CT-means Fuzzy Clustering Algorithm

Eklavya Sharma

December 23, 2017

Abstract

We propose a fuzzy clustering algorithm - CT-means. It is similar to C-means, but is faster when number of dimensions is low and number of clusters is high. This improvement in speed is achieved by using data structures which support nearest neighbor queries, like KD-trees. The algorithm creates c clusters where membership value of a point is non-zero in t clusters closest to it and zero in all other clusters. CT-means is therefore a generalization of the K-means ($t = 1$) and C-means ($t = c$) algorithms. We also discuss methods for choosing an appropriate value for t. Generally, a point's membership value is significant only for a few clusters close to it. If we set t to the number of significant clusters, we can get results very similar to fuzzy C-means clustering with a reduced running time.

1 Introduction

1.1 Fuzzy clustering vs Hard clustering

Clustering is a problem where n objects have to be assigned to clusters such that similar objects belong to the same cluster and different objects belong to different clusters.

A hard clustering algorithm takes n points and assigns them to c clusters. Each point belongs to one and only one cluster. A hard clustering algorithm therefore outputs a membership array and coordinates of the centroids. The i^{th} element of the membership array is the cluster to which the i^{th} point belongs to.

A fuzzy clustering algorithm, on the other hand, can assign a point to multiple clusters. Each point belongs to every cluster with a certain membership degree. A fuzzy clustering algorithm therefore outputs a membership matrix instead of a membership array. The membership matrix gives the

membership value of each point in each cluster. The sum of membership values of a point in different clusters sums to 1.

1.2 CT-means

We propose a fuzzy clustering algorithm CT-means. This algorithm computes the membership value of a point only for the t clusters whose centroids are closest to it. The membership value of that point for all other clusters is 0. Note that t need not be the same for all points. It can also vary across iterations. The CT-means algorithm is a generalization of the K-means algorithm ($t = 1$) and C-means algorithm ($t = c$).

For any fuzzy clustering algorithm, a point's membership value is supposed to be high for clusters closer to a point and low for clusters far away from a point. In practice [citation needed], with fuzzy clustering algorithms like C-means, the membership value of a point is significant only for a few nearby clusters and is very close to zero for all other clusters. This is the core idea behind the CT-means algorithm. CT-means is expected to perform well when t for a point is chosen to be equal to or slightly more than the number of significant clusters for that point. In this paper, we discuss the CT-means algorithm and strategies to choose an appropriate value of t for each point.

2 Overview of Algorithm

The n points are represented by a n by d matrix X where the i^{th} row x_i represents a d-dimensional point. The membership matrix is a n by c matrix U where the entry in i^{th} row at j^{th} column ($u_{i,j}$) is the membership value of the i^{th} point in the j^{th} cluster. The c centroids are represented by a c by d matrix C where the i^{th} row c_i represents the coordinates of the i^{th} centroid (in d dimensions).

Since t need not be the same for different points, let t_i denote the value of t for the i^{th} point.

The CT-means algorithm, like the C-means and K-means algorithms, involves several iterations. First, c initial centroids are chosen. Then in each iteration, the membership matrix is calculated and centroids are updated. The centroids will stabilize over time and the algorithm can be stopped then. Any stopping criterion used in C-means can also be used in CT-means.

In each iteration, this is how membership values are calculated and centroids are updated:

$$u_{i,j} = \left(\sum_{k=1}^{t_i} \left(\frac{|x_i - c_j|}{|x_i - c'_{i,k}|} \right)^{\beta} \right)^{-1} = \frac{|x_i - c_j|^{-\beta}}{\sum_{k=1}^{t_i} |x_i - c'_{i,k}|^{-\beta}}$$

$$c_j = \frac{\sum_{i=1}^{n} u_{i,j}^m x_i}{\sum_{i=1}^{n} u_{i,j}^m}$$

where $\beta = \frac{2}{m-1}$ and $c'_{i,k}$ is the k^{th} closest centroid to x_i.

3 Objective Function and Convergence

CT-means tries to minimize the objective function

$$\sum_{i=1}^{n} \sum_{j=1}^{c} u_{i,j}^m |x_i - c_j|^2$$

This is the same objective function as C-means, but the algorithm is different because there is a constraint on the membership matrix (that for each point, all clusters except the t nearest clusters will have membership value 0 for that point).

3.1 Centroid Update

During centroid calculation, we have to choose the centroids in such a way that the objective function is minimized while keeping the membership matrix fixed.

Define

$$S_j = \sum_{k=1}^{n} u_{k,j}^m \qquad\qquad p_{i,j} = \frac{u_{i,j}^m}{S_j}$$

Therefore,

$$\sum_{i=1}^{n} p_{i,j} = 1$$

Also define

$$\mu_j = \sum_{i=1}^{n} p_{i,j} x_i \qquad\qquad \sigma_j^2 = \sum_{i=1}^{n} p_{i,j} |x_i - \mu_j|^2$$

$$\begin{aligned}
&\sum_{i=1}^{n} p_{i,j} |x_i - c_j|^2 \\
&= \sum_{i=1}^{n} p_{i,j} |(x_i - \mu_j) - (c_j - \mu_j)|^2 \\
&= \sum_{i=1}^{n} p_{i,j} (|x_i - \mu_j|^2 + |c_j - \mu_j|^2 - 2(x_i - \mu_j) \cdot (c_j - \mu_j)) \\
&= \left(\sum_{i=1}^{n} p_{i,j} |x_i - \mu_j|^2 \right) + \left(\sum_{i=1}^{n} p_{i,j} |c_j - \mu_j|^2 \right) \\
&- 2 \left(\sum_{i=1}^{n} p_{i,j} (x_i - \mu_j) \cdot (c_j - \mu_j)) \right) \\
&= \sigma_j^2 + \left(|c_j - \mu_j|^2 \sum_{i=1}^{n} p_{i,j} \right) - 2(c_j - \mu_j) \cdot \left(\sum_{i=1}^{n} p_{i,j} (x_i - \mu_j) \right) \\
&= \sigma_j^2 + |c_j - \mu_j|^2
\end{aligned}$$

Objective function

$$\begin{aligned}
&= \sum_{j=1}^{c} \sum_{i=1}^{n} u_{i,j}^m |x_i - c_j|^2 \\
&= \sum_{j=1}^{c} S_j \sum_{i=1}^{n} p_{i,j} |x_i - c_j|^2 \\
&= \sum_{j=1}^{c} S_j (\sigma_j^2 + |c_j - \mu_j|^2) \\
&= \sum_{j=1}^{c} S_j \sigma_j^2 + \sum_{j=1}^{c} S_j |c_j - \mu_j|^2
\end{aligned}$$

The second sum is non-negative and we can make it zero by setting c_j equal to μ_j for all $1 \leq j \leq c$. Therefore, this minimizes the objective function. This gives us the update equation for centroids which implies that centroid update in CT-means decreases the objective function in each iteration.

3.2 Membership Matrix Update

Let's focus our attention on calculating the i^{th} row of the membership matrix. For convenience, assume that c_j is the j^{th} closest centroid to x. This assumption will not affect correctness since numbering of clusters is arbitrary and we can renumber them to enforce this assumption. Also, let $x = x_i$, $t = t_i$ and $u_j = u_{i,j}$ for convenience.

The contribution of a point to an objective function is given by

$$\sum_{j=1}^{c} u_j^m |x - c_j|^2$$

Let's call this the partial objective function (POF). We will prove that a membership matrix row update reduces the POF. Therefore a membership matrix update reduces the objective function.

Using the rearrangement inequality, we can see that the centroid closer to x should have a higher membership value than a centroid far away from x if we want to minimize the POF.

A corollary of this is that only the t centroids closest to x can have non-zero membership values under the constraint that there can be at most t non-zero membership values for a point.

Therefore, the POF can also be written as

$$\sum_{j=1}^{t} u_j^m |x - c_j|^2$$

Attempting to minimize the POF gives us a constrained optimization problem where the constraint is $\sum_{j=1}^{t} u_j = 1$.

Using Lagrange's method of multipliers, we get the Lagrangian as

$$\left(\sum_{j=1}^{t} u_j^m |x - c_j|^2 \right) - \lambda \left(\sum_{j=1}^{t} u_j - 1 \right)$$
$$= \left(\sum_{j=1}^{t} \left(u_j^m |x - c_j|^2 - \lambda u_j \right) \right) - \lambda$$

On taking partial derivatives and setting them to 0, we get these equations:

$$mu_j^{m-1}|x-c_j|^2 = \lambda \qquad\qquad \sum_{j=1}^{t} u_j = 1$$

On solving these equations, we get

$$\begin{aligned} u_j &= \left(\sum_{k=1}^{t} \left(\frac{|x-c_j|}{|x-c_k|} \right)^{\frac{2}{m-1}} \right)^{-1} \\ &= \frac{|x-c_j|^{\frac{-2}{m-1}}}{\sum_{k=1}^{t} |x-c_k|^{\frac{-2}{m-1}}} \\ &= \frac{|x-c_j|^{\frac{-2}{m-1}}}{s_t} \end{aligned}$$

where $s_t = \sum_{k=1}^{t} |x-c_k|^{\frac{-2}{m-1}}$.

This comes out to be the same as the membership update equation for CT-means algorithm. Therefore, the membership update equation for CT-means algorithm leads us to a critical point.

We'll now prove that this point is a local minimum.

Proof. First, observe that

$$u_j^{m-1}|x-c_j|^2 = s_t^{1-m}$$

Now let's move an infinitesimally small distance away from the critical point. To do that, change u_j to $u_j + \delta_j$, where $\sum_{j=1}^{t} \delta_j = 0$ to ensure that the changed membership values add up to 1.

Change in POF

$$= \sum_{j=1}^{t} ((u_j + \delta_j)^m - u_j^m)|x - c_j|^2$$

$$= \sum_{j=1}^{t} \left(m u_j^{m-1} \delta_j + \binom{m}{2} u_j^{m-2} \delta_j^2 + O(\delta_j^3) \right) |x - c_j|^2$$

(using binomial theorem)

$$= \sum_{j=1}^{t} m u_j^{m-1} \delta_j |x - c_j|^2 + \sum_{j=1}^{t} \binom{m}{2} u_j^{m-2} |x - c_j|^2 \delta_j^2 + O(\delta_j^3)$$

$$= \sum_{j=1}^{t} m s_t^{m-1} \delta_j + \sum_{j=1}^{t} \binom{m}{2} u_j^{m-2} |x - c_j|^2 \delta_j^2 + O(\delta_j^3)$$

$$= 0 + \sum_{j=1}^{t} \binom{m}{2} u_j^{m-2} |x - c_j|^2 \delta_j^2 + O(\delta_j^3)$$

Here the sum of δ_j^2 terms is always positive if $m \neq 2$. When $m = 2$, There are no higher order δ_j terms. Therefore, the change in POF is non-negative when moving slightly away from the critical point. This means that the critical point is a point of local minimum.

□

To prove that this point is a global minimum we must prove that values of POF at the boundaries are higher.

The constraints on membership values are $0 \leq u_j \leq 1$. Therefore, a boundary condition is achieved when u_j is either 0 or 1 for some j. If u_k is set to 1 for some k, all other u_j will be 0 (because they must sum to 1). The minimum POF over all k at this boundary will therefore be the same as the minimum POF at $t = 1$. If u_k is set to 0 for some k, we have $t - 1$ non-zero values of u_j to optimize. The minimum POF over all k at this boundary will therefore be the same as the minimum POF at $t - 1$.

Now we'll prove using mathematical induction on t that the membership update equation gives us the globally minimized POF.

Proof. Let $P(r)$ be the predicate 'POF is globally minimized according to the membership update equation when $t = r$'.

Base case:

When $t = 1$, only u_1 can be positive. Since all u_j should sum to 1, $u_1 = 1$.

According to the membership update equation

$$u_1 = \frac{|x - c_1|^{\frac{-2}{m-1}}}{\sum_{k=1}^{t} |x - c_k|^{\frac{-2}{m-1}}} = \frac{|x - c_1|^{\frac{-2}{m-1}}}{|x - c_1|^{\frac{-2}{m-1}}} = 1$$

Therefore, the membership update equation gets us to global minimum at $t = 1$. Hence $P(1)$.

Inductive step:

Let $r \geq 2$. Assume $P(r-1)$ is true (induction hypothesis).

The POF at boundary conditions for $t = r$ is the same as either the minimum POF at $t = 1$ or the minimum POF at $t = r - 1$. Since $t = r - 1$ is a more relaxed condition than $t = 1$ (or the same condition if $r = 2$), the minimum POF at $t = 1$ is less than or equal to the minimum POF at $t = r - 1$.

Therefore, to prove that the membership update equation gives us the minimum POF at $t = r$, we have to prove that the POF obtained using the membership update equation at $t = r$ is less than the minimum POF at $t = r - 1$. According to induction hypothesis, minimum POF at $t = r - 1$ is given by the membership update equation with $t = r - 1$.

Let v_j be the membership value at $t = r - 1$ and u_j be the membership value at $t = r$.

$$\begin{aligned}
&\text{POF at } t = r \\
&= \sum_{j=1}^{r} u_j^m |x - c_j|^2 \\
&= \sum_{j=1}^{r} (u_j^{m-1} |x - c_j|^2) u_j \\
&= \sum_{j=1}^{r} s_r^{1-m} u_j \\
&= s_r^{1-m} \sum_{j=1}^{r} u_j \\
&= s_r^{1-m}
\end{aligned}$$

Similarly, POF at $t = r - 1$ equals s_{r-1}^{1-m}.

$$
\begin{aligned}
s_r &= \sum_{k=1}^{r} |x - c_k|^{\frac{-2}{m-1}} \\
&= s_{r-1} + |x - c_k|^{\frac{-2}{m-1}} \\
&\geq s_{r-1}
\end{aligned}
$$

That implies $s_r^{1-m} \leq s_{r-1}^{1-m}$ ($\because m < 1$), which implies that POF at $t = r$ using the membership update equation is less than the POF at $t = r - 1$ using the membership update equation.

This means that POF is minimized at $t = r$ by using the membership update equation.

By the principle of mathematical induction, it can be concluded that the membership update equation minimizes the POF for all values of t.

□

Since the membership values obtained using the membership update equation minimizes the objective function for fixed centroids, membership matrix update lowers the objective function in each iteration.

Since both centroid update and membership matrix update lower the objective function in each iteration, CT-means converges to a local minimum of the objective function.

4 Detailed Description of Algorithm

Since many entries in the membership matrix are expected to be 0, the membership matrix is stored as a sparse matrix.

4.1 Centroid Updates

$$
c_j = \frac{\sum_{i=1}^{n} u_{i,j}^m x_i}{\sum_{i=1}^{n} u_{i,j}^m}
$$

All non-zero entries in the j^{th} column of the membership matrix are used to calculate c_j. Therefore, time taken for centroid updates is $O(\bar{t}nd)$. Here $\bar{t}$ is the average t_i over all points.

4.2 Membership Matrix Updates

Let's focus our attention on calculating the i^{th} row of the membership matrix. Let's denote the i^{th} point by x. For convenience, assume that c_j is the j^{th} closest centroid to x and let $u_j = u_{i,j}$. This assumption will not affect correctness since numbering of clusters is arbitrary and we can renumber them to enforce this assumption.

$$\begin{aligned} u_j &= \left(\sum_{k=1}^{t} \left(\frac{|x - c_j|}{|x - c_k|} \right)^{\beta} \right)^{-1} \\ &= \frac{|x - c_j|^{-\beta}}{\sum_{k=1}^{t} |x - c_k|^{-\beta}} \end{aligned}$$

To calculate this efficiently, we need to find the t nearest neighbors of x in the set of centroids.

A naive way to do this is to calculate distance of x from all c clusters, get the smallest t of them in sorted order and use that to calculate membership values. This will take time $O(cd + t \log c)$ if we use a heap.

A better method is to use a data structure which can calculate nearest neighbors of x without calculating all c distances. Examples of such data structures include KD-trees [citation needed], R-trees [citation needed] and ball trees [citation needed]. In this paper, we only consider KD-trees because they are simple to analyze and implement.

If the value of t is known beforehand, we can simply query the data structure for the nearest t neighbors. But we can't do that if the value of t is decided adaptively. For example, instead of getting the t nearest neighbors in one go, one could get neighbors iteratively in decreasing order of closeness to x. When we reach a neighbor which is so far away that we expect its membership value to be very low, we stop getting more neighbors.

Getting neighbors iteratively can be much more expensive than getting them in a single query, depending on the data structure used for nearest neighbor queries. We could settle for a compromise by getting neighbors in batches.

5 KD-Tree for Nearest Neighbor Queries

A KD-tree is a space partitioning data structure for organizing points in d-dimensional space. A KD-tree is a binary tree in which every node v contains

a d-dimensional point value(v). It can be thought of as a generalization of a binary search tree to d-dimensional points.

Let value$(v)_i$ be the i^{th} coordinate of value(v). Every internal node u at depth h in a KD-tree satisfies the following property:

- For each node v in the left sub-tree of u, value$(v)_{h\%d} \leq$ value$(u)_{h\%d}$.
- For each node v in the right sub-tree of u, value$(v)_{h\%d} \geq$ value$(u)_{h\%d}$.

(Here $a\%b$ is the remainder obtained when a is divided by b.)

Therefore, the root node splits points along the first dimension, the nodes at the next level split points along the second dimension, and so on, cyclically repeating dimensions.

5.1 Construction

Construction of a KD-tree is best done in an offline manner. The median of the points along the first dimension will be calculated. The median will become the root node, all points with first dimension less than the median will be part of the left sub-tree and the rest will go to the right sub-tree. Now each sub-tree will be created recursively, but the splitting dimension will be incremented.

Let the height of the tree be h and number of nodes in the tree be n. Since the difference between the number of nodes in the two sub-trees is at most 1, the tree will be balanced. Therefore, $h \approx \log_2 n$.

Number levels from 0 onward. Therefore, the root is at level 0, it's children are at level 1, and so on. Number of points in the first i levels $= \sum_{j=0}^{i-1} 2^j = 2^i - 1$. Therefore, $n - (2^i - 1)$ points reach level i during construction for further splitting.

$$
\begin{aligned}
&\text{Time taken to construct level } i \\
&= \text{Time taken to split data at each node in level } i \\
&\approx 2^i \left(\text{Time taken to split } \frac{n - 2^i + 1}{2^i} \text{ points} \right) \\
&\leq 2^i k \left(\frac{n - 2^i + 1}{2^i} \log_2 \frac{n - 2^i + 1}{2^i} \right) \\
&\qquad\qquad \text{(finding median of } n \text{ items takes } O(n \log_2 n) \text{ time)} \\
&= k(n - 2^i + 1)(\log_2(n - 2^i + 1) - i) \\
&\leq kn \log_2 n
\end{aligned}
$$

Since there are around $\log_2 n$ levels, construction time is $O(n \log_2^2 n)$.

5.2 Auxiliary data

For efficient nearest neighbor queries, we require upper and lower bound on coordinates of points for each dimension. Therefore, each node u will store two vectors $\text{cmin}(u)$ and $\text{cmax}(u)$. For each node v in the sub-tree at u, $\text{cmin}(u)_i \leq \text{value}(v)_i \leq \text{cmax}(u)_i$. For every leaf node v, $\text{cmin}(v)_i = \text{value}(v)_i = \text{cmax}(v)_i$.

Let $\min(x, y, z)$ denote the component-wise minimum of 3 points x, y, z. Similarly, let $\max(x, y, z)$ denote the component-wise maximum of 3 points x, y, z.

For a node u with children v_1 and v_2,

- $\text{cmin}(u) = \min(\text{value}(u), \text{cmin}(v_1), \text{cmin}(v_2))$
- $\text{cmax}(u) = \max(\text{value}(u), \text{cmax}(v_1), \text{cmax}(v_2))$

This is how cmin and cmax will be calculated for each node in the KD-tree. The tree will now occupy $O(nd)$ space and time to calculate auxiliary data is $O(nd)$.

5.3 Iterative Nearest Neighbors

The following algorithm will output points in descending order of proximity from an input point x upon successive calls. It is therefore a stateful algorithm. The state will be captured by a min-heap.

The measure of proximity that a KD-tree will support is Minkowski norm. Minkowski norm between points x and y is defined as

$$|x - y| = \sum_{i=1}^{d} |x_i - y_i|^w$$

where w is a hyper-parameter. Let $d(v)$ be the norm of x from a node v.

$$d(v) = \sum_{i=1}^{d} \begin{Bmatrix} \text{cmin}(v)_i - x_i & \text{if } x_i < \text{cmin}(v)_i \\ 0 & \text{if } \text{cmin}(v)_i \leq x_i \leq \text{cmax}(v)_i \\ x_i - \text{cmax}(v)_i & \text{if } x_i > \text{cmax}(v)_i \end{Bmatrix}^w$$

The norm of a point x from a node v is a lower bound on the minimum Minkowski norm between x and a point in the sub-tree of v.

The min-heap will store nodes of the KD-tree and points. The heap-priority of a node v is $d(v)$. The heap-priority of a point is it's distance from x. Let r be the root of the KD-tree. Initially the heap contains just one node - r.

A heap-update operation is defined as this algorithm:

---

if heap is empty **then**
  return NULL;
else
  Pop an item v from the heap.
  if v is a point **then**
    return v.
  else if v is a leaf node **then**
    return value(v).
  else
    insert value(v) and v's children in the heap.
    return v.
  end if
end if

To get a nearest neighbor, repeatedly apply the heap-update operation till the returned value is a point.

Worst-case time complexity of a heap-update operation is $O(\log_2 n + d)$.

Let $l(n)$ be the number of leaves in a balanced KD-tree with n nodes. The maximum possible number of heap-update operations equals the number of points in the KD-tree plus the number of internal nodes in the KD-tree. This is equal to $2n - l(n)$.

$l(n)$ satisfies the recurrence relation

$$l(n) = l\left(\left\lfloor \frac{n-1}{2} \right\rfloor\right) + l\left(\left\lceil \frac{n-1}{2} \right\rceil\right)$$

with base cases $l(0) = 0$ and $l(1) = 1$.

By using strong mathematical induction, it can be proved that

$$\frac{n+1}{3} \leq l(n) \leq \frac{n+1}{2} \ \forall\, n \geq 1$$

Hence, the maximum number of heap-update operations is at most $\frac{5n}{3}$.

5.4 Efficiency

The number of heap-update operations executed until k nearest neighbors are obtained equals the number of heap update operations required to be

executed for k points to be returned. In the best case, most of the first few heap-update operations will return points. In the worst case, heap-update operations will continually return nodes and only start returning points when there are hardly any nodes left in the heap.

In a KD-tree on n points in d dimensions, iteratively getting the k nearest neighbors of a point x in descending order of proximity takes time $O(n(d + \log_2 n))$ in the worst case. In the best case, time taken is $O(k(d + \log_2 n))$.

But even in the best case, this algorithm is useful for CT-means only if time taken is less than C-means. Time taken for a row update for C-means is $O(cd)$ and time taken in the best case for CT-means is $O(t(d + \log_2 c))$. So this algorithm is useful only if we stop at a sufficiently small value of t, i.e. the number of significant clusters for a point is small. The number of significant clusters will have to be less than a constant fraction of $\frac{cd}{d+\log_2 c}$.

6 Strategies for choosing t

6.1 Fixed-t strategy

As per this strategy, t is fixed and same for all points. Choosing t equal to 1 gives the K-means algorithm.

Generally a higher t will be better if the number of dimensions is high. Sometimes, there could be domain-specific information which can help make a better choice.

There are drawbacks of this strategy. First, we might use a very low value of t for some points or a very high value for t for some points or both. Using a very low t means that the algorithm fails to consider some nearby significant clusters. Using a very high value of t would increase computational cost.

An advantage of this strategy is that it doesn't require iterative nearest neighbor queries. Some data structures may be very good at non-iterative nearest neighbor queries but not so good at iterative nearest neighbor queries. Those data structures can be used with this strategy.

K-means is a very popular clustering algorithm, which is an instance of this strategy with $t = 1$. Other fixed values of t might give similar or better results in practice.

6.2 Variable-t strategy

This strategy gets nearest-neighbors iteratively, stopping when it sees appropriate. Assume that when it stopped, t nearest neighbors were obtained.

After that, it computes the membership value of a point for those t nearest centroids.

Let c_j be the j^{th} closest centroid to a point x. Let u_j be the corresponding membership value chosen by the CT-means algorithm and μ_j be the corresponding membership value chosen by the C-means algorithm. Let $e_j = |u_j - \mu_j|$ be the j^{th} error. We will now discuss a strategy which will choose a value of t such that $e_j \leq \alpha$ for all j.

$$\mu_j = \frac{|x - c_j|^{-\beta}}{\sum_{k=1}^{c} |x - c_k|^{-\beta}}$$

$$u_j = \begin{cases} \dfrac{|x - c_j|^{-\beta}}{\sum_{k=1}^{t} |x - c_k|^{-\beta}} & \text{if } j \leq t \\ 0 & \text{if } j > t \end{cases}$$

$$v_j = \begin{cases} \dfrac{|x - c_j|^{-\beta}}{\left(\sum_{k=1}^{t} |x - c_k|^{-\beta}\right) + (c - t)|x - c_t|^{-\beta}} & \text{if } j \leq t \\ \dfrac{|x - c_t|^{-\beta}}{\left(\sum_{k=1}^{t} |x - c_k|^{-\beta}\right) + (c - t)|x - c_t|^{-\beta}} & \text{if } j > t \end{cases}$$

Lemma 6.1. $j \leq t \Rightarrow v_j \leq \mu_j \leq u_j$

Proof.

$$
\begin{aligned}
&(k > t \Rightarrow |x - c_k| \geq |x - c_t|) \\
&\Rightarrow (k > t \Rightarrow 0 \leq |x - c_k|^{-\beta} \leq |x - c_t|^{-\beta}) \\
&\Rightarrow \sum_{k=t+1}^{c} 0 \leq \sum_{k=t+1}^{c} |x - c_k|^{-\beta} \leq \sum_{k=t+1}^{c} |x - c_t|^{-\beta} \\
&\Rightarrow 0 \leq \sum_{k=t+1}^{c} |x - c_k|^{-\beta} \leq (c - t)|x - c_t|^{-\beta} \\
&\Rightarrow \sum_{k=1}^{t} |x - c_k|^{-\beta} \leq \sum_{k=1}^{c} |x - c_k|^{-\beta} \leq \sum_{k=1}^{t} |x - c_k|^{-\beta} + (c - t)|x - c_t|^{-\beta} \\
&\Rightarrow \frac{|x - c_j|^{-\beta}}{\sum_{k=1}^{t} |x - c_k|^{-\beta} + (c - t)|x - c_t|^{-\beta}} \leq \frac{|x - c_j|^{-\beta}}{\sum_{k=1}^{c} |x - c_k|^{-\beta}} \leq \frac{|x - c_j|^{-\beta}}{\sum_{k=1}^{t} |x - c_k|^{-\beta}} \\
&\Rightarrow (j \leq t \Rightarrow v_j \leq \mu_j \leq u_j)
\end{aligned}
$$

□

Theorem 6.2.

$$\sum_{j=1}^{c} u_j = \sum_{j=1}^{c} \mu_j = \sum_{j=1}^{c} v_j = 1$$

Proof.

$$\sum_{j=1}^{c} u_j = \sum_{j=1}^{t} u_j = \sum_{j=1}^{t} \frac{|x - c_j|^{-\beta}}{\sum_{k=1}^{t} |x - c_k|^{-\beta}} = \frac{\sum_{j=1}^{t} |x - c_j|^{-\beta}}{\sum_{k=1}^{t} |x - c_k|^{-\beta}} = 1$$

$$\sum_{j=1}^{c} \mu_j = \sum_{j=1}^{c} \frac{|x - c_j|^{-\beta}}{\sum_{k=1}^{c} |x - c_k|^{-\beta}} = \frac{\sum_{j=1}^{c} |x - c_j|^{-\beta}}{\sum_{k=1}^{c} |x - c_k|^{-\beta}} = 1$$

$$
\begin{aligned}
&\sum_{j=1}^{c} v_j = \sum_{j=1}^{t} v_j + \sum_{j=t+1}^{c} v_j \\
&= \sum_{j=1}^{t} \frac{|x - c_j|^{-\beta}}{\left(\sum_{k=1}^{t} |x - c_k|^{-\beta}\right) + (c - t)|x - c_t|^{-\beta}} \\
&+ \sum_{j=t+1}^{c} \frac{|x - c_t|^{-\beta}}{\left(\sum_{k=1}^{t} |x - c_k|^{-\beta}\right) + (c - t)|x - c_t|^{-\beta}} \\
&= 1
\end{aligned}
$$

□

Since u and v sum to 1, they lie between 0 and 1 and they decrease with increasing distance to a centroid, u and v are valid membership functions. Additionally, u satisfies the constraint of having at most t non-zero values.

Theorem 6.3. $j > t \Rightarrow \mu_j \leq u_t$

Proof.

$$
\begin{aligned}
&j > t \\
&\Rightarrow |x - c_j| \geq |x - c_t| \\
&\Rightarrow |x - c_j|^{-\beta} \leq |x - c_t|^{-\beta} \\
&\Rightarrow \mu_j \leq \mu_t \\
&\Rightarrow \mu_j \leq u_t && \text{(by lemma 6.1)}
\end{aligned}
$$

□

Theorem 6.4. $u_j - \mu_j \leq u_1 - v_1$

Proof.

$$
\begin{aligned}
&j \leq t \\
&\Rightarrow |u_j - \mu_j| \\
&\leq |u_j - v_j| && \text{(by lemma 6.1)} \\
&= \frac{|x - c_j|^{-\beta}}{|x - c_1|^{-\beta}}(u_1 - v_1) \\
&= \frac{\mu_j}{\mu_1}(u_1 - v_1) \\
&\leq (u_1 - v_1) && (\mu_1 \geq \mu_j)
\end{aligned}
$$

□

Theorem 6.5. *u_t, u_1 and $-v_1$ decrease with increasing t.*

Proof.

$$P = \sum_{k=1}^{t} |x - c_k|^{-\beta} \qquad d = |x - c_t|^{-\beta}$$
$$d_1 = |x - c_1|^{-\beta} \qquad h = d - |x - c_{t+1}|^{-\beta}$$

$$d - h = |x - c_{t+1}|^{-\beta} > 0$$

$$|x - c_t| \leq |x - c_{t+1}| \Rightarrow |x - c_t|^{-\beta} \geq |x - c_{t+1}|^{-\beta}$$
$$\Rightarrow d \geq d - h \Rightarrow h \geq 0$$

$$f(t) = u_t = \frac{|x - c_t|^{-\beta}}{\sum_{k=1}^{t} |x - c_k|^{-\beta}}$$
$$= \frac{d}{P}$$

$$f(t+1) = \frac{|x - c_{t+1}|^{-\beta}}{\sum_{k=1}^{t+1} |x - c_k|^{-\beta}}$$
$$= \frac{d - h}{P + (d - h)}$$

$$f(t) - f(t+1)$$
$$= \frac{d}{P} - \frac{d - h}{P + (d - h)}$$
$$= \frac{d(d - h) + Ph}{(P + (c - t)(d - h))(P + (c - t)d)}$$

Since $d(d-h) > 0$ and $Ph > 0$, $f(t)-f(t+1) > 0$. Therefore, u_t increases with t.

$$\begin{aligned}
&\text{Denominator of } v_1 = g(t) \\
&= \sum_{k=1}^{t} |x - c_k|^{-\beta} + (c-t)|x - c_t|^{-\beta} \\
&= P + (c-t)d
\end{aligned}$$

$$\begin{aligned}
&g(t+1) \\
&= \sum_{k=1}^{t+1} |x - c_k|^{-\beta} + (c-t-1)|x - c_{t+1}|^{-\beta} \\
&= (P + (d-h)) + (c-t-1)(d-h) \\
&= P + (c-t)(d-h)
\end{aligned}$$

$$\begin{aligned}
&g(t) - g(t+1) \\
&= (P + (c-t)d) - (P + (c-t)(d-h)) \\
&= (c-t)h > 0
\end{aligned}$$

Since denominator of v_1 decreases with t and numerator of v_1 is independent of t, v_1 increases with t. Therefore, $-v_1$ decreases with t.

Numerator of u_1 is independent of t and denominator of u_1 increases with t. Therefore, u_1 decreases with t.

□

Using theorem 6.5, $u_1 - v_1$ and u_t both decrease as t increases. So by using theorem 6.3 and theorem 6.4, we get this stopping condition:

$$(u_1 - v_1 \leq \alpha) \text{ and } (u_t \leq \alpha)$$

Getting nearest neighbors incrementally means that t will be increased from 1 to at most c. Whenever t is incremented, we can update the value of u_1, v_1 and u_t in $O(1)$.

6.3 Stronger variable-t strategy

We can also obtain a variable-t strategy which imposes an even stronger constraint on errors. We now discuss a t-choosing strategy which ensures that:

- The sum of first t errors is less than α.
- The sum of last $c-t$ errors is less than α.

Theorem 6.6.

$$\sum_{j=1}^{t}(u_j - \mu_j) = \sum_{j=t+1}^{c} \mu_j \leq \sum_{j=t+1}^{c} v_j = \sum_{j=1}^{t}(u_j - v_j)$$

Proof.

$$\sum_{j=1}^{t}(u_j - \mu_j) = \sum_{j=1}^{t} u_j - \sum_{j=1}^{t} \mu_j = 1 - \sum_{j=1}^{t} \mu_j$$
$$= \sum_{j=1}^{c} \mu_j - \sum_{j=1}^{t} \mu_j = \sum_{j=t+1}^{c} \mu_j$$

$$\sum_{j=1}^{t}(u_j - v_j) = \sum_{j=1}^{t} u_j - \sum_{j=1}^{t} v_j = 1 - \sum_{j=1}^{t} v_j$$
$$= \sum_{j=1}^{c} v_j - \sum_{j=1}^{t} v_j = \sum_{j=t+1}^{c} v_j$$

$$(j \leq t \Rightarrow v_j \leq \mu_j \leq u_j)$$
$$\Rightarrow \sum_{j=1}^{t} v_j \leq \sum_{j=1}^{t} \mu_j$$
$$\Rightarrow 1 - \sum_{j=1}^{t} \mu_j \leq 1 - \sum_{j=1}^{t} v_j$$
$$\Rightarrow \sum_{j=1}^{c} \mu_j - \sum_{j=1}^{t} \mu_j \leq \sum_{j=1}^{c} v_j - \sum_{j=1}^{t} v_j$$
$$\Rightarrow \sum_{j=t+1}^{c} \mu_j \leq \sum_{j=t+1}^{c} v_j$$

□

$$S_t = \sum_{j=t+1}^{c} v_j = \frac{(c-t)|x-c_t|^{-\beta}}{\sum_{k=1}^{t} |x-c_k|^{-\beta} + (c-t)|x-c_t|^{-\beta}}$$

Theorem 6.7. *S_t decreases as t increases.*

Proof.

$$P = \sum_{k=1}^{t} |x-c_k|^{-\beta} \qquad d = |x-c_t|^{-\beta} \qquad h = d - |x-c_{t+1}|^{-\beta}$$

$$|x-c_t| \le |x-c_{t+1}| \Rightarrow |x-c_t|^{-\beta} \ge |x-c_{t+1}|^{-\beta}$$
$$\Rightarrow d \ge d-h \Rightarrow h \ge 0$$

$$\begin{aligned} S_t &= \frac{(c-t)|x-c_t|^{-\beta}}{\sum_{k=1}^{t} |x-c_k|^{-\beta} + (c-t)|x-c_t|^{-\beta}} \\ &= \frac{(c-t)d}{P+(c-t)d} \end{aligned}$$

$$\begin{aligned} S_{t+1} &= \frac{(c-t-1)|x-c_{t+1}|^{-\beta}}{\sum_{k=1}^{t+1} |x-c_k|^{-\beta} + (c-t-1)|x-c_{t+1}|^{-\beta}} \\ &= \frac{(c-t-1)(d-h)}{(P+(d-h)) + (c-t-1)(d-h)} \\ &= \frac{(c-t-1)(d-h)}{P+(c-t)(d-h)} \end{aligned}$$

$$\begin{aligned} &S_t - S_{t+1} \\ &= \frac{(c-t)d}{P+(c-t)d} - \frac{(c-t-1)(d-h)}{P+(c-t)(d-h)} \\ &= \frac{(c-t)d(P+(c-t)(d-h)) - (c-t-1)(d-h)(P+(c-t)d)}{(P+(c-t)(d-h))(P+(c-t)d)} \\ &= \frac{hP(c-t) + (c-t)d(d-h) + P(d-h)}{(P+(c-t)(d-h))(P+(c-t)d)} \end{aligned}$$

Since all terms in the numerator are non-negative, $S_t - S_{t+1} > 0$.

□

Theorem 6.6 states that the sum of errors in the first t terms and the sum of errors in the remaining $c-t$ terms is the same and they are both less than S_t. Also, theorem 6.7 states that S_t decreases with increasing t. Therefore, $S_t \leq \alpha$ (α is a constant) can be used as a stopping criterion.

Getting nearest neighbors incrementally means that t will be increased from 1 to at most c. Whenever t is incremented, we can update the value of S_t in $O(1)$.

7 Conclusion

Updating centroids takes time $O(ndt)$. t is the average number of significant clusters per point. For C-means, this time becomes $O(ndc)$. Therefore, CT-means is at least as fast as C-means in this step. (On ignoring difference between access speed of sparse matrix vs dense matrix)

In C-means, updating membership values takes time $O(ncd)$. CT-means without nearest neighbor queries takes time $O(n(cd + t\log_2 c))$ for membership matrix update.

CT-means with nearest neighbor queries takes time $O(c(d + \log_2^2 c))$ for constructing a KD-tree and $O(np(d + \log_2 c))$ for updating membership matrix. Here p is the maximum number of heap-update operations, which is at least t and at most $\frac{3c}{2}$.

The practical utility of this algorithm depends on two factors:

1. The value of t: A higher value of t requires more computation. Whether a low value of t can give good clusters or good convergence would depend a lot on the dataset. For variable-t strategy, the clusters and convergence rate are expected to be very similar to C-means, but the running time will be high if the average number of significant clusters for a point is high. In general, number of significant clusters is expected to be high for high-dimensional data [proof needed].

2. The efficiency of nearest-neighbors querying: If the data structure for finding nearest neighbors is not efficient, which is the case with KD-trees and other popular data structures, then running time will be high. In the worst case, most algorithms would scan all points in the data structure. This is especially true for high-dimensional data. In practice, KD-trees are not useful when number of dimensions is greater

than around 20 [citation needed]. The efficiency of nearest neighbor queries generally varies across datasets.

Therefore, whether CT-means is faster than C-means in practice can be determined only by experimentation. The theoretical results only claim that it's not terribly slow compared to C-means.