BBM411/AIN411 Fundamentals of Bioinformatics (Fall 2023)

Assignment 2

Çağrı Çakıroğlu

QUESTION 1

1-) What is represented in the 2-axis of the Ramachandran plot, what kind of information they provide, and why this is important?

Answer:

In a Ramachandran plot, one axis shows the phi angle and the other shows the psi angle of proteins. These angles tell us how proteins twist and fold. This is important because the shape of a protein affects how it works in our bodies.

2-) What are forces acting on atoms of amino acids that cause the formation of secondary and tertiary structures of proteins?

Answer:

In proteins, the secondary structures, like helices and sheets, are made mainly because of hydrogen bonds. For the tertiary structure, where the protein folds into a complex 3D shape, forces like ionic bonds, water-fearing (hydrophobic) interactions, and sulfur-sulfur (disulfide) bridges are important. These forces help the protein to get into the right shape to do its job in the body.

3-) ***Define homology in terms of biomolecular sequence similarities.***

Answer:

Homology in biomolecules, like DNA, RNA, or proteins, means that two sequences are similar because they come from a common ancestor. This similarity can be in their DNA, RNA, or amino acid sequences. When two sequences are homologous, it usually means they have a similar structure and function because they share a common history.

4-) ***Give one example way to extract biological data (a.k.a. transforming a biological sample into data) by briefly explaining it. Which one is cheaper, sequencing DNA or protein, why?***

Answer:

DNA sequencing is a way to turn a sample, like blood, into data by reading the order of the DNA parts (A, T, C, G). It's usually cheaper than protein sequencing because the technology for DNA is better and it's easier to do. Protein sequencing is more complex and costly because proteins have more complicated structures.

QUESTION 2

True / Predicted	H	E	T	U
H	**0**	**32**	**17**	**0**
E	**0**	**19**	**6**	**0**
T	**0**	**3**	**5**	**0**
U	**0**	**0**	**0**	**0**

	Prec	Recall	F1	Acc
H	0	0.0	0	0.4024
E	0.3519	0.76	0.481	0.5
T	0.1786	0.625	0.2778	0.6829
U	0	0	0	1.0

SSE HITS:

-RegionFinder

```python
def RegionFinder(property_element: str, protein_sequence: str, amino_acid_properties: Dict[str, Dict[str, float]]) -> List[Tuple[int, int]]:
    """
    Identifies regions in a protein sequence where a specific property exceeds a threshold.

    This function scans a given protein sequence to find regions where a specified property
    of amino acids, such as hydrophobicity or charge, exceeds a predefined threshold. This
    can be useful in protein analysis for identifying domains or motifs of interest.

    Args:
        property_element (str): The property to evaluate (e.g., hydrophobicity, charge).
        protein_sequence (str): The amino acid sequence of the protein.
        amino_acid_properties (dict): A dictionary mapping each amino acid to its properties,
                                      which are themselves represented as a dictionary.

    Returns:
        List[Tuple[int, int]]: A list of tuples, where each tuple contains the start and end
                               indices (0-based) of a region in the protein sequence where
                               the specified property exceeds a threshold.


    """

    def expand_region(start_idx: int, threshold: float) -> int:
        """Expands the region to include adjacent amino acids that meet the threshold."""
        # Implementation of expand_region function
        pass

    region_threshold = 1.0
    minimum_matching_amino_acids = 4
    region_window_size = 6
    sequence_length = len(protein_sequence)
    identified_regions = []

    for start_idx in range(sequence_length - region_window_size + 1):
        if sum(amino_acid_properties[protein_sequence[i]][property_element] > region_threshold
               for i in range(start_idx, start_idx + region_window_size)) >= minimum_matching_amino_acids:
            end_idx = expand_region(start_idx, region_threshold)
            identified_regions.append((start_idx, end_idx))

    return identified_regions
```

OVERLAP TREATMENT:

-MarkProteinRegions

```python
def mark_protein_regions(protein_sequence: List[str], target_regions: List[Tuple[int, int]], region_marker: str) -> List[str]:
    """
    Annotates specified regions in a protein sequence with a given marker.

    This function allows for the marking of specific regions within a protein sequence.
    It is useful in bioinformatics for highlighting domains, active sites, or other
    regions of interest in protein sequences.

    Args:
        protein_sequence (List[str]): The amino acid sequence of the protein, represented as a list of single-letter codes.
        target_regions (List[Tuple[int, int]]): A list of tuples, each indicating the start and end indices (0-based)
                                                of regions to be marked.
        region_marker (str): The marker symbol used to annotate the specified regions.

    Returns:
        List[str]: The annotated protein sequence, with specified regions marked by the given marker.

    """

    marked_sequence = protein_sequence.copy()
    for start_idx, end_idx in target_regions:
        for position in range(start_idx, end_idx + 1):
            if 0 <= position < len(protein_sequence):
                marked_sequence[position] = region_marker
    return marked_sequence
```

-Find_overlapping_regions

```python
def find_overlapping_regions(alpha_marked_sequence: List[str], beta_marked_sequence: List[str]) -> List[Tuple[int, int]]:
    """
    Identifies overlapping regions between two sequences marked with specific characters.

    This function is designed to find overlapping regions in two protein sequences or similar
    biological sequences, where specific regions have been marked with distinct characters
    ('a' for alpha regions in one sequence and 'b' for beta regions in the other). It is
    particularly useful in comparative analysis of protein domains or structural features.

    Args:
        alpha_marked_sequence (List[str]): The sequence marked with 'a' to indicate alpha regions.
        beta_marked_sequence (List[str]): The sequence marked with 'b' to indicate beta regions.

    Returns:
        List[Tuple[int, int]]: A list of tuples, each representing the start and end indices
                               (0-based) of overlapping regions between the two sequences.


    """

    overlapping_regions = []
    in_overlap = False
    start_idx = 0

    for position, (alpha_char, beta_char) in enumerate(zip(alpha_marked_sequence, beta_marked_sequence)):
        is_overlap = alpha_char == 'a' and beta_char == 'b'
        if is_overlap and not in_overlap:
            start_idx = position
            in_overlap = True
        elif not is_overlap and in_overlap:
            end_idx = position - 1
            overlapping_regions.append((start_idx, end_idx))
            in_overlap = False

    # Capture overlap at the end of the sequences
    if in_overlap:
        overlapping_regions.append((start_idx, len(alpha_marked_sequence) - 1))

    return overlapping_regions


# Assuming alpha_H and beta_E are already defined
overlapping_regions = find_overlapping_regions(alpha_H, beta_E)
```

-create_combined_ab_prediction

```python
def create_combined_ab_prediction(alpha_marked_sequence: List[str], beta_marked_sequence: List[str]) -> List[str]:
    """
    Generates a combined sequence prediction integrating alpha and beta region markings.

    This function is designed for use in protein structure analysis, where alpha helices and
    beta sheets are marked separately in two sequences. It combines these two sequences into
    one, where beta regions take precedence over alpha regions in the combined prediction.

    Args:
        alpha_marked_sequence (List[str]): The sequence marked with 'a' to indicate alpha helices.
        beta_marked_sequence (List[str]): The sequence marked with 'b' to indicate beta sheets.

    Returns:
        List[str]: A combined sequence with each position marked as 'a' for alpha helix, 'b' for
                   beta sheet, or '_' for unmarked regions.


    """

    combined_prediction = [
        'b' if beta_char == 'b' else alpha_char
        for alpha_char, beta_char in zip(alpha_marked_sequence, beta_marked_sequence)
    ]

    return combined_prediction
```

-refine_ab_prediction

```python
def refine_ab_prediction(combined_prediction_sequence: List[str]) -> List[str]:
    """
    Enhances a combined alpha-beta prediction sequence by grouping and filtering small regions.

    This function is useful in protein structure prediction where small, likely inaccurate
    predictions of alpha helices and beta sheets need to be filtered out. It groups consecutive
    regions of the same type and replaces any small group (less than 5 residues) with unmarked
    regions, represented by '_'.

    Args:
        combined_prediction_sequence (List[str]): The combined sequence with 'a' for alpha helices,
                                                  'b' for beta sheets, and '_' for unmarked regions.

    Returns:
        List[str]: A refined sequence where small alpha or beta regions are replaced with
                   unmarked regions to improve the prediction's accuracy.


    """

    # Group the sequence and calculate cumulative lengths
    grouped_sequence = [[region, len(list(group))] for region, group in groupby(combined_prediction_sequence)]
    for i in range(1, len(grouped_sequence)):
        grouped_sequence[i][1] += grouped_sequence[i-1][1]

    # Refine the sequence by filtering out small regions
    for i, (region_type, cumulative_length) in enumerate(grouped_sequence):
        start_index = grouped_sequence[i-1][1] if i > 0 else 0
        region_length = cumulative_length - start_index

        # Replace small regions (less than 5 residues) with '_'
        if region_type != '_' and region_length < 5:
            for position in range(start_index, cumulative_length):
                combined_prediction_sequence[position] = '_'

    return combined_prediction_sequence
```

-indetify_turns_in_sequence

```python
def identify_turns_in_sequence(protein_sequence: str, amino_acid_properties: Dict[str, Dict[str, float]], alpha_beta_prediction: List[str]) -> Tuple[List[str], List[str]]:
    """
    Identifies turns in a protein sequence and updates the alpha-beta prediction to include turns.

    This function is intended for protein structure analysis. It uses a set of properties for
    each amino acid to identify turns in the protein sequence. A turn is marked when certain
    criteria, based on the properties of a group of four amino acids, are met. The function
    updates the alpha-beta prediction and also creates a sequence marking the turns.

    Args:
        protein_sequence (str): The amino acid sequence of the protein.
        amino_acid_properties (Dict[str, Dict[str, float]]): A dictionary mapping each amino acid
                                                             to its properties.
        alpha_beta_prediction (List[str]): The existing alpha-beta prediction sequence, with 'a'
                                           for alpha helices, 'b' for beta sheets, and '_' for
                                           unmarked regions.

    Returns:
        Tuple[List[str], List[str]]: A tuple containing the updated alpha-beta prediction sequence
                                     and a new turns sequence marked with 'T' for turns.


    """

    turns_sequence = ['_'] * len(protein_sequence)
    for start in range(len(protein_sequence) - 3):
        total_alpha, total_beta, total_turns, bend_factor = 0, 0, 0, 1

        # Calculate properties for a window of 4 amino acids
        for offset in range(4):
            amino_acid = protein_sequence[start + offset]
            total_alpha += amino_acid_properties[amino_acid]['Pa']
            total_beta += amino_acid_properties[amino_acid]['Pb']
            total_turns += amino_acid_properties[amino_acid]['Pt']
            bend_factor *= amino_acid_properties[amino_acid][f'f{offset}']

        # Determine if the window represents a turn
        if bend_factor > 0.000075 and total_turns / 4 > 1 and total_turns > total_alpha and total_turns > total_beta:
            for position in range(start, start + 4):
                alpha_beta_prediction[position] = 't'
                turns_sequence[position] = 'T'

    return alpha_beta_prediction, turns_sequence
```

QUESTION 3

output:

O00762 UBE2C_HUMAN
MASQNRDPAATSVAAARKGAEPSGGAARGPVGKRLQQELMTLMMSGDKGISAFPESDNLFKWVGTI
HGAAGTVYEDLRYKLSLEFPSGYPYNAPTVKFLTPCYHPNVDTQGNICLDILKEKWSALYDVRTILLSIQ
SLLGEPNIDSPLNTHAAELWKNPTAFKKYLQETYSKQVTSQEP H 9..10; H 14..16; H 20..21; H
41..45; H 50..53; H 83..85; H 116..118; H 153..155; H 176..179 E 4..5; E 25..26; E 31..33; E 60..62;
E 66..67; E 71..73; E 77..79; E 89..91; E 95..98; E 106..108; E 126..128; E 131..133; E 137..139; E
143..144; E 148..149; E 160..162; E 165..167; E 170..172 T 34..36; T 54..56; T 99..101; T 109..111;
T 119..121 Probability of path: 2^(-1015.9434025928475)

True / Predicted	H	E	T	U
H	11	19	3	19
E	6	10	4	9
T	0	3	3	2
U	12	18	5	55

```
      Prec          Recall        F1            Acc
H     0.3793        0.2115        0.2716        0.6704
E     0.2           0.3448        0.2532        0.6704
T     0.2           0.375         0.2609        0.905
U     0.6471        0.6111        0.6286        0.6369

Overall Accuracy: 0.4413
```

For the HMM Predictor Model, we have the following scores:

- Precision (Prec): This tells us how many of the predicted structures of a certain type were actually correct. For instance, when the HMM model predicted an alpha-helix (H), it was correct about 37.93% of the time.

- Recall: This score indicates how well the model captured all the actual structures. For alpha-helices, the model only identified about 21.15% of them correctly.

- F1 Score: This is the harmonic mean of precision and recall, giving us a balance between the two. It's especially useful when the costs of false positives and false negatives are roughly equivalent. The HMM model scored 0.2716 for alpha-helices, suggesting there's a need for improvement as the F1 score is quite low.

- Accuracy (Acc): This metric assesses the overall correctness of predictions, not just for one structure type but across all categories. The HMM model had a high accuracy for turns (T) at 90.5%, indicating it's quite adept at predicting turns.

The Overall Accuracy of the HMM model is 0.4413, meaning that when it makes a prediction about any element of the protein structure, it is correct about 44.13% of the time.

In contrast, the Chou-Fasman Method shows a different pattern:

- Precision: It didn't manage to correctly predict any alpha-helices, indicated by a precision score of 0 for H. However, it did better with beta-strands (E), with a precision of 35.19%.

- Recall: Its recall score for beta-strands is quite high at 76%, suggesting that when beta-strands are present, the Chou-Fasman method can detect them three-quarters of the time.

- F1 Score: The best F1 score for the Chou-Fasman method is for beta-strands at 0.481, which is a moderate score and indicates a better balance between precision and recall for this structure type compared to others.

- Accuracy: The method has a high accuracy score for turns (T) at 68.29%. But, this is somewhat misleading because the Chou-Fasman method didn't predict any unstructured regions (U), which inflates its accuracy for other categories due to the way accuracy is calculated.

The Overall Accuracy of the Chou-Fasman method is lower at 0.2927, meaning it's correct about 29.27% of the time across all predictions.

To summarize, the HMM model has a more balanced performance across different types of protein structures but still has room for improvement, especially for helices (H) and sheets (E). The Chou-Fasman method seems particularly weak in predicting helices but shows some strength in predicting sheets.

Improving these models could involve:

- Enhancing the training data with more diverse examples to help the model learn more comprehensive patterns.

- Adjusting the algorithms to better capture the characteristics of different structure types.

- Incorporating additional biological insights that may influence protein structure formation.