Visual Assessment of (Cluster) Tendency (VAT) Optimizations

1 Original VAT

1.1 Initialization

Utilized `np.max` and `np.argmax` to determine the two indices corresponding to the maximum distance. These two indices formed the starting point for the set I.

1.2 Main Loop

For each iteration, the algorithm performs a search over the set J (using nested loops) to find the index that has the minimum distance to the indices in I. This is the core of the iterative process. The use of `np.min` and `np.argmin` inside the loop over J adds a computational overhead, especially as the size of J grows.

2 Optimized VAT

2.1 Initialization

The start is similar to the original VAT by identifying the two indices with the maximum distance to begin the set I.

2.2 Main Loop

Instead of using nested loops, this version leverages *NumPy's vectorized operations* to compute minimum distances. Specifically:

- The use of `np.min` and `np.argmin` over entire slices of the matrix, like `R[I][:, J]`, eliminates the need for nested loops, thereby speeding up this critical step.
- It efficiently updates the sets I and J using array operations like `np.append` and `np.delete`, ensuring the correct indices are added/removed without multiple iterations.

3 Optimized VAT with Priority Queue (pq)

3.1 Initialization

Here, the initialization is quite different. Instead of just finding the maximum distance, this version constructs a graph from the dissimilarity matrix.

3.2 MST and Priority Queue

- Utilizes *networkx* to compute a Minimum Spanning Tree (MST) from the graph.
- The edges of the MST are processed using *priority queues* (implemented using `heapq`). This ensures that edges (and thus, the related indices) are efficiently retrieved in the order of their increasing weights.

3.3 Main Loop

The loop's operation is predicated on the MST structure. Instead of determining the next index solely based on distance (as in the original and first optimized versions), this method leverages the structure of the MST. The choice of indices is thus influenced by the overall structure of the data, which the MST captures. Using `heapq.heappop`, it efficiently retrieves the next smallest edge from the MST, ensuring that the chosen indices respect the global structure of the data.

4 Summary

The **Optimized VAT** enhanced the original algorithm mainly by cutting down on iterative processes. It achieved this by embracing NumPy's array capabilities, notably using operations over entire slices of matrices instead of element-by-element checks.

The **Optimized VAT with pq** brought a fresh perspective to the VAT optimization challenge. By introducing graph-based techniques and priority queues, it ensured that the order of indices was not just based on local distance values but also respected the overall data structure. This was achieved mainly through the MST, which inherently captures the intrinsic relationships among data points.