Structural and Data-Driven Approaches to Drug Discovery

Fergus Boyles, Patrick Brennan, and Thomas Hadfield
Doctoral Training Centre - University of Oxford
03/11/2020

Protein-Ligand Docking

In this practical you will explore protein-ligand docking by docking the protease inhibitor Indinavir, which was used as part of an early highly active antiretroviral treatment (HAART) for HIV/AIDS, into a crystal structure of HIV-1 protease.

Requirements

PyMOL (https://pymol.org/2/)

You can install PyMol using conda:

```
conda install -c schrodinger pymol-bundle
```

This will also make the PyMOL Python API available in your conda environment.

OpenBabel (https://open-babel.readthedocs.io/en/latest/Installation/install.html)

You can install OpenBabel using conda:

```
conda install -c conda-forge openbabel
```

This will also make the OpenBabel Python API available in your conda environment.

RDKit (https://www.rdkit.org/docs/GettingStartedInPython.html)

You can install RDKit using conda:

```
conda install -c conda-forge rdkit
```

Smina (https://sourceforge.net/projects/smina/)

This is a fork of the popular docking program AutoDock Vina (http://vina.scripps.edu/) that features an enhanced user interface and scoring options. A pre-built binary is available for Unix-based systems.

If you're using Windows, you can install OpenBabel and Smina in the WSL. In WSL, your Windows drives are mounted under the `/mnt` folder. For example, in WSL, your Windows home directory in your C drive will look something line

```
/mnt/c/Users/<username>
```

You can place files here from the WSL to make them available in Windows.

Step 1: Prepare your data

For this practical we'll be docking a ligand for which there is a crystal structure of the bound protein-ligand complex in the PDB. The structure, 1HSG, features the protease inhibitor Indinavir in complex with HIV-1 protease. Before going any further, download this structure and inspect it in PyMol. See if you can identify any important interactions between the protein and the ligand.

We've prepared files containing the protein structure, the crystallographic ligand binding mode, and the SMILES string of the ligand.

To better reflect a real-world docking scenario, we are going to start with the SMILES string of the ligand. This is a 2D ASCII representation of the ligand commonly used in the cheminformatics world. Before we can try docking the ligand, we first need to generate a 3D conformer for the docking program to use as a starting point. Simply re-docking the conformer extracted from the pdb file would bias the docking search toward the crystallographic binding pose and lead to an unrealistically high estimate of docking performance.

We've provided a script, `get_conformer.py`, which uses the cheminformatics toolkit RDKit to generate a 3D conformer from a SMILES string, which you can use to create your conformer by running:

```
python generate_conformer.py ligand.smi ligand.sdf
```

The sdf file format, based on the chemical file table, is commonly used to store 3D structures of small molecules. Open your new conformer in PyMOL to verify that you have a 3D structure for the ligand.

Next, we need to prepare the protein and ligand files for docking. Smina, like the AutoDock Vina upon which it is based, uses the pdbqt format (an extension of the pdb format) to store additional information used by the docking engine, such as atom types, partial charges, and

the location of rotatable bonds in the ligand. We can use OpenBabel to convert our pdb and sdf files into pdbqt files. For the ligand, we simply run:

```
obabel ligand.sdf -O ligand.pdbqt
```

Compare the contents of ligand.sdf and ligand.pdbqt - notice that ligand.pdbqt includes explicit information about where the rotatable bonds are in the ligand. This is used by Smina to sample the conformational space of the ligand.

By default, OpenBabel identifies rotatable bonds in the molecule when generating a pdbqt file. This is what we want for our ligand. However, for our protein, which is a very large molecule with many rotatable bonds, identifying all of these would take OpenBabel a very long time. To tell it to treat the protein as a rigid molecule, we can run OpenBabel with the '-xr' flag.

```
obabel protein.pdb -O protein.pdbqt -xr
```

The 'x' means 'output' and the 'r' means 'rigid, i.e., write the output as a rigid molecule. You can see exactly what the input and output options are for a specific file format by running:

```
obabel -L <format>
```

Compare the contents of protein.pdb and protein.pdbqt. Notice how each atom record has been extended to include a special atom type. These capture additional contextual information about the atom, (e.g. C = aliphatic carbon; A = aromatic carbon) and are used by the docking engine to evaluate possible interactions.

Smina can generate pdbqt files on the fly using OpenBabel, but it can be valuable to pre-generate these files if you plan on running many docking experiments, or want to manually inspect them beforehand.

Step 2: Run the docking

Now we have everything we need to dock our ligand! To run Smina, we need to specify a region around the protein in which it will attempt to dock the ligand. To do this, we can either specify a search box by providing the (x,y,z) coordinates of the centre of the search box and the (x,y,z) dimensions of the search box, or we can provide an additional ligand file that Smina will use to automatically identify the search space. Since we already have the coordinates of the ligand from the original pdb file, we can use this to save time. Note that this doesn't bias Smina toward the original binding mode, as it simply computes the centroid of the ligand and adds a search box around that point.

To run Smina with default parameters, using the crystal ligand to define the search box:

```
smina.static -r protein.pdbqt -l ligand.pdbqt -o docked.pdbqt\
             --autobox_ligand crystal_ligand.sdf
```

This will take about a minute on a modern 8-core CPU. Smina automatically detects the number of available cores and, if possible, uses all of them. You can restrict Smina to a specific number of cores using the `--cpu` option.

Once your docking has finished, Smina will print a summary of the binding poses generated, sorted (from best to worst) by the binding affinity predicted by Smina's scoring function. The RMSD (in Angstroms) of each pose relative to the highest-ranked pose is also shown.

You can convert the docked poses into an sdf file using OpenBabel:

```
obabel docked.pdbqt -O docked.sdf
```

Load your docked poses and the crystal pose into PyMOL and inspect them. Are any of them similar to the crystal pose? Is the pose ranked highest by Smina the best pose?

To quantify the accuracy of the docked poses, we can compute their heavy-atom RMSD with respect to the crystal pose. You could write your own script to do this, or we can use `obrms` tool included with OpenBabel:

```
obrms docked.sdf crystal_ligand.sdf
```

A heavy atom RMSD of less than 2 Angstroms is typically considered to be 'native-like'. How accurate were your poses? Does this line up with what you saw visually in PyMol, and does Smina's ranking correlate with the quality of the poses?

Smina's scoring function estimates the free energy of binding in kcal/mol. Our ligand, MK1, has an experimentally-determined free energy of binding of -43.47kJ/mol (source: https://www.rcsb.org/structure/1hsg), equivalent to -10.39 kcal/mol. How does this compare to the values predicted by Smina for each of your docked poses? Does the pose ranked highest by Smina have the most accurate predicted free energy of binding? What about the lowest-RMSD pose?

Step 3: Tweak the docking parameters

Smina's default parameters are quite sensible, and often yield reasonable results for many protein-ligand complexes. However, Smina has a lot of options that can be used to control the docking search. To see them all, run

```
smina --help
```

For this tutorial, let's focus on three options:

- exhaustiveness - this is (surprisingly) the level of exhaustiveness of the docking search. It is roughly proportional to the time spent sampling conformational space.
- autobox_add - this is the size of the search box that is constructed around the centre point. A larger search space allows Smina to attempt to dock the ligand in more

locations around the active site, but will increase run time and may result in the ligand being docked far from the correct location.

- num_modes, energy_range, and min_rmsd_filter - these control the (maximum) number of poses Smina returns, as well as the range of (predicted) energies and rmsd relative to the best pose that are considered. Increasing the number of modes and the energy range and reducing the rmsd filter will allow Smina to return more poses, but can also reduce the diversity of the poses.

By default, Smina uses --exhaustiveness 8 and --autobox_add 4. Try setting the exhaustiveness to 20. How does this affect the run time? Does this lead to any better results? Next, try increasing the search box padding from 4 to 8. Again, how does this affect the run time and what effect does it have on the results?

Try increasing the number of poses Smina is allowed to generate, and tweak the size of the energy and rmsd filters. Do you generate any more poses, and are they sufficiently distinct to be useful?