A Mean Shift Clustering Implementation for VTK

Release 0.00

David Doria

September 23, 2010

Rensselaer Polytechnic Institute, Troy NY

Abstract

Mean shift clustering is an excellent technique for clustering points when the number of clusters is not known. We present a implementation (vtkMeanShiftClustering) of the simplest version of the algorithm written in a VTK context.

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

Contents

1 Introduction

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

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

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

1. For each point in the data set:
   - Center a kernel function at the point and computed the weighted average of all points in the data set. Call this weighted average *CurrentCenter*.
   - Shift the kernel to the weighted average and repeat these two steps until the distance between the kernel position of successive iterations is less than *ConvergenceThreshold*
   - If a cluster whose center is less than *MinDistanceBetweenClusters* from *CurrentCenter* does not exist, create a new cluster and set its center to *CurrentCenter*.
2. Make a single pass through the points assigning them to the closest cluster.

3 Details

3.1 Kernel

We provide two kernels:

1. Uniform kernel - all points within the *WindowRadius* of the current point are used with equal weights to compute the center. This kernel can be selected by calling $meanShiftFilter->SetKernelToUniform()$;
2. Finite width Gaussian kernel - points farther from the query point contribute less to the center computation. This kernel can be selected by calling $meanShiftFilter->SetKernelToGaussian()$;

3.2 Parameters

There are a few parameters of the algorithm that can be specified:

- `double WindowRadius` - Only points within this radius of the current point will be used in the weighting function to determine the new kernel center
- `double ConvergenceThreshold` - In the inner loop of the algorithm (shifting the kernel), this is the distance that successive kernel positions must be less than for
- `unsigned int MaxIterations` - If we haven't converged before this number of iterations, quit when we reach this number.
- `double MinDistanceBetweenClusters` - A new cluster will not be created if it is less than this distance from an existing cluster.

3.3 Extras

We color the output points so that points from the same cluster are the same (random) color.

4 Code Snippet

4.1 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.

```
vtkSmartPointer<vtkPointSource> pointSource1 =
  vtkSmartPointer<vtkPointSource>::New();
pointSource1->SetCenter(0,0,0);
pointSource1->SetRadius(1);
pointSource1->SetNumberOfPoints(40);
pointSource1->Update();

vtkSmartPointer<vtkPointSource> pointSource2 =
  vtkSmartPointer<vtkPointSource>::New();
pointSource2->SetCenter(5,5,5);
pointSource2->SetRadius(1);
pointSource2->SetNumberOfPoints(40);
pointSource2->Update();
```

```
vtkSmartPointer<vtkPointSource> pointSource3 =
  vtkSmartPointer<vtkPointSource>::New();
pointSource3->SetCenter(-5,5,0);
pointSource3->SetRadius(1);
pointSource3->SetNumberOfPoints(40);
pointSource3->Update();

vtkSmartPointer<vtkAppendPolyData> appendFilter =
  vtkSmartPointer<vtkAppendPolyData>::New();
appendFilter->AddInputConnection(pointSource1->GetOutputPort());
appendFilter->AddInputConnection(pointSource2->GetOutputPort());
appendFilter->AddInputConnection(pointSource3->GetOutputPort());
appendFilter->Update();
```

4.2 Clustering

To perform the clustering, the user simply must instantiate a vtkMeanShiftClustering object, set its input to the data set they wish to cluster, and specify the expected size of the clusters.

```
vtkSmartPointer<vtkMeanShiftClustering> meanShiftFilter =
    vtkSmartPointer<vtkMeanShiftClustering>::New();
meanShiftFilter->SetInputConnection(appendFilter->GetOutputPort());
meanShiftFilter->SetWindowRadius(1.5); //radius should be bigger than expected clusters
meanShiftFilter->SetKernelToGaussian();
meanShiftFilter->SetGaussianVariance(1.0);
meanShiftFilter->Update();
```