Adaptive K-Means Clustering

Sanjiv K. Bhatia
Department of Mathematics & Computer Science
University of Missouri – St. Louis
St. Louis, MO 63121
sanjiv@acm.org

Abstract

Clustering is used to organize data for efficient retrieval. One of the problems in clustering is the identification of clusters in given data. A popular technique for clustering is based on K-means such that the data is partitioned into K clusters. In this method, the number of clusters is predefined and the technique is highly dependent on the initial identification of elements that represent the clusters well. A large area of research in clustering has focused on improving the clustering process such that the clusters are not dependent on the initial identification of cluster representation. In this paper, I advance an adaptive technique that grows the clusters without regard to initial selection of cluster representation. As such, the technique can identify K clusters in an input data set by merging existing clusters and by creating new ones while keeping the number of clusters constant. The technique has been used to achieve an impressive speedup of a search process when other efficient search techniques may not be available.

Introduction

Clustering is a technique that is used to partition elements in a data set such that similar elements are assigned to same cluster while elements with different properties are assigned to different clusters. Clustering is used to perform effi cient search of elements in a data set. Clustering is particularly effective in multi-dimensional data that may be otherwise diffi cult to organize in an effective manner. An example of such data can be spectral reflectance and emittance properties of materials used in computation of their spectral signature. Such data is typically represented as a floating-point number for each property such that different properties may be represented on vastly different scales. The number of properties and the use of different scale for each property make it extremely hard to organize such data by conventional means. The problem is harder if the search is expected to return a data item that is closest in properties to the item being searched for, in case the item does not exist in the search space. We cannot use any of the conventional data structures such as a sorted array or a hash table to solve this search problem. We cannot use sorted arrays due to multidimensional nature of data. Hash tables cannot be used because we may want to retrieve an item that is closest in properties to a specifi ed item when the specifi ed item does not exists in the data set. Clustering provides an elegant solution to this problem while providing a fast search capability for the same.

One of the earliest clustering techniques in the literature is the K-means clustering method (Anderberg 1973; Romesburg 1984). In this technique, clustering is based on the identifi cation of K elements in the data set that can be used to create an initial representation of clusters. These K elements form the *cluster seeds*. The remaining elements in the data set are then assigned to one of these clusters. Even though the method seems to be straightforward, it suffers from the fact that it may not be easy to clearly identify the initial K elements, or the seeds for the clusters. This drawback led the researchers to look into alternative methods that provide an improvement over K-means. Some of these techniques include genetic algorithm based clustering (Bandyopadhyay & Maulik 2002) and fuzzy clustering (Belacel, Hansen, & Mladenovic 2002).

In this paper, I have presented a technique that allows the partitioning of a given data set without having to depend on the initial identifi cation of elements to represent clusters. The technique is based on rearranging the clusters to better reflect the partitions when new elements are added. In addition, some clusters may be merged and new clusters are created as needed. I also describe a technique where the clustering can be based on a specifi ed threshold in which case the number of clusters is unknown until all the elements have been clustered. Both the techniques are adaptive in nature and have been inspired by clustering of documents proposed in the realm of information retrieval (Yu, Wang, & Chen 1985; Bhatia & Deogun 1998).

This paper is organized as follows. In the next section, I present the algorithm for adaptive K-means clustering. This is followed by another section to describe a second clustering algorithm that is based on specifi cation of threshold. Then, I present the asymptotic analysis of search based on clustered data. This is followed by an application where I have been able to successfully generate a perceptually uniform color table under a specifi ed set of constraints.

Algorithm for Adaptive K-Means Clustering

The adaptive K-means clustering algorithm starts with the selection of K elements from the input data set. The K elements form the seeds of clusters and are randomly selected. The properties of each element also form the properties of the cluster that is constituted by the element.

The algorithm is based on the ability to compute distance between a given element and a cluster. This function is also used to compute distance between two elements. An important consideration for this function is that it should be able to account for the distance based on properties that have been normalized so that the distance is not dominated by one property or some property is not ignored in the computation of distance. In most cases, the Euclidean distance may be sufficient. For example, in the case of spectral data given by n-dimensions, the distance between two data elements $E_1 = \{E_{11}, E_{12}, \ldots, E_{1n}\}$ and $E_2 = \{E_{21}, E_{22}, \ldots, E_{2n}\}$ is given by

$$\sqrt{(E_{11} - E_{12})^2 + (E_{12} - E_{22})^2 + \cdots + (E_{1n} - E_{2n})^2}$$

It should be pointed out that for performance reasons, the square root function may be dropped. In other cases, we may have to modify the distance function. Such cases can be exemplified by data where one dimension is scaled different compared to other dimensions, or where properties may be required to have different weights during comparison.

With the distance function, the algorithm proceeds as follows:

Compute the distance of each cluster from every other cluster. This distance is stored in a 2D array as a triangular matrix. We also note down the minimum distance $d_{\min}$ between any two clusters C_{m_1} and C_{m_2} as well as the identification of these two closest clusters.

For each unclustered element E_i, compute the distance of E_i from each cluster. For assignment of this element to a cluster, there can be three cases as follows:

1. If the distance of the element from a cluster is 0, assign the element to that cluster, and start working with the next element.
2. If the distance of the element from a cluster is less than the distance $d_{\min}$, assign this element to its closest cluster. As a result of this assignment, the cluster representation, or centroid, may change. The centroid is recomputed as an average of properties of all elements in the cluster. In addition, we recompute the distance of the affected cluster from every other cluster, as well as the minimum distance between any two clusters and the two clusters that are closest to each other.
3. The last case occurs when the distance $d_{\min}$ is less than the distance of the element from the nearest cluster. In this case, we select the two closest clusters C_{m_1} and C_{m_2}, and merge C_{m_2} into C_{m_1}. Also, we destroy the cluster C_{m_2} by removing all the elements from the cluster and by deleting its representation. Then, we add the new element into this now empty cluster, effectively creating a new cluster. The distances between all clusters are recomputed and the two closest clusters identified again.

The above three steps are repeated until all the elements have been clustered. There is a possibility that the algorithm identifies a number of singletons or single-element clusters if the distance of some elements is large from other elements. These elements are known as *outliers* and can be accounted for by looking for clusters with an extremely small number of elements and removing those elements from clustering consideration, or handled as exceptions.

Threshold-Based Clustering Algorithm

In the threshold-based clustering algorithm, the number of clusters is unknown. However, two elements are classified to the same cluster if the distance between them is below a specified threshold. The algorithm proceeds as follows:

- Select an element from the give data set. This element is assigned as the seed of a cluster by itself.
- For every unclassified element, find its distance from the centroid of the existing clusters. If the distance is less than the threshold, assign the element to this cluster. Recompute the centroid of the cluster as the average of all properties of all elements in the cluster. If no such cluster can be found after examining all current clusters, assign the element as the seed for a new cluster.
- If, as a result of the above step, the distance of new cluster to another cluster is smaller than the threshold, merge the two close clusters together, and recompute the cluster distances.
- The algorithm stops after all the elements have been assigned to one or the other cluster.

It is easy to see that if the threshold is small, all the elements will get assigned to different clusters. If the threshold is large, the elements may get assigned to just one cluster. Thus, this algorithm is sensitive to the specification of threshold. The threshold can be reduced or increased to get an appropriate number of clusters by repeated iterations.

Analysis of Clustering Algorithms

The two clustering algorithms presented in this paper are easy to analyze. The adaptive K-means algorithm is analyzed first.

In the adaptive K-means algorithm, the number of elements in each cluster decides the number of comparisons for each search. Since there are K clusters, we have to perform K comparisons to find the most promising cluster for further exploration. In the worst case, all the clusters except one are single-element clusters. If the cluster with $N - K$ elements is selected for further exploration, the algorithm performs K comparisons to get the appropriate cluster, and $N - K$ comparisons to look for the appropriate element within that cluster. Thus, the algorithm results in $O(N)$ comparisons which makes it perform just like linear search.

In the best case, the data elements are distributed uniformly across clusters. In such a case, each cluster contains N/K elements. A search in this case will take at most K comparisons to determine the appropriate cluster, and N/K comparisons to determine the appropriate element. Thus, the number of comparisons is given by $O(K + N/K)$. If $K = \sqrt{N}$, the number of comparisons equals $O(\sqrt{N})$.

In the average case, the data is distributed across the clusters such that some clusters will have a number of elements while some other clusters may contain few elements. The analysis of such a case is more complex but lies between $O(N)$ and $O(\sqrt{N})$, as shown below in three cases with search in sparse cluster, dense cluster, and averaging over sparse and dense clusters.

Sparse Cluster. If the element is in a sparse cluster, we can safely assume a constant number of comparisons for search. Thus, for elements in i sparse clusters, the number of comparisons is $K \cdot i$, with K comparisons to identify the cluster to which the element belongs and almost negligible (constant number) of comparisons to actually search within clusters.

Dense Cluster. For elements in dense clusters, we make the assumption that elements are uniformly distributed across the dense clusters. Obviously, the number of dense clusters is small compared to the number of sparse clusters. As an example, if there are two dense clusters, we can assume that each of the two dense clusters has $N/2$ elements each. Thus, search of $N - i$ elements in $K - i$ clusters requires the number of comparisons given by

$$(N-i)\left(K + \frac{N}{K-i}\right)$$

with K comparisons to determine the cluster and $\frac{N}{K-i}$ comparisons to search within the dense clusters.

Averaging Over Sparse and Dense Clusters. The average number of comparisons over all the elements is given by

$$\begin{aligned}
& \frac{1}{N}\left[K \cdot i + (N-i)\left(K + \frac{N}{K-i}\right)\right] \\
= & \frac{1}{N}\left[K \cdot i + (N-i)K + \frac{N(N-i)}{K-i}\right] \\
= & \frac{1}{N}\left[KN + \frac{N(N-i)}{K-i}\right] \\
= & K + \frac{N-i}{K-i} \\
= & K + \frac{K^2-i}{K-i} \qquad \text{Substituting } N = K^2
\end{aligned}$$

It is easily proved by induction that $(K^2 - i) < (K - i)^2$. Therefore, the above expression can be changed to

$$\begin{aligned}
< & K + \frac{(K-i)^2}{K-i} \\
= & K + (K-i) \\
< & 2K \qquad \text{Since } i < K \\
= & O(\sqrt{N}) \qquad \text{Substituting } N = K^2
\end{aligned}$$

From the above analysis, it is clear that the search takes $O(\sqrt{N})$ comparisons on an average. However, the major advantage of the algorithm is when an exact match is not found and the algorithm needs to return the closest match. In such a case, the algorithm still performs $O(\sqrt{N})$ comparisons while the linear search needs $O(N)$ comparisons in case of multi-dimensional data. Furthermore, the algorithm can be used to determine the kth closest match by rejecting a number of elements in a cluster, or by selecting a cluster that may contain the kth closest element.

The analysis of threshold-based algorithm proceeds in a similar manner. In case the threshold is too small, the number of clusters will be large, resulting in a linear search for an element. In case the threshold is too large, the number of clusters will be small, resulting in an almost linear behavior for search again. The best search performance is achieved when the number of clusters is such that the elements are evenly distributed across clusters. If the number of clusters is $K = \sqrt{N}$, the performance of the algorithm is the same as the adaptive K-means algorithm. This algorithm requires a careful assignment of threshold, either by examination of data to be clustered or by repeated iterations.

Application: Creation of Perceptually Uniform Colors

In this section, I'll describe the task of generating a set of n colors that are uniformly distributed in perceptual color space (Poynton 1995). A color space is defined as the set of possible colors in a given color representation. For example, an RGB color is defined as a tuple with three values for red, green, and blue components. Each of these values can change from 0 to 255 to account for 8-bit components. The RGB color space is made up of all possible colors with different combinations of values of the three components. RGB color space is not perceived uniformly by humans because different colors are perceived with different weights by the eye.

The task to determine perceptually uniform distribution of colors was simple on first look with the knowledge that a perceptual color space is defined by CIE as L*a*b* space (Ald 1992) and it is a simple matter of converting L*a*b* colors to RGB color space using standard techniques. However, the n desired colors were constrained to be inside a gamut specified by a triangle in xyY color space which made the problem more interesting.

I solved the problem by iterating over all the colors in L*a*b* space and ignoring the colors that are not inside the specified color gamut. After the colors were plotted in xyY space (with Y kept as 1 for maximum brightness), we noticed a concentration of colors near the triangle edges. This happened because L*a*b* colors are being forced into an xyY gamut; with colors in L*a*b* that have plenty of space between them, resulting in colors very close together in xyY space. I solved the problem by generating a table of $2n$ colors in L*a*b* space, and then clustering the resulting colors in xyY space.

In L*a*b* space, the value of L ranges from 0 to 100, while a and b range between -128 to $+127$. I worked with floating point values incrementing the counters by 1 for each of L, a, and b, resulting in 64 million colors. When I constrained the colors in the given gamut, I ended with a little more than 55,000 colors in perceptual space that needed to be uniformly distributed. I wanted to generate a color table of 4,096 colors. Therefore, I clustered the constrained colors into 8,192 clusters in L*a*b* space, and clustered the resulting elements into 4,096 colors in xyY space.

The color distribution in L*a*b* space is shown in Figure 1. The ring results because of reclustering of colors in xyY space constrained to the triangular gamut shown in Figure 2. In Figure 3, I show the distribution of the same colors in RGB color space, and the final color palette is presented in Figure 4.

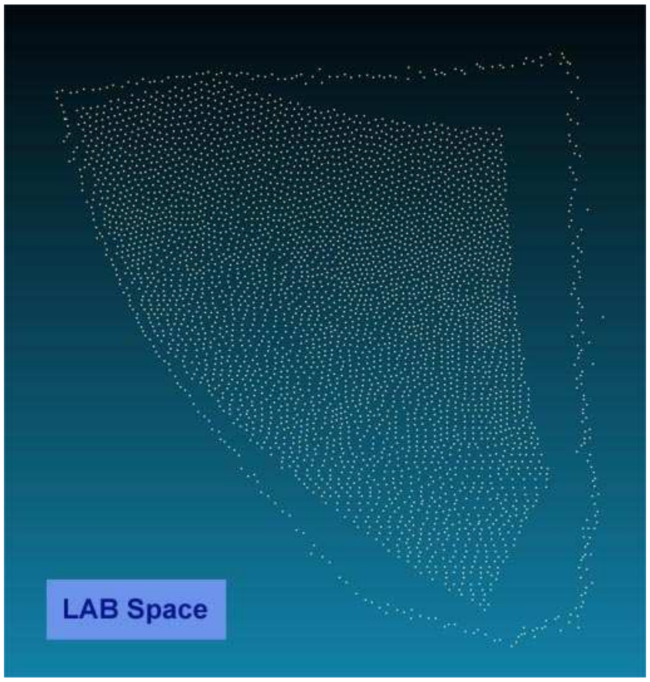

Figure 1: Color distribution in L*a*b* space

Conclusion

In this paper, I have presented an algorithm to perform K-means clustering that is adaptive in nature, and is not dependent on the selection of K seeds to initialize the clusters. The algorithm has been successfully tested in multiple application where it has performed very well, resulting in good data partitioning and impressive speedup of search in the resulting data structures. I have described the application of creating perceptually uniform color distribution table in detail. Another application where I could achieve a seven fold speedup is in the assignment of infra-red material codes to composite materials created from a set of specified materials. I am currently working on the application of this algorithm in creating a hierarchical index to perform efficient storage and retrieval of images using the JPEG 2000 standard and wavelets.

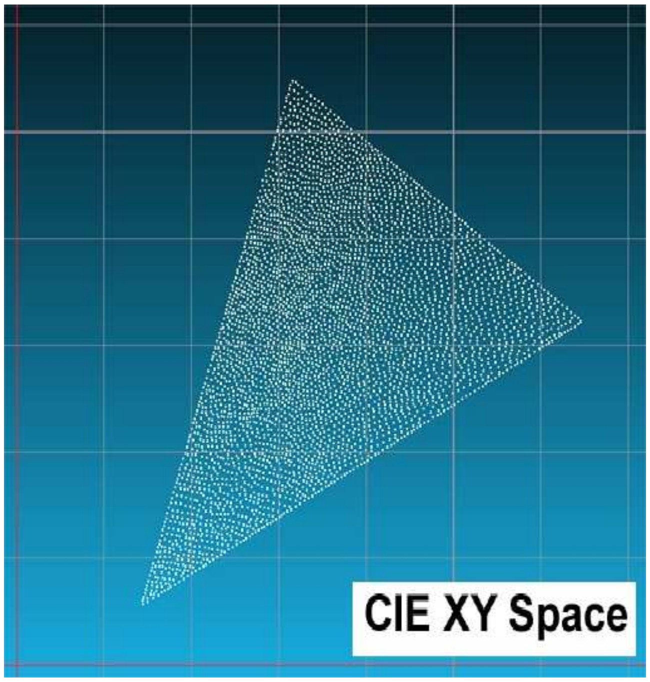

Figure 2: Color distribution in xyY space

Acknowledgement

The work reported in this paper was performed at Visual Simulation Systems Division of FlightSafety International. Andrew Lindberg created the figures used in this paper.

References

Aldus Corporation, Seattle, WA. 1992. *TIFF Revision 6.0*.

Anderberg, M. R. 1973. *Cluster Analysis for Applications*. New York, NY: Academic Press.

Bandyopadhyay, S., and Maulik, U. 2002. Generic clustering for automatic evolution of clusters and applications to image classification. *Pattern Recognition* 35:1197–1208.

Belacel, N.; Hansen, P.; and Mladenovic, N. 2002. Fuzzy J-means: A new heuristic for fuzzy clustering. *Pattern Recognition* 35:2193–2200.

Bhatia, S. K., and Deogun, J. S. 1998. Conceptual clustering in information retrieval. *IEEE Transactions on Systems, Man, and Cybernetics* 28(3):427–436.

Poynton, C. 1995. A guided tour of color space. In *New Foundations for Video Technology: Proceedings of the SMPTE Advanced Television and Electronic Imaging Conference*, 167–180.

Romesburg, H. C. 1984. *Cluster Analysis for Researchers*. Belmont, CA: Lifetime Learning Publications.

Yu, C. T.; Wang, Y. T.; and Chen, C. H. 1985. Adaptive document clustering. In *Proceedings of the Eighth ACM SIGIR*, 197–203.

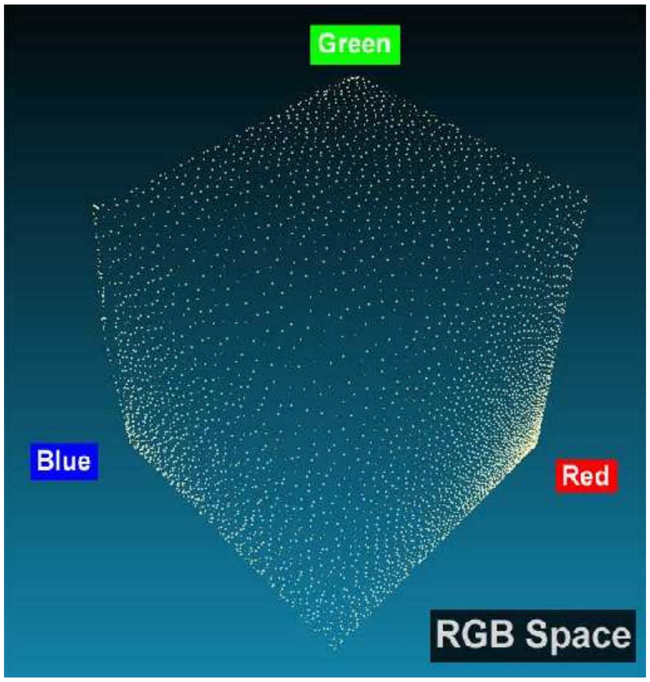

Figure 3: Color distribution in RGB space

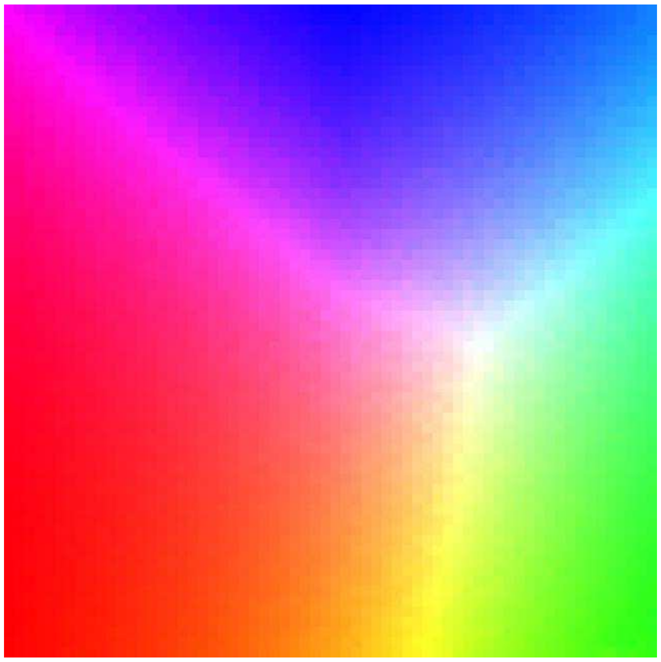

Figure 4: Color palette of ﬁ nal color table