Distributed mean shift clustering with approximate nearest neighbours

Gaël Beck, Tarn Duong, Hanene Azzag and Mustapha Lebbah
Computer Science Laboratory (LIPN)
University Paris-North – Paris 13
F-93430 Villetaneuse, France
Email: {beck, duong, hanane.azzag, mustapha.lebbah}@lipn.univ-paris13.fr

***Abstract*—We introduce an efficient distributed implementation of nearest neighbour mean shift clustering (NNMS). The computationally intensive nature of NNMS has so far restricted its application to complex data sets where a flexible clustering with non-ellipsoidal clusters would be beneficial. A parallel implementation of the standard serial NNMS algorithm on its own brings insufficient performance gains so we introduce two further algorithmic improvements: a normal scale (NS) choice of the optimal number of nearest neighbours, and locality sensitive hashing (LSH) to approximate nearest neighbour searches. Combining these improvements into a single distributed algorithm DNNMS offers the potential for an efficient method for Big Data Clustering.**

Keywords: Big Data Clustering, gradient ascent estimation, k-means, locality sensitive hashing LSH, normal scale, unsupervised learning

I. Introduction

The goal of clustering (or unsupervised learning) is to assign cluster membership to unlabelled candidate points where the number and location of these clusters are unknown. We focus on the class of modal clustering methods where clusters are defined in terms of the local modes of the probability density function which generates the data. The most well-known modal clustering method is the k-means clustering. As the k-means clustering is based on the normal mixture densities, it is restricted to finding ellipsoidal clusters which can be inappropriate for complex data sets. Mean shift clustering is a generalisation of the k-means clustering where it computes arbitrarily shaped clusters as defined as the basins of attraction to the local modes created by the density gradient ascent paths [1].

To compute these gradient ascent paths, nearest neighbour methods are well-suited as they adapt to the local data structure. Current nearest neighbour mean shift clustering NNMS algorithms contain computational bottlenecks posed by a multiple pass grid based search for the optimal number of nearest neighbours, and exact nearest neighbour searches. We propose a new algorithm DNNMS which resolves these computational inefficiencies: (a) an efficient normal scale (NS) or 'rule of thumb' choice of the nearest neighbours which avoids a grid search, (b) locality sensitive hashing (LSH) for approximate nearest neighbour searches to replace the exact nearest neighbour calculations, and (c) a distributed platform implementation.

II. Methods

A. Mean shift clustering

The mean shift method, introduced by [1], for a d-dimensional point $\boldsymbol{x}$, generates a sequence of points $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \dots\}$ which follows the gradient density ascent paths using the recurrence relation

$$\boldsymbol{x}_{j+1} = \frac{1}{k} \sum_{\boldsymbol{X}_i \in k\text{-nn}(\boldsymbol{x}_j)} \boldsymbol{X}_i \tag{1}$$

where $\boldsymbol{X}_1, \dots, \boldsymbol{X}_n$ is a random sample drawn from a common density f, the k nearest neighbours of $\boldsymbol{x}$ are $k\text{-nn}(\boldsymbol{x}) = \{\boldsymbol{X}_i : \|\boldsymbol{x} - \boldsymbol{X}_i\| \le \delta_{(k)}(\boldsymbol{x})\}$ as $\delta_{(k)}(\boldsymbol{x})$ is the k-th nearest neighbour distance, and $\boldsymbol{x}_0 = \boldsymbol{x}$. Eq. (1) gives the mean shift method its name since the current iterate $\boldsymbol{x}_j$ is shifted to the sample mean of its k nearest neighbours for the next iterate $\boldsymbol{x}_{j+1}$. The convergence of the sequence $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \dots\}$ to a local mode for the kernel version of Eq. (1) for a wide class of kernels was established by [2] for fixed bandwidths. This convergence remains valid when the fixed bandwidth is replaced with a nearest neighbour distance which decreases as the iteration number increases.

The gradient ascent paths towards the local modes produced by Eq. (1) form the basis of Algorithm 1, our nearest neighbour mean shift clustering method NNMS. The inputs to the NNMS are the data sample $\boldsymbol{X}_1, \dots, \boldsymbol{X}_n$ and the candidate points $\boldsymbol{x}_1, \dots, \boldsymbol{x}_m$ which we wish to cluster (these can be the same as $\boldsymbol{X}_1, \dots, \boldsymbol{X}_n$, but this is not required); and the tuning parameters: the number of nearest neighbours k, the tolerance under which subsequent iterations in the mean shift update are considered to be convergent ε_1, the maximum number of iterations $j_{\max}$, the tolerance under which two cluster centres are considered to form a single cluster ε_2, and the minimum cluster size $s_{\min}$. The output are the cluster labels of the candidate points $\{\mathrm{c}(\boldsymbol{x}_1), \dots, \mathrm{c}(\boldsymbol{x}_m)\}$. There are three main sub-routines to Algorithm 1. Lines 1–6 correspond to the gradient ascent paths in Eq. (1) which are iterated until subsequent iterates are less than ε_1 apart or the maximum number of iterations $j_{\max}$ are reached. The output from these lines are the final iterates $\boldsymbol{x}_1^*, \dots, \boldsymbol{x}_m^*$. Lines 7–8 concern merging the final iterates within ε_2 distance of each other into a single cluster, creating an initial clustering of $\boldsymbol{x}_1^*, \dots, \boldsymbol{x}_m^*$. In Lines 9–13, if the smallest cluster is less than the minimum cluster size $s_{\min}$, then it is iteratively merged into next nearest cluster, to produce cluster labels $\mathrm{c}(\boldsymbol{x}_1^*), \dots, \mathrm{c}(\boldsymbol{x}_m^*)$. Line 14 assigns these cluster labels to the original data $\boldsymbol{x}_1, \dots, \boldsymbol{x}_m$.

Algorithm 1 NNMS – Nearest neighbour mean shift clustering, with exact k-nn

```
    Input: ${\boldsymbol{X}_1, \dots, \boldsymbol{X}_n\}, \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_m\}, k, \varepsilon_1, \varepsilon_2, j_{\max}, s_{\min}$
    Output: $\{\mathrm{c}(\boldsymbol{x}_1), \dots, \mathrm{c}(\boldsymbol{x}_m)\}$
    /* Compute gradient ascent paths */
 1: for $\ell := 1$ to $m$ do
 2:     $j := 0$; $\boldsymbol{x}_{\ell,0} := \boldsymbol{x}_\ell$;
 3:     $\boldsymbol{x}_{\ell,1} :=$ mean of $k$-nn of $\boldsymbol{x}_{\ell,0}$;
 4:     while $\|\boldsymbol{x}_{\ell,j+1}, \boldsymbol{x}_{\ell,j}\| > \varepsilon_1$ or $j < j_{\max}$ do
 5:         $j := j+1$; $\boldsymbol{x}_{\ell,j+1} :=$ mean of $k$-nn of $\boldsymbol{x}_{\ell,j}$;
 6:     $\boldsymbol{x}_\ell^* := \boldsymbol{x}_{\ell,j}$;
    /* Create clusters by merging near final iterates */
 7: for $\ell_1, \ell_2 := 1$ to $m$ do
 8:     if $\|\boldsymbol{x}_{\ell_1}^* - \boldsymbol{x}_{\ell_2}^*\| \le \varepsilon_2$ then $\mathrm{c}(\boldsymbol{x}_{\ell_1}^*) := \mathrm{c}(\boldsymbol{x}_{\ell_2}^*)$;
    /* Merge small clusters */
 9: $C^* :=$ cluster with minimum cardinality;
10: while $\mathrm{card}(C^*) < s_{\min}$ do
11:     $C' :=$ nearest other cluster to $C^*$;
12:     for $\boldsymbol{x}_\ell^* \in C^*$ do $\mathrm{c}(\boldsymbol{x}_\ell^*) := \mathrm{c}(C')$;
13:     $C^* :=$ cluster with minimum cardinality;
14: for $\ell := 1$ to $m$ do $\mathrm{c}(\boldsymbol{x}_\ell) := \mathrm{c}(\boldsymbol{x}_\ell^*)$;
```

B. Normal scale choice of the number of nearest neighbours

The critical tuning parameter for mean shift clustering is the choice of the number of nearest neighbours k. The pioneering work of [3], [4] established the oracle local and global squared error optimal selectors for nearest neighbour density estimators, though these authors did not consider data-based selectors. A data-based grid search to minimise clustering quality indices, such as the Silhouette index, was considered by [5]. Our proposed normal scale or 'rule of thumb' selector is

$$k_{\mathrm{NS}} = v_0[4/(d+4)]^{d/(d+6)} n^{6/(d+6)} \tag{2}$$

where $v_0 = \pi^{d/2}\Gamma((d+2)/d))$ is the hyper-volume of the d-dimensional unit ball. The derivation of Eq. (2) is given in [6], which follows the assertion that tuning parameter selection based on the density gradient rather than on the density itself is more suitable for mean shift clustering [7]. The complexity of k_{NS} is $O(1)$ which is in contrast with $O(n)$ grid searches for selecting an optimal k since the number of possible search values is usually set to be proportional to n.

C. Approximate nearest neighbours with locality sensitive hashing

The most computationally intensive step in the NNMS is the computation of the k nearest neighbours, rather than the selection of the number of nearest neighbours. For each of the candidate points, this requires computing and sorting the distances $\|\boldsymbol{X}_i - \boldsymbol{x}_j\|, i = 1, \dots, n, j = 1, \dots, m$, which is $O(mn \log n)$. For the usual case where m is the same order as n, this implies that mean shift clustering with exact nearest neighbours is infeasible for large sample sizes.

One promising complexity reduction approach relies on computing approximate nearest neighbours rather than exact neighbours. Amongst these, locality sensitive hashing (LSH), introduced by [8], [9], is a probabilistic method based on a random scalar projection of multivariate data point $\boldsymbol{x}$

$$L(\boldsymbol{x}; w) = (\boldsymbol{Z}^T \boldsymbol{x} + U)/w$$

where $\boldsymbol{Z} \sim N(0, \mathbf{I}_d)$ is a standard d-variate normal random variable and $U \sim \mathrm{Unif}(0, w)$ is a uniform random variable on $[0, w)$, $w > 0$. A hash table whose buckets are based on the integer values $\lfloor L(\boldsymbol{X}_i; w) \rfloor, i = 1, \dots, n$, is constructed. Due to the statistical properties of the normal distribution, close points in the original multivariate space tend to fall in the same bucket and distant points tend to fall into different buckets in the hash table, as verified by [10]. Larger values of w imply fewer buckets with more accurate preservation of characteristics of $\boldsymbol{X}_i$, whereas smaller values of w imply more buckets with less accuracy. We prefer to parametrise the LSH by the number of buckets M in the hash table. So we set $w = 1$ without loss of generality $L_i \equiv L(\boldsymbol{X}_i; 1)$. These scalar projections are sorted into their order statistics $L_{(1)} < \cdots < L_{(n)}$, and their range is divided into M partition intervals of width $w = (L_{(n)} - L_{(1)})/M$ where $I_j = [L_{(1)} + w(j-1), L_{(1)} + wj]$, $j = 1, \dots, M$. The hash value of $\boldsymbol{x}$ is the index of the interval in which $L(\boldsymbol{x}; 1)$ falls

$$H(\boldsymbol{x}) = j\mathbf{1}\{L(\boldsymbol{x}; 1) \in I_j\} \tag{3}$$

where $\mathbf{1}\{\cdot\}$ is the indicator function. To search for approximate nearest neighbours, the reservoir of potential nearest neighbours is set to the bucket which contains the hash value. This reservoir is enlarged if necessary by concatenating the adjacent buckets. The approximate k nearest neighbours to $\boldsymbol{x}$ are the k nearest neighbours drawn only from the reduced reservoir $R(\boldsymbol{x})$: $k\text{-}\widetilde{\mathrm{nn}}(\boldsymbol{x}) = \{\boldsymbol{X}_i \in R(\boldsymbol{x}) : \|\boldsymbol{x} - \boldsymbol{X}_i\| \le \delta_{(k)}(\boldsymbol{x})\}$ where $\delta_{(k)}(\boldsymbol{x})$ is the nearest neighbour distance to $\boldsymbol{x}$. The approximation error in the nearest neighbours to $\boldsymbol{x}$ induced by searching in $R(\boldsymbol{x})$ rather than the full data set is probabilistically controlled, see [10].

Algorithm 2 is the NNLSH, an approximate nearest neighbour search with LSH with the hash function provided by Eq. (3). The inputs are the data sample $\boldsymbol{X}_1, \dots, \boldsymbol{X}_n$, the candidate points $\boldsymbol{x}_1, \dots, \boldsymbol{x}_m$, and the tuning parameters: the number of nearest neighbours k and the number of buckets in the hash table M. In Line 1, the hash table is created by applying the LSH to the data values $\boldsymbol{X}_1, \dots, \boldsymbol{X}_n$. In Lines 2–6, for each candidate point $\boldsymbol{x}_\ell$, the approximate k nearest neighbours $k\text{-}\widetilde{\mathrm{nn}}(\boldsymbol{x}_\ell)$ are computed from within the reservoir $R(\boldsymbol{x}_\ell)$.

The proposal where the NNLSH is substituted into the NNMS was made by [11], which reduces the complexity to $O((mn/M)\log(n/M))$. The number of buckets M is a crucial tuning parameter. Despite intense interest in the LSH [12], there do not yet exist optimal methods for selecting the number

Algorithm 2 NNLSH – Approximate k-nn with LSH

```
    Input: $\{\boldsymbol{X}_1, \dots, \boldsymbol{X}_n\}, \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_m\}, k, M$
    Output: $\{k\text{-}\widetilde{\mathrm{nn}}(\boldsymbol{x}_1), \dots, k\text{-}\widetilde{\mathrm{nn}}(\boldsymbol{x}_m)\}$
    /* Create hash table with $M$ buckets */
1:  for $i := 1$ to $n$ do $H_i := H(\boldsymbol{X}_i)$;
    /* Search for approx nn in adjacent buckets */
2:  for $\ell := 1$ to $m$ do
3:      $R(\boldsymbol{x}_\ell) := \{\boldsymbol{X}_i : H_i = H(\boldsymbol{x}_\ell), i \in \{1, \dots, n\}\}$
4:      while $\mathrm{card}(R(\boldsymbol{x}_\ell)) < k$ do
5:          $R(\boldsymbol{x}_\ell) := R(\boldsymbol{x}_\ell) \cup$ adjacent bucket;
6:      $k\text{-}\widetilde{\mathrm{nn}}(\boldsymbol{x}_\ell) := k$-nn from $R(\boldsymbol{x}_\ell)$ to $\boldsymbol{x}_\ell$;
```

of buckets, so we will examine its heuristic performance in the next section.

Implementing the approximate NNMS in a distributed system with a master processor and N slave processors further reduces the complexity to $O(mn/(MN)\log(n/(MN)))$. This is our proposed DNNMS in Algorithm 3. Its inputs and outputs are the same as for Algorithm 1. For the j-th iteration, the gradient ascent paths are collated in an $m \times d$ matrix $\boldsymbol{x}_j = [\boldsymbol{x}_{1,j}; \dots; \boldsymbol{x}_{m,j}]$. In Lines 1–6, these are iterated until a global convergence $\|\boldsymbol{x}_{j+1} - \boldsymbol{x}_j\| \leq \epsilon_2 \equiv \|\boldsymbol{x}_{1,j+1} - \boldsymbol{x}_{1,j}\|, \dots, \|\boldsymbol{x}_{m,j+1} - \boldsymbol{x}_{m,j}\| \leq \epsilon_2$ or the maximum number of iterations is exceeded $j > j_{\max}$. Some redundant calculations are performed here whenever some of the $\boldsymbol{x}_{\ell,j}$ have already converged, but this form of computation is required for effective MapReduce parallelisation [13]. The MapReduce paradigm is most efficient if the serial algorithms are redesigned from iterating over each candidate point to treating all candidates simultaneously. Lines 7–14 for cluster merging from Algorithm 1 are reused without major modification as MapReduce is not required here.

Algorithm 3 DNNMS – Distributed nearest neighbour mean shift clustering, with approximate k-nn using LSH

```
    Input: $\{\boldsymbol{X}_1, \dots, \boldsymbol{X}_n\}, \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_m\}, k, \varepsilon_1, j_{\max}, \varepsilon_2, s_{\min}, M$
    Output: $\{\mathrm{c}(\boldsymbol{x}_1), \dots, \mathrm{c}(\boldsymbol{x}_m)\}$
    /* Compute gradient ascent paths */
1:  $j := 0$; $\boldsymbol{x}_0 := [\boldsymbol{x}_{1,0}; \dots; \boldsymbol{x}_{m,0}]$;
2:  $\boldsymbol{x}_1 :=$ mean of $k\text{-}\widetilde{\mathrm{nn}}$ of $\{\boldsymbol{X}_1, \dots, \boldsymbol{X}_n\}$ to $\boldsymbol{x}_0$
3:  while $\|\boldsymbol{x}_{j+1} - \boldsymbol{x}_j\| > \varepsilon_1$ or $j < j_{\max}$ do
4:      $j := j + 1$;
5:      $\boldsymbol{x}_{j+1} :=$ mean of $k\text{-}\widetilde{\mathrm{nn}}$ of $\boldsymbol{x}_j$; /* use Algorithm 2 */
6:  $\boldsymbol{x}^* := [\boldsymbol{x}_{1,j}; \dots; \boldsymbol{x}_{m,j}]$;
7:  Same as Lines 7–14 in Algorithm 1;
```

III. Results

A. Serial clustering for simulated multivariate data

We evaluate the clustering performance of our proposed nearest neighbour mean shift clustering NNMS (Algorithm 1) with the normal scale choice k_{NS}. An alternative nearest neighbour median shift clustering NNMS2, where the sample mean of the k nearest neighbours is replaced by a component-wise sample median, and the choice of k is based on a grid search to minimise the Silhouette index [5]. The kernel mean shift clustering, with a normal kernel and the plug-in selector from [7] is labelled as KMS. For these mean shift methods, we set the other tuning parameters as follows: the mean shift iteration tolerance ε_1 is 0.005 times the maximum marginal data range, the maximum number of mean shift iterations is $j_{\max} = 100$, the cluster merging tolerance is $\varepsilon_2 = 10\varepsilon_1$, and minimum cluster size is $s_{\min} = 0.05n = 50$. The 'gold standard' parametric clustering method is the k-means normal mixture based method with a BIC procedure for selecting an optimal number of clusters [14], labelled KM. We restrict ourselves to this small number of clustering methods as these are conveniently available as public R packages: NNMS2 in clues [5], KMS in ks [15] and KM in mclust [16].

We examine the d-dimensional 4-crescent density as defined in [6]. Fig. 1(a) displays the scatter plot of $n = 1000$ from the 4-crescent for $d = 5$. There are four crescent saddle-shaped clusters, with the two smaller clusters in the lower right posing particular difficulty to separate cleanly. The nearest neighbour methods NNMS (Fig. 1(b)) and NNMS2 (Fig. 1(c)) are able to correctly locate the 4 crescent clusters, whereas the kernel mean shift KMS (Fig. 1(d)) and k-means KM (Fig. 1(e)) both produce 10 clusters by splitting the 4 main clusters into smaller clusters.

The box plots of the Adjusted Rand Index (ARI) from 100 trials of $n = 1000$ random samples in Fig. 1(f) indicate that all three mean shift methods give higher ARI values than k-means (KM). Values of ARI close to one indicate closely matched cluster labellings, and values close to and less than zero indicate mismatched cluster labellings [17]. Whilst KMS and NNMS2 perform well, NNMS performs overall the best. We note that NNMS2 requires a multiple pass grid search to choose the number of nearest neighbours, whereas NNMS can achieve similar performance with a single calculation of k_{NS}.

Table I shows the numerical performance measures of the clustering methods. The NNMS has the highest mean of and the smallest spread of the ARI values. Since execution times are highly dependent on the system utilised, we normalise these execution times by the mean NNMS execution time. The ratios for the other clustering methods are greater than 1, indicating that the grid-based search of the number of nearest neighbours for the NNMS2 and of the number of mixture components for the KM, and the dense nature of the kernel mean shift KMS leads to computational bottlenecks in comparison to the NNMS.

TABLE I
Performance measures of the clustering methods for 100 trials of $n = 1000$ samples

	Clustering method			
	NNMS	NNMS2	KMS	KM
ARI	0.99±0.01	0.87±0.14	0.68±0.09	0.61±0.07
Execution time	1.00±0.24	1.98±0.71	3.48±0.36	12.41±0.23

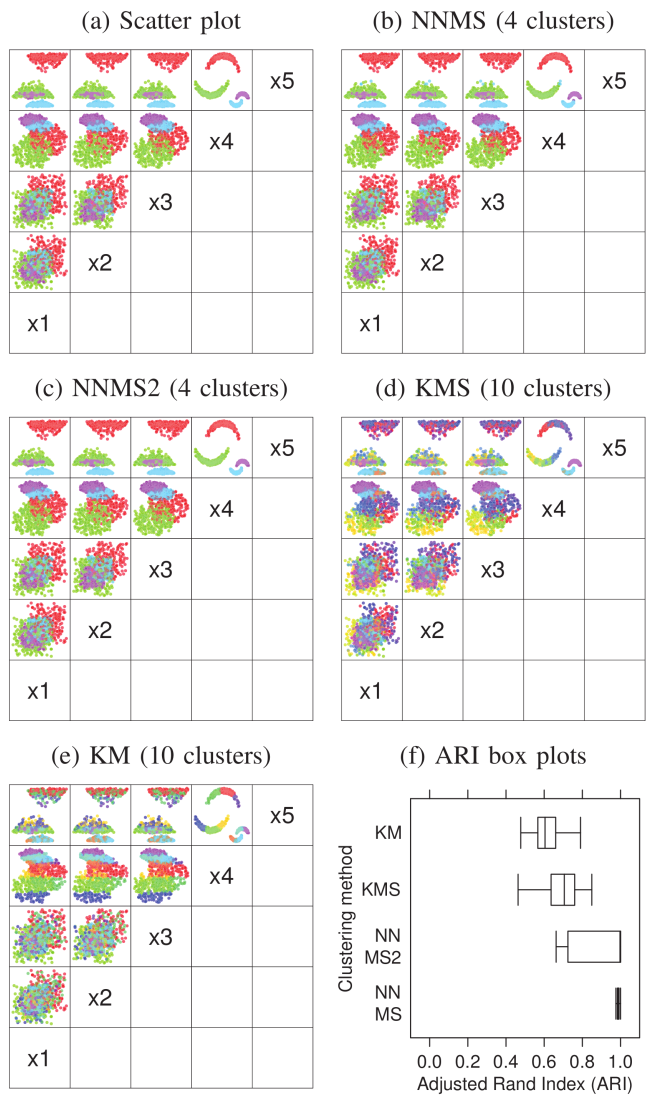


Fig. 1. 4-crescent density. (a) Scatter plot for a single $n = 1000$ random sample. (b) Nearest neighbour mean shift with normal scale choice NNMS. (c) Nearest neighbour median shift with Silhouette index choice NNMS2. (d) Kernel mean shift KMS. (e) k-means KM. (f) Box plots for ARI for 100 trials of $n = 1000$ samples for the clusterings.

B. *Distributed clustering for simulated multivariate data*

We implement DNNMS on a Spark Scala ecosystem [18] to investigate the effect on the execution time according to (a) the number of slave processors N and (b) the number of buckets M in the LSH to process a 160000×5 matrix. This code available on the GitHub platform https://github.com/Spark-clustering-notebook/Mean-Shift-LSH. In Fig. 2(a), are the execution times for $N = 2, 3, \dots, 8$ slave processors for $M = 200$. There are three curves: solid black for the total time, dashed red for the time for computing the nearest neighbours gradient ascent paths (Lines 1–6 in Algorithm 3) and dotted green for the cluster merging (Line 7 in Algorithm 3). The nearest neighbour calculations are more computationally intensive, though these decrease markedly as the number of slaves N increases. The cluster merging operations decrease more slowly as N increases, as they depend on the number of clusters rather than n. In Fig. 2(b) is the effect of the number of buckets $M = 15, 35, 75, 125, 250, 500, 1000$ in the LSH for $N = 2$. A decreasing (total) execution time with increasing number of buckets is observed. From this figure, we observe that the rate in the decrease in execution time is faster when increasing the number of buckets in the LSH than increasing the number of slave processes. It is noteworthy that after about $M = 200$ buckets, there is a diminishing decrease. At least from a computational point of view, $M = 200$ appears to be a good heuristic choice for these data. We focus in the next subsection on the effect of the number of buckets on clustering quality in the context of image segmentation.

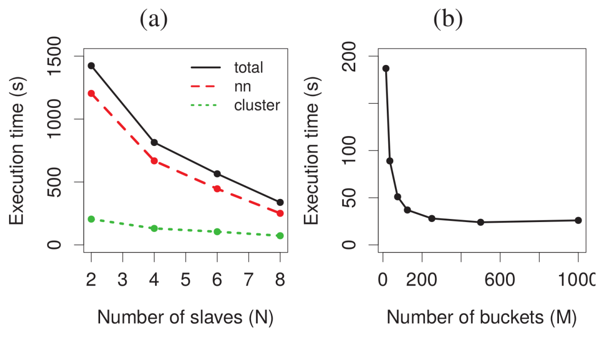


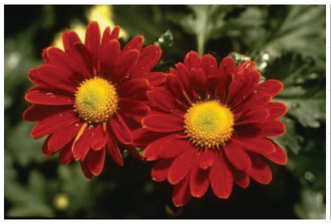
Fig. 2. Execution times for distributed nearest neighbour mean shift clustering DNNMS. (a) Number of slave processors N. (b) Number of buckets M in locality sensitive hashing.

C. *Distributed image segmentation*

A resurgence in interest in the mean shift algorithm is due to its application to image segmentation [19] where an image is transformed into a colour space in which clusters correspond to segmented regions in the original image. The 3-dimensional $L^*u^*v^*$ colour space [20, Eqs. 3.5-8a–f] is a common choice. Since an image is a 2-dimensional array of pixels, let (x, y) be the row and column index of a pixel. The spatial and colour (range) information of a pixel can be concatenated into a 5-dimensional vector (x, y, L^*, u^*, v^*) in the joint spatial-range domain. To illustrate this, we take image #171 from the colour training set from the Berkeley Segmentation Dataset and Benchmark http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds. In Fig. 3(a–b) are the original RGB 481×321 pixels JPEG image and the scatter plot of the $n = 154401$ joint spatial-range (x, y, L^*, u^*, v^*) coordinates.

An image segmentation algorithm based on the kernel mean shift was introduced in [2] which we adapt for use with NNMS. The tuning parameters for the NNMS and DNNMS are $k_{\text{NS}} = 2463, \varepsilon_1 = 0.005$ times the maximum marginal data range, $j_{\max} = 100, \varepsilon_2 = 10\varepsilon_1, s_{\min} = 1544$. We compute the DNNMS-$M$ with $M = 200, 500, 1000$ buckets. Execution times are 20, 13 and 10 minutes respectively, a vast improvement in comparison to an overnight run for the NNMS on a desktop PC. For the image segmentation quality,

(a) RGB



(b) Spatial-range

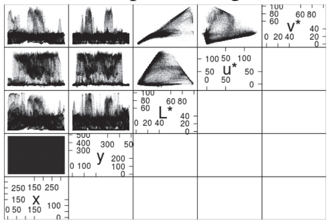
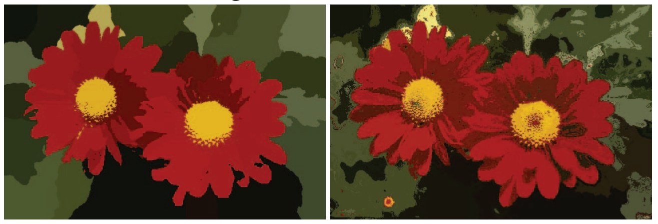


Fig. 3. Colour image representations. (a) RGB image 481×321 pixels. (b) Scatter plot of $n = 154401$ transformed (x, y, L^*, u^*, v^*) spatial-range values.

in Fig. 4(a) is the NNMS with exact nearest neighbours. This segmented image offers a considerable reduction in image complexity, whilst is able to detect the centres of the flowers including some of the fine granular structure, the contours of the outer edge of the petals and some internal shading, and differences in the background foliage. Due to the random nature of the LSH projections to approximate the nearest neighbours in the DNNMS, these clusters are less compact and more diffuse than those in the NNMS where the LSH projections are not used. For the DNNMS-200 in Fig. 4(b) with approximate nearest neighbours with $M = 200$ buckets, some finer details are more visible than without LSH. For the DNNMS-500 and the DNNMS-1000 in Fig. 4(c–d), the yellow flower centres are less clearly delimited from the petals, and there is considerable bleeding of the petals into the foliage. We observe that it is a coincidence that $M = 200$ buckets is a suitable choice as it is also in Fig. 2 and that further investigation is required for its optimal choice in general.

(a) NNMS (1 night) (b) DNNMS-200 (20 min)



(c) DNNMS-500 (13 min) (d) DNNMS-1000 (10 min)

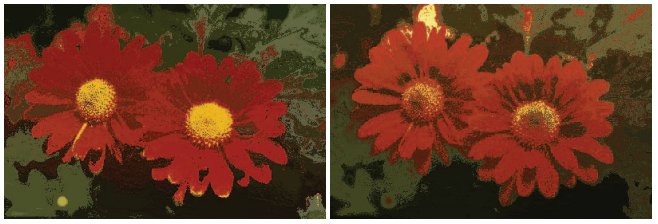
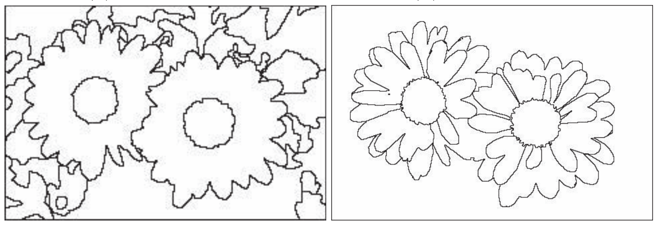


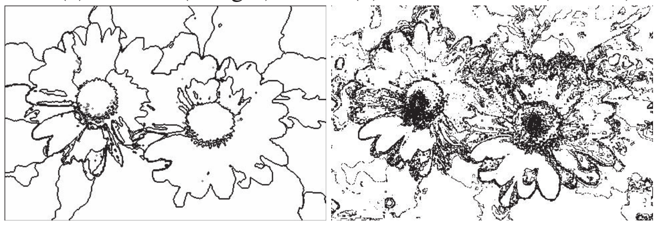
Fig. 4. Colour image segmentation using nearest neighbour mean shift clustering. (a) NNMS with serial exact nearest neighbours. (b–d) DNNMS-M with distributed approximate nearest neighbours using localilty sensitive hashing with $M = 200, 500$ and 1000 buckets. The execution times are 1 night, 20, 13 and 10 minutes respectively.

The Berkeley Segmentation Dataset and Benchmark provides human expert segmentations of their images for comparisons. In Fig. 5(a–b) are two such edge detections made by Users #1107 and #1123. User #1107 focuses on segmenting the background foliage and the overall flower shape, whilst ignoring the detail of the flower petals, whereas User #1123 concentrates on segmenting the individual petals in the foreground. We focus on the NNMS and the DNNMS-200 (Fig. 5(c–d)), as the DNNMS-500 and the DNNMS-1000 (Fig. 5(e–f)) give insufficient quality of edge detection. Both the NNMS and the DNNMS-200 are able to segment, in a single process with a single set of tuning parameters, simultaneously the background foliage and foreground petal shapes, providing an automatic segmentation which combines the results of two human experts focusing on different areas of the image.

(a) User #1107 (b) User #1123



(c) NNMS (1 night) (d) DNNMS-200 (20 min)



(e) DNNMS-500 (13 min) (f) DNNMS-1000 (10 min)

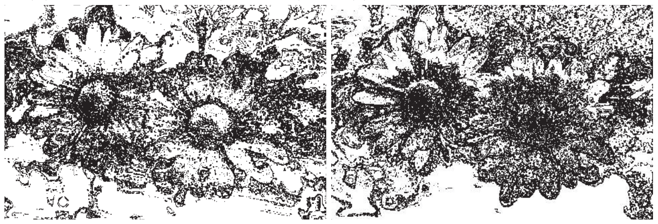


Fig. 5. Edge detection of segmented images. (a–b) Two human experts: users #1107 and #1123. (c) NNMS with serial exact nearest neighbours. (d–f) DNNMS-M with distributed approximate nearest neighbours using localilty sensitive hashing with $M = 200, 500$ and 1000 buckets.

IV. Conclusion

We have introduced several improvements to the standard nearest neighbour mean shift clustering algorithm. The first is a single pass normal scale selector for the optimal number of nearest neighbours. The second is an approximate nearest neighbour search via locality sensitive hashing. The third is an implementation on a distributed computing ecosystem. We demonstrated that these improvements greatly decrease the execution time whilst maintain the quality of clustering of the exact mean shift clustering. These improvements render it possible to apply mean shift for Big Data Clustering in the near future. To achieve this, further refinement of the crucial tuning parameters, i.e. the number of nearest neighbours and the number of buckets in the locality sensitive hashing for approximate nearest neighbours, is required.

Acknowledgements

This research has been partially supported by the Square Predict project within a 'Projet investissement d'avenir (PIA)' 2013–2016 grant provided by the French government.

References

[1] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Trans. Inform. Theory*, 21:32–40, 1975.

[2] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Trans. Pattern Anal.*, 24:603–619, 2002.

[3] D. O. Loftsgaarden and C. P. Quesenberry. A nonparametric estimate of a multivariate density function. *Ann. Math. Statist.*, 36:1049–1051, 1965.

[4] K. Fukunaga and L. Hostetler. Optimization of k-nearest-neighbor density estimates. *IEEE Trans. Inform. Theory*, 19:320–326, 1973.

[5] X. Wang, W. Qiu, and R. H. Zamar. CLUES: A non-parametric clustering method based on local shrinking. *Comput. Statist. Data Anal.*, 52:286–298, 2007.

[6] T. Duong, G. Beck, H. Azzag, and M. Lebbah. Nearest neighbour estimators of density derivatives, with application to mean shift clustering. 2016. Submitted.

[7] J. E. Chacón and T. Duong. Data-driven density estimation, with applications to nonparametric clustering and bump hunting. *Electron. J. Statist.*, 7:499–532, 2013.

[8] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 604–613. ACM, 1998.

[9] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th Annual Symposium on Computational Geometry*, pages 253–262. ACM, 2004.

[10] M. Slaney and M. Casey. Locality-sensitive hashing for finding nearest neighbors. *IEEE Signal Proc. Mag.*, 25:128–131, 2008.

[11] Y. Cui, K. Cao, G. Zheng, and F. Zhang. An adaptive mean shift algorithm based on LSH. *Procedia Engineering*, 23:265–269, 2011.

[12] S. Har-Peled, P. Indyk, and R. Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory Comput.*, 8:321–350, 2012.

[13] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Commun. ACM*, 51:107–113, 2008.

[14] C. Fraley and A. E. Raftery. Model-based clustering, discriminant analysis, and density estimation. *J. Amer. Statist. Assoc.*, 97:611–631, 2002.

[15] T. Duong. ks: Kernel density estimation and kernel discriminant analysis for multivariate data in R. *J. Statist. Softw.*, 21(7):1–16, 2007.

[16] C. Fraley, A. E. Raftery, T. B. Murphy, and L. Scrucca. mclust version 4 for R: Normal mixture modeling for model-based clustering, classification, and density estimation. Technical Report No. 597, Department of Statistics, University of Washington, 2012.

[17] L. Hubert and P. Arabie. Comparing partitions. *J. Classif.*, 2:193–218, 1985.

[18] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 15–28, 2012.

[19] D. Comaniciu. An algorithm for data-driven bandwidth selection. *IEEE Trans. Pattern Anal.*, 25:281–288, 2003.

[20] W. K. Pratt. *Digital Image Processing: PIKS Inside*. John Wiley and Sons, New York, 3rd edition, 2001.