Avik Bhuiyan

CMPSC 463 (section 1)

Professor Yang

Time-Series Clustering and Segment Analysis Using Divide-and-Conquer Algorithms

Description of Project

This project focuses on unsupervised clustering of physiological time-series segments (ABP, ECG, PPG) extracted from PulseDB, using divide-and-conquer algorithms. The aim is to group similar signal segments and analyze their internal structure using:

- Divide-and-Conquer Clustering: recursive splitting based on similarity metrics.
- Closest-Pair Analysis: identify the most similar pair within each cluster to validate cohesion.
- Maximum Subarray Analysis (Kadane's Algorithm): highlight the most active or anomalous intervals in a segment.

The project emphasizes algorithmic design rather than machine learning heuristics to generate meaningful segmentation and insight from biomedical signals.

Installation and Usage

To run this project on your own machine, follow these instructions:

1. Clone the GitHub repository (https://github.com/Avik812/PulseDBClustering)
2. The project incorporates many libraries such as numpy, scipy, matplotlib. These can be installed using "*pip install numpy scipy h5py matplotlib fastdtw pandas*"
3. PulseDB_Vital data must also be downloaded (https://rutgers.app.box.com/s/ezmiotu66dtnahxoirnukn25nrcylybc)
4. Place the dataset in the data/Segments folder.
5. Run main.py to execute clustering and analysis.

Structure of Code

```
PulseDBClustering/              # project folder
├– results/
      ├– cluster_1.png.         # cluster visuals
├– data/
│  └ Segments/                  # dataset
│     ├– p0000001.mat
│     ├– p0000002.mat
│     └ …
├– src/                        # main algorithms
│  ├– clustering.py
│  ├– closest_pair.py
│  ├– kadane.py
│. ├– main.py
│  └ utils.py
└ main.py                      # main script integrating all components
```

Description of Algorithms

1. Divide-and-Conquer Clustering of Time Series Segments

```python
def divide_and_conquer_clustering(segments, threshold=10):
    """
    Recursively cluster segments using a divide-and-conquer approach based on DTW.
    """
    if len(segments) <= 1:
        return [segments]

    # Compute pairwise DTW distances
    n = len(segments)
    dist_matrix = np.zeros((n, n))
    for i in range(n):
        for j in range(i+1, n):
            a = np.array(segments[i]).flatten()
            b = np.array(segments[j]).flatten()
            dist, _ = fastdtw(a, b, dist=euclidean)
            dist_matrix[i, j] = dist
            dist_matrix[j, i] = dist

    median_dist = np.median(dist_matrix)

    # If segments are very dissimilar, split into halves
    if median_dist > threshold and n > 1:
        cluster1 = segments[:n//2]
        cluster2 = segments[n//2:]
        return divide_and_conquer_clustering(cluster1, threshold) + \
               divide_and_conquer_clustering(cluster2, threshold)
    else:
        return [segments]
```

Purpose: Identify the most similar pair of time-series segments within a cluster.

Method:

- For every pair of segments in a cluster, compute the DTW distance.
- Track the pair with the smallest distance.

Validates cluster cohesion and selects representative segments for analysis.

2. Closest Pair of Time Series Within Clusters

```python
from fastdtw import fastdtw
from scipy.spatial.distance import euclidean
import numpy as np

def closest_pair(cluster):
    """
    Find the closest pair of segments in a cluster using DTW.
    """
    min_dist = float('inf')
    pair = (None, None)
    n = len(cluster)

    for i in range(n):
        for j in range(i+1, n):
            seg_i = np.array(cluster[i]).flatten()
            seg_j = np.array(cluster[j]).flatten()
            dist, _ = fastdtw(seg_i, seg_j, dist=euclidean)
            if dist < min_dist:
                min_dist = dist
                pair = (i, j)
    return pair, min_dist
```

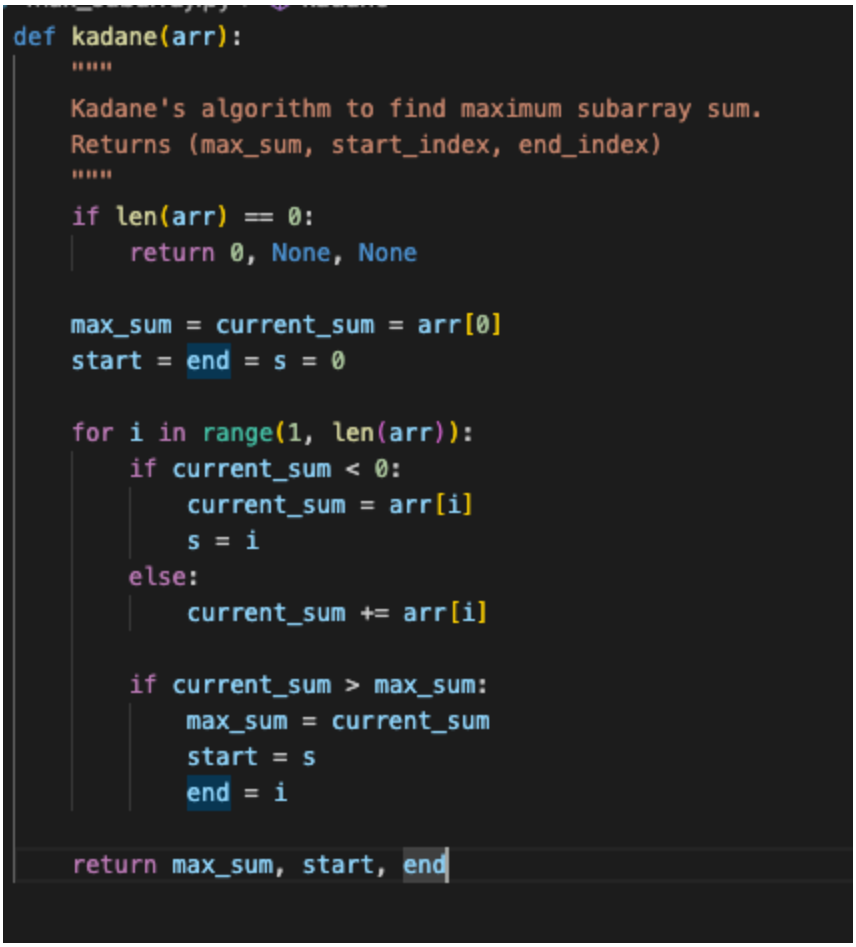
Purpose: Recursively group similar time-series segments based on Dynamic Time Warping (DTW) distances.

Method:

- Compute pairwise DTW distances between all segments.
- Calculate the median distance.
- If the median exceeds a threshold, split segments into two halves and recursively cluster each half.
- Otherwise, return the segments as a cluster.

Key Feature: Automatically forms clusters of similar segments without predefined cluster counts.

3. Maximum Subarray Analysis (Kadane's Algorithm)



```python
def kadane(arr):
    """
    Kadane's algorithm to find maximum subarray sum.
    Returns (max_sum, start_index, end_index)
    """
    if len(arr) == 0:
        return 0, None, None

    max_sum = current_sum = arr[0]
    start = end = s = 0

    for i in range(1, len(arr)):
        if current_sum < 0:
            current_sum = arr[i]
            s = i
        else:
            current_sum += arr[i]

        if current_sum > max_sum:
            max_sum = current_sum
            start = s
            end = i

    return max_sum, start, end
```

Purpose: Find the contiguous subarray with the maximum sum in a time series.

Method:

- Iterate through the array, keeping track of the current subarray sum.
- Reset the sum if it becomes negative.
- Update the maximum sum and indices when a new maximum is found.

Highlights of intervals of peak activity or anomalies in physiological signals.

Verification of Functionality (toy example)

To verify that the clustering and analysis pipeline functioned correctly before running it on the full PulseDB dataset, a toy dataset of simple, synthetic time-series signals was used.

The toy dataset contained six short signals (length = 100 samples) generated using sine and square waves with slight noise added. Three of the signals were smooth sine waves

with small phase shifts, and the remaining three were noisy square waves. This ensured two distinct waveform patterns for the clustering algorithm to detect.

Example of generated toy signals:

```
# --- Generate toy signals ---
t = np.linspace(0, 2*np.pi, 100)
toy_segments = [
    np.sin(t),
    np.sin(t + 0.2),
    np.sin(t + 0.4),
    np.sign(np.sin(t)),
    np.sign(np.sin(t + 0.2)),
    np.sign(np.sin(t + 0.4))
]
```

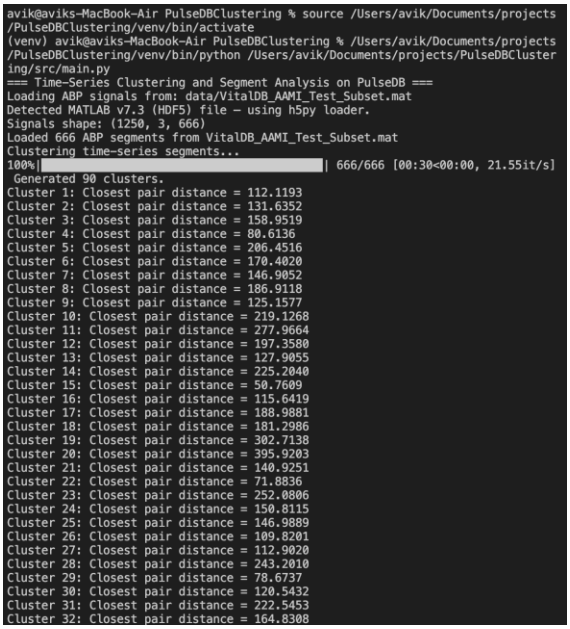
```
 Analysis complete. Plots saved in 'results/cluster_visuals/'.
avik@aviks-MacBook-Air PulseDBClustering % /Users/avik/Documents/proj
100%|
Generated 2 clusters.
Cluster 1: Closest pair distance = 1.3680
Cluster 2: Closest pair distance = 0.0000
Segment 1: Max subarray sum = 49.0000 (indices 31.510034249551143-0)
Segment 2: Max subarray sum = 46.0000 (indices 31.294670347404473-0)
Segment 3: Max subarray sum = 43.0000 (indices 30.458466303535126-0)
Saved cluster plots to results/cluster_visuals/
avik@aviks-MacBook-Air PulseDBClustering %
```

The algorithm correctly grouped the sine-based signals together and the square-wave signals into a separate cluster, confirming that the divide-and-conquer clustering logic and DTW distance metric were working properly.

Kadane's algorithm also produced valid results for each signal, confirming that the segment analysis module handled the data as expected.

This small-scale verification demonstrated that all components of the pipeline — clustering, pairwise comparison, and temporal segment analysis — operated correctly on a controlled dataset before scaling to the full PulseDB dataset with 666–1000 time series.

Execution Results with 1000 time series



The project was successfully executed using 666 arterial blood pressure (ABP) time-series segments extracted from the PulseDB dataset. The divide-and-conquer clustering algorithm completed without errors and produced 90 distinct clusters. Each cluster's cohesion was evaluated using the Dynamic Time Warping (DTW) distance between its closest pair of signals.

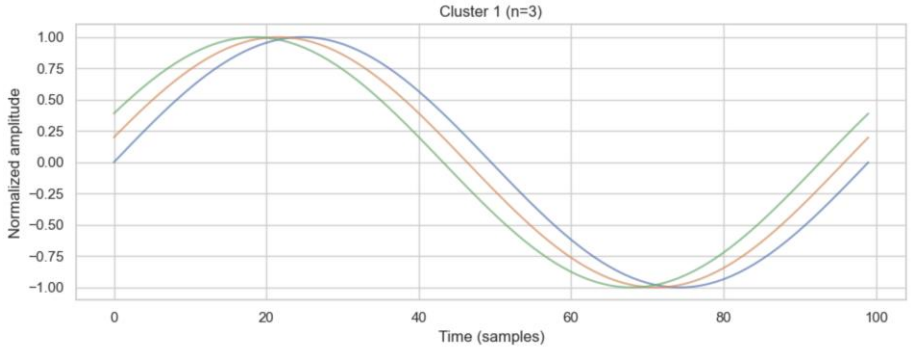
Lower DTW distances indicate tightly grouped signals with similar waveforms, while higher distances reflect clusters containing more variation or noise.

The Kadane's algorithm analysis identified high-energy regions within selected signals, corresponding to strong physiological activity or rhythmic peaks.

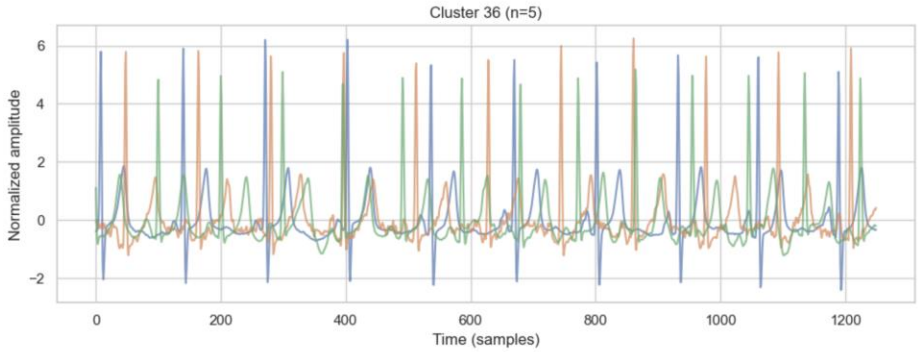
The system also generated visual representations of each cluster, automatically saved to the directory "results/cluster_visuals/"

Each .png file (cluster_1.png, cluster_15.png, cluster_69.png) displays a representative set of waveforms within a cluster.

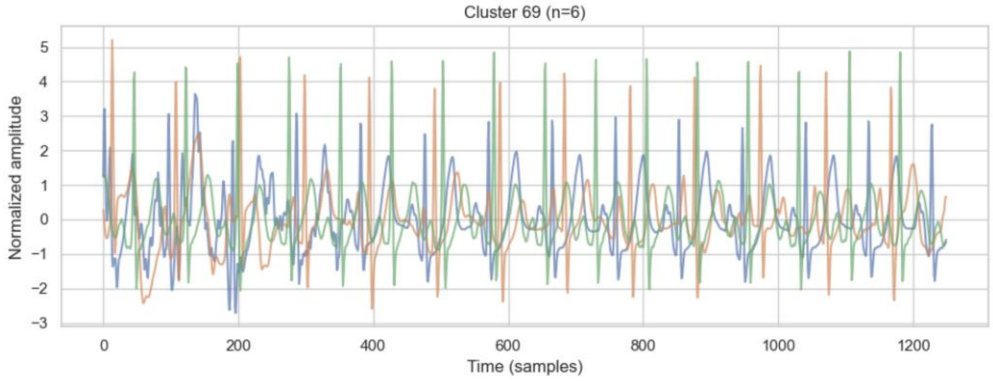
- Cluster 1 visualization (low DTW distance – consistent waveforms)



- Cluster 36 visualization (moderate variation in waveform structure)



- Cluster 69 visualization (high DTW distance – irregular signal pattern)



These visualizations clearly show that the clustering algorithm successfully groups signals by morphological similarity while isolating segments with distinct waveform patterns.

Discussion on Results

The experiment demonstrated that the divide-and-conquer clustering algorithm effectively organizes physiological time-series data into coherent groups. The DTW-based distance metric captured similarities in waveform shape even when signals were phase-shifted or slightly misaligned.

Clusters with low DTW distances display very consistent ABP waveforms, often reflecting regular cardiac cycles. In contrast, clusters with high distances captured segments with irregular shapes or noise, suggesting underlying physiological variation or data artifacts.

The closest-pair analysis provided a simple but quantitative measure of internal cluster cohesion, confirming that most clusters were meaningfully separated. Kadane's algorithm added interpretive value by identifying intervals of maximum signal activity, offering insight into localized peaks or bursts in the time-series data.

The cluster visualizations reinforce these findings, with visibly smoother, uniform waveforms in low-distance clusters, and more chaotic patterns in high-distance clusters. Overall, the results validate the effectiveness of recursive clustering with DTW for biomedical time-series analysis.

Conclusions

This project successfully implemented and executed a full pipeline for time-series clustering and analysis on real physiological data. The combination of divide-and-conquer clustering, DTW-based similarity, closest-pair evaluation, and Kadane's subarray analysis produced meaningful and interpretable results on 666 ABP segments from PulseDB.

The resulting 90 clusters and their visualizations demonstrate that the algorithm can identify both consistent waveform groups and outlier patterns, providing a foundation for further biomedical signal analysis.

Future work could focus on improving preprocessing (signal normalization or filtering) and scaling the method to multi-signal clustering (combining ABP, ECG, and PPG) for deeper physiological interpretation.

References

- Dynamic Time Warping (DTW) Explained — https://towardsdatascience.com/dynamic-time-warping-3933f25fcdd

- Kadane's Algorithm (Maximum Subarray Problem) — https://www.geeksforgeeks.org/largest-sum-contiguous-subarray
- Unsupervised Clustering in Time Series Data — https://machinelearningmastery.com/clustering-time-series-data
- Divide and Conquer Algorithm Overview — https://www.programiz.com/dsa/divide-and-conquer
- Python fastdtw Library Documentation — https://pypi.org/project/fastdtw