Tutorial

It is important to have completed those steps:

1. Have downloaded the biter folder
2. Have executed the setup.py in order to create the necessary environment

To run biter you just have to type the input file type and the file itself (pdb or mol2). It will generate a solution file that can be called using chimera to show the distribution of probabilities through all the residues to take part on a binding site.

$ python biter.py -i pdb protein.pdb

As said, this will generate a .cmd file that can be called using chimera

$ chimera protein\_bindingProbability.cmd

You can get only the ML solution by running:

$ python biter.py -i pdb -b protein.pdb

Or just the geometric pocket detector:

$ python biter.py -i pdb -p protein.pdb

Theory

The binding sites detector is coded mainly as a machine learning approach. It first computes some data and using the pytorch package it trains an artificial intelligence that is capable to predict if each atom of a molecule takes part on a binding site or not. Lately, to filter the solutions, a clustering algorithm is computed in order to detect regions with high positive solution density. To get every data row passed to the model, it must compute some features:

1. Every row contains the atom that we are evaluating and its 15 nearest neighbors.
2. For every neighbor:
   1. SASA value
   2. Direction respect the main atom
   3. The secondary structure of the residue of which the atom pertains
   4. Lenard-Jones potential
   5. Hydrophobicity of the residue of which the atom pertains
   6. 10 nearest distances to geometric possible binding points
3. SASA value:

The SASA (solvent-accessible surface area) value is computed using the Biopython package.

It uses “the “rolling ball” algorithm developed by Shrake & Rupley algorithm, which uses a sphere (of equal radius to a solvent molecule) to probe the surface of the molecule.” [https://biopython.org/docs/dev/api/Bio.PDB.SASA.html]

1. Direction respect the main atom

The objective of this precalculus is to both describe the protein structure using several features and adding as much relevant information as possible.

One of the ways to describe the structure is to know the direction of the 15 nearest neighbors of each atom.

This is basically computed using numpy arrays, and subtractions that result on a direction

1. Secondary structure

The secondary structure of the protein is computed using the Φ and Ψ angles for the helix structure. For the beta sheet, the angle computed is the resulting between the N-H bond and the O.

Alpha helix:

For the alpha helix, the number of residues between one and another residue goes from 3 to 5. Also, the distance between those residues is less than 3.4 Å. Also Φ has to be 92 ± 35 %, and Ψ must be 98 ± 35 %.

Beta sheet:

For the beta sheet first the distance between Ni and Oj must be < 3.2 Å. Then a H is placed in the same plane as Cα-N-C at a distance of the midpoint of Cα-C, but in the opposite direction. Having placed the H in between Ni and Oj it is time to calculate the angle being H the vertex. The angle must be 180 ± 15 %.

1. Lennard-Jones potential

It returns a matrix with the potential for every pair of atoms.

ε and σ are obtained from a library that has those values from each pair of atoms [https://github.com/choderalab/ambermini/blob/master/share/amber/dat/leap/parm/parm99.dat].

1. Hydrophobicity

In order to transform the letter symbol of each residue and to add hydrophobicity information, a library that contains different hydrophobicity types is used [https://www.cgl.ucsf.edu/chimera/docs/UsersGuide/midas/hydrophob.html#anote].

1. Geometry based.

To add more information about the geometry of the molecule its used a script that calculates points around the protein that are placed on concave regions.

This script uses the SASA values to know which atoms do have accessible area, then it places points from [x-3, y-3, z-3] to [x+3, y+3, z+3] with a 3 value step. Once it has the points it calculates whether each point is inside another atom or not. If not, it throws other points until reached a distance of 15 or a collision with another atom. Once it has all the collisions it calculates the area of first sphere that is occupied by the collisions. A draw in 2D molecule representation is shown for a better understanding.

Diagrama

Descripción generada automáticamente

In this 2D example we can see 2 atoms (red spheres) with SASA > 0. In this particular example each atom throws possible points (green dots). Those points can be inside of another atom, that would be the case of the green dots inside the protein, or outside the protein. Later on, using the outside of protein green points, lines are thrown in 45º difference in all directions from distance 2 to 15 or until collision.

Once we have all the collisions we know the θ and ϕ angles (in the case of the sphere). In the case of 2D protein we would have just one angle. Now, we have to know which angles do provide a collision with the protein. If it is 0.5 or higher of the sphere we know we are in a concave site. That would be the case of 2 and 3 green dots.

When working in 3D, the area of the sphere is computed using θ and ϕ angles, so in that case we search for an area equal or grater than 2π.

Once the program has all the points, it adds 10 features foreach atom, those features being the 10 nearest distances to those points using KDTrees algorithm.

All this information is fed to the machine learning approach. For every molecule, it’s feed each single atom (avoiding H because some pdb may not have this information).

The data used is a curated database that each protein has at least one known binding site, so that leads to a good data used. Different techniques such as upgrading the minority group could be applied.

The algorithm for training opens the folder called scPDB and loads one at time protein.mol2 file. It is continuously learning, the program does not need to learn on one single dataframe. This is more useful, because the computational resources needed are less that when trying to load large amounts of information in a single time.

Every protein loop the program loads the existing model and tries to improve it, when all the atoms of the protein have passed through the ML, the new model is saved, also the protein used is written down on a file. This way the algorithm can be paused at any time, and the information lost will just be the modified model with the protein running at the pause moment.