BBM411/AIN411: Fundamentals of (Introduction to) Bioinformatics

(Fall 2023)

Assignment 2

Question 1

A) The Ramachandran plot graphically represents the ψ and ϕ angles of amino acid residues, essential for predicting and understanding the protein's structure, highlighting possible conformations and aiding in the study of protein folding and function.

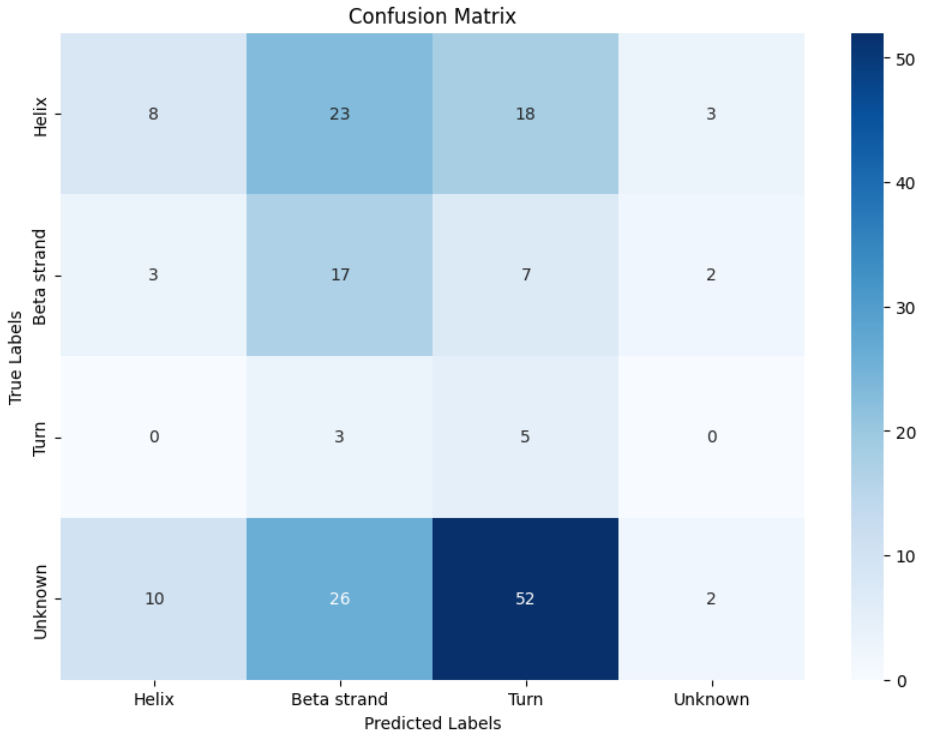
B) Protein structure formation involves various forces: Hydrogen bonds stabilize secondary structures; ionic and hydrophobic interactions, along with Van der Waals forces, influence tertiary structures; disulfide bonds and metal coordination further solidify the structure.

C) Homology in biomolecular sequences refers to sequences deriving from a common ancestor, used to infer evolutionary relationships, differing from mere sequence similarity which doesn't always imply common ancestry.

D) RNA-Seq analyzes gene expression by sequencing RNA extracted and converted to cDNA from a sample, while DNA sequencing, cheaper due to technological advances, PCR amplification, and scalability, offers a broader application in genomic research.

Question 2

- **Helix (H)**
  - Precision: 38.10%
  - Recall: 15.38%
  - F1-Score: 21.92%
- **Beta strand (E)**
  - Precision: 24.64%
  - Recall: 58.62%
  - F1-Score: 34.69%
- **Turn (T)**
  - Precision: 6.10%
  - Recall: 62.50%
  - F1-Score: 11.11%



- **Unknown (U)**
  - Precision: 28.57%
  - Recall: 2.22%
  - F1-Score: 4.12%
- **Overall Accuracy**: 17.88%

1. **Prediction of Alpha Helices and Beta Sheets**:

The **predict_alpha_helices** and **predict_beta_sheets** functions attempt to predict the locations of alpha helices and beta strands, respectively, by checking if a sequence segment has a propensity above the threshold and then extending these regions to the left and right until the propensities fall below the threshold.

2. **Extension of SSE Regions**:

Inside both **predict_alpha_helices** and **predict_beta_sheets** functions, the **extend_helix** and **extend_sheet** methods implement the logic for extending the regions of identified helices and sheets.

3. **Prediction of Turns**:

The **predict_turns** function looks for turns in the protein sequence using a calculated bending propensity (P_bend) and the average propensity to form turns (P_turn_avg).

```
def predict_alpha_helices(sequence, chou_fasman_table):
    alpha_helices = []

    def is_helix_forming(segment):
        return sum(chou_fasman_table[residue]['P_a'] > 1 for residue in segment) >= 4

    def extend_helix(start, end):
        # Extend to the right
        right_pos = end + 1
        while right_pos != sequence_length:
            counter_right = sum(chou_fasman_table[sequence[k]]['P_a'] < 1 for k in range(max(0, right_pos - 3), right_pos+1))
            if counter_right == 4:
                end = right_pos - 4
                break
            right_pos += 1

        # Extend to the left
        left_pos = start - 1
        while left_pos != -1:
            counter_left = sum(chou_fasman_table[sequence[k]]['P_a'] < 1 for k in range(left_pos, min(sequence_length, left_pos + 4)))
            if counter_left == 4:
                start = left_pos + 4
                break
            left_pos -= 1

        return start, end

    for i in range(sequence_length - 5):
        segment = sequence[i:i+6]
        if is_helix_forming(segment):
            start, end = extend_helix(i, i + 5)
            alpha_helices.append((start, end))

    return alpha_helices
```

```
def predict_beta_sheets(sequence, chou_fasman_table):
    beta_sheets = []

    def is_sheet_forming(segment):
        return sum(chou_fasman_table[residue]['P_b'] > 1 for residue in segment) >= 4

    def extend_sheet(start, end):
        # Extend to the right
        right_pos = end + 1

        while right_pos != sequence_length:
            counter_right = sum(chou_fasman_table[sequence[k]]['P_b'] < 1 for k in range(max(0, right_pos - 3), right_pos+1))
            if counter_right == 4:
                end = right_pos - 4
                break
            right_pos += 1

        # Extend to the left
        left_pos = start - 1
        while left_pos != -1:
            counter_left = sum(chou_fasman_table[sequence[k]]['P_b'] < 1 for k in range(left_pos, min(sequence_length, left_pos + 4)))
            if counter_left == 4:
                start = left_pos + 4
                break
            left_pos -= 1

        return start, end

    for i in range(sequence_length - 5):
        segment = sequence[i:i+6]
        if is_sheet_forming(segment):
            start, end = extend_sheet(i, i + 5)
            beta_sheets.append((start, end))

    return beta_sheets
```

```
def predict_turns(sequence, chou_fasman_table):
    turns = []

    for i in range(sequence_length - 3):
        segment = sequence[i:i+4]
        P_bend = chou_fasman_table[segment[0]]['f_i'] * chou_fasman_table[segment[1]]['f_i1'] * chou_fasman_table[segment[2]]['f_i2'] * chou_fasman_table[segment[3]]['f_i3']
        P_turn_avg = sum([chou_fasman_table[residue]['P_turn'] for residue in segment]) / 4
        sum_P_turn = sum([chou_fasman_table[residue]['P_turn'] for residue in segment])
        sum_P_alpha = sum([chou_fasman_table[residue]['P_a'] for residue in segment])
        sum_P_beta = sum([chou_fasman_table[residue]['P_b'] for residue in segment])

        if  (P_bend > 0.000075) and (P_turn_avg  > 1) and (sum_P_turn > sum_P_alpha) and (sum_P_turn > sum_P_beta):

            turns.append((i, i + 3))

    return turns
```

4. **Finding Overlaps**:

The **find_overlaps** function is used to detect overlaps between predicted alpha helices and beta sheets. It checks for positions in the sequence where both alpha and beta structures are predicted to occur.

```
def find_overlaps(alpha_helices, beta_sheets, sequence_length):
    alpha_booleans = [False] * sequence_length
    beta_booleans = [False] * sequence_length
    for start, end in alpha_helices:
        alpha_booleans[start:end+1] = [True] * (end - start + 1)
    for start, end in beta_sheets:
        beta_booleans[start:end+1] = [True] * (end - start + 1)
    overlaps = []
    start = -1
    end = -1
    for i in range(sequence_length):
        if alpha_booleans[i] and beta_booleans[i]:
            if start == -1:
                start = i
            end = i
        else:
            if start != -1:
                overlaps.append((start, end))
                start = -1
                end = -1
    if start != -1:
        overlaps.append((start, end))
    return overlaps
```

5. **Resolving Overlaps**:

The **compare_structure_propensity** function seems to compare the sum of propensities for helices (**sum_P_alpha**) and sheets (**sum_P_beta**) to determine which structure is more likely in the overlapping regions.

```
def compare_structure_propensity(start, end, chou_fasman_table, sequence):
    def sum_propensity(start, end, structure_type):
        if structure_type == 'alpha':
            return sum(chou_fasman_table[sequence[i]]['P_a'] for i in range(start, end + 1))
        elif structure_type == 'beta':
            return sum(chou_fasman_table[sequence[i]]['P_b'] for i in range(start, end + 1))
        else:
            return 0

    sum_P_alpha = sum_propensity(start, end, 'alpha')
    sum_P_beta = sum_propensity(start, end, 'beta')

    return 'alpha' if sum_P_alpha > sum_P_beta else 'beta'
```

6. **Consolidating Structures**:

The **consolidate_structures** function is used to merge adjacent or overlapping helices or strands into a single continuous structure.

7. **Adjusting Short Structures and Final Labeling**:

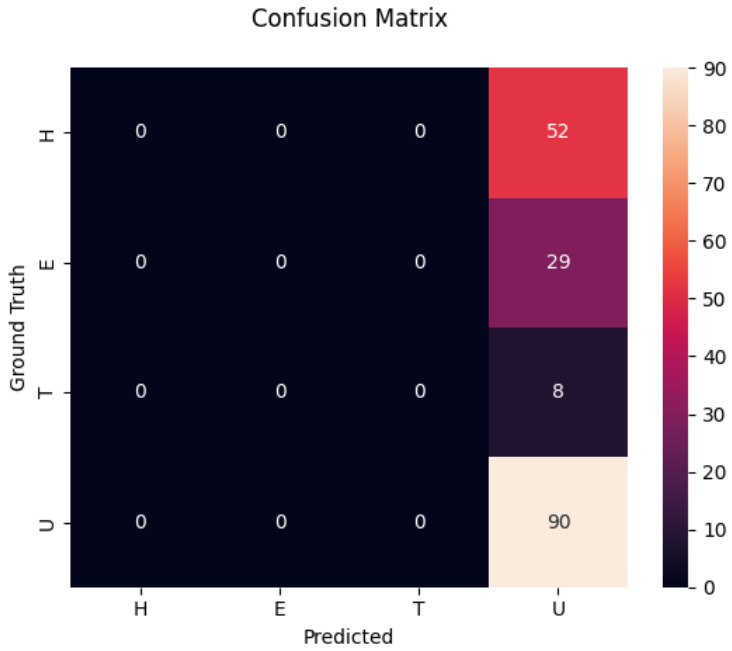
The code then iterates through the **alpha_line** and **beta_line** arrays to remove any predicted structures that are shorter than the minimum length requirement (likely 5 residues for helices and strands). The final SSE labels are then determined by combining the alpha, beta, and turn predictions, with preference given to the dominant structure in cases of overlap.

```
    alpha_line = ['-'] * sequence_length
    for start, end in alpha_helices:
        alpha_line[start:end+1] = ['H'] * (end - start + 1)
    beta_line = ['-'] * sequence_length
    for start, end in beta_sheets:
        beta_line[start:end+1] = ['E'] * (end - start + 1)
    overlap_line = ['-'] * sequence_length

    for start, end in overlaps:
        structure_type = compare_structure_propensity(start, end, chou_fasman_table, sequence)
        if structure_type == 'alpha':
            overlap_line[start:end+1] = ['H'] * (end - start + 1)
            beta_line[start:end+1] = ['-'] * (end - start + 1)
        else:
            overlap_line[start:end+1] = ['E'] * (end - start + 1)
            alpha_line[start:end+1] = ['-'] * (end - start + 1)

    counter = 0
    for i in range(len(alpha_line)):
        if alpha_line[i] == 'A':
            counter += 1
        else:
            if (counter != 0 and counter < 5):
                for j in range(counter):
                    alpha_line[i-j-1] = '-'
            counter = 0
    counter = 0
    for i in beta_line:
        if i == 'E':
            counter += 1
        else:
            if (counter != 0 and counter < 5):
                for j in range(counter):
                    beta_line[i-j-1] = '-'
            counter = 0
    for i in range(len(alpha_line)):
        if alpha_line[i] == 'H':
            overlap_line[i] = 'H'
    for i in range(len(beta_line)):
        if beta_line[i] == 'E':
            overlap_line[i] = 'E'
```

Question 3

	Prec	Recall	F1	Acc
H	0.0000	0.0000	0.0000	0.6338
E	0.0000	0.0000	0.0000	0.7563
T	0.0000	0.0000	0.0000	0.9184
U	0.5028	1.0000	0.6691	0.5028

Overall Accuracy: 0.5028



Reviewing the confusion matrix, Model shows a strong bias towards predicting the 'U' class, irrespective of the actual secondary structure present. Notably, the model did not correctly predict any instances of helices, strands, or turns, as indicated by the zero values across the precision, recall, and F1-score metrics for H, E, and T. For the 'U' class, while the recall was perfect (1.0000), indicating that all actual 'U' instances were predicted as 'U', the precision was only about 50%, which implies that only half of the 'U' predictions were accurate.

The overall accuracy of the model was calculated to be 0.5028. However, this metric does not reflect the model's ability to distinguish between the different classes of secondary structures, as it is disproportionately influenced by the high frequency of 'U' predictions.

Considering these findings, it is clear that the model's current predictive capability for H, E, and T structures is lacking. The performance for 'U' suggests that while the model can recognize sequences that do not conform to the typical secondary structures, it struggles to identify and categorize the conventional secondary structures accurately.

To enhance the model's predictive performance:

- **Data Balancing**: Adjust the training dataset to ensure a more balanced representation of each secondary structure class.
- **Model Optimization**: Experiment with the model's architecture and parameters to improve its ability to capture the distinctive patterns of secondary structures.
- **Feature Engineering**: Integrate additional features such as amino acid physicochemical properties and evolutionary profiles to provide the model with richer contextual information.

c) It didn't do as well as Chou-Fasman. It was good at predicting 'U' (unknown areas) but didn't correctly predict helices (H), strands (E), or turns (T) even once.

The Chou-Fasman model is simple and uses known rules about amino acids to guess structures. It seems to work better. Maybe our tool needs a better way to learn from data or we need to check if we have enough different examples in our training set.

To make our tool better, we could:

- Use a more balanced set of examples for training.
- Try changing the settings to see if it can learn better.
- Use more complex models like LSTM, which are good at learning from sequences.
- Add more information to our data, like details about the amino acids that might help the model make better guesses.