Python Polypeptide Prediction (p-cubed) Project Documentation

Authors: Isaias Ardaya, Minoru Nakano, Gayathri Ravindra, Queenie Tsang

Submitted on April 12, 2019

Table of Contents

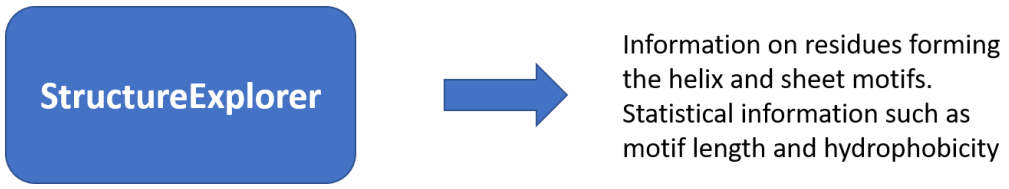
Purpose and Overview of the Project

Purpose of our project is to predict protein secondary structures and visualize the result. Our product is a secondary structure protein prediction program, relying on the support vector machine module for machine learning.

The secondary structures of proteins can fall into 3 general categories: Alpha helix, beta sheet and coil. These secondary structures will largely determine the biological function of the protein. Current sequencing techniques make it so that we can determine the amino acid sequence of proteins at a quicker rate than the respective proteins 3-D structure.

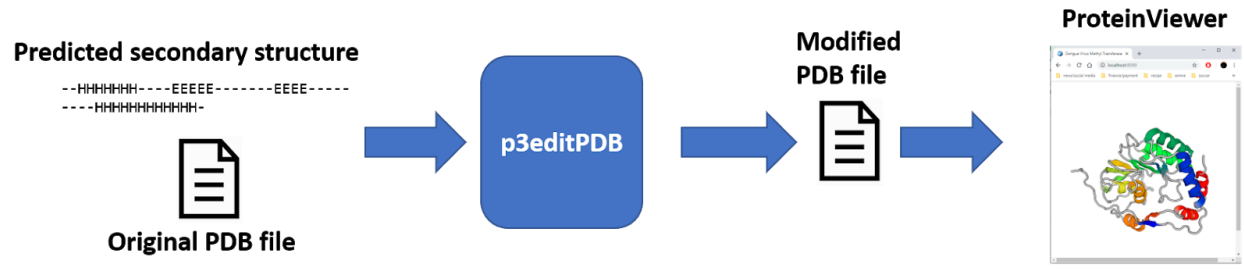
This system consists of three modules: StructureExplorer, StructurePrediction and p3editPDB/ProteinViewer. Overview of each module is illustrated as follows and described in detail in “Module Overview”:

StructureExplorer



StructurePrediction

p3editPDB/ProteinViewer



Module Overview

1. StructureExplorer

StructureExplorer was designed to extract the various structures within a protein in sequential order. The underlying amino acid sequences were also extract in the same manner. This information was used to generate statistics to help test a hypothesis about amino acid properties: That hydrophobic (meaning being more averse to water; the opposite of hydrophilicity) nature of certain amino acids was an important quality for determining the secondary structures of proteins.

This module went through several iterations and test programs (that were named aminoTest.py, aminoTest2.py etc) culminating in the program StructureExplorer.py. The programs functionality can be broken into 3 sections: Reading & Extraction, Motif Identification, Motif Analysis. This program was run on a Windows 8 machine, with Python version 3.7. The folder rs126 must be in the same directory as the helixTest.py file.

StructureExplorer.py

This program began by taking the following table from Kyte & Doolitte (1982) and turning it into a Python dictionary. The lower and more negative scores indicate a more hydrophobic amino acid.

Amino Acid	One Letter Code	Hydropathy Score
Isoleucine	I	4.5
Valine	V	4.2
Leucine	L	3.8
Phenylalanine	F	2.8
Cysteine	C	2.5
Methionine	M	1.9
Alanine	A	1.8
Glycine	G	-0.4
Threonine	T	-0.7
Serine	S	-0.8
Tryptophan	W	-0.9
Tyrosine	Y	-1.3
Proline	P	-1.6
Histidine	H	-3.2
Glutamic acid	E	-3.5
Glutamine	Q	-3.5
Aspartic acid	D	-3.5
Asparagine	N	-3.5
Lysine	K	-3.9
Arginine	R	-4.5

Reading and Extraction (readFile, extractSequenceSet, extractStructure)

The program begins by reading in the files in the rs126 directory, extracting the sequence and structure information from each file. That information would look something like this:

```
OrigSeq:    GPSQPTYPGDDAPVEDLIRFYDNLQQYLNVVTRHRY
cons:       ------------HHHHHHHHHHHHHHHHHH-----
```

Using regular expressions, the ‘OriSeq:’ and ‘cons:’ are removed from each line (please note the space is added for the formatting of this particular document; it is not present in the original files). The motif finder is primarily focused on the second line, showing either “-“, “H” or ”E”, for coil, helix or sheet respectively.

Motif Identification (motifFinder)

After extraction, the motif finder takes that sequence in as a string and using regex, can extract each individual structure from the string regardless of length and places it into a list. This is done in a recursive fashion, until the string length is equal to zero. Taking the above example, when these elements are placed in a list it would look something like this:

```
motifs =['-------------', 'HHHHHHHHHHHHHHHHHH', '-----']
```

Simultaneously, the sequence string is manipulated in similar fashion to get a list like this:

```
motifsAmino =['GPSQPTYPGDDAPV', 'EDLIRFYDNLQQYLNVV', 'TRHRY']
```

(Please note that motifs and motifsAmino are the names given in the program to the respective lists holding structure and amino acid information)

In both the motifs and motifsAmino lists, at each corresponding index, the elements are of equal length.

Motif Analysis (kdAnalysis)
Motif Analysis is composed of the function kdAnalysis and a for loop with the following syntax: `for (i,j) in zip(motifsAmino, motifs)`. In this case 'i' iterates through motifsAmino and 'j' iterates through motifs. This loop would generate the following information: The total count of each motif, average length of each type of motif; and the average hydrophobicity score for each motif. This information will be tracked with three new lists: motifCount, motifLength, and motifTKD. These lists have 3 indices each (for the 3 types of structures) with a numerical value of zero; and will have new values appended to each index

Each element in motifsAmino was passed into the function (kdAnalysis. Being a string (of amino acids), that element was converted into a list so that each separate amino acid could access the kd values (hydrophobicity scores) from the dictionary.

Example: 'TRHRY' is converted to ['T', 'R', 'H', 'R', 'Y']

The kdAnalysis function returns the sum of kd values for the element from the motifsAmino list that was passed into it. This sum is placed into the list, motifTKD. Simultaneously the for loop mentioned above is gathering information about the elements in the motifs list; it access the index[0] of each element to determine which of the three structures it is; and obtains the length of each structure as well.

Output
After those steps we can finally generate the output:

COIL COUNT: 1585 HELIX COUNT: 587 SHEET COUNT: 978

How many of each structure are contained with the file in the rs126 folder.

AVG_LEN_COIL: 7.661829652996845
AVG_LEN_HEL: 10.928449744463373
AVG_LEN_SHE: 4.806748466257669

The average length of each structure, in terms of amino acids.

COIL-HYDRO: -5.5634700315457435
HELIX HYDRO: -1.1328790459965923
SHEET HYDRO: 5.27822085889571

The average hydrophobicity score for each structure

From this we can that that each motif group has a different average amino acid length (Coil: 7.66, Helix: 10.93, Sheet: 4.81) as well as a different hydrophobicity score Coil: -5.56, Helix: -1.13, Sheet: 5.27). This last piece of information was sufficient to conclude that hydrophobicity does indeed play a pivotal role in determining secondary structure.

2. Secondary Structure Prediction

Secondary Structure Prediction module uses Supported Vector Machine (SVM) machine learning model to predict residues in an amino acid sequence that are likely to form either alpha helix or beta sheet. The module consists of two main programs: p3Train.py and p3Predict.py. The module also consists of four other python libraries with support functions to be used for the two main programs. They include;

1. aminoHelper.py
   Contains variables associated with amino acid properties, and a class with a function that calculates feature scores for each amino acid.

2. featureHelper.py
   Contains functions used to calculate and evaluate feature-related values for the training dataset.

3. p3DataBuilder.py
   Contains functions relevant to building training and test datasets.

4. p3DataReader.py
   Contains functions related to reading amino acid sequence data into the program.

SVM - Brief Background

Supported Vector Machine is a type of supervised machine learning process to that uses a set of training samples and labels that indicates which group each sample belongs to.[1] Upon training a model, SVM classifies test samples with unknown labels into different groups. For the Secondary Structure Prediction module, each amino acid in a protein sequence is treated as one test sample, and a label for that amino acid indicates whether the amino acid is classified as an “alpha-helix-forming”, “beta-sheet-forming” or “neither”. The secondary structure prediction module then uses a trained model to classify each amino acid residue in a previously unknown sequence into the aforementioned three groups.

p3Train.py

p3Train.py is responsible for reading in a training dataset for the machine learning model, quantifying and calculating features to be used for the prediction process, training the SVM model and saving the trained machine learning model in joblib format which is subsequently used to predict secondary structure of previously unknown amino acid sequences by p3Predict.py.
Currently, the program utilizes a training dataset called TD9078[2] (found in “p-cubed/StructurePrediction/td9078.csv” file) which consists of 9078 amino acid sequences of random sequence pattern with varying numbers and lengths of secondary structures, and corresponding secondary structure sequences as the following example:

```
Seq: NLSELDRLLLELNAVQHNPP
Sec: CCCHHHHHHHHHHCCCCCC
```

Sequence denoted as “Seq” is an amino acid sequence, and the sequence below denoted as “Sec” indicates which amino acids in the sequence are involved in forming secondary structures. In the example above, an alpha helix of 11 residues are formed between the fourth Glutamic Acid (E) and the fourteenth Alanine (A). These amino acids are marked with “H”, indicating that these are helix forming amino acids in this sequence. Beta-sheet forming amino acids, if they are present in the sequence, would be marked with letter “E”. Letter “C” indicates residues that do not participate in either of the secondary structures. These residues can also be marked with “-”. Although there are other types and subtypes of protein secondary structures, the program trains the models to predict secondary structure forming amino acids based on “three structural states”

which groups different secondary structures into three groups: alpha helix (H), beta sheet(E), and neither (C or -).
Currently, the training process uses three numerical features to represent each amino acid in the training sequences: average neighboring helix score (avgH), average neighboring sheet score (avgE) and average neighboring hydrophobicity score (avgP). avgH and avgE are calculated on training-set basis. Upon reading in a subset of sequences from the TD9078 training dataset, each of the 20 amino acids are assigned "helix score" and "sheet score" based on the frequency of each amino acid forming the two secondary structures in the test sequence set. The detailed method of calculating the helix and sheet score is outlined in Wang, et al (2004). avgH and avgE are calculated by using the helix and sheet scores assigned for each amino acid when the test sequence dataset is pre-processed. The data pre-processing is performed by iterating through each amino acid in the test dataset sequences with a window of a particular length. The detailed method of calculating the avgH and avgE within a window is outlined in Firdaus (2013). Calculation of avgP uses a predetermined set of hydrophobicity scores for each of the 20 amino acids called Kyle-Doolittle scale,[5] and follows an aforementioned process of iterating through the amino acids in the training sequences with a window.

p3Predict.py

p3Predict.py reads in a set of amino acid sequences and uses the trained model to predict the amino acids within the sequence set that are either alpha-helix or beta-sheet forming. The program currently reads in 12 randomly selected amino acid sequences (located in "p-cubed/StructurePrediction/test" directory) from RS126 protein sequence dataset, saved in .concise file format [6].
Upon predicting the secondary structure for the 12 test sequences, the program displays prediction statistics such as percentage of correct predictions for each secondary structure.
Finally, the program calls editPDB() function of the p3editPDB module which generates a PDB file for one of the test sequences (currently the protein "1eca") to be used for the 3D protein structure visualization.

3. PDB File Modification

p3editPDB.py

This file retrieves all the rs126FileNames from the folder and writes them into a file called rs126FileNames.txt. The next thing it does is open a rs126FileNamesOnly.txt file and remove the ".concise" from the lines, then write the first four characters of each of the new file names into a text file called rs126FileNamesOnly.txt. The four character pdb file names are stored into a list to be utilized by Biopandas[6] which can be obtained from

http://dx.doi.org/10.21105/joss.00279. We had to restrict the file names to four characters to meet the requirement of the biopandas.pdb module. The Biopandas module was imported into Spyder for retrieving the respective PDB file for a protein from the web and to allow us to save a the final PDB file to be used by the ProteinViewer module. Taking these file names from rs126, we combined them with a numeric key and placed these into a dictionary so that the user is able to select which PDB file they need to retrieve to match the protein that was initially predicted using the p3predict.py file. The requested PDB file was filtered for only the ATOM and HETATM records (which are required for visualization) and was stored under the name final.PDB in the user's current directory. The rest of the commands involved using the amino acid sequence and secondary structure sequence input to append the necessary information regarding secondary structure in the final.PDB file. The file was initially opened and the index was set to 1. We also imported the regex function into Spyder. Through the use of a variety of for loops which iterate through the aminoAcidSeq or secStrSeq we generated the necessary information for secondary structure visualization using PV.js. For example, the first column required is the recordName which indicates the type of secondary structure a protein contains (helix, sheet). We designed a for loop to iterate through the secondary structure prediction sequence and print "HELIX" any time a new series of H's was found. A similar logic was used to fill in the nine other columns (which included 1. Record Name (HELIX), 2. serNum, 4. initResName, 5. initChainID, 6. initSeqNum, 7. endResName,
9. endSeqNum, 10. helixClass and 14. lengthrequired) for the final.pdb file.
For initChainID and helixClass, we assumed the values were assumed to be "A" and "1" respectively since these values were most often found for these columns in the sample PDB files within our sample dataset. It is also currently not possible to extract initChainID and helixClass from the predicted sequence and predicted secondary structure.
In order to format the columns with the appropriate spacing we used the ljust and rjust commands. The write command was used to write the commands into the final.pdb file and the file was then closed.

ProteinViewer

ProteinViewer module renders the PDB file generated by p3editPDB module, along with the original PDB file of "1eca" on a simple HTML page, and visualizes 3D protein structures based on the PDB file contents. The module utilizes a JavaScript library called PV.js [7]. Along with the minified version of the PV.js library, the module contains an HTML file which was scripted based on an example usage demonstrated in the PV.js website [7].

Using P3 Modules

1. Running Environment and Required Python Modules

Environment: This program runs on Python version 3.4.0 or higher. The program was developed and tested on Windows 8 and 10 operating systems.

In order to run the p3 modules, the following publically available Python modules are required:

- NumPy version 1.8.2 or higher
- SciPy 0.13.3 or higher
- joblib 0.13.2
- Biopandas 0.2.4

2. How to obtain the motif details from the training dataset

From the command line cd into the directory that contains the StructureExplorer.py and rs126 folder. Execute the program with the command Python StructureExplorer.py, the following information will be displayed:

Counts for each structure
COIL COUNT: HELIX COUNT: SHEET COUNT:

Average amino acid length of structure
AVG_LEN_COIL:
AVG_LEN_HEL:
AVG_LEN_SHE:

Average hydrophobicity score for each structure
COIL-HYDRO:
HELIX HYDRO:
SHEET HYDRO:

3. How to train the machine learning model.

In order to train a SVM model;

1. Open p3Train.py file which is located in “p-cubed/StructurePrediction/” directory with a text editor.

2. On line 16 of the program, edit the number of amino acid sequences to be used for the training (the second parameter of the getTD9078() function). Default is set to the first 1000 sequences stored in td9078.csv file.

3. On line 39 of the program, edit the output joblib file name.

4. From the command prompt, navigate to “p-cubed/StructurePrediction/” directory.

5. Execute the program with the following command:

```
Python p3Train.py
```

   Once the training is complete, the program displays “training complete” message. Please note that the training time depends on the total number of amino acids contained in the training sequences.

6. The trained model will be saved as a joblib file in “p-cubed/StructurePrediction/models/” directory.

4. How to predict secondary structures of unknown amino acid sequence.

In order to predict secondary structures of previously unknown amino acid sequence:

1. Open p3Predict.py file which is located in “p-cubed/StructurePrediction/” directory.

2. On line 18, change the number of amino acid sequences (the second parameter of the getTD9078() function) to the same number of sequences that was used for the model training.

3. On line 24, specify the training model to be used for prediction by changing the parameter for the load() function.

4. As discussed in the “Module Overview” section, p3Predict.py currently uses 12 randomly selected amino acid sequences from RS126 dataset as test cases.

5. Execute the program with the following command:

```
Python p3Predict.py
```

6. Upon completing the prediction process for the 12 amino acid sequences, the program displays prediction statistics as seen below:

```
Total number of residues: 2033
Total number of Helix residues: 647
Total number of Sheet residues: 362
Total number of correct predcition: 1267
Total number of correct helix prediction: 469
Total number of correct sheet prediction: 3
Pct total correct prediction: 62.32169208066897
Pct correct helix prediction: 72.48840803709427
Pct correct sheet prediction: 0.8287292817679558
```

7. The program also calls the editPDB() function in p3editPDB module to generate a PDB file of “1eca” protein with predicted secondary structures as “final.pdb” in p-cubed/ProteinViewer/ directory.

5. How to visualize the 3D protein structure.

Upon running p3Predict.py, a PDB file of “1eca” protein with predicted secondary structure information is generated as “final.pdb” in “p-cubed/ProteinViewer/” directory. The 3D structure of the protein can be viewed by using Python’s built in HTTP module with the following steps:

1. With Command Prompt, navigate to “p-cubed/ProteinViewer/” directory.

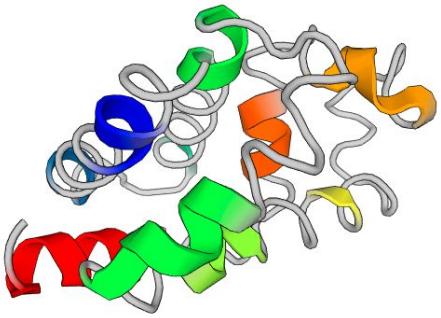
2. Create a simple web server by typing the following command:

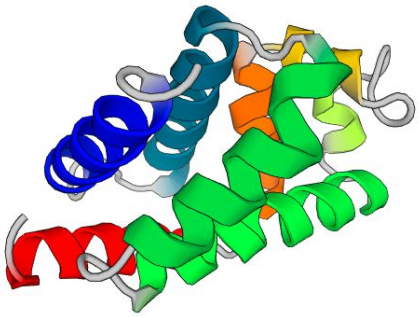
```
Python -m http.server
```

3. Open a WebGL-enabled web browser and go to localhost:8000.

4. A simple html page will display the 3D structure of 1eca protein, highlighted with the predicted secondary structures, along with the actual structure of 1eca protein obtained from the PDB website as shown below:

Predicted Structure **Actual Structure**





Test Plan

1. Motif Finder

Python File Name	Function Name	Test #	Input	Expected Output/Behavior
StructureExplorer.py	motifFinder (recursive function)	1	2 arguments: the strings extracted from the .concise files, one is the structure and the other the amino acid sequence	Will produce two lists that contain the seperate structures as elements
		2	Inproperly truncated strings	RecursionError, because the base condition can never get met
	kdAnalysis	1	A string containing the elements for motifsAmino list	Will produce the hydrophobicity scores for the motifs
		2	A string contained the elements for motifs list	Produces a KeyError since the dictionary does not recognize any Key that is “-”.

2. Secondary Structure Prediction

Python File Name	Function Name	Test #	Input	Expected Output/Behavior
aminoHelper.py	calculateSecStructScore	1	A list containing a list of valid amino acid sequences and another list of corresponding secondary structure sequence.	Output; A list containing helix scores, and another list containing sheet scores for each of the 20 amino acids.
		2	A list containing a list of invalid amino acid sequences and another list of corresponding secondary structure sequence.	Output; The function will raise ValueError as invalid characters are not found in the list of valid amino acid characters.
featureHelper.py	isStructureResidue	1	Character ‘H’	Returns 1
		2	Character ‘E’	Returns 2
		3	Character ‘Z’	Returns 0
	assignAminoIdx	1	fragment: “ARNDCQ” amino list: ['A', 'R', 'N', 'D', 'C', 'Q', 'E', 'G', 'H', 'I', 'L', 'K', 'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V']	Returns [0, 1, 2, 3, ,4 5]
		2	fragment: “XZB” amino list: ['A', 'R', 'N', 'D', 'C', 'Q', 'E', 'G', 'H', 'I', 'L', 'K', 'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V']	The function throws ValueError as the characters in the fragment variable is not found in the amino list.
	getNeighboringAminos	1	fragment: “ARNDCQE” window: 7 amino list: ['A', 'R', 'N', 'D', 'C', 'Q', 'E', 'G', 'H', 'I', 'L', 'K', 'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V']	Returns [0, 1, 2, 4, 5, 6]
		2	fragment: “XZBXZBX” window: 7 amino list: ['A', 'R', 'N', 'D', 'C', 'Q', 'E', 'G', 'H', 'I', 'L', 'K', 'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V']	The function throws ValueError as the characters in the fragment variable is not found in the amino list.
	getAvgNeighboringSecStructScore	1	fragment: “ARNDCQE” window: 7 feature score: [1.8, -4.0, -3.5, -3.5, 2.5, -3.5, -3.5,	Returns -1.7

			0.4, -3.2, 4.5, 3.8, -3.9, 1.9, 2.8, -1.6, -0.8, -0.7, -0.9, -1.3, 4.2, 0.0] amino list: ['A', 'R', 'N', 'D', 'C', 'Q', 'E', 'G', 'H', 'I', 'L', 'K', 'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V']	
		2	fragment: “XZBXZBX” window: 7 feature score: [1.8, -4.0, -3.5, -3.5, 2.5, -3.5, -3.5, 0.4, -3.2, 4.5, 3.8, -3.9, 1.9, 2.8, -1.6, -0.8, -0.7, -0.9, -1.3, 4.2, 0.0] amino list: ['A', 'R', 'N', 'D', 'C', 'Q', 'E', 'G', 'H', 'I', 'L', 'K', 'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V']	The function throws ValueError as the characters in the fragment variable is not found in the amino list.
p3DataBuilder.py	buildPredictedSequence	1	result: [0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 2, 2, 2, 0, 0]]	Returns “---HHHHH----EEE--”
		2	result: [3 ,3 ,3, 6, 6, 6, 6, 7, 7, 7]]	Returns “----------”
p3DataReader	readFile	1	Correct file name	Returns content of the file
		2	Incorrect file name	The program throws FileNotFoundError.
	extractSequenceSet/extractStructure These functions are also used in StructureExplorer.py	1	File content of a concise file.	Returns a list containing string values of an amino acid sequence, and secondary structure sequence.
		2	File content of any other file format	Out of range exception
	getRS126Dataset	1	Correct directory	Returns a list that contains a list of amino acid sequences and another list of corresponding secondary structure sequence.
		2	Incorrect or non-existent directory	The program throws FileNotFoundError.
	getTD9078Dataset	1	Correct directory	Returns a list that contains a list of amino acid sequences and another list of corresponding secondary structure sequence.
		2	Incorrect or non-existent directory	The program throws FileNotFoundError.
p3Train.py	N/A	1	Running the program with correct training data directory	The program completes without error
		2	Running the program with	The program throws

			incorrect training data directory.	FileNotFoundError.
p3Predict.py	N/A	1	Running the program with correct training data directory	The program completes without error
		2	Running the program with incorrect training data directory.	The program throws FileNotFoundError.

3. PDB File Modification

Python File Name	**Function Name**	**Test #**	**Input**	**Expected Output/Behavior**
p3editPDB.py	editPDB	1	Valid sequence and file index number.	Creates final.pdb file in p-cubed/ProteinViewer directory.
		2	Invalid sequence and invalid file index number.	Throws out of range error.

Reflection on Solution

Strengths of the software

One of the strengths of our software is the modularized design. Three modules were built separately and work independently without a high degree of coupling between each module. This makes it easier to modify each module without impacting other modules.
Another strength of the software is that it manages to demonstrate the end-to-end process of protein sequence analysis, data pre-processing, making predictions on previously unknown sequences, modifying a PDB file and visualizing the 3D protein model with the predicted secondary structures. By establishing the end-to-end process, we will be able to add features and modifications to the program in the future while assessing their impact on the entire process.

Weaknesses of the software

Although our software can demonstrate the entire process of the secondary structure prediction and visualization, the current version is not versatile to be used out-of-box. For example, if the user wants to change the resources and assets used for the software such as the training dataset, trained models, or a pdb file to be used for visualization, it needs to be modified programmatically. Currently, the implemented modules are not designed to handle changes in these resources, files and assets without changes in the actual program files.
In addition, the current version of the software provides very limited usability, as all the modules need to be accessed and executed through the command line.

Things to be done differently for the second time

Although our software has a limited usability and lacks versatility to be used out-of-box, the modular approach in developing the software worked well in terms of work distribution and maintaining a steady project progress. However, we were unable to perform thorough testing on all the functions and features in the software due to lack of time. In the future, we would like to allocate a series of testing periods each time a feature is implemented, so that the software is tested as it is being developed, instead of performing all the testing tasks at the end of the development phase.

References

1. Cortes, Corinna; Vapnik, Vladimir N. (1995). "Support-vector networks". *Machine Learning*. 20 (3): 273–297. CiteSeerX 10.1.1.15.9362. doi:10.1007/BF00994018.
2. Protein Secondary Structure - Curated dataset for protein secondary structure prediction. Obtained from https://www.kaggle.com/alfrandom/protein-secondary-structure.
3. Wang, LH., et al. (2004). “Predicting Protein Secondary Structure by a Support Vector Machine Based on a New Coding Scheme”. *Genome Informatics* 15(2): 181–190.

4. Firdaus, Syeda Nadia. (2013) “Protein Structural Class Prediction Using Predicted Secondary Structure And Hydropathy Profile”. MSc thesis. Ryerson University Library. Obtained from https://digital.library.ryerson.ca/islandora/object/RULA:2841.
5. Kyte J, Doolittle RF (May 1982). "A simple method for displaying the hydropathic character of a protein". *J. Mol. Biol.* 157 (1): 105–32. CiteSeerX 10.1.1.458.454. doi:10.1016/0022-2836(82)90515-0. PMID 7108955.
6. Sebastian Raschka. Biopandas: Working with molecular structures in pandas dataframes. *The Journal of Open Source Software,* 2(14), jun 2017. doi: 10.21105/joss.00279. URL http://dx.doi.org/10.21105/joss.00279.
7. Rost and Sander dataset (RS126). Obtained from http://www.compbio.dundee.ac.uk/jpred/legacy/data/pred_res/
8. Marco Biasini, . (2015, July 21). pv: v1.8.1 (Version v1.8.1). Zenodo. http://doi.org/10.5281/zenodo.20980.