Robust Single Linkage Algorithm and Extract Flat Clustering

Taoran Xue

April 2017

1 Instruction

In unsupervised learning algorithm, there exists two problem to solve. First is that if our data set is infinity, in another word, its size can be increasing in a time, we need a cluster algorithm to fit the distribution density function of the data set. Second, if some bad or noise point occasionally draws between two cluster, it will influence the cluster result different from the original distribution of probability density function.

2 Minimum spanning tree

Before we go through today's topic, let's have a quick review of my last presentation about density based hierarchy clustering algorithm. Because we are trying to give a robust single linkage algorithm based on previous method.

For each i, set $r(x_i)$ to the distance from x_i to its kth nearest neighbor. As r grows from 0 to ∞:

1. Construct a graph G_r with nodes $\{x_i : r(x_i) \leq r\}$. Include edge (x_i, x_j) if $\|x_i - x_j\| \leq \alpha r$.
2. Let $\mathbb{C}_n(r)$ be the connected components of G_r.

As we know, real data is messy and has corrupt data, and noise. So we need to make a new algorithm with single linkage algorithm and it can be sensitive to noise. As I mentioned, if a single noise data point is in the wrong place can act as a "thin bridge" between two cluster, gluing them together.

So the authors (Chaudhuri, Dasgupta) of previous paper[1] give a method to robustly process against noise, and, in the paper presentation, I proved that this algorithm can deal with noise problem. Their idea is to define a new version of minimal spanning tree.

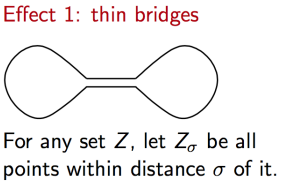

Figure 1: "Thin bridge" effect.

Definition 1. *Set $r_k(x_i)$ to the distance to kth nearest neighbor. For any $r = max\{r_k(x_i)\}$, connect points x_i and x_j, if $||x_i - x_j|| \leq \alpha r$.*

In another word, we can convert it to our new definition – **mutual reachability distance**:

Definition 2. *Set $core_k(x_i)$ to the distance to kth nearest neighbor.*

$$d_{\text{mrd}_k}(x_i, x_j) = max\{core_k(x_i), core_k(x_j), ||x_i - x_j||\}$$

As we have mutual reachability distance matrix, we can build the minimum spanning tree very efficiently via Prim's algorithm.

3 Single linkage hierarchy algorithm

Given the minimal spanning tree, the next step is to convert that into the hierarchy of connected components. This is most easily done in the reverse order: sort the edges of the tree by distance (in increasing order) and then iterate through, creating a new merged cluster for each edge.

4 Condense tree

Hierarchy cluster algorithm always gets a large and complicated cluster hierarchy tree. So we try to condense down the cluster hierarchy into a smaller tree. As you can see in the hierarchy above it is often a cluster split is one or two points splitting off from a cluster. We need to eliminate cluster which has fewer points than the **minimum cluster size**.

Initially, we breadth-first search the whole hierarchy tree. When we arrive one cluster, there will be three conditions.

- If left child cluster point number is greater than minimum cluster size, but right side is not, Figure 2 (a), keep the left branch and ignore right cluster;

- If left and right child clusters are both greater than minimum cluster size, Figure 2 (b), we consider that a cluster split and let the split persist the whole tree;
- If left and right child clusters are both fewer than minimum cluster size, Figure 2 (c), we ignore the two cluster.

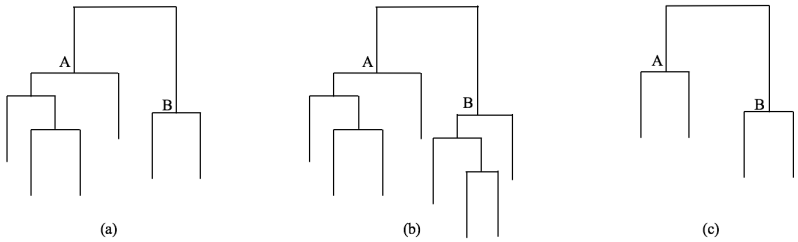

Figure 2: Condense tree.

Algorithm 1: Condense hierarchy cluster tree

Input: H[m] ← hierarchy tree contains a tuple of (child1,child2,distance,childrenSize), minClusterSize ← The minimum size of clusters to consider

Output: T[n] ← condense tree contains a tuple of (parent,child,λ,childrenSize)

```
nodeList ← BFS(H);
for r ← 0 to m do
    if nodeList[r] is ignored then
        Pass;
    end
    left ← nodeList[r].child1;
    leftCount ← H[left].childrenSize;
    right ← nodeList[r].child2;
    rightCount ← H[right].childrenSize;
    if leftCount ≥ minClusterSize and rightCount ≥ minClusterSize then
        T[p++] ← (nextLabel, ++nextLabel, 1/distance, leftCount);
        T[p++] ← (nextLabel, ++nextLabel, 1/distance, rightCount);
    end
    if leftCount < minClusterSize then
        for tmpNode in BFS(left) do
            if tmpNode is leaf then
                T[p++] ← (nextLabel, tmpNode, 1/distance, 1);
            end
            Ignore tmpNode;
        end
    end
    if rightCount < minClusterSize then
        for tmpNode in BFS(right) do
            if tmpNode is leaf then
                T[p++] ← (nextLabel, tmpNode, 1/distance, 1);
            end
            Ignore tmpNode;
        end
    end
end
```

5 Cluster stability

For a condense tree, intuitively, we want to extract the cluster that persists and has a long lifetime. It means the cluster is relatively stable unlikely to split into two clusters after it created. So we give a new definition to measure the persistence of a cluster.

Definition 3. *Let* $\lambda = \frac{1}{d_{\text{mrd}_k}}$. *For each cluster we give* λ_{birth} *and* λ_p *to be the lambda value when the cluster split off then became it's own cluster, and the lambda value (if any) when the cluster split into smaller clusters respectively.*

Now, for each cluster C compute the stability to as:

$$S(C) = \sum_{p \in C} (\lambda_p - \lambda_{\text{birth}})$$

Algorithm 2: Calculate stability of each cluster

Input: $\mathsf{T}[n] \leftarrow$ condense tree in reverse topological order contains a tuple of (parent,child,λ,childrenSize)
Output: $\mathsf{S}[n] \leftarrow$ stability of every node in condense tree
for $r \leftarrow 0$ **to** n **do**
  currChild $\leftarrow \mathsf{T}[r]$.child;
  curr$\lambda \leftarrow \mathsf{T}[r].\lambda$;
  if currChild = prevChild **then**
    min$\lambda \leftarrow$ `Min(`minλ, currλ`)`;
  else
    birthλ[currChild] $\leftarrow$ minλ;
    prevChild $\leftarrow$ currChild;
    min$\lambda \leftarrow$ currλ;
  end
end
for $r \leftarrow 0$ **to** n **do**
  $\mathsf{S}[r] \leftarrow \mathsf{S}[r] + \mathsf{T}[r].\lambda - \text{birth}\lambda[\mathsf{T}[r].\textit{parent}] \times \mathsf{T}[r]$.childrenSize;
end

6 Flat clustering

To make hierarchy cluster result to flat cluster result, we require that once the cluster is selected, then the descendant of it cannot be selected. Because hierarchy cluster is nested, selected cluster contains all cluster of its descendant.

Definition 4. *Set* $SC(C)$ *is the sum of the stabilities of the child cluster of cluster* C.

$$SC(C) = \sum_{q \in C} S(c)$$

Algorithm 3: Abstract cluster with stabilities

Input: $\mathsf{S}[n] \leftarrow$ stability of every node in condense tree sorted in reverse topological order

Output: $\mathsf{L}[n] \leftarrow$ **True** if index of cluster is selected; **False** otherwise

$\mathsf{L} \leftarrow \{\textbf{True}\}$;
for $r \leftarrow 0$ **to** n **do**
    childList $\leftarrow$ {list of node whose parent is r};
    subtreeStabilities $\leftarrow \sum_{c \in \mathsf{childList}} \mathsf{S}[c]$;
    if subtreeStabilities $> \mathsf{S}[r]$ **then**
        $\mathsf{L}[r] \leftarrow$ **False**;
        $\mathsf{S}[r] \leftarrow$ subtreeStabilities;
    else
        for tmpNode ***in*** `BFS(`r`)` **do**
            if tmpNode $\neq r$ **then**
                L[tmpNode] $\leftarrow$ **False**
            end
        end
    end
end