Want Drugs? Use Python.

Michał Nowotka*,†, George Papadatos†, Mark Davies†, Nathan Dedman†, Anne Hersey†

Abstract—We describe how Python can be leveraged to streamline the curation, modelling and dissemination of drug discovery data as well as the development of innovative, freely available tools for the related scientific community. We look at various examples, such as chemistry toolkits, machine-learning applications and web frameworks and show how Python can glue it all together to create efficient data science pipelines.

Index Terms— drugs, drug-design, chemistry, cheminformatics, pipeline

1 INTRODUCTION

ChEMBL [ChEMBL12], [ChEMBL14] is a large open access database resource in the field of computational drug discovery, chemoinformatics, medicinal chemistry [MedChem] and chemical biology. Developed by the Chemogenomics team at the European Bioinformatics Institute, the ChEMBL database stores curated two-dimensional chemical structures and standardised quantitative bioactivity data alongside calculated molecular properties. The majority of the ChEMBL data is derived by manual extraction and curation from the primary scientific literature, and therefore covers a significant fraction of the publicly available chemogenomics space.

In this paper, we describe how Python is used by the ChEMBL group, in order to process data and deliver high quality tools and services. In particular, we cover the following topics:

1. Distributing data
2. Performing core cheminformatics operations
3. Rapid data analysis and prototyping
4. Curating data

2 DATA DISTRIBUTION

ChEMBL offers two basic channels to share its contents: SQL dump downloads via FTP and web services. Both channels have different characteristics - data dumps are typically used by organizations ready to host their own private instance of the database. This method requires downloading a SQL dump file and hosting on a machine (physical or virtual). This approach can be expensive, both in terms of time and hardware infrastructure costs. An alternative approach to accessing the ChEMBL data, is to use the dedicated web services. This method, supported with detailed online documentation and examples, can be used by developers, who wish to create simple widgets, web sites, RIAs or mobile applications, that consume chemical and biological data.

The ChEMBL team uses Python to deliver the SQL dumps and web services to end users. In the case of the SQL dumps, the Django ORM (Object Relational Mapping) is employed to export data from a production Oracle database into two other popular formats: MySQL and PostgreSQL. The Django data model, which describes the ChEMBL database schema, is responsible for translating incompatible data types, indicating possible problems with data during the fully automated migration process. After data is populated to separate Oracle, MySQL and PostgreSQL instances, the SQL dumps in the respective dialects are produced.

The Django ORM is also used by the web services [WS15]. This technique simplifies the implementation of data filtering, ordering and pagination by avoiding raw SQL statements inside the code. The entire ChEMBL web services code base is written in Python using the Django framework, Tastypie (used to expose RESTful resources) and Gunicorn (used as an application server). In production, Oracle is used as a database engine and MongoDB for caching results. As a plus, the ORM allows for the same codebase to be used with open source database engines.

Currently, the ChEMBL web services provide 18 distinct resource endpoints, which offer advanced filtering and ordering of the results in JSON, JSONP, XML and YAML formats. The web services also support CORS, which allows them to be accessed via AJAX calls from web pages. There is also an online documentation, that allows users to perform web services calls from a web browser.

The web services codebase is Apache 2.0 licensed and available from GitHub. The code is also registered in the Python Package Index (PyPI), which allows quick deployment by third-party organizations hosting the ChEMBL database.

3 PERFORMING CORE CHEMINFORMATICS OPERATIONS

There are some commonly used algorithms and methods, that are essential in the field of cheminformatics. These include:

1. 2D/3D compound depiction.
2. Finding compounds similar to the given query compound with some similarity threshold.
3. Finding all compounds, that have the given query compound as substructure.
4. Computing useful descriptors, such as molecular weight, polar surface area, number of rotatable bonds etc.
5. Converting between popular chemical formats/identifiers such as SMILES, InChI, MDL molfile.

There are several software libraries, written in different languages, that implement some or all of the operations described above. Two of these toolkits offer robust and comprehensive functionality, coupled with a permissive license, namely RDKit (developed and maintained by Greg Landrum) and Indigo (created by GGA software, now Epam). They both provide Python bindings and database cartridges, that, among other things, allow performing substructure and similarity searches on compounds stored in RDBMS.

The ChEMBL web services that we’ve described so far are focused on the retrieval of structured data stored in databases. Talking with colleagues, we’ve identified a gap in efficient pipelines, that allow researchers to handle data process and curating chemical datasets, and we thus focused on building additional cheminformatics-focused services. To fix this gap, the Beaker project was setup. Beaker [Beaker14] exposes most functionality offered by RDKit using REST. This means that the functionality RDKit provides, can now be accessed via HTTP, using any programming language, without requiring a local RDKit installation.

Following a similar setup to the data part of ChEMBL web services, the utils part (Beaker) is written in pure Python (using Bottle framework), Apache 2.0 licensed, available on GitHub, registered to PyPI and has its own live online documentation. This means, that it is possible to quickly set up a local instance of the Beaker server.

In order to facilitate Python software development, the ChEMBL client library has been created. This small Python package wraps around Requests library, providing more convenient API, similar to Django QuerySet, offering lazy evaluation of results, chaining filters and caching results locally. This effectively reduces the number of requests to the remote server,
which speeds up data retrieval process. The package covers full ChEMBL web services functionality, allowing users to retrieve data as well as perform chemical computations without installing chemistry toolkits.

The following code example demonstrates how to retrieve all approved drugs for a given target:

```python
from chembl_webresource_client.new_client

# Receptor protein-tyrosine kinase erbB-2
chembl_id = "CHEMBL184"

activities = new_client.mechanism
    .filter(target_chembl_id=chembl_id)
compound_ids = [x['molecule_chembl_id']
    for x in activities]
approved_drugs = new_client.molecule
    .filter(molecule_chembl_id__in=compound_ids)
    .filter(max_phase=4)

approved_drugs
```

Another example will use Beaker to convert approved drugs from the previous example to SDF file and compute maximum common substructure:

```python
from chembl_webresource_client.utils import utils

smiles = [drug['molecule_structures']]
canonical_smiles = [drug for drug in smiles]
mols = [utils.smiles2ctab(smile) for smile in smiles]
sdf = ''.join(mols)
result = utils.mcs(sdf)
```

4 Rapid data analysis and prototyping

Access to a very comprehensive cheminformatics toolbox, consisting of a chemically-aware relational database, efficient data access methods (ORM, web services, client library), specialized chemical toolkits and many other popular general-purpose libraries, scientific and data science libraries, facilitates sophisticated data analysis and rapid prototyping of advanced cheminformatics applications.

This is complemented by an IPython notebook server, which executes Python code along with rich interactive plots and markdown formatting to improve sharing results with other scientists.

In order to demonstrate capabilities of the software environment used inside ChEMBL a collection of IPython notebooks has been prepared. They contain examples at different difficulty levels, covering following topics:

1. Retrieving data using raw SQL statements, Django ORM, web services and the client library.
2. Plotting charts using matplotlib and D3.js.
3. Detailed RDKit tutorial.
4. Machine learning - classification and regression using scikit-learn.
5. Building predictive models - ligand-based target prediction tutorial using RDKit, scikit-learn and pandas.
6. Data mining - MDS tutorial, mining patent data provided by the SureChEMBL project.
7. NoSQL approaches - data mining using Neo4j, fast similarity search approximation using MongoDB.

Since many notebooks require quite complex dependencies (RDKit, numpy, scipy, lxml etc.) in order to execute them, preparing the right environment may pose a challenge to non-technical users. This is the reason that the ChEMBL team has created a project called myChEMBL [myChEMBL14].

myChEMBL encapsulates an environment consisting of the ChEMBL database running on PostgreSQL engine with RDKit chemistry cartridge, web services, IPython Notebook server hosting collection of notebooks described above, RDKit and Indigo toolkits, data-oriented Python libraries, simple web interface for performing substructure and similarity search by drawing a compound and many more.

myChEMBL comes preconfigured and can be used immediately. The project is distributed as a Virtual Machine, that can be downloaded via FTP or obtained using Vagrant by executing the following commands:

```bash
vagrant init chembl/mychembl_20_ubuntu
vagrant up --provider virtualbox
```

There are two variants - one based on Ubuntu 14.04 LTS and the second one based on CentOS 7. Virtual Machine disk images are available in vmdk, qcow2 and img formats. Docker containers are available as well. The scripts used to build and configure machines are available on GitHub so it is possible to run them on physical machines instead of VMs.

Again, Python plays important role in configuring my-ChEMBL. Since Docker is designed to run one process per container and ignores OS-specific initialization daemons such as upstart, systemd etc. myChEMBL ships with supervisor, which is responsible for managing and monitoring all core myChEMBL services (such as Postgres, Apache, IPython server) and providing a single point of entry.

5 Target prediction

The wealth and diversity of structure-activity data freely available in the ChEMBL database has enabled large scale data mining and predictive modelling analyses [Ligands12], [Targets13]. Such analyses typically involve the generation of classification models trained on the structural features of compounds with known activity. Given a new compound, the model predicts likely biological targets, based on the enrichment of structural features against known targets in the training set. We implemented our own classification model using:

1. a carefully selected subset of ChEMBL as a training set stored as a pandas dataframe,
2. structural features computed by RDKit,
3. the naive Bayesian classification method implemented in scikit-learn.

As a result, ChEMBL provides predictions of likely targets for known drug compounds available online (e.g. in https://www.ebi.ac.uk/chembl/compound/inspect/CHEMBL502), along with the models themselves available to download (ftp://ftp.ebi.ac.uk/pub/databases/chembl/target_predictions/). This is complemented with an IPython Notebook tutorial on using these models and getting predictions for arbitrary input structures.

Furthermore, similar models have been used in a publicly available web application called ADME SARfari [Sarfari].
This resource allows cross-species target prediction and comparison of ADME (Absorption, Distribution, Metabolism, and Excretion) related targets for a particular compound or protein sequence. The application uses SQLAlchemy as an ORM, contained within a web framework (Pyramid & Cornice) to provide an API and HTML5 interactive user interface.

6 Curation of Data

Supporting and automating the process of extracting and curating data from scientific publications is another area where Python plays a pivotal role. The ChEMBL team is currently working on a web application, that can aid in-house expert curators with this challenging and time-consuming process. The application can open a scientific publication in PDF format or a scanned document and extract compounds presented as images or identifiers. The extracted compounds are presented to the user in order to correct possible errors and save them to database. The system can detect compounds already existing in database and take appropriate action.

In addition to processing scientific papers and images, curator interface can handle the most popular chemical formats, such as SDF files, MDL mofles, SMILES and InChIs. Celery is used as a synchronous task queue for performing the necessary chemistry calculations when a new compound is inserted or updated. This system allows a chemical curator to focus on domain specific tasks and no longer interact directly with the database, using raw SQL statements, which can be hard to master and difficult to debug.

7 Discussion

Python has become an essential technology requirement of the core activities undertaken by ChEMBL group, in order to streamline data distribution, curation and analysis in the field of computational drug discovery. The tools built using Python are robust, flexible and web friendly, which makes them ideal for collaborating in a scientific environment. As an interpreted, dynamically typed scripting language, Python is ideal for prototyping diverse computing solutions and applications. The combination of a plethora of powerful general purpose and scientific libraries, that Python has at its disposal, (e.g. scikit-learn, pandas, matplotlib), along with domain specific toolkits (e.g. RDKit), collaborative platforms (e.g. IPython Notebooks) and web frameworks (e.g. Django), provides a complete and versatile scientific computing ecosystem.

8 Acknowledgments

We acknowledge the following people, projects and communities, without whom the projects described above would not have been possible:

1. Greg Landrum and the RDKit community (http://www.rdkit.org/)
2. Francis Atkinson, Gerard van Westen and all former and current members of the ChEMBL group.
3. All ChEMBL users, in particular those who have contacted chembl-help and suggested enhancements to the existing services

References

[ChEMBL12] A. Gaulton, L.I. Bellis, A.P. Bento et al. ChEMBL: a large-scale bioactivity database for drug discovery, Nucl. Acids Res., 40(database issue):D1100-D1107, January 2012.
[ChEMBL14] A.P. Bento, A. Gaulton, A. Hersey et al. The ChEMBL bioactivity database: an update, Nucl. Acids Res., 42(D1):D1083-D1090, January 2014.
[MedChem] G. Papadatos, J.P. Overington. The ChEMBL database: a taster for medicinal chemists, Future Med Chem., 6(4):361-364, March 2014.
[WS15] M. Davies, M. Nowotka, G. Papadatos et al. ChEMBL web services: streamlining access to drug discovery data and utilities, Nucl. Acids Res., April 2015.
[Beaker14] M. Nowotka, M. Davies, G. Papadatos et al. ChEMBL Beaker: A Lightweight Web Framework Providing Robust and Extensible Cheminformatics Services, Challenges, 5(2):444-449, November 2014.
[myChEMBL14] M. Davies, M. Nowotka, G. Papadatos et al. myChEMBL: A Virtual Platform for Distributing Cheminformatics Tools and Open Data, Challenges, 5(2):334-337, November 2014.
Fig. 4: Extracting data from a scientific publication.

[Ligands12] J. Besnard, G.F. Ruda, V.Setola et al. Automated design of ligands to polypharmacological profiles, Nature, 492(7428):215–220, December 2012.

[Targets13] F. Martínez-Jiménez, G. Papadatos, L. Yang et al. Target Prediction for an Open Access Set of Compounds Active against Mycobacterium tuberculosis, PLoS Comput Biol, 9(10):e1003253, October 2013.

[Sarfari] M. Davies, N. Dedman, A. Hersey et al. ADME SARfari: comparative genomics of drug metabolizing systems, Bioinformatics, 31(10):1695-7, May 2015.
