A K-Means++ Clustering Implementation for VTK

Release 0.00

David Doria

September 28, 2010

Rensselaer Polytechnic Institute, Troy NY

Abstract

K-Means clustering is an excellent technique for clustering points when the number of clusters is known. We present a implementation (vtkKMeanClustering) of the algorithm written in a VTK context. We also implement the K-Means++ initialization method which finds the global optimum much more frequently than a naive/random initialization.

The code is currently hosted at http://github.com/daviddoria/KMeansClustering .

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3220]

Contents

1 Introduction

K-Means clustering is an excellent technique for clustering points when the number of clusters is known. It is not the intention of this document to explain any theoretical aspects of the algorithm. Rather, we intuitively explain the algorithm and introduce an implementation to be used to cluster points stored in vtkDataSet objects.

While VTK has built in the ability to perform KMeans clustering (vtkKMeansStatistics), the intention of this paper is to provide an "easy to modify" implementation of the algorithm. The input is in a natural format for clustering points, and the implementation, while potentially less efficient, is very straight forward.

The code is currently hosted at http://github.com/daviddoria/KMeansClustering .

2 Algorithm

The goal is to label a set of points in such a way that points that naturally form a cluster are assigned the same label. Graphically, from a set of unlabeled points we want to obtain the result in Figure 1.

Figure 1: Labeled points after clustering

Figure 2 shows the colored points (squares) and their associated cluster center the same color sphere.

Figure 2: Labeled points with cluster centers

Figure 3 shows a zoomed in version of one of the clusters in Figure 2.

Figure 3: Zoomed in on a single cluster

The algorithm is as follows:

1. Initialize cluster centers (see next section)
2. For each point in the data set, determine which cluster center it is closest to and assign it that label.
3. Find the center of the points currently marked as belonging to the same cluster.
4. Repeat until the labels do not change in successive iterations.

3 Initialization

3.1 Random

One can set the initial cluster centers to either random points in the space (preferably within the bounds of the data) or to random points in the data set. Using either of these methods can lead to very poor results. This typically happens when two clusters start very near each other.

3.2 K-Means++

This initialization procedure attempts to initialize cluster centers far way from each other.

1. Choose the first center to be a random data point.
2. Create a vector of the distance from every data point to the nearest center.
3. Select one of the data points to add as the next cluster center randomly, using the distance vector to bias the random selection to points that are farther away from an existing cluster.

4 Extras

We color the output points so that points from the same cluster are the same (random) color. We also provide a second output which contains the cluster centers.

5 Code Snippet

5.1 Demo Data

For testing data, we create three groups of points and append them together without tracking any membership information. That is, we now have simply one group of points that we know contains three distinct clusters.

```
void GenerateData(vtkPolyData* polydata, int n)
{
  vtkSmartPointer<vtkPoints> points =
    vtkSmartPointer<vtkPoints>::New();

  vtkMath::RandomSeed(time(NULL));
  for(unsigned int i = 0; i < n; i++)
    {
    points->InsertNextPoint(1.0 + vtkMath::Random(-.1, .1), 1.0 + vtkMath::Random(-.1, .1), 1.
    points->InsertNextPoint(-1.0 + vtkMath::Random(-.1, .1), -1.0 + vtkMath::Random(-.1, .1),
    points->InsertNextPoint(1.0 + vtkMath::Random(-.1, .1), -1.0 + vtkMath::Random(-.1, .1), 1
    }

  polydata->SetPoints(points);
}
```

5.2 Clustering

To perform the clustering, the user simply must instantiate a vtkKMeansClustering object and set its input to the data set they wish to cluster.

```
vtkSmartPointer<vtkKMeansClustering> kmeans =
  vtkSmartPointer<vtkKMeansClustering>::New();
kmeans->SetK(3);
kmeans->SetInputConnection(input->GetProducerPort());
kmeans->Update();
```

5.3 Obtaining the result

The colored (labeled) input points are on the first output port, and the cluster centers are on the second output port:

```
vtkPolyData* labeledPoints = kmeans->GetOutput(0);
vtkPolyData* clusterCenters = kmeans->GetOutput(1);
```