

Clustering of Points

Afshin Sepehri and Nargess Memarsadeghi
University of Maryland

Spring 2004

Given a set of data points, we worked on their clustering and visualization of results. We assumed that for each data point P , we were given its position (x, y, z) , its color (r, g, b) , and its normal (n_1, n_2, n_3) where $x, y, z \in [-1, 1]$, $r, g, b \in [0, 1]$, and $\|n\| = 1$. In case the points do not satisfy these conditions, an extra normalizing step would be required. We approached this project in three steps: rendering the input data and clustering results, clustering algorithms, and evaluation of the results. We describe each part more below.

1. **Clustering algorithms:** We mainly focused on two clustering algorithms for this project: *K-Means* and *Fuzzy K-Means*. Both of these algorithms depend highly on calculation of a distance function.

For two points P_1 and P_2 , a general approach for defining their distance would be a linear combination as follows:

$$d(P_1, P_2) = k_{pos}d_{pos}(P_1, P_2) + k_{col}d_{col}(P_1, P_2) + k_{norm}d_{norm}(P_1, P_2) \quad (1)$$

where d_{pos} , d_{col} and d_{norm} are three distance measures depending on position, color, and normal of the points respectively. In this project considering the range of the point parameters, we defined these distance measures as

$$d_{pos}(P_1, P_2) = d_{eucl}^2(P_{x1}, P_{x2}, P_{y1}, P_{y2}, P_{z1}, P_{z2})/12 \quad (2)$$

$$d_{col}(P_1, P_2) = d_{eucl}^2(P_{r1}, P_{r2}, P_{g1}, P_{g2}, P_{b1}, P_{b2})/3 \quad (3)$$

$$d_{norm}(P_1, P_2) = (1 - \langle P_1, P_2 \rangle)/2 \quad (4)$$

where d_{eucl} is the Euclidean distance of the two points' position or color vectors, and $\langle P_1, P_2 \rangle$ is the dot product of the normals of the two points. k_1, k_2 , and k_3 are weights assigned to each of these distance measures with conditions $k_1 + k_2 + k_3 = 1$ and $k_1, k_2, k_3 \geq 0$. We usually consider position values to have greater importance in cluster determination of points than color and normal values. Next important features will be color and normal values accordingly. In our program, user can select these parameters in percent manually.

2. **Implemented Algorithms:** We implemented the following algorithms:

- (a) **K-Means:** This is an iterative algorithm which clusters points so that sum of distances of points from their cluster centers are minimized. The algorithm takes desired number of clusters, k , as input, and outputs k centers, and reports the cluster to which each point belongs to. The algorithm starts by selecting k centers at random from data points as initial set of centers. Then, the algorithm proceeds by calculating distances of each point from each cluster center as (1). Each

point is assigned to a cluster whose center is closer to that point than any other cluster center. After assigning clusters to each data point, each cluster center is updated to be the vector mean of all points in that particular cluster. Calculating distances and updating centers are repeated until centers converged. It is important to notice that since we are always updating centers in direction of the minimizer of our objective function, the algorithm will always converge, however, it may converge to a local minima rather than a global one.

Finally, for evaluation purposes, we calculate average distance of points in each cluster from their cluster center, as well as over all average distance of all points from their cluster centers.

- (b) **Fuzzy K-Means:** Implementation of this algorithm is very similar to the K-Means. Instead of calculating exact distances of points to all cluster centers, we calculate a membership function indicating how much each point belongs to each cluster. Output of what so called membership function is a value between 0 and 1, so that sum of membership function values of each point to all clusters is 1. Each point is assigned to a cluster for which it had the highest membership value. More details on how distances and memberships are calculated, as well as how centers are updated can be found at the below mentioned link.

<http://www.usyd.edu.au/su/agric/acpa/fkme/FkME.html>

Similar to K-Means, this algorithm converges when centers are converged (in this case within a threshold). Average distances are also calculated at the end of the algorithm as described for K-Means algorithm.

- (c) **Automatic K-means:** This algorithm is very similar to K-Means except that instead of getting number of clusters, k , as input, it decides automatically what would be the best k for a given data set. Here is how the algorithm decided on best k to report:

The algorithm starts by setting $k = 1$. K-Means gets run with $k = 1$. This decision is made based on overall average distance of all points from their cluster centers. That is, we keep increasing k and running K-Means until we see that overall average distortion has changed less than a threshold (this is called *ACCURACY* in our code and can be determined manually by the user). Please notice that it is important to let the stopping criterion be viewing small change in overall distortion rather than stopping if distortion did not decrease. The reason is, if we keep increasing k as long as distortions are decreasing, for all data set, the k which is going to be reported will be equal to number of points (where every point is a single cluster by itself and overall distortion is zero).

- (d) **Automatic Fuzzy K-Means:** This is implemented exactly like Automatic K-Means algorithm, except that for each k we run the Fuzzy K-Means algorithm instead of the K-Means algorithm.

3. **Algorithms' Coding Design:** The code is implemented in C++, using OpenGL and MFC. We implemented it by designing and implementing the following data structures:

- (a) *TPoint*: any object of this type is representative of a point which has information about its color, position, normal vector and its cluster. In other words, it has (r, g, b) , (x, y, z) , and (N_x, N_y, N_z) values. It also has an integer indicating which cluster the point belongs to (this value gets determined in clustering algorithms, and initially one can assign random cluster labels to each point. Simple operations needed for points, such as assignment, additions, and testing for equality, are also implemented to support objects of this type.
- (b) *TPointList*: contains an array of *TPoints* as well as an integer indicating number of points in the array.
- (c) *TCluster*: contains a *TPoint* representing center of the cluster, as well as an integer representing number of points in the cluster. In order to make the process of updating centers and calculations of average distances easy, we store some additional information in each object of type *TCluster* as well. That is, we also have a *TPoint* which is the vector sum of all points in the cluster, and a float that stores sum of distances of all points in the cluster from their center. This way, as

points are visited and distances are calculated, these values can easily be updated (assuming we initialized them to zero vector and number accordingly upon instantiation of the object), and thus, averages can be easily calculated without the need of going through the data an additional time.

- (d) *TClusterList*: is mainly composed of an array of *TClusters* and an integer indicating number of *TClusters* in the array. it also has a field to store overall average distance of all points from their cluster centers. This is our base class for all our clustering algorithms from which classes related to each clustering algorithm are inherited. Thus, main functions needed in all clustering algorithms (ie. one that gets called to perform clustering, update centers,etc) are defined as virtual functions so that for each clustering algorithm, its associated function in the inherited class, with the same name in the base class, gets called.
- (e) *TKMeansClusterList*: is inherited from *TClusterList*.
- (f) *TFuzzyKMeansClusterList*: is also inherited from *TClusterList*
- (g) *TAutoKMeansClusterList*: is inherited from *TKMeansClusterList*.
- (h) *TFuzzyAutoKMeansClusterList*: is inherited from *TFuzzyKMeansClusterList*.

4. **User Interface:** The user interface is implemented using MFC and OpenGL.

As the user runs the program, a window will appear which has a toolbar with various icons on top of it. These icons as well as some key combinations support various operations, mainly reading data, manipulating it, and writing the output:

- (a) Input: the software can either read data in the particular *.PT* format from a file and render it on screen, or the users can generate random data if they choose to. All data points values, regardless of whether they have been read from a file or generated by user, are stored in a normalized manner: color and position fields have values between 0 and 1, and normal vector fields have values between -1 and 1.
- (b) Capabilities: user can perform any of the four mentioned clustering algorithms by clicking on their associated icons. User can also rotate, zoom, and translate the images through mouse movements. In particular rotation can be done by right clicking and dragging the image, translation is performed by left clicking and dragging the image, and zooming is performed as a translation of camera in z direction by holding *Ctrl* key and left mouse button and dragging the mouse. Rotation about $z - axis$ can be also performed by holding *Ctrl* key and right mouse button and dragging the mouse.
- (c) Output: after performing each of the mentioned operations on the image, as well as performing any clustering, an output image is rendered on the screen. For clustering algorithms' outputs, all points in one cluster have the same color. In particular, for K-Means all point's in one cluster will have the same (r, g, b) value as their center, which can be the average color of colors of all points' in that cluster or can be assigned randomly. The method of color assignment can be selected through program Options. The final result image can be stored on a file in the same format as input, that is *.PT*.

5. **Evaluation:** We compared the performance of the two algorithm by looking at values of the objective function that they were trying to minimize. That is, for each cluster, we look at sum of distances of points from their centers (where *distance* will be defined similar to above), and we can come up with an average overall distortion value (by calculating average of distances of points from their cluster center), and then see which method gave better results, which means lower overall distortion. To do a right comparison, it is important that distortions are calculated in the same manner. To achieve

this, we added a post processing function to Fuzzy K-Means so that it calculates distances and overall distortion of the final result in the same manner as K-Means does.

6. Conclusion:

In summary, given a set of data points consisting of their position, color, and normal information, we implement two clustering algorithms that consider all the mentioned features of points in its distance function. We will also provide the user the rendering and visualization capabilities for the results, as well as the capability to translate, zoom, and rotate the results and the input image.

We ran our algorithms on the provided test data, and for a given data set and a particular k , K-Means algorithm gives better and lower overall distortion in results than Fuzzy K-Means.