Self-Tuning Spectral Clustering & Sampling Framework

Haowen (2025) – Extended for Scientific Workflows

1 SPC Function (library)

1.1 Overview

This repository provides a Python implementation of **self-tuning spectral clustering**. It is designed to handle high-dimensional data efficiently, offering flexible clustering methods and spectral embeddings.

Key features:

- Locally scaled affinity computation
- Laplacian matrix construction (random walk & symmetric)
- Eigen decomposition and spectral embedding
- Multiple clustering backends (KMeans, Gaussian Mixture Models, QR-based clustering)
- Eigengap heuristic for automatic cluster number estimation
- Sampling strategies: random, temporal, farthest point sampling (FPS), local-scaled FPS, and hybrid methods

1.2 Affinity & Laplacian

- `local_scaled_affinity`: Builds an affinity matrix using k-nearest neighbors
- `Laplacian_matrix`: Constructs Laplacian (random walk or symmetric)
- `eigenthings`: Eigen decomposition of Laplacian

1.3 Clustering

- `clustering`: Spectral clustering with multiple assignment methods
- `cluster_qr`: QR-based clustering
- `eigerngap`: Eigengap heuristic for cluster number selection
- `Spectral_embeding`: Full spectral embedding pipeline
- `process_cluster`: Handles hierarchical sub-clustering and model persistence

1.4 Correlation Map

- `Reconstruct_correlation_map`: Rebuilds correlation maps using pairwise distances based on the order and function we chose.

1.5 Sampling

We have already tried different sampling methods, hoping the training data set affinity/correlation matrix captures the structure of whole data set. However, in the current data set, none of these approaches have shown improvement.

- `Sampler` class: Unified interface for sampling strategies
  - Random sampling
  - Temporal random samples
  - Farthest Point Sampling (FPS)
  - Local Scaled FPS (density-aware)
  - Hybrid temporal-FPS
- `estimate_local_density`: Computes local density via k-nearest neighbors
- `scale_features_by_density`: Adjusts the feature space based on density

1.6 Dependencies

- Python $\geq$ 3.8
- NumPy, SciPy, Pandas, Xarray
- Matplotlib
- scikit-learn
- tqdm, joblib

2 Iteration (SPC_process)

2.1 Initialization

- A root cluster dictionary is created with metadata:
  - `Cluster_Nr`: Cluster index number
  - `Cluster_Frames`: All frames initially assigned to cluster 0
  - `Nr_Frames`: Number of frames (updated later)
  - `Threshold`: Reclustering threshold
  - `Order`: Correlation order
  - `Metric`: Similarity metric (`correlation` or `cosine`)
  - `eigenvalues`: Placeholder array
- Clusters are stored in a list for iterative processing.

2.2 Parameters

- `first_threshold = 0.05` (root reclustering threshold)
- `rc_threshold = 0.15` (subcluster reclustering threshold)
- `force_iterations = 3` (forced split iterations)
- `order = 1` (subcluster reclustering correlation order)
- `metric = "correlation"` (similarity metric)

2.3 Iterative Loop

The workflow iterates over clusters until no further reclustering is possible:

1. **Sampling**: Root cluster: sampled directly from the full fragment space. Subclusters: sampled from sub-fragment space. We recommend using a small training data set size for the first several iterations and a large training data set size for the rest. **SPC.Sampler**:
   - Max samples: Upper limit for the number of training samples.
   - Min samples: Lower limit for the number of training samples.
   - Proportion: The ratio of the training dataset size to the total size of the dataset under classification.
2. **Correlation Map Construction**: Train/test correlation maps are built using the chosen metric.
3. **Spectral Embedding**: Eigenvalues and eigenvectors are computed from Laplacian matrices.
4. **Reclustering Condition**:
   - If smallest eigenvalue $<$ threshold, reclustering is triggered.
   - Root cluster: number of clusters = count of eigenvalues below threshold.
   - Subclusters: multiple criteria (eigengap, minimum eigenvalue, fixed $n = 2$).
5. **Clustering**:
   - Spectral clustering applied to eigenvectors (KMeans).
   - Train/test labels merged into unified cluster labels.
6. **Sub-cluster Processing**: Hierarchical sub-clusters are generated and stored.
7. **Termination**: Loop ends when all clusters are processed.

2.4 Outputs

- Cluster metadata: number of frames, thresholds, correlation order, metric, eigenvalues/eigenvectors.
- Console feedback: iteration progress, eigenvalue checks, reclustering decisions, timing.
- Final result: number of clusters determined after iterative refinement.

3 Example Output

```
========== Clustering of Cluster Index: 0 ==========
Cluster size: 5000 Train size: 3000 Test size: 2000
Calculating Clustering
Smallest Eigenvalue = 0.0321
Reclustering condition satisfied (Eigenvalue = 0.0321)! Recluster number 2
Time for embedding: 12.45 seconds
Iterative clustering algorithm finished!
You determined 2 clusters! (Threshold 0.05)
```