Enhancing Protein Structure Analysis

Introduction

The Residue Interaction Network Generator (RING) and Ribbon Diagrams have revolutionized the study of protein structures, enabling researchers to delve into the intricate details of residue interactions. The aim of this powerfull tool in the field of structural biology is to provide a simple and intuitive way to visualize the interactions between residues in a protein structure. In our case we are going to develop a software that will be able to predict the interactions between residues in a protein structure, given a dataset of protein structures and its corresponding interactions.

Dataset

The first look of the project were given to datasets provided to understand the problem and the data. The datasets provide different type of geometrical information about the protein structures. We are going to summarize, the data are devided in two parts the source amino acid and the target amino acid. The source amino acid is the amino acid that is going to interact with the target amino acid. The data for the source and target aminoacids are provided in a csv file with the following columns:

- **pdb_id**: The id of the protein structure.
- **ch**: The chain of the protein structure.
- **resi**: The residue number of the amino acid.
- **resn**: The name of the amino acid.
- **ss8**: The secondary structure of the amino acid, calculated with DSSP.
- **rsa**: The relative solvent accessibility of the amino acid, calculated with DSSP.
- **phi**: The phi angle of the amino acid, ramachandran plot used to determine the angles.
- **psi**: The psi angle of the amino acid, ramachandran plot used to determine the angles.
- **up**: Half sphere exposure of the amino acid.
- **down**: Half sphere exposure of the amino acid.
- **ss3**:
- **a1 to a5**: Atchley scale of the amino acid.

Adding New data

To enhance the accuracy of our predictions regarding interactions between amino acids, we recognized that the existing dataset was insufficient. As a result, we made the decision to augment the dataset with additional data. Prior to this decision, we conducted preliminary experiments using simple prediction models. These models and their outcomes will be presented in detail later in the report, providing a clearer understanding of our rationale.

The first step involved incorporating a new set of data obtained from the DSSP module. To achieve this, we made modifications to the code provided by our professor, specifically in the calcfeatures.py file. During this process, we also identified and rectified errors present in the code, which were caused by the use of deprecated libraries. By addressing these issues, we ensured the proper functionality of the code and obtained the necessary data from the DSSP module.

These additions to the dataset, obtained through modifications and improvements to the code, significantly bolstered the quality and comprehensiveness of the data used for predicting amino acid interactions. The subsequent sections of the paper will delve into the details of the modifications made to the code and the specific improvements achieved:

```
if file_extension == ".pdb":
     structure = PDBParser(QUIET=True).get_structure(pdb_id, args.pdb_file)
 elif file_extension == ".cif":
     structure = MMCIFParser(QUIET=True).get_structure(pdb_id, args.pdb_file)
 else:
     print("Error: Unsupported file extension.")
```

By implementing these changes, our code is now capable of processing and extracting relevant data from both CIF and PDB files. This flexibility allows us to incorporate a broader range of structural data sources, enhancing the scope and versatility of our analysis.

Overall, this modification provides us with greater flexibility in data acquisition and enables us to explore a wider range of structural information, contributing to a more comprehensive analysis of amino acid interactions.

We encountered challenges with the deprecated version of the DSSP module, which caused conflicts with our Python version. To address this issue, we took a two-step approach.

First, we used an external software tool to generate the DSSP file separately. This allowed us to obtain the required structural information for further analysis. In the second step, we carefully examined the code and identified the specific problem causing the conflict. By updating the module and replacing the deprecated dssp function with mkdssp, we resolved the compatibility issues.

The following code snippet demonstrates the implementation of this update:

```
dssp = {}
 try:
     dssp = dict(DSSP(structure[0], args.pdb_file, dssp="mkdssp"))
 except Exception:
     logging.warning("{} DSSP error".format(pdb_id))
```

By resolving the compatibility issues, we were able to obtain the required structural information from the DSSP module. This enabled us to incorporate a broader range of structural data sources, enhancing the scope and versatility of our analysis.

To address the absence of PDB files associated with the feature_ring dataset provided, we developed a Python script named **collect_pdb.py**. This script efficiently downloads all the necessary PDB files related to the dataset. The process is straightforward: the script takes the filename as input from the feature_ring folder and proceeds to download the corresponding PDB files.

Furthermore, after the successful download of the PDB files, the script automatically executes the calcfeatures.py script for each downloaded PDB file. This allows us to calculate the new features specific to each PDB file by making appropriate calls from the terminal.

By utilizing the **collect_pdb.py** script, we ensure the availability of all the required PDB files for the feature_ring dataset. This enables us to generate accurate and up-to-date features for further analysis.

Looking at the data

Upon careful examination of the data and file, we have observed instances where certain rows are identical, except for the interactions recorded. This occurs because a single amino acid can interact with multiple other amino acids. As a result, we encounter a challenge of having duplicated data entries within the dataset.

To address this issue, we have implemented a solution involving the creation of a new column called "interaction." This column serves as a list to store the various interactions involving the amino acid.

To achieve this, we utilize a dictionary structure. The key of the dictionary comprises the **pdb_id**, **s_resi**, and **t_resi** values. By using this key, we can identify and select the duplicated rows within the same file of the dataset. Subsequently, we append the corresponding interactions to the "interaction" column.

By employing this approach, we effectively consolidate the duplicated data entries and store the diverse interactions of the amino acid within a single row. This enhances the clarity and efficiency of our dataset, avoiding unnecessary duplication of information.

Example of duplicated rows:

pdb_id	s_resi	...	t_resi	...	Interaction
1b0y	28	...	76	...	HBOND
1b0y	28	...	76	...	VDW

Solution provided:

pdb_id	s_resi	...	t_resi	...	Interaction
1b0y	28	...	76	...	[HBOND, VDW]

This will increase the number of possible interactions and will make the dataset more realistic.

Data preprocessing

In our dataset, it is important to note that not all possible interactions between residues are present. This occurs because some interactions may not be captured in the protein's structure. This poses a challenge as it results in a significant amount of data that cannot be utilized for analysis.

To address this issue, we have made a deliberate decision. Rather than excluding those rows entirely, we have introduced a new category called **unclassified** to represent these interactions. By including this category, we aim to create a more comprehensive and robust model. We understand that this approach may result in a potential loss of accuracy in the final model. However, after careful consideration, we believe it is the best solution to handle the absence of certain interactions in the dataset.

By adding the **unclassified** category, we acknowledge the presence of unobserved or unclassified interactions, ensuring that they are accounted for in our analysis. This approach allows us to capture a broader range of potential interactions, even though they may not be directly observable in the protein structure.

Normalization of the data

The normalization proposed is done by normalize the data depending on the type of data. The normalization is done by using the MinMaxScaler from sklearn. The normalization is done in the following way:

1. **Normalization of Angles:** The first normalization function focuses on columns representing angles in the dataset. Angles are mathematical values that describe the shape or orientation of certain features.

2. **Normalization of Acetylcholine Features:** The second normalization function deals with specific columns related to acetylcholine features. Acetylcholine is a neurotransmitter that plays a role in various biological processes.

3. **Normalization of Relative Solvent Accessibility (RSA):** The third normalization function focuses on columns representing the relative solvent accessibility (RSA). RSA refers to how exposed or buried a certain part of a molecule is within its environment.

4. **Normalization of Half-Sphere Coordinates:** The fourth normalization function addresses columns containing half-sphere coordinates. These coordinates describe the position or orientation of certain features.

5. **Normalization of Categorical Features:** The final normalization function deals with categorical features in the dataset. Categorical features represent different categories or labels, such as types of structures or residues. To transform these categorical labels into numerical representations, a technique called "LabelEncoder" is used. This transformation enables the categorical features to be used effectively in machine learning algorithms and analysis.

By applying these normalization techniques, the data is prepared in a standardized and comparable format. This ensures that the various features are on a consistent scale, allowing for accurate analysis, modeling, and interpretation of the data.

Multiclass Classification Problem

In machine learning and statistical classification, multiclass classification or multinomial classification is the problem of classifying instances into one of three or more classes (classifying instances into one of two classes is called binary classification). Multiclass classification should not be confused with multi-label classification, where multiple labels are to be predicted for each instance. We take also in account to use this last possibility by codify our interaction as one hot encoding vector for our prediction.