Mean Shift Clustering

Giacomo Ravara, Simone Casini

February 27, 2020

Università degli Studi di Firenze

giacomo.ravara@stud.unifi.it
simone.casini3@stud.unifi.it

Overview

Introduction

Mean Shift

- Mean shift is a clustering algorithm used in many applications, such as image segmentation.
- Computationally expensive
- Easy to parallelize

Mean Shift

- Non-parametric mode finding algorithm
- Define a *KDE* surface
- At each step, each point is shifted towards the nearest mode

Mean Shift

At each iteration, given a point y_i, the vector towards the next point y_i' is determined by a weighted average between y_i and each point x_j in the dataset:

$$\mathrm{m_i} = m(\mathrm{y_i}) = \frac{\sum_{j=1}^{n} K(\|\mathrm{y_i} - \mathrm{x_j}\|)\mathrm{x_j}}{\sum_{j=1}^{n} K(\|\mathrm{y_i} - \mathrm{x_j}\|)} - \mathrm{y_i}$$

$$= \frac{\sum_{j=1}^{n} w_{ij}\mathrm{x_j}}{\sum_{j=1}^{n} w_{ij}} - \mathrm{y_i} = \mathrm{m_i} - \mathrm{y_i}$$

The only parameters of the algorithm are the bandwidth and the choice of the *kernel* used.

Gaussian Kernel

- There are many different types of kernel, the most used is the *Gaussian kernel*:

$$k(x) = \frac{1}{(\sqrt{2\pi}\sigma)} e^{-\frac{x}{2\sigma^2}}$$

- The standard deviation σ is the bandwidth parameter, that determines the number of final clusters.

Data Structure

The data structure consists of two list of points:

- list containing the original points X
- list containing the points during the iterations Y

The data structures are the same for both the sequential and the parallel algorithm.

Sequential Version

Algorithm 1 Sequential Mean Shift

for each point X **do**
  $iter \leftarrow 0$
  $Y_0 \leftarrow X$
  while $t < MaxIterations$ **do**
    $Y_{t+1} = \textsc{Shifter}(Y_t, X, bandwidth)$
    $\textsc{Swap}(Y_t, Y_{t+1})$
    $t \leftarrow t + 1$
  Return Y_{t+1}

OpenMP Version

OpenMP

- OpenMP is an API for shared-memory programming.
- The program is parallelized with *compiler directives*
- It follows a *fork-join* thread model.

Mean Shift OpenMP

- Mean Shift is easily parallelizable.
- Each point is shifted indepently from the other points.
- In OpenMP we use the directive *pragma* to parallelize.

Implementation

Algorithm 2 OpenMP Mean Shift

```
#pragma
for each point X do
    iter ← 0
    Y_0 ← X
    while t < MaxIterations do
        Y_{t+1} = SHIFTER(Y_t, X, bandwidth)
        SWAP(Y_t, Y_{t+1})
        t ← t + 1
    Return Y_{t+1}
```

Implementation

The best scheduling strategy for the mean shift algorithm is the *dynamic scheduling*. Using this strategy ensure that each thread will never stops its execution waiting for other threads.

Implementation

Threads	Static	Dynamic
2	124.54s	114.49s
3	87.07s	76.29s
4	63.72s	57.20s
5	52.44s	45.72s
6	44.13s	38.25s

Comparison between static and dynamic with 36864 points

Java Parallel Version

Java Parallel Version

- Uses the same data structures.
- Divide points between threads.
- Each point is shifted until convergence.

Implementation

Algorithm 3 Java Mean Shift

for $t < num\_threads$ **do**

    CreateThreadMeanshift ($Y_t, X, bandwidth$)

    Shifter

Return ShiftedPoints

Implementation

It is possible an alternative implementation, without a static division of points between threads.
The threads share an index to iterate the shared list of points, the index is defined as an AtomicInteger to avoid race condition.

Implementation

Threads	Static	Dynamic
2	135.10s	133.20s
3	93.90s	88.35s
4	77.87s	66.68s
5	64.25s	53.27s
6	56.26s	45.09s

Comparison between static and dynamic with 36864 points

Results

Context

- The performances are measured by evaluating the speedup
- Changing the number of threads and using different input sizes
- Using Microsoft Azure Virtual Machine, with Intel Xeon E5-2690 v3 @ 2.60GHz, 6 cores.

Results

BandWidth	1	
Number of points	16384	
Sequential C++	25.05s	
Sequential Java	25.24s	
Threads	OpenMP	Java
2	12.54s (1.99X)	13.07s (1.93X)
3	8.38s (2.98X)	8.62s (2.93X)
4	6.31s (3.96X)	6.52s (3.86X)
5	5.05s (4.95X)	5.25s (4.80X)
6	4.24s (5.90X)	4.47s (5.64X)

Tabelle 1: Comparison between sequential and OpenMP/Java

Results

BandWidth	10	
Number of points	16384	
Sequential C++	31.74s	
Sequential Java	33.34s	
Threads	OpenMP	Java
2	15.07s (2.10X)	17.62s (1.89X)
3	10.05s (3.15X)	11.70s (2.84X)
4	7.53s (4.21X)	8.83s (3.77X)
5	6.04s (5.24X)	7.11s (4.68X)
6	5.04s (6.29X)	5.99s (5.56X)

Results

BandWidth	25	
Number of points	16384	
Sequential C++	47.01s	
Sequential Java	50.30s	
Threads	OpenMP	Java
2	23.18s (2.03X)	25.77s (1.95X)
3	15.52s (3.03X)	17.18s (2.92X)
4	11.59s (4.06X)	12.89s (3.90X)
5	9.26s (5.08X)	10.30s (4.84X)
6	7.76s (6.06X)	8.74s (5.75X)

Results

BandWidth	25	
Number of points	4096	
Sequential C++	2.86s	
Sequential Java	3.05s	
Threads	OpenMP	Java
2	1.43s (2.00X)	1.64s (1.80X)
3	0.98s (2.92X)	1.03s (2.94X)
4	0.72s (3.94X)	0.78s (3.88X)
5	0.57s (4.97X)	0.64s (4.74X)
6	0.47s (6.03X)	0.57s (5.30X)

Results

BandWidth	25	
Number of points	36864	
Sequential C++	228.61s	
Sequential Java	245.05s	
Threads	OpenMP	Java
2	114.37s (1.99X)	131.75s (1.86X)
3	76.30s (2.99X)	87.96s (2.79X)
4	57.05s (4.00X)	66.02s (3.72X)
5	45.77s (4.99X)	53.46s (4.59X)
6	38.20s (5.98X)	44.88s (5.46X)

Results

Example of Image Segmentation, using MeanShift

END