CS-E4600 Algorithmic Methods of Data Mining Answers to Homework 3

Adam Ilyas 72581

November 12, 2018

Francsico Caldas, Zeyneb Erdogan

Problem 1: clustering given a hierarchy

Consider again a set X of n points in $\mathbb{R}^d$. In addition, this time, we are given a hierarchy tree T over the points of X.

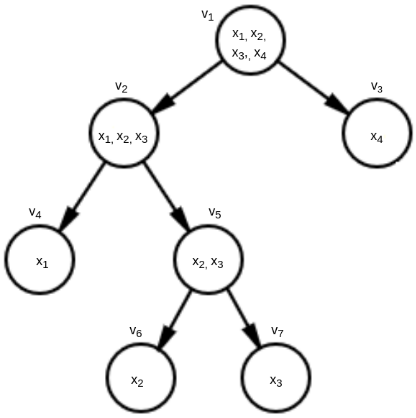
The tree T is binary and each node $v \in T$ corresponds to a subset of X. Let us denote by $X(v)$ the set of points that correspond to node v. There are exactly n leaves in T , each one corresponding to exactly one point in X. If v, u, w are nodes of T with u and w being the children of v then $X(v) = X(u) \cup X(w)$. Thus, the root r of the tree corresponds to the whole set X , i.e., $X(r) = X$. We want to solve the k-means problem on X , but the resulting clustering is constrained by the nodes of the tree. More precisely, we want to solve the following problem.

k-means on trees: We are given a set X of n points in $\mathbb{R}^d$, a hierarchy binary tree T over X , as described above, and an integer k. We want to find k nodes $\{v_1, \ldots, v_k\}$ of T , such that $\bigcup_{i=1}^{k} X(v_i) = X$ and $X(v_i) \cap X(v_j) = \emptyset$; for all $i \neq j$, and such that the objective function

$$\sum_{i=1}^{k} var(v_i)$$

is minimized, where $var(v_i) = \sum_{x \in X(v_i)} ||x - c(v_i)||^2$ and where $c(v)$ is the mean of all points in $X(v)$.

Question 1.1 Show that the problem of k-means on trees can be solved optimally in polynomial time.



Ans. The first thing to note is that the binary tree T containing elements of X, and the integer K which represents the cluster which we want to obtain, has already been given to us.

Given we have n number of data points in X, thus we have n leaves. Let's denote the total number of nodes as $b = 2n - 1$

We maintain a $b \times k$ table for us to store the combination of clusters and the minimum $\sum_{i=1}^{k} var(v_i)$ it gives, for every node.

	$k=1$	$k=2$	$\dots$	$k = k_{given}$
$v = v_1$	$var(v_1), s_{1,1}$	$d(v_1, 2), s_{1,2}$	$\dots$	$d(v_1, k), s_{1,k}$
$v = v_2$	$var(v_2), s_{2,1}$	$d(v_2, 2), s_{2,2}$	$\dots$	$d(v_2, k), s_{2,k}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$v = v_b$	$var(v_b), s_{b,1}$	$d(v_b, 2), s_{b,2}$	$\dots$	$d(v_b, k), s_{b,k}$

Where the rows represents the node, the columns represent the number of clusters k. We then fill each cell with 2 information (the value of the variance, and the combination of element in each cluster)

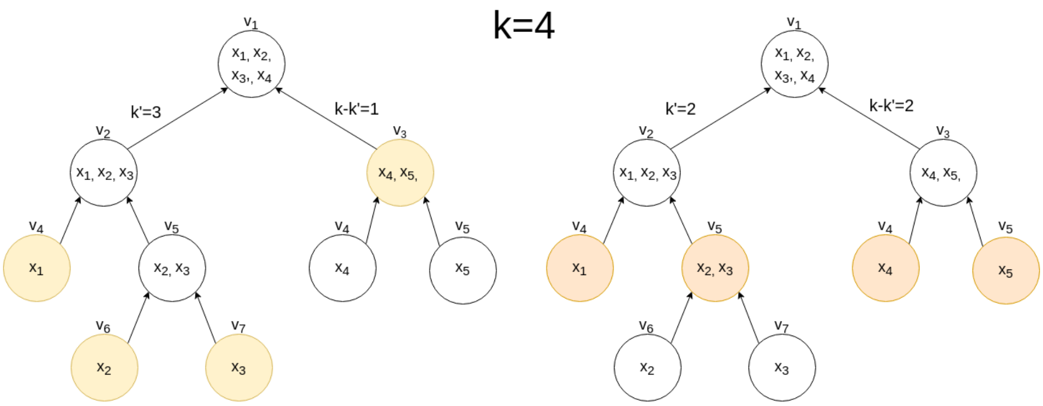
1) We initialize the first row as the $var(v_i)$: $\sum_{x \in X(v_i)} ||x - c(v_i)||^2$ for each node v_i, and also $X(v_i)$

2) We then fill the rest with $d(v_i, k), s_{i,k}$ where both are calculated.

$$d(v_i, k) = \min_{0<k'<k} \{d(v_{left}, k') + d(v_{right}, k - k')\}$$

3) Our solutions lies on the first row, k^{th} column: $d(v_1, k_{given})$ and $s_{1,k}$

Figure 1: Visualisation of $d(v_i, k) = \min_{0<k'<k}\{d(v_{left}, k') + d(v_{right}, k - k')\}$



We are taking every combination of clustering, which already calculated the clustering configuration and value of the level below by refering to the table.

For example, in the image above we see that on the clusters are being divided to:

- Left image: $d(v_2, k = 3)$ and $d(v_3, k = 1)$
- Right image: $d(v_2, k = 2)$ and $d(v_3, k = 2)$

Which we already know the value and configuration of.

Question 1.2 Show the correctness of your algorithm.

Ans. The function $d(v, k)$ that we defined is based on dynamic programming and we store the values in a table. Thus, the basis of our algorithm is that we have already calculated the minimum variance and configuration of clusters that gives this minimum variance for each k.

Question 1.3 What is the complexity of your algorithm?

We iteratively fill our table of the $2n - 1$ rows by k columns. For each element of the table, we have to calculte for $k' - 1$ (from 1 to $k' - 1$). Thus our complexity is $\mathcal{O}(n \cdot k^2)$

Problem 2 : clustering a data stream

Question 2.1 Prove that the `STREAMING-FURTHEST` algorithm produces a clustering that has at most k cluster centers.

Ans.

(Contradiction Hypothesis) We are going to proof by contradiction by assuming that the number of clusters to be $k+1$ which is more than k.

For the next point x_{new} to be a new center, the distance to it's closest center must be more than $2d_*$ ($d(x_{new}, x_i) > 2d_*$ where d is the distance metric)

(Consequence of contradiction hypothesis) For us to have $k+1$ this means we have least $k+1$ points in C and all these points are apart from each other by $2d_*$)

However, optimal k-clustering means produces optimal d_* distance such that we have k clusters. This means that we have that two of the $k+1$ points are in the same cluster. (let's call the center of this cluster to be c^*, where $c^* \in C$)

Let the 2 point in the same cluster to be p_1 and p_2. Thus the distance of the these points to point c^* is: $d(p_1, c^*)$ and $d(p_2, c^*)$. Due to optimal k-means clustering, we know that these 2 points are less than d_* away from c^*

$$
\begin{aligned}
d(p_1, c^*) &\leq d^* \\
d(p_2, c^*) &\leq d^*
\end{aligned}
$$

Since they belong Using triangular inequality we get:

$$
\begin{aligned}
d(p_1, p_2) &\leq d(p_1, c^*) + d(p_1, c^*) \\
d(p_1, p_2) &\leq d^* + d^* = 2d^*
\end{aligned}
$$

Which is a contradiction to our hypothesis that there are $k+1$ clusters. As such, the number of clusters at most $\leq k$.

Question 2.2 Prove that the STREAMING-FURTHEST algorithm is still a factor-2 approximation of the optimal clustering C^* on the data stream X .

(Contradiction Hypothesis) There is a point (p) in a cluster (the furthest point) which is $d_m(p, c) > 2d^*$ from the center of the cluster (c)

it is also known that

> all centers $\in C$ are at least $2d*$ apart
>
> all points have a center within d^* distance,

A consequence of our hypthesis that $d_m(p, c) > 2d^*$ is that there will be a cluster with 2 points that are more than $2d^*$ from each other, which is a contradiction because we would end up with $k + 1$ clusters and as such, d^* is not actually optimal. (We have shown this in the previous question)

Thus, STREAMING-FURTHEST is still a 2-factor approximation.

Question 2.3 Suggest how to modify the STREAMING-FURTHEST algorithm for the (realistic) case that the cost d^* of the optimal clustering is not known.

Input a stream of data points X
Output: clustering of points in X in k clusters

read the first k points of X $(x_1, x_2, \ldots, x_k)$
define the set of cluster centers as $C \leftarrow \{x_1, x_2, \ldots, x_k\}$
$d_{min} \leftarrow$ minimum of all distances between every pair of cluster in C
$c_{small} \leftarrow$ cluster in C with smallest distance
while X has stream **do**
  read the next point x in the stream
  compute the distance d_m from x to its closest cluster center $c_{closest}$
  if $d_m < d_{min}$ **then**
    replace c_{min} in C with x
    Calculate and set new d_{min} and c_{small} for the new C
  else
    Assign the new x datapoint to its closest cluster in C
return C

Problem 3 : monitoring a graph

We are monitoring a graph, which arrives as a stream of edges$E = e_1, e_2, \ldots$. We assume that exactly one edge arrives at a time, with edge e_i arriving at time i, and the stream is starting at time 1. Each edge e_i is a pair of vertices $(u_i\ ,\ v_i\ )$, and we use V to denote the set of all vertices that we have seen so far.

We assume that we are working in the sliding window model. According to this model, at each time T only the W most recent edges are considered active. Thus, the set of active edges $E(T, W)$ at time T and for window length W is

$$E(T, W) = \begin{cases} e_{T-W+1}, \ldots, e_T, & \text{if } T > W \\ e_1, \ldots, e_T, & \text{if } T < W \end{cases}$$

We then write $G(T, W) = (V, E(T, W))$ to denote the graph that consists of the active edges at time T , given a window length W

Question 3.1 Propose a streaming algorithm for deciding the connectivity of G(T, W).

Ans.
We know that

1) the smallest length (in terms of edges) of a connected graph in n points is $|C_n \backslash e|$

2) Any connected graph without cycles as the same length

Also we see that any cycle is 2-connected so when we remove a edge from a cycle the graph remains connected.

Input: stream of edges E (contains $e_{T-W+1}, \ldots, e_T,$), Integer W
Output: : Boolean c_t graph connectivity at time t {True, False}

$E = \{\}$ ▷ is a list that records the active edges
while has Stream **do**
  if t<W **then**
    $E \leftarrow E \cup \{e_t\}$ ▷ read e_t and store it on E
  if $t \geq W$ **then**
    $E \leftarrow E \cap e_T$ ▷ read e_t and store it on E
    $V \leftarrow$ all vertices of E
    $G \leftarrow$ create a graph from edges E and vertices V
    Initialize hashMap $visited \leftarrow \{v_i : False\}$ for all $v_i \in V$
    while True **do**
      $hasCycle \leftarrow checkCycle(G, v, NULL)$ ▷ Check cycles in E
      if $hasCycle$ **then**
        $E \leftarrow E \backslash e_{oldest}$ ▷ remove the oldest edge from the cycle.
      else
        BREAK
    if $|E| < N - 1$ **then**
      $c_T = False$
    else
      $c_T = True$

function CHECKCYCLE($G, v, parent$)
  $visited[v] \leftarrow True$
  for all v_n neighbours of v **do**
    if $visited[v_n]$ **then**
      return True
    else
      return `checkCycle`($\{G, v_n, v\}$)
  return False

Question 3.2 Prove the correctness of your algorithm.

We checks the the connectivity of the graph given by the sliding window, based on 2 cases:

1. When $T < W$: For the first W-1 edges it collects the edges in a list, that list as the information of the $G(T, W)$.

2. When $t \geq W$:

   – it checks for cycles in the graph G(T,W), if it exists one or more cycles it always removes the oldest edge in the graph such that there at least one less cycle on the graph.

   – If the acyclic cycle obtained has $|E| < n - 1$ (being n the number of total vertices,and $|E|$ the edges of the acyclic graph) then we know for sure that is not connected, because n-1 is the lower bound for edge connectivity.

Question 3.3 How much space does your algorithm use?

The space complexity for the algorithm is the space needed to store the edges in the window E and the total number of vertices N but as the maximum size of the window is N we have at the most $\mathbb{O}(N)$ space.

Question 3.4 What is the update time of your algorithm?

The most expensive process in our algorithm is the recursivee cycle check `checkCycle` which has the average running time of $\mathbb{O}(|V| + |E|)$ being V the vertices and E the edges. as it is based on a `depthFirstSearch` algorithm.

Our E that we store for each time t contains N vertices at most (when no edges gets removed), and as such the run time is is $\mathbb{O}(N)$.