Iterative Label Spreading
Segmentation Algorithm

Daniel Tan

January 11, 2022

Definitions

I found that clearly defining definitions helped with coming up with segmentation methods.

Note: The definitions are not rigorous as there many undefined terms but it provides a good intuition (it would be interesting to properly define things with respect to the statistics regarding random walks).

Given a set of points, S, in a metric space, (X, d), what makes the labelling of points in S as means to cluster meaningful?

Definition: Given a set of points, S, in metric space, (X, d), a set of subsets, $C = \{C_1, C_2, \dots, C_n\}$ is a *clustering* if the following conditions are met.

- For any $C_i \in C$ there does not exist a clustering of C_i with only one cluster
- Any C_i, C_j are disconnected. (See definition)
- If C_i, C_j are not disconnected then the 'density' of points in C_i is significantly different from C_j

Definition: Two clusters, X, Y are disconnected if the distance between the two sets, d(X, Y)) is significantly different from the average distance between closest points within each cluster. The distance between the two sets is the usual minimum distance between two points from each set.

Definition: Two non disconnected clusters, X, Y, are valid clusters with respect to the union, $X \cup Y$, if the density of each significantly different from each other.

What is significantly different?, what is density when looking at the $R_{\min}$ series? The interpretation of these ambiguities are made clear in the description of the given algorithm. Though still not rigorous

In the Context of Rmin

Identifying Disconnected Clusters

Using the definition given above it and an understanding of how ILS works traversing between two disconnected cluster will appear as a peak in the series. So if p is the distance jumped between clusters then we will have

$$\mu_b + n \cdot \sigma_b < p \tag{0.1}$$

$$\mu_a + n \cdot \sigma_a < p \tag{0.2}$$

where n is the number of standard deviations required and (μ_b, σ_b) and (μ_a, σ_a) if the mean and standard deviation of R_{min} before and after the peak respectively.

Identifying Connected Clusters

Two connected clusters with differing densities will appear in the R_{min} series as a threshold difference. That is a given segmentation point, p will only be significantly different from one side. This is,

$$\mu_b + n \cdot \sigma_b < p \tag{0.3}$$

$$\text{OR } \mu_a + n \cdot \sigma_a < p \tag{0.4}$$

General Structure

- 1. Check if a given point exceed the significance level given
- 2. Naively Cluster the suggested segmentation points (multiple points will return for the same segmentation)
- 3. Return the maximum significance point within each cluster

Why do we have to naively cluster segmentation points?
Suppose we have two disconnected clusters both of size n. Then the R_{min} series, $\{x_i | i \in \mathbb{N}, i < 2n\}$, will contain the true segmentation point, x_n. Since ILS labels the closest point, before ILS traverses to the new cluster the distance it's jumps to get to outer points of the first cluster will be larger. As such most likely there exists some x_{n-k} where $k << n$ such that x_{n-k} also meets the required definition.

Algorithm

This first algorithm iterates through the list calculating the mean and standard deviation of points before and after as well as the position of the maximums all of which are linear calculations with respect to the minimum cluster size. So we have a time complexity of $O(m \cdot n)$ where m is the minimum cluster size and n is length of the list but typically $m << n$ so it linear.

The second simply iterates through the suggested segmentation points and groups the points such that the width from the first to the last point is less the three quarters of the minimum cluster size. Then chooses the point which has the highest significance. This seems to work well even though it seems like a bad idea. The time complexity of this is linear with respect to the number of points given, $O(n)$.

Algorithm 1 Finding Possible Segmentation points

```
1: INPUT: $R_{\min}$ (list of length n), $m$ (minimum cluster size), $s$ (number of standard deviations)
2: segmentation points = empty list                              ▷ Points will be added
3: number of standard deviations = empty list
4: $i = m$                                                       ▷ iterator
5: while $i < n - m$ do                                          ▷ Iterate through points in $R_{\min}$
6:     point = $R_{\min}[i]$
7:     sublist before = $R_{\min}[i - m + 1 : i + 1]$            ▷ get sublist before and after point
8:     sublist after = $R_{\min}[i : i + m]$
9:     if point is maximum of sublist before OR sublist after then
10:        $\mu_B$ = mean(sublist before)
11:        $\sigma_B$ = standard deviation(sublist before)
12:        $\mu_A$ = mean(sublist after)
13:        $\sigma_A$ = standard deviation(sublist after)
14:        if point > $\mu_b + s \cdot \sigma_B$ OR point > $\mu_A + s \cdot \sigma_A$ then
15:            add point index to segmentation point list
16:            add number of standard deviations it exceeded the mean to number of standard
   deviation list
17:        end if
18:    end if
19:    i = i+1
20: return (list) segmentation points, (list) number of standard deviations =0
```

Algorithm 2 Final Segmentation Points

```
1: INPUT: points (list of possible segmentation points), number of standard deviations (list
   corresponding to points), $m$ (minimum cluster size)
2: width = floor(3.4 · $m$)                                      ▷ Arbitrary Choice
3: final peaks = empty list                                      ▷ list of points will be added, each list
4: final standard deviations = empty list
5: grouping peaks = empty list                                   ▷ points that segment the same clusters
6: group standard deviations = empty list
7: i = 0                                                         ▷ Iterator
8: while i < length of points do
9:     if points[i+1] - [points[i] < width then
10:        add points[i] to the grouping peaks
11:        add number of standard deviations[i] to grouping standard deviations
12:        width = width subtract the distance between the two peaks
13:    else
14:        add grouping peaks to final peaks (list of list)
15:        add grouping standard deviations
16:        reset grouping points to the next points
17:        reset grouping standard deviations to the next standard deviation
18:        width = (int) 3/4 times minimum cluster size
19:    end if
20:    Indexs = list of indexs of maximum for each list in final standard deviations
21:    final points = list of points which are indexed be Indexs list from final peaks return
   final points
```