Comp 564: Assignment 2

Protein Structure and System Biology

Due on April 12th, 2019.

- To some extent, collaborations are allowed, but you must indicate the name of all collaborators (including instructors) on your answers. Uncredited collaborations will be penalized.
- Unless specified, all answers must be justified.
- Partial answers will receive credits.
- Answers should be submitted electronically to the instructor.

Exercise 1 (35 points) We aim to implement (in Python) a simple neural network (a multi-layer perceptron) to predict if a residue at index i is in contact with another residue or not (regardless of the index of the other residue). Here, we will focus on building a predictor for G protein-coupled receptors (GPCRs).

You will use the template `http://www.cs.mcgill.ca/~jeromew/teaching/564/W2019/HW2/HW2Q1.txt` (replace `.txt` with `.py` after you downloaded the file) to implement your program. In addition, you will also need to install the following requirements:

- The Python packages Biopython and scikit-learn (Note: use `pip`).
- `DSSP` (`https://anaconda.org/salilab/dssp`). This works on windows when using Windows Ubuntu Bash.
- `tmhmm` (`http://www.cbs.dtu.dk/cgi-bin/nph-sw_request?tmhmm`). Note that this software uses `perl`. The software does not work on Windows; use the Trottier computers or the web-server if you do not have access to another OS.

First, you will build a data set for training and testing your methods. Go to the Protein Data Bank (PDB) website and use the "advanced search" tool to retrieve all PDB entries having the keyword "GPCR" (i.e. use the field "text search") and less than 90% of sequence identity. The "advanced search" interface is accessible at `http://www.rcsb.org/pdb/search/advSearch.do?search=new`. This request should return to you 131 structure files.

Next, you will implement a feed-forward neural network with a single hidden layer of 156 neurons. Your input features will be:

- The amino acid type for which you want to make a prediction, as well as the type of the four amino acids before and after that position.
- The secondary structure type of each of those 9 residues.
- Each example will be structured as a list of 9 tuples (A, B) where A is the amino acid one-letter code, and B is the secondary structure type. The residues should be ordered by sequence position, with the residue of interest (the one on which the prediction is done) in the middle.

Since GPCRs are all-α proteins, we will only consider two types of secondary structures `H` (transmembrane helices) and `C` (coil).

- **Task 1:** Implement the function **get_PDB_info** that returns sequence and structure information from a PDB file, as well as the intramolecular contacts in that PDB file. A contact is defined as two residues whose C_α are less than $5A$ apart. Those residues must be separated by at least 9 other amino acids in sequence order for this contact to be considered meaningful. You will be required to implement 8 helper functions to complete this task. Feel free to add more helper methods if they are helpful to you.

- **Task 2:** Implement the function **split_dataset**, which builds a meaningful training set and test set from the dataset you assembled in task 1 to train a neural network on. You are provided with the neural network implementation to evaluate the accuracy of your model. Do not modify the code outside of the functions you are asked to implement.
- **Task 3:** Report the accuracy of predictions on the training sets and test sets of your neural network when you only use the sequence features.
- **Task 4:** Report the accuracy of predictions on the training sets and test sets of your neural network when you also include the secondary structure assignment obtained with `DSSP` (`DSSP` is a program generating secondary structure annotation from a PDB file).
- **Task 5:** Same as above but using the secondary structure prediction made with `tmhmm` instead of `DSSP`.
- **Task 6:** Discuss your results. How do you explain the differences of accuracy between the various experiments?

Refer to the comments within the provided python template for more detailed instructions.

Exercise 2 (10 points) The Hart and Istrael approximation algorithm for the `HP` square lattice protein folding model (1995) assumes it exists ant index p such that at least half of the odd `H` (hydrophobic amino acids at odd indexes) are on one side, and at least one half of the even `H` on the other side. Prove that this index p always exists.

Exercise 3 (20 points) We will analyze the result of our MD simulation. We simulated a system for 2000 pico seconds with a mutated version of Amylin. Your job is to create the RMSD graph for this simulation along with a short movie showing the protein in action. To do this, you will have to:

1. Read and execute the molecular dynamics tutorial available at `http://www.cs.mcgill.ca/~jeromew/teaching/564/W2019/COMP564_MD_tutorial_W2019.pdf`. A zip file including all files you will need for this tutorial is available at `http://www.cs.mcgill.ca/~jeromew/teaching/564/W2019/COMP564_MD.zip` (Note: You do not have to return any answer for this, but it will provide you the knowledge you need to complete this exercise).

2. Retrieve the files `npt-nopr.tpr` and `npt-nopr.trr` in the tutorial material (`http://www.cs.mcgill.ca/~jeromew/teaching/564/W2019/COMP564_MD.zip`). These two files store the result of the simulation trajectory of every atom.

3. Use the 2 commands on slide 8 of the tutorial. The first command will create the RMSD graph, and the second will create the movie.

4. Generate a graph from a software that opens .xvg extensions

5. Open your `molecule-movie.pdb` file in PYMOL and click "File→Save as→Movie" and choose .avi, .mpg or .mov.

Note: If you use GROMACS5, there are slight changes in the commands. A tutorial is available at `http://www.cs.mcgill.ca/~jeromew/564/W2019/COMP564_GROMACS5_tutorial_W2019.zip`

Exercise 4 (35 points) We want to implement a simplified version of the ISORANK algorithm covered in class (`http://www.pnas.org/content/105/35/12763`) to compare 2 protein-protein interaction (PPI) networks N_1 and N_2. Here, we will not include the sequence similarity score in our model. The functional similarity score R_{ij} between two nodes i and j is thus equal to the network similarity score defined as:

$$R_{ij} = \sum_{u \in \mathcal{N}(i)} \sum_{v \in \mathcal{N}(j)} \frac{R_{uv}}{|\mathcal{N}(u)| \cdot |\mathcal{N}(v)|} \tag{1}$$

Where i is a node of the network N_1, and j a node of the network N_2. $\mathcal{N}(x)$ is the list of neighbours of a node x. To calculate the R_{ij}, we represent the system of equations given by Equation 1 as a product of matrices $R = A \cdot R$, where:

$$R = \begin{pmatrix} R_{(a_1,b_1)} \\ R_{(a_1,b_2)} \\ \vdots \\ R_{(a_m,b_n)} \end{pmatrix} \tag{2}$$

$$A = \begin{pmatrix} a_{(a_1,b_1),(a_1,b_1)} & a_{(a_1,b_1),(a_1,b_2)} & \cdots & a_{(a_1,b_1),(a_m,b_n)} \\ a_{(a_2,b_1),(a_1,b_1)} & a_{(a_2,b_1),(a_1,b_2)} & \cdots & a_{(a_2,b_1),(a_m,b_n)} \\ \vdots & \vdots & \vdots & \vdots \\ a_{(a_m,b_n),(a_1,b_1)} & a_{(a_m,b_n),(a_1,b_2)} & \cdots & a_{(a_m,b_n),(a_m,b_n)} \end{pmatrix} \tag{3}$$

(4)

We obtain the values of the $a_{(i,j),(u,v)}$ using the following formula:

$$a_{(i,j),(u,v)} = \begin{cases} \frac{1}{|\mathcal{N}(u)| \cdot |\mathcal{N}(v)|} & \text{if } (i,u) \in N_1 \text{ and } (j,v) \in N_2 \\ 0 & \text{otherwise} \end{cases}$$

To solve this system of equations, we calculate the eigenvector R (i.e. finding R such that $R = A \cdot R$).

Once the values R_{ij} are defined, we compute the network alignment using a greedy approach. We align the node i and j with the largest R_{ij} and remove i and j from the list of available nodes. Then, we find the next largest value R_{xy} and align x with y. We iterate this process until no other node is available, or all remaining values R_{ij} are below a threshold λ.

1. Download the files `N1.txt`, `N2.txt`, and `N3.txt`, storing networks N_1, N_2, and N_3 you will use for this question at `http://www.cs.mcgill.ca/~jeromew/teaching/564/W2019/HW2/`. Each row stores an edge of the network, which are indicated by their identifiers separated by a space character (you can ignore the sign of the interaction if it has one).

2. Implement the proposed algorithm. You can use existing libraries to solve the eigenvector problem (e.g. `numpy.linalg` in Python). Your program must take as an input 2 networks A and B and return a list of values "$a_i\, b_j$", where a_i is the node of A aligned with node b_j of B. Each row must contain one and only one alignment of nodes, themselves separated by a space character.
   Notes:

   - If you want to solve the eigenvalue problem efficiently (e.g. using the power method), the matrix A needs to be sparse. You should carefully calculate the values of coefficient of A. But you should also notice that R can be sparsified as well. You will try to find a rule allowing you to set to 0 some R_{ij} before computing them.
   - Specify the precise command line used to run your program. Do not use any library or dependency not installed on SOCS servers.

3. Use your program to align N_1 with N_2, N_1 with N_3, and N_2 with N_3. Print the common subgraphs. Which networks are the most similar? Justify your answer.

4. Let us assume now that the edges are weighted (e.g. the weight could represent the strength of the interaction). Propose modification of the algorithm that will align nodes connected by an edge with similar weight (Note: We do not ask you to implement this algorithm).