Pairwise Sequence Alignment of APOE and TREM2 gene in Alzeihmer's pathway

Project Report

Presented To

Professor Aarohi Chopra

Department of Computer Science

San José State University

By

Hina Dawar and Saumi Shah

CS22B

May 2024

TABLE OF CONTENTS

Abstract

This study explores the functional relationship between the APOE and TREM2 genes in Alzheimer's disease (AD) progression. The FileReader class processes genomic data using Python, while the DotPlot class depicts conserved regions. Furthermore, the GlobalAlignment class measures sequence similarity, providing insight into AD pathogenesis. The results highlight the important sequence conservation between TREM2 and APOE, indicating their role in AD pathogenesis.

Keywords: Dot plots, Global alignment, Alzheimer’s, Sequence conservation

I. Introduction

Alzheimer’s disease(AD) has devastated millions of memories and dismantled lives. AD is a neurodegenerative disorder that progresses slowly with age and causes dementia (memory loss) and loss of thinking, learning, and organization skills. [1] Though there is currently no cure, understanding the mechanisms behind AD symptoms can aid in the development of treatments. APOE, a major human gene associated with dementia in AD, transports cholesterol in the brain, maintains brain homeostasis, and ensures synaptic (connections between brain nerve cells stability [2]. Lipid-rich particles produced by APOE interact with TREM2 receptors found in the brain's immune cells, inhibiting its role in carrying out functions like clearing plaques and unwanted substances, eventually leading to dementia-related symptoms [2].

Literature suggests a high functional dependence between the APOE and TREM2 genes in causing Alzheimer's symptoms. Confirmation of this relationship through pairwise sequence alignment of APOE and TREM2 DNA sequences will reveal functional, structural, and evolutionary conservation if found. This report aims to confirm this relationship using dot plots and global alignment using Python. High prioritization is given to the global alignment method due to its computational complexities. Program goals, structure, execution and challenges are all discussed in the report.

Dot plots visually represent data to identify direct and inverted repeats [3], whereas global alignment yields quantitative alignment scores for measuring similarities and differences between APOE and TREM2. The FileReader class is designed to open, read, and clean data files, making instances of this class usable in the next class. DotPlot class is composed of FileReader class and relies heavily on it, to plot an image of the 2D numpy dot plot matrix for qualitative analysis of conserved regions. Finally, Dot Plot class is associated with the Global Alignment class that calculates alignment scores to assess sequence conservation. Fig 1 shows a UML diagram illustrating individual class attributes, methods, and relationships.

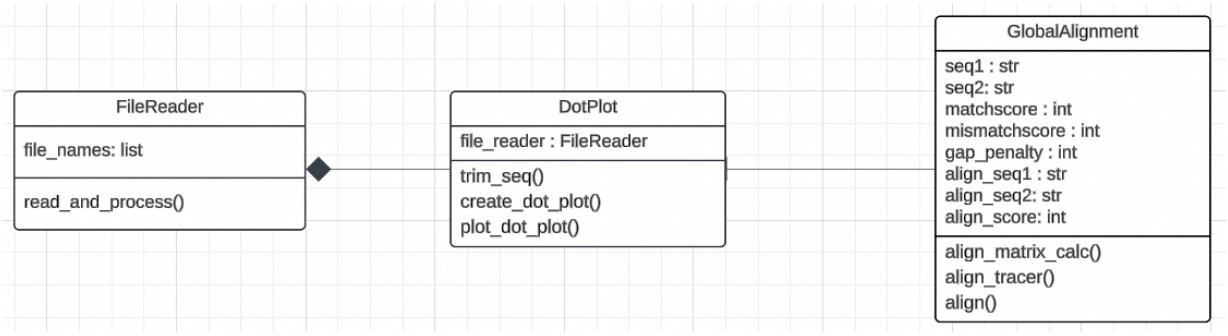

Fig 1: UML diagram illustrating methods, attributes, and relationships among all classes

II. FileReader

The FileReader class takes a list of text files as an attribute and reads and processes the file in the read_and_process() method. Metadata from each text file is removed using .pop(0), and white space (leading and trailing) is removed using .split(), so a new list of nucleotide strings is generated for each file.

III. DotPlot

This class uses the processed list of nucleotide strings from the FileReader class. Between two DNA lists, the length of the bigger list is reduced to the length of the shorter list to produce two lists of

equal length in the trim_seq() method. This ensures clarity and efficiency when running the code and interpreting the results. To ensure only two lists can be used, exception handling is performed in the create_dot_plot() method such that ValueError will be raised if more than two lists are used. This method initializes a matrix using np.zeros from the numpy library. Both rows and columns contain as many zeroes as the length of the shorter DNA lists. Upon iteration of the length of both lists, the 2D matrix is filled with a value of one, if the strings in both sequences at the same iteration match. If unmatched, the matrix remains zero as initialized. The filled matrix is then plotted using the pyplot module within the matplot library using .imshow() and the figure is labeled in the plot_dot_plot() method. Fig 2 shows the code snippet of the create_dot_plot and plot_dot_plot method.

```
    def create_dot_plot(self):
        sequences = self.file_reader.read_and_process()
        if len(sequences) < 2:
            raise ValueError("Need at least two sequences to create a dot plot.")
        seq1, seq2 = self.trim_seq(sequences[0], sequences[1])
        dot_matrix = np.zeros((len(seq1), len(seq2)), dtype=int)
        for i in range(len(seq1)):
            for j in range(len(seq2)):
                if seq1[i] == seq2[j]:
                    dot_matrix[i, j] = 1
        return dot_matrix

    def plot_dot_plot(self, dot_matrix, length):
        plt.figure(figsize=(10, 8))
        plt.imshow(dot_matrix, cmap='viridis', interpolation='nearest')
        plt.title('Dot Plot Matrix')
        plt.xlabel('Sequence 1 (Length: {})'.format(length))
        plt.ylabel('Sequence 2 (Length: {})'.format(length))
        plt.colorbar()
        plt.show()
```

Fig 2 shows the code snippet of the create_dot_plot and plot_dot_plot method.

IV. GlobalAlignment

The GlobalAlignment class performs global alignment between two sequences. Fig3 shows the constructor method initializing the class with essential parameters such as seq1, seq2, match score, mismatch score, and gap penalty. It also creates variables to store the aligned sequences (align_seq1,align_seq2), aligned marker, and alignment scores (align_score).

```
class GlobalAlignment:
    def __init__(self, seq1, seq2, match_score, mismatch_score, gap_penalty):
        self.seq1 = seq1
        self.seq2 = seq2
        self.match_score = match_score
        self.mismatch_score = mismatch_score
        self.gap_penalty = gap_penalty
        self.align_seq1 = ""
        self.align_seq2 = ""
        self.align_score = 0
```

Fig 3: Constructor method of GlobalAlignment class

```
def align_matrix_calc (self):
    k = len(self.seq1)
    l = len(self.seq2)
    res = []
    for seq in range (k+1):
         res.append([0] * (l + 1))



    for i in range(1, k + 1):
        res[i][0] = res[i - 1][0] + self.gap_penalty
    for j in range(1, l + 1):
        res[0][j] = res[0][j - 1] + self.gap_penalty


    for i in range(1, k + 1):
        for j in range(1, l + 1):
            if self.seq1[i - 1] == self.seq2[j - 1]:
                diagonal_box = res[i - 1][j - 1] + self.match_score
            else:
                diagonal_box = res[i - 1][j - 1] + self.mismatch_score
            top_box = res[i - 1][j] + self.gap_penalty
            beside_box = res[i][j - 1] + self.gap_penalty
            res[i][j] = max(diagonal_box,top_box , beside_box)

    return res
```

Fig 4. Alignment matrix class

Figure 4 shows the alignment matrix method initializes it to store alignment scores and fills it with match score = +1, mismatch = -1, and gap = -1. It calculates the length of two sequences (seq1 and seq2) and stores that value in variables k and l, respectively. Moreover, It creates an empty list called res and stores its alignment score. As per the algorithm, the scores are computed by considering the values from the box above and the box to the left, adding the gap penalty. The algorithm assigns the score from the diagonal box. If the base pairs match, the match score is added; otherwise, the mismatch score is added. According to the Needleman-Wunsch algorithm, the highest score will be selected, and then the matrix will be updated.

```
def align_tracer(self, res):
    p = len(self.seq1)
    q = len(self.seq2)

    while p > 0 or q > 0:
        if p > 0 and q > 0 and res[p][q] == res[p - 1][q - 1] + self.match_score:
            self.align_seq1 = self.seq1[p - 1] + self.align_seq1
            self.align_seq2 = self.seq2[q - 1] + self.align_seq2

            p -= 1
            q -= 1

        elif p > 0 and res[p][q] == res[p - 1][q] + self.gap_penalty:
            self.align_seq1 = self.seq1[p - 1] + self.align_seq1
            self.align_seq2 = "_" + self.align_seq2

            p -= 1

        else:
            self.align_seq1 = "_" + self.align_seq1
            self.align_seq2 = self.seq2[q - 1] + self.align_seq2

            q -= 1
```

Fig 5: Alignment tracer class to perform tracing

The align_tracer method determines aligned sequences by tracing back via the alignment matrix, as shown in Figure 5. It travels along the path with the highest alignment score, starting in the bottom-right corner and ending in the top-left corner. Until both sequences have the same length, the while loop keeps going. According to character matches or mismatches, it assigns scores and adds characters to align_seq1 and align_seq2. It adds a "-" character to align_seq2 and appends the character from seq1 to align_seq1 if there is a gap in seq1. The tracing across the alignment matrix is finished when both sequences approach zero length, achieved by iterating the loop.

```
    def align(self):
        align_matrix = self.align_matrix_calc()
        self.align_tracer(align_matrix)
        self.align_score = align_matrix[len(self.seq1)][len(self.seq2)]

# Example usage:
file_names = ['APOE.txt', 'TREM2.txt']
```

Fig 6: align method to align the DNA sequences

Figure 6 notes that the align() method regulates the alignment process by computing the alignment matrix and then tracing back to find the aligned sequences. Finally, it calculates the alignment score and gives the aligned sequences.

V. EXECUTION AND TESTING

Upload the two text files, for example, APOE.txt and TREM2.txt, using the ‘Files’ tab on the left sidebar in Google Colab. Under the file_names variable, input the list of two .txt files containing DNA sequences. Press shift + enter to execute the cell. The FileHandler, DotPlot, and GlobalAlignment classes are designed to work with individual files to produce the dot plot and global alignment scores.

VI. RESULTS

The qualitative assessment using the dot plot in Fig 8 reveals that the APOE and TREM2 genes have high sequence conservation as there are more yellow lines than in violet, representing more overall high sequence conservation than dissimilarities (in violet)

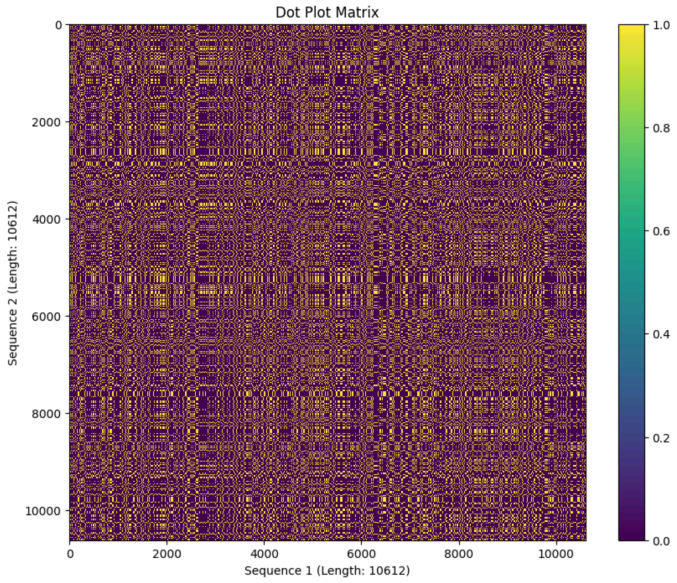

Fig 7: Dot plot between APOE and TREM2 gene

This visual representation, when followed with a quantitative score from the GlobalAlignment class, reveals more information about the alignment of the two genes. The global alignment score is 758, which indicates that the APOE and TREM 2 sequences are highly similar. This suggests that they may share functional properties that play a role in Alzheimer's disease, given the known involvement of both genes in Alzheimer's pathology through literature. Additionally, this displays the strength of pairwise sequence alignment in research.

VII. CHALLENGES

The current program takes 2 minutes and 26 seconds to run, which displays a slow run time due to code complexity. Additionally, significant run time issues were experienced, especially when using seaborn heatmaps for dot plots resulted in our RAM crashing. Plt.imshow() provided faster runtime due to low memory usage. Finally, understanding and programming global alignment without using the Biopython package was difficult, as it involved numerous iterations and calculations. SJSU BIOL 123B course notes were used to understand the global alignment matrix and calculations.

REFERENCES

[1] Y. Jung, Y. Kim, M. Bhalla, S. Lee, and J. Seo, “Genomics: New Light on Alzheimer’s Disease Research,” *International Journal of Molecular Sciences*, vol. 19, p. 3771, Nov. 2018, doi: 10.3390/ijms19123771.

[2] C. M. Wolfe, N. F. Fitz, K. N. Nam, I. Lefterov, and R. Koldamova, “The Role of APOE and TREM2 in Alzheimer′s Disease—Current Understanding and Perspectives,” *International Journal of Molecular Sciences*, vol. 20, no. 1, 2019, doi: 10.3390/ijms20010081.

[3] D. W. Mount, *Bioinformatics: sequence and genome analysis*. Cold Spring Harbor, N.Y: Cold Spring Harbor Laboratory Press, 2001