Supplementary Materials to: GDSCTools for Mining Pharmacogenomic Interactions in Cancer

Thomas Cokelaer,1,∗, Elisabeth Chen 2, Francesco Iorio 3, Michael P. Menden 4, Howard Lightfoot 2, Julio Saez-Rodriguez 3,5,∗,1, Mathew J. Garnett 2,∗,1

1 Institut Pasteur – Bioinformatics and Biostatistics Hub – C3BI, USR 3756 IP CNRS – Paris, France
2 Welcome Trust Sanger Institute, Welcome Trust Genome Campus, Cambridge, UK
3 European Molecular Biology Laboratory, European Bioinformatics Institute (EMBL-EBI), Welcome Trust Genome Campus, Cambridge, UK
4 Oncology Innovative Medicines and early Development, AstraZeneca, Cambridge, UK
5 RWTH Aachen University, Faculty of Medicine, Joint Research Centre for Computational Biomedicine, Aachen, Germany

1 Supplementary

1.1 Code and Installation

GDSCTools source code is available on GitHub website at http://www.github.com/cancerXgene/gdsctools and a pre-compiled version is available on Bioconda channel. It has an extended documentation hosted on http://gdsctools.readthedocs.io. We have also included a large set of functional tests to assess results’ reproducibility. Changes made to the code are tested automatically via the Travis continuous integration framework so that changes that affect the analysis or normal behaviour may be caught early. Finally, GDSCTools makes use of existing open source libraries such as scikit-learn (Pedregosa et al., 2011) for the machine learning tools, and statsmodels (Seabold et al., 2010) for advanced statistical analysis.

GDSCTools can be installed from the source code. However, we recommend using the Anaconda framework. Information can be found on https://www.continuum.io/downloads. Once the software is installed, an executable called conda provides pre-compiled versions of many scientific libraries. In addition, GDSCTools itself is exposed on one of the Anaconda channel called Bioconda. Consequently, having Anaconda pre-installed makes the installation of GDSCTools easier and quicker. The commands needed to select the Anaconda channel to be used (once for all) are the following ones:

conda config --add channels r
conda config --add channels defaults
conda config --add channels conda-forge
conda config --add channels bioconda

Then, GDSCTools can be installed as follows:

conda install gdsctools

This will take care of all dependencies required by GDSCTools. Further details can be found in the http://gdsctools.readthedocs.io website (installation section).

1.2 Data

1.2.1 IC\textsubscript{50} indicators

The first data object that GDSCTools uses by default should contain the IC\textsubscript{50} indicators, summarising the effect of drug treatment across a large collection of cell lines using experimental protocols detailed in (Garnett et al., 2012; Iorio et al., 2016). These indicators were derived by applying a curve-fitting algorithm to raw cell counts data, via a multilevel mixed model (Vis et al., 2016). The IC\textsubscript{50} data structure, in keeping with GDSCT reporting standards, stores micromolar IC\textsubscript{50} values in natural log scale. Users data will need to log transform their own data if appropriate. These (or any other user defined drug response indicators) must be stored in a CSV file, which can be optionally compressed (gzip format). In this file, the header must contain an entry named COSMIC_ID: this column will contain the COSMIC identifiers of the cell lines, one per line. The following entries should contain drug identifiers (one integer per column).

The order of the columns is not relevant. Each row should contain IC\textsubscript{50}s for a given cell line (identified through its COSMIC_ID), across all the tested drugs. Here is an example of the data format for 2 cell lines and 3 drugs

<table>
<thead>
<tr>
<th>COSMIC_ID</th>
<th>IC\textsubscript{50}</th>
<th>IC\textsubscript{50}</th>
<th>IC\textsubscript{50}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20.1</td>
<td>40.0</td>
<td>20.0</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
<td>0.8</td>
<td>10.0</td>
</tr>
<tr>
<td>3</td>
<td>222222</td>
<td>1.2</td>
<td>10.0</td>
</tr>
</tbody>
</table>

Further details can be found within the GDSCTools on-line documentation http://gdsctools.readthedocs.io/en/latest/data.html. Worthy of note is that in this data object, IC\textsubscript{50}s can be replaced by any kind of scalar data (e.g., AUCs). To read the IC\textsubscript{50}s file shown above, the following commands should be used (assuming that the object is saved into a file named is\textsubscript{50}.csv):

```python
from gdsctools import *

ic50 = IC50("is50.csv")
```

This allows the data to be accessed as a DataFrame and used with various descriptive statistics and plotting functions.

1.2.2 Genomic features

The second data set required by GDSCTools is the Genomic Features data set. All the implemented analyses are performed on the cell lines included in both the IC\textsubscript{50}s and the Genomic Features data object and this intersection is determined at the level of COSMIC identifiers. As a consequence, cell lines that are not included in both matrices will be discarded. In addition to the COSMIC identifiers, the Genomic Features file should contain the following columns: TISSUE_FACTOR, MSI_FACTOR and MEDIA_FACTOR that can be used in the ANOVA or linear regression models as explained hereafter. All remaining columns should refer to individual genomic features, whose status (positive or negative) in a generic i-th cell line should be indicated with a binary entry in the i-th line.

An example is reported below.
Genomic Features

In parallel, a subset of the genomic features and IC\textsubscript{50}s used in (Iorio et al., 2016) are provided inside GDSCTools, which includes IC\textsubscript{50}s of 265 drugs across 988 cell lines. In parallel, a Genomic Features file encompassing the status of 677 genomic features (copy number alteration and cancer gene mutations) on the same set of cell lines is provided. Alternatively, GDSCTools contains built-in functions to retrieve and to analyse more data from the GDSC database. This currently encompasses data sets including 29,214 gene variants, 2,436 copy number variations (deletion and amplification) and 10,581 differentially methylated gene promoters across 1,002 cancer cell lines. For instance, the following code downloads all GDSC data from the database. This currently encompasses data sets including 29,214 gene variants, 2,436 copy number variations (deletion and amplification) and 10,581 differentially methylated gene promoters across 1,002 cancer cell lines. From GDSCTools Python library, which is fully documented on http://gdsctools.readthedocs.io, users can read the IC\textsubscript{50}s and Genomic Features files, perform the analysis and create HTML reports highlighting the identified significant interactions and meaningful models. Here is the code to perform these tasks:

```python
from gdsctools import *
data = GDSC1000()
data/download_data()
data.filter_by_genes("Core Genes")
```

More generally, we provide the OmniBEM Builder module that allows the user to merge different levels of annotations from the GDSC web-portal into a single gene-level view that merges together different types of alterations (for example mutations and copy number amplifications involving the same gene). Users can additionally specify which sets of genomic annotations to integrate as well as upload and integrate their own sets of genomic annotations.

1.3 ANOVA

1.3.1 Details

GDSCTools implements functions to perform a systematic analysis of variance (ANOVA) to identify statistically significant interactions between genomic features and drug responses. To this aim IC\textsubscript{50}s and Genomic Features data object must be created first (as explained in the previous section). The implemented model is fully detailed in (Iorio et al., 2016) and (Garnett et al., 2012). Briefly, for each drug a drug response vector is assembled consisting of n IC\textsubscript{50}s values, derived from treating n cell lines with the drug under consideration, as explained in the previous sections. The implemented model is linear with no interaction terms, dependent variables represented by the described vector and independent factors including tissue type, and screening medium (for the pan-cancer analysis only), microsatellite instability status (for the cancer types with positive samples for this feature) and the status of a genomic feature. For all the tested gene-drug associations, an indication of their effect size is estimated considering the pooled standard deviation of the analysed IC\textsubscript{50}s population (Cohen’s d), or the individual standard deviations (quantified through two different Glass deltas), for the IC\textsubscript{50}s populations of the cell lines that are respectively positive or negative for a given genomic feature. P-values and all other statistical scores are obtained from the fitted models. A genomic-feature/drug pair is tested only if at least n cell lines are contained in the two sets resulting from the dichotomy induced by the status of the considered genomic-feature (for example at least 3 positive cell lines and at least 3 negative cell lines), and n can be defined by the user.

The resulting p-values are corrected (all together those obtained in the pan-cancer analysis and on a cancer type basis those obtained in a given cancer-specific analysis), with a user-chosen method among Bonferroni (Bonferroni, 1953) or Benjamini-Hochberg (Benjamini-Hochberg, 1995).

1.3.2 Examples

From GDSCTools Python library, which is fully documented on http://gdsctools.readthedocs.io, users can read the IC\textsubscript{50}s and Genomic Features files, perform the analysis and create HTML reports highlighting the identified significant interactions and meaningful models. Here is the code to perform these tasks:

```python
from gdsctools import *
anova = ANOVA(ic50_test,
genomic_features_test)
results = anova.anova_all()
results.volcano()
report = ANOVAReport(anova, results)
report.create_html_pages()
```
controls the combination of L1 and L2 penalties. For the Elastic Net model, we fix \( \rho = 0 \) while for the Elastic Net analysis, we fix \( \rho = 1 \).

\[ Y_d = X_w + \epsilon \]

\[ \min_{w} \frac{1}{2N} ||Y_d - Xw||^2_2 + \alpha \rho ||w||_1 + \frac{\alpha(1-\rho)}{2} ||w||^2_2, \]

where \( Y_d \) as defined before contains the IC_{50} for all cell lines for a given drug and \( X \) contains the genomic features for the same cell lines. Here we will use the notations used in the scikit-learn library (Pedregosa et al., 2011). The mixing parameter \( \rho \) (with \( 0 \leq \rho \leq 1 \)) controls the combination of L1 and L2 penalties. For \( \rho = 1 \) the penalty is an L1 penalty (Lasso) while for \( \rho = 0 \) we have an L2 penalty (Ridge). In the Elastic Net analysis, we fix \( \rho = 0.5 \) but it can be changed by the user.

The equation above allows for learning a sparse model where few of the weights are non-zero like Lasso, while still maintaining the regularisation properties of Ridge. Elastic-net is useful when there are multiple features which are correlated with one another. Lasso is likely to pick one of these at random, while elastic-net is likely to pick both.

Before proceeding with an analysis, we need to minimize the function and optimize the alpha parameter. In order to avoid over-fitting, we hold out part of the available data as a test set and perform a cross validation on a training set. When performing a \( k \) folds, we train the model with \( k-1 \) of the training data and the resulting model is validated on the remaining part of the data. The performance measure reported by \( k \)-fold cross validation is the average of the values computed on the \( k \) – 1 models. The metric used to select the best model is the Pearson correlation between predicted and actual drug responses. We scan the range of \( \alpha \) parameter and select the best \( \alpha \). In Fig. 5 we show the Pearson coefficient along the log of an \( \alpha \) parameter.

1.5 Running analysis with Snakemake pipelines

In parallel computing, an embarrassingly parallel problem is one where little or no effort is needed to separate the problem into a number of parallel tasks. In GDSCTools, each drug can be analyzed independently of the others. The analysis is therefore an embarrassingly problem. In GDSCTools, developers can write their own pipelines and run analysis locally, however, we also provide Snakemake pipelines that can be easily run on various clusters (e.g., LSF, SLURM).

1.5.1 Linear models

The initialisation of the pipeline works as follows:

```bash
$ gdsctools_regression -I ic50.csv -F features.csv --method lasso --output-directory analysis
```

This command creates a directory called analysis where a pipeline encoded with the Snakemake framework (Köster and Rahmann, 2012) is copied. The pipeline filename is regression.rules. In addition, a configuration file named config.yaml is also provided. A snapshot of the pipeline workflow is shown in Fig. 4. This is a simple example with 4 drug responses. Of course, real case examples would include hundreds of them.

Each drug is analysed in the same way with a linear model analysis (e.g., Lasso). The results of the analysis as well as images representing the weights are stored in sub directories. Finally, HTML reports are created for each drug and a summary page is also created.
Fig. 5. Tuning of the $\alpha$ parameter of a linear regression model. Using a 10-folds cross validation, for a given drug and a set of genomic features, we scan the $\alpha$ parameter space to obtain the best $\alpha$ that maximizes the Pearson correlation (indicated by the green vertical line).

compare the best model obtained with a null hypothesis (where $Y$ variable is randomized).

Here is the configuration, which can easily be edited and adapted to your needs.

```bash
1 regression:
2 method: lasso
3 kfold: 10
4 randomness: 50
5
6 input:
7 ic50: ic50.csv
8 genomic_features: gf.csv
```

Once the configuration file is available, one can start the analysis as follows. On a local computer (using 4 CPUs):

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

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

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

where --j 40 indicates that we wish to use 40 cores.

References


