Computational Biology

Team

Bitiutskii Artem
Semenenko Aleksandr
Fomin Dmitrii

Supervisor

Clovis Galiez

November 2022

Abstract. The goal of the current project is to model the structure of the protein that carries the mutation responsible for high resistance to tetracycline of a Salmonella variant.

1. Introduction

We initially process the data, then develop a piece of software that detects the covariations and creates the contact map. Finally, we use FT-KOMAR to predict the 3D structure.

2. Sequence alignment

The data contains strings of the form bellow

$> B4D6M1/13 - 59$

$LLDAA - F - R - V - C - S - E - -Q - - - - - G - -I - H - - - - - G - A - -TTRE - - - - - IADVA - - - -G - - - V - N - -E - V - T - -LF - R - H - F - - - - - G - -NKEKLIAAL$

First, the omissions, denoted by dashes $(-)$, should be removed along columns if the latter consists only of them. Doing that, we intend to simplify the initial data and then work with essential columns which have at least one amino acid. The corresponding part of the code is titled "Data preparation" in our notebook. After removing columns with more than 30% of skips $(-)$, we want to figure out how many dashes remained by plotting the histogram (see Fig. 1).

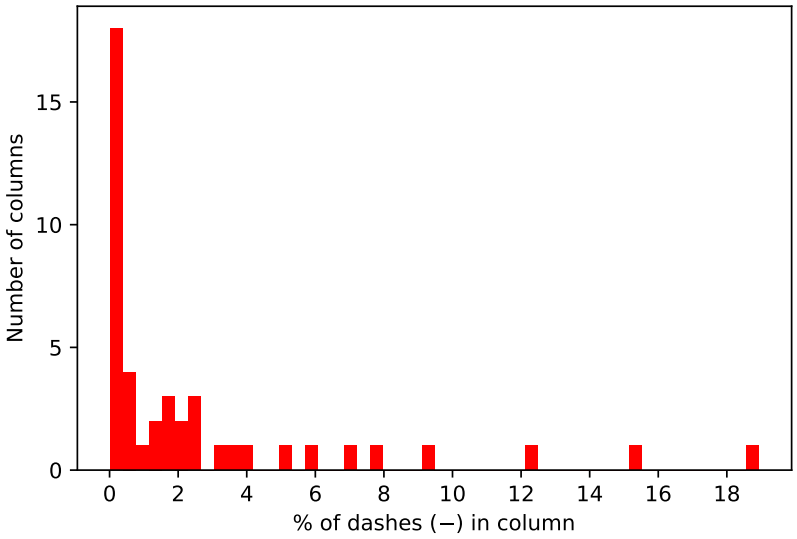

Figure 1: The histogram where the columns represent the frequency of appearing dash $(-)$ in the column.

3. Calculating co-variations

Recall corresponding one-letter codes from the table of standard amino acids:

A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V

So, 20 standard amino acids can be used to synthesize proteins. The structure of the protein is stable mainly thanks to the electrostatic interaction between amino acids being in contact. Thus, the latter should be investigated more precisely.

Let the dependence between sequences (columns) is measured by mutual information. Then we say that there is a contract between i-th and j-th amino acids if the mutual information $\mathrm{MI}(i,j)$ is greater than thresholding τ:

$$\mathrm{MI}(i,j) > \tau,$$

where

$$\mathrm{MI}(i,j) = \sum_{a,b} p(x_i = a, x_j = b) \log \frac{p(x_i = a, x_j = b)}{p(x_i = a)p(x_j = b)}. \tag{1}$$

In our notations, x_i is the i-th column where x_{im} is the amino acid in this column at m-th position.

It is convenient to rewrite i-th column, e.g. $[Q, A, Q, C, C, Q, \ldots]^T$ in one - hot representation, i.e. $[Q, A, Q, C, C, Q, \ldots]^T \xrightarrow{\{x_{ij}=Q\}} [1, 0, 1, 0, 0, 1, \ldots]^T$.

Given column-vector x_i and amino acid a, denote the set of indices $K_i(a) = \{m : x_{im} = a\}$ of nonzeros elements in one - hot representation of x_i. Then

$$p(x_i = a) = \frac{1}{N} \sum_{m \in K_i(a)} 1 = \frac{|K_i(a)|}{N} \tag{2}$$

and

$$p(x_i = a, x_j = b) = \frac{1}{N} \sum_{m \in K_i(a) \cap K_j(b)} 1 = \frac{|K_i(a) \cap K_j(b)|}{N}. \tag{3}$$

Substituting (2) with (3) in (1) yields

$$\mathrm{MI}(i,j) = \frac{1}{N} \sum_{a,b} |K_i(a) \cap K_j(b)| \log \frac{N|K_i(a) \cap K_j(b)|}{|K_i(a)||K_j(b)|}.$$

However, mentioned rule

$$\mathrm{MI}(i,j) > \tau \Rightarrow \text{there is a connection between } i \text{ and } j$$

leads to predicting extra contacts. That is why we apply the following correction

$$\mathrm{MI}'(i,j) = \mathrm{MI}(i,j) - \frac{1}{N} \sum_{k} (\mathrm{MI}(k,j) + \mathrm{MI}(i,k))$$

To see the difference between these two ways of calculating mutual information, we present two contact maps below.

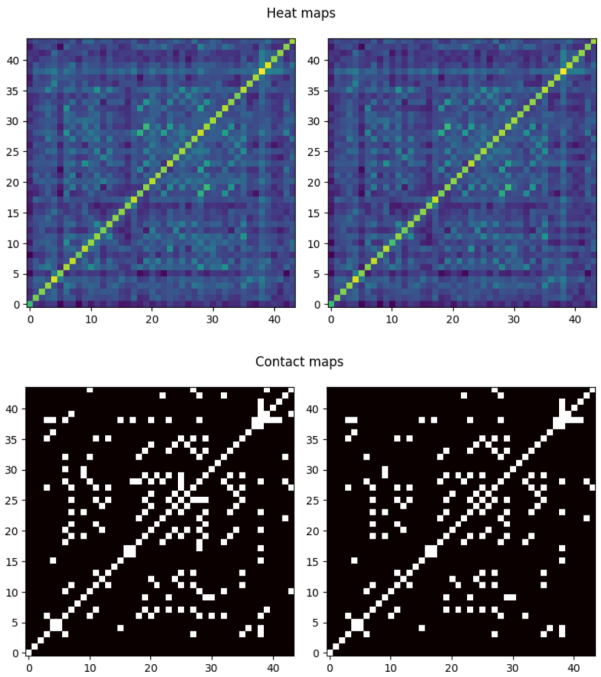

Figure 2: Calculation mutual information in two ways. The first column corresponds to MI and the second to MI′.

Then when τ is fixed, we can predict contacts between amino acids using the rule: $\mathrm{MI}'(i,j) > \tau$.

4. Protein contact map

To compute a protein contact map, we implemented two functions that calculate mutual information between x_i and x_j amino acids (see Fig. 7 and Fig. 8 in Appendix). We also compare them with $normalized\_mutual\_info\_score$ from the *sklearn* library.
Now we create heat maps after calculating mutual information using mentioned three functions:

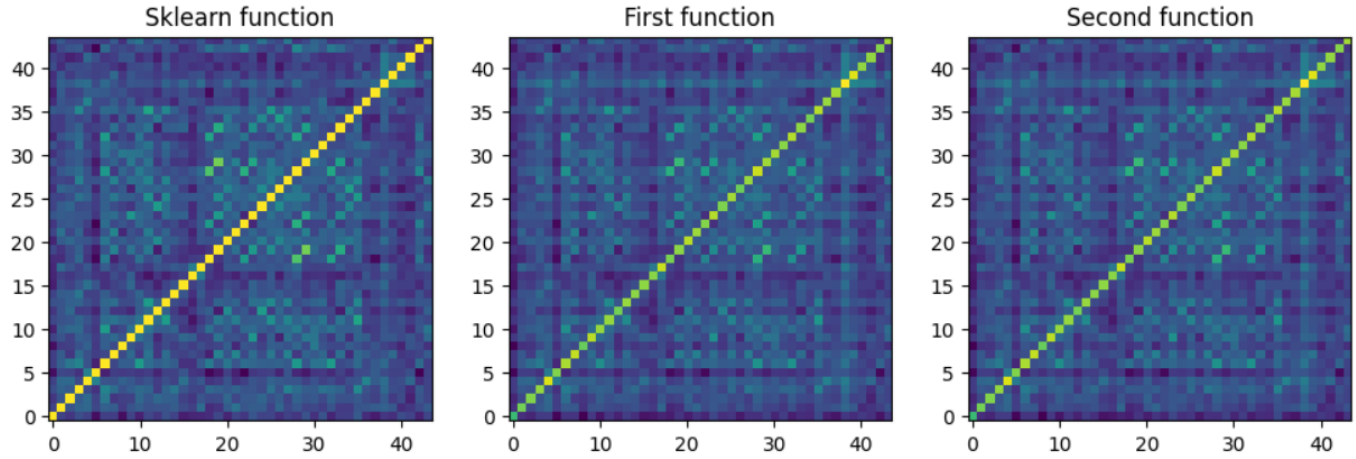

Figure 3: Heat maps from left to right for *sklearn* implementation, our the first function and the second respectively.

As we can see, all these heat maps have minor differences, therefore, next, we will use the *sklearn* mutual information function as the most convenient for us due to its normalization (it is easier to select a threshold).

5. Getting contact map from tetR.pdb

The next step is to get a contact map from *tetR.pdb*. After that, we are planning to compare it with our contact maps. To do this, we use $PDBParser$ from the *biopython* library. We pay attention only to the lines corresponding to "ATOM" and "CA" and read 3 coordinates. Next, we compute the Euclidean distance between i-th and j-th elements. It brings us to the following contact map in Fig. 4 (in this case $\tau = 9$Å‡):

6. Threshold choice

Now we have to find the threshold τ such that our contact map is as similar as possible to the contact map obtained in the previous step. One straightforward way to do it is simply to use grid search, some results of which are presented in Fig. 5. We chose the set $\{0.73, 0.75, 0.77\}$ of τ's.

‡ This particular value was discussed with our supervisor and also mentioned in some articles, e.g. here.

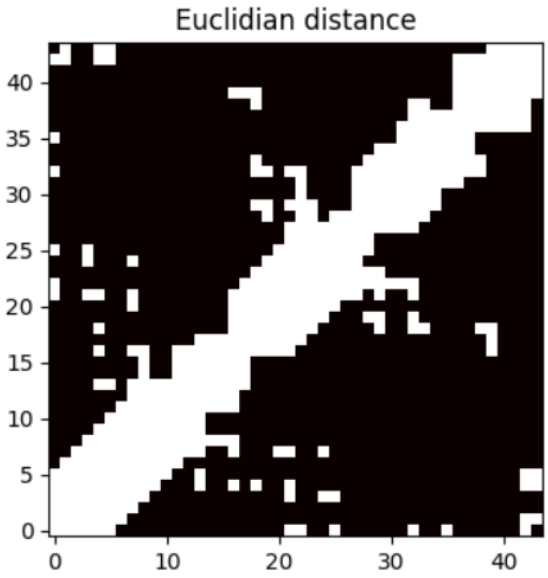

Figure 4: Contact map from *tetR.pdb*

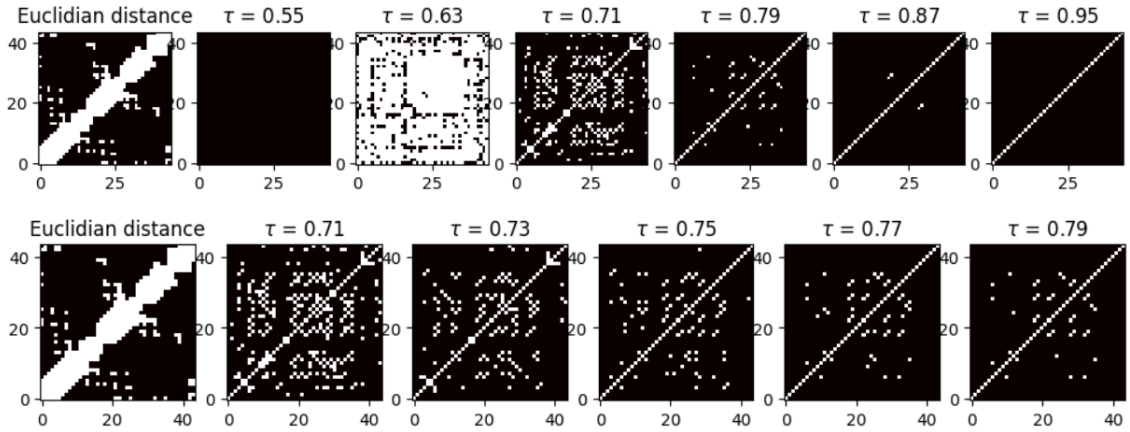

Figure 5: Contact maps for different threshold τ.

7. 3D structure

Having saved our contact map in the required format, we conduct a series of experiments: choose $\tau \in \{0.73, 0.75, 0.77\}$, then run FT-COMAR and PyMOL multiple times, in order to prevent stochastic influence on the result after using FT-COMAR tool. Averaging RMSD's for corresponding τ, we are ready to fill the table bellow

τ	0.73	0.75	0.77
Average RMSD	5.81	7.16	8.46

As a result, $\tau = 0.73$ appears to be the best option. Fig. 6 depicts 3D protein structure. One can say these two hardly coincide. However, we should take into

consideration two remarks at least:

- FT-COMAR works stochastically.
- We considered only a part of all the information predicting the structure (the rows containing "ATOM" and "CA" were taken).

The main form is preserved, and hence, these two are quite similar.

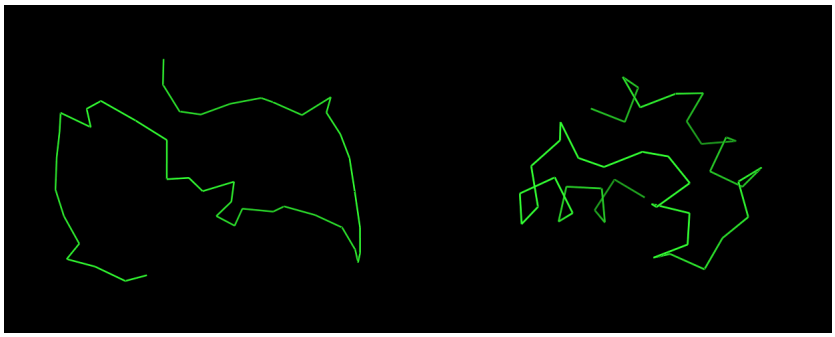

Figure 6: 3D structures: the left one is ours, and the right one is original.

8. Conclusion

We developed the tools for creating a contact map. Following a grid search and RMSD's analysis, we chose the threshold $\tau = 0.73$. At last, we plotted the 3D structure of the protein, which in turn turned out to be similar to the real one. At any rate, we should keep in mind that the current approach to modelling protein structure exhibits simplicity and therefore might provide insufficiently accurate results.

9. Appendix

```
def mutual_information(x_col, y_col):
  """
  The first mutual information function
  """
  summation = 0.0
  values_x = np.array(list(set(x_col)))
  values_y = np.array(list(set(y_col)))
  n = len(x_col)
  try:
    values_y.remove('nan')
    values_x.remove('nan')
  except:
    pass

  for x_i in values_x:
    for y_j in values_y:
      px = sum(np.where(x_col == x_i,1,0)) / n
      py = sum(np.where(y_col == y_j,1,0)) / n

      first = np.where(x_col==x_i,1,0)
      second = np.where(y_col==y_j,1,0)
      agreg = np.array([1 if first[i] * second[i] == 1 else 0 for i in range(len(first)) ])

      pxy = sum(agreg)/n

      if pxy > 0.0:
        summation += pxy * np.log( pxy  / (px*py) )
  return summation
```

Figure 7: The first function.

```
def prob_i(column, A, N):
    A = str(A)
    return sum(np.where(column == A,1,0))/N

def prob_joint_ij(column_i, column_j, A, B, N):
    A = str(A)
    B = str(B)
    c1 = np.where(column_i == A,1,0)
    c2 = np.where(column_j == B,1,0)
    n_ij = 0
    for i in range(c1.size):
      if c1[i] == c2[i] == 1:
        n_ij += 1
    return n_ij/N

def mutual_info(column_i, column_j, labels, N = None):
    N = len(column_i)
    mutual_info = 0
    for i in labels:
        for j in labels:
            p_i = prob_i(column_i, i, N)
            p_j = prob_i(column_j, j, N)
            R_ij = prob_joint_ij(column_i, column_j, i, j, N)
            if (R_ij > 0 and p_i > 0 and p_j > 0):
                mutual_info += R_ij*np.log( R_ij / (p_i * p_j))
    return mutual_info

def mutual_info_new(mi_matrix):
  """
  Our second mutual information function
  """
  new_mi_matrix = np.zeros((n,n))
  for i in range(n):
    for j in range(n):
      for k in range(n):
        s = mi_matrix[k, j] + mi_matrix[i, k]
      new_mi_matrix[i, j] = mi_matrix[i, j] - 1/n * s
  return new_mi_matrix
```

Figure 8: The second function.