MEDIA_FACTOR column in addition to MSI and tissue columns
  - shift attribute is now read-only and set automatically
  - add a function to fill media column automatically
  - print function is more verbose
  - volcano: uses new naming convention for the columns as described above.
  - split *anova* module (create *anova_report*) (issue #98).
  - *readers*: improved DrugDecoder and renamed into DrugDecode (issue #102 and #101)
  - add new settings and code to apply pvalue correction at drug level rather than global level.
  - add new module to find chemblId/ChemSpider from drug name.

Version 0.9

0.9.10

- NEW:
  - add settings as json file in the HTML report
  - ANOVAResults has now a volcano() method
  - add read_settings method in ANOVA
  - add code in the HTML tree directory to reproduce HTML report and results
- CHANGES:
  - anova_one_drug now returns an ANOVAResults object
  - Restructure data package tree directory (#83)
  - **Default header have changed:**
    * COSMIC ID –> COSMID_ID
    * Sample Name –> SAMPLE_NAME
    * MS-instability Factor Value –> MSI_FACTOR
    * Tissue Factor Value –> TISSUE_FACTOR

    Previous values will still be accepted but deprecation warning added.

- BUGS:
  - Fixes #89 (tight layout buggy under MAC)

0.9.9

- CHANGES:
  - add new regression method: Ridge/Lasso/ElasticNet in *gdsctools.anova.ANOVA*
  - Rename some of the settings to have a more uniform naming convention in *gdsctools.settings.ANOVASettings*
  - Add new module related to fitting ot logistic function parameters (*gdsctools.logistics*)

0.9.8

- BUG:
  - javascript were not included in version 0.9.7 had to rename js directory into javascript to avoid known bug in distutils. Maybe solved in the future but for bow just renamed the directory.

0.9.7

- MSI/Sample/Tissue columns in the genomic features are not required anymore.
- FDR lines in volcano plots are now using interpolation and therefore more precisily placed. Fixes #57
- volcano plot improvments. Fixes #79, #80, #81
- Fixes issue #72 to get the drug_decoder information from the ANOVA class.
- Fixes issue #76 to drop IC50 cosmic Id not found in the genomic feature matrix
- Readers (e.g. IC50) can now read CSV files with commented lines (# character) issue #78
- Readers can now ignored columns that are not named (usually first column of index exported by excel document)
- IC reader figure out automatically if the prefix "Drug" has been used. It so, it drops other irrelevant columns. Useful if genomic features and IC50 are mixed together.
- IC50 and GenomicFeatures, DrugDecode now accepts both TSV and CSV format (gziped or not)
- add more datasets for testing purposes
- double checked results on BLCA tissue v17 and v18
- Finalise a first version of the standalone application
- ReadTheDocs documentation is now on line gdsctools.readthedocs.org
- GDSCTools has now all features of the original R version
- With in addition: - a standalone application - test suite - documentation
- benchmarking for the analysis in about 20 minutes 265 drugs and 680 features across 980 cell lines. HTML report takes as much time.

Version 0.3

- Cancer specific now included and tested on BRCA and BLCA cases.

Version 0.2

First working version with HTML output.

Version 0.1

First working version of gdsctools with test and documenation. Tested against version17. A standalone app is also provide as a command line argument (named **gdsctools_anova**).

FAQS

Contents

Installation

The *Installation* should guide you to install **GDSCTools** but would you have any trouble, please let us know and send a ticket on gdsctools github.

Usage

Any trouble using **gdsctools_anova** executable, please let us know: create a ticket on gdsctools github.

I cannot see any plots or figures

First solution is to import the show function from pylab:

```
from pylab import show
show()
```

The figures should appear now but you will have to close them to get back to the shell. To get a better interaction, exit from the shell and start a new one typing:

```
ipython --pylab
```

Under MAC, the default backend for pylab may cause some trouble with this message:

```
RuntimeError: CGContextRef is NULL
```

If so, try another backend:

```
ipython --pylab=qt
```

About Python

Shall I use Python 2 or Python 3

GDSCTools is currently compatible for the version 2.7 but also 3.3, 3.4 and 3.5. We do not have any strong recommendation except to use whatever version is already installed on your system. We will not maintain the code for version below 2.7 although the code may work for those versions.

How do I get documentation on Python?

The standard documentation for the current stable version of Python is available at https://docs.python.org/3/. PDF, plain text, and downloadable HTML versions are also available at https://docs.python.org/3/download.html.

I've never used Python before. Is there a Python tutorial?

There are numerous tutorials and books available. The standard documentation includes The Python Tutorial.

Glossary

ADAF acronym for All Drug All Feature mode

CLI A Command Line Interface (CLI) is an interface where the user types a command (text) and presses the return key to execute that command.

COSMIC COSMIC is a global resource for information on somatic mutations in human cancer, combining curation of the scientific literature with tumor resequencing data from the Cancer Genome Project at the Sanger Institute, U.K. See the COSMIC website or pubmed reference for more details

CSV acronym for Comma Separated Values, a file format. Extension must be *.csv*.

EC50 The EC50 is the half maximal Effective Concentration and refers to the concentration of a drug, antibody or toxicant which induces a response halfway between the baseline and maximum after a specified amount of time. It is used as a measure of drug's potency

FDR In genome research, it is common to examine a large number of features. When testing multiple hypotheses, one must guard against an abundance of false positive results. A criterion for error control is the false discovery rate (FDR), which is the expected proportion of falsely rejected hypotheses. This error rate is equal to FWER when all null hypotheses are true but is smaller otherwise. Benjamini and Hochberg, proposed a step-down procedure to control FDR for independent test statistics. This method is currently recommended in **GDSCTools** and is the default method gor multiple testing correction. See e.g., article

IC50 The IC50 is the half maximal Inhibitory Concentration that is a measure of effectiveness of a drug or substance in inhibiting a specific biological or biochemical function.

Sometimes, IC50 values are converted to the pIC50 scale that is $-\log 10(IC50)$.

pIC50 is usually given in terms of molar concentration (mol/L or M). Therefore to obtain pIC50, an IC50 should be specified in units of M. When IC50 is expressed in microM or nanoM, it will need to be converted to M before conversion to pIC50.

IPython Python comes with a dedicated shell, which can be started in a terminal by typing **python**. However, an improved shell is provided with the IPython environment, which should be installed independently. Note that conda distribution usually comes with ipython already installed.

MEDIA A factor used in the regression analysis. Stands for Growth MEDIA. MEDIA_FACTOR ir one of the possible column filled in Genomic Feature.

MoBEM Data structure used in GDSC to store genomic information. This is a CSV file with specific header: (SAMPLE, TISSUE, TYPE, GENE)

MSI Microsatellite Instability (MSI) are markers indicating the presence or absence of a MSI shift, allele homozygosity/heterozygosity, and loss of heterozygosity (LOH) observed in the tumor sample for each participant

ODAF acronym for One Drug All Feature mode

ODOF acronym for One Drug One Feature mode

OLS An ordinary least squares (OLS) or linear least squares is a method for estimating the unknown parameters in a linear regression model, with the goal of minimizing the differences between the observed responses in some arbitrary dataset and the responses predicted by the linear approximation of the data

PANCAN alias for set of cancer tissues (unlike cancer-specific tissue).

shell A shell is a program that provides the traditional, text-only user interface for Linux and other Unix-like operating systems. It is a specialised *CLI* that is a command-line shell (e.g., bash) where users can execute programs.

TCGA The Cancer Genome Atlas (TCGA) is a project to catalogue genetic mutations responsible for cancer, using genome sequencing and bioinformatics. See TCGA homepage

Terminal Under Unix-like operating systems, a terminal is a program that run a shell. A unix terminal (Linux or Mac) starts with a specialised shell (e.g. bash shell). Under Windows, the "command prompt" available in All Programs -> Accessories is an entry point to a terminal for typing computer commands

TSV acronym for Tabular Separated Values, a file format. Extension must be *.tsv*.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

E

F

G

T

U

V

W

X

Y