Journal of Parallel and Distributed Computing 134 (2019) 128–139

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

A distributed approximate nearest neighbors algorithm for efficient large scale mean shift clustering

Gaël Beck *, Tarn Duong, Mustapha Lebbah, Hanane Azzag, Christophe Cérin

Computer Science Laboratory of Paris North (LIPN, CNRS UMR 7030), University of Paris 13, F-93430 Villetaneuse, France

ARTICLE INFO

Article history:
Received 29 November 2018
Received in revised form 13 May 2019
Accepted 31 July 2019
Available online 28 August 2019

Keywords:
Clustering
Gradient ascent
Nearest neighbors
Spark
MapReduce

ABSTRACT

Mean Shift clustering, as a generalization of the well-known k-means clustering, computes arbitrarily shaped clusters as defined as the basins of attraction to the local modes created by the density gradient ascent paths. Despite its potential for improved clustering accuracy, the Mean Shift approach is a computationally expensive method for unsupervised learning. We introduce two contributions aiming to provide approximate Mean Shift clustering, based on scalable procedures to compute the density gradient ascent and cluster labeling, with a linear time complexity, as opposed to the quadratic time complexity for the exact clustering. Both propositions are based on Locality Sensitive Hashing (LSH) to approximate nearest neighbors. When implemented on a serial system, these approximate methods can be used for moderate sized datasets. To facilitate the analysis of Big Data, a distributed implementation, written for the Spark/Scala ecosystem is proposed. An added benefit is that our proposed approximations of the density gradient ascent, when used as a pre-processing step in other clustering methods, can also improve the clustering accuracy of the latter. We present experimental results illustrating the effect of tuning parameters on cluster labeling accuracy and execution times, as well as the potential to solve concrete problems in Big Clustering.

1. Introduction

The goal of clustering or unsupervised learning is to assign cluster membership to unlabeled candidate points where the number and location of these clusters are unknown. Due to the exponential increase of the size of datasets collected to tackle increasingly complex scientific challenges, scalable versions of unsupervised learning and other machine learning methods have become critical. The complexity of the machine learning implies that improvements in distributed hardware on their own are incapable to providing scalable machine learning. In order to optimally exploit the hardware, their mathematical foundations must also be re-formulated with scalability as a priority.

Current Mean Shift clustering [6,13,14] algorithms contain computational bottlenecks with both kernel and nearest neighbor approaches: due to the exact evaluation of the kernel function, and the exact nearest neighbor searches respectively. We propose a new algorithm which resolves the computational inefficiencies of the nearest neighbor Mean Shift by using Locality Sensitive Hashing (LSH) [9,17,25] for approximate nearest neighbor searches to replace the exact nearest neighbor calculations. Compared to kernel approaches of Mean Shift clustering, which are $O(n^2)$ where n is the size of the dataset, our nearest neighbors approach enables a scalable $O(n)$ implementation of the gradient ascent and cluster labeling. In addition to this scalability, nearest neighbors Mean Shift is a sparse method, and so it can be applied to the high dimensional datasets, unlike kernel Mean Shift which is limited to a maximum of 5 or 6 dimensions in practice.

Furthermore, existing programming paradigms for dealing with parallelism, such as MapReduce [10] and Message Passing Interface (MPI) [20], have been demonstrated to be the best practical choices for implementing these clustering algorithms. Most parallel and distributed clustering algorithms follow the general framework depicted in Fig. 1 [2,18,23,30].

1. **Partition.** Distribution and partitioning of data takes place over machines.
2. **Local Clustering.** The clustering algorithm is applied on each machines' data independently.
3. **Global Clustering.** Previous clustering results are aggregated into a global clustering.
4. **Refinement of Local Clusters.** Eventually global clustering results can be used to refine the local clusters.

MapReduce allows an unexperienced programmer to develop parallel programs and create a program capable of using computers in a cloud. Indeed the MapReduce paradigm has become popular since data are stored on a distributed file system, which

* Corresponding author.
E-mail address: beck.gael@gmail.com (G. Beck).

https://doi.org/10.1016/j.jpdc.2019.07.015
0743-7315/

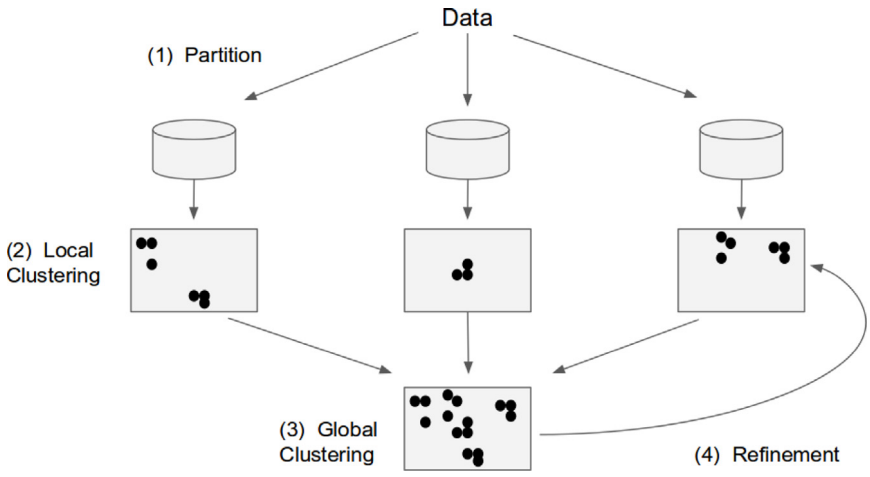


Fig. 1. The general framework of most parallel and distributed clustering algorithms [2].

offers data replication, as well as for its ability to execute computations locally on each data node. Thus, we implement this approximate nearest neighbor Mean shift clustering algorithm on a distributed Apache Spark/Scala framework [29], which allows us to carry out clustering on Big datasets. Users specify the computation in terms of a *map* and a *reduce* function, and the underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, handles machine failures, and schedules inter-machine communication to make efficient use of the network and disks.

The organization of the paper is as follows. We review related works in Section 2. In Section 3 we briefly introduce the problem, prior to elaborating our algorithmic contributions for the gradient ascent and cluster labeling in Section 4. Section 5 is related to the experiments conducted on the Grid'5000 testbed where we examine the role of the key tuning parameters for both accelerating the execution time and controlling the clustering quality. Concluding remarks are made in Section 6.

2. Related works

For comprehensive reviews of clustering, see for example the monographs in [1,5]. The principal scalable clustering algorithm is the well-known k-means [3]. This algorithm has the advantage of having a single parameter k which stands for the number of desired clusters, which needs to be specified a priori. The algorithm works by moving k prototypes towards the centroids formed by their closest data points. This process is repeated iteratively until the intra-class variance (the sum of squared distances from each data point within a cluster to its corresponding prototype) is minimized. DBScan [16] is a well-known density based algorithm. It takes two parameters, ε which defines the radius of the hypersphere and *minPts* which is the minimum number of points above which a corresponding hypersphere is considered to be sufficiently dense. Each time the density threshold is reached, the data points in the same hypersphere belong to the same cluster, and the process is extended to include more data points until the data density falls under the threshold determined by ε and *minPts*. The remaining data points are considered to be noise. One notable advantage of this algorithm is its ability to automatically detect the number of clusters with arbitrary shape, but it remains difficult to tune it correctly. Like DBScan, Mean Shift is a density based algorithm and can detect automatically the number of clusters with arbitrary shape. Most studies on the Mean Shift clustering have focused on the kernel versions [24,27,28]. The latter authors compared Gaussian, Cauchy and generalized Epanechnikov kernels to study the behavior of tuning parameters of kernel Mean Shift clustering. Fewer studies have been carried out on nearest neighbors Mean Shift, with recent contributions on their theoretical (such as convergence) and practical issues from [4,11]. These authors did not fully resolve the scalability issues, and so were only able to demonstrate their findings with moderate values of the number of nearest neighbors. In this paper, we introduce fully scalable algorithms for the two most computationally intensive steps of density gradient ascent and cluster labeling, due to an improved LSH algorithm and a novel labeling algorithm.

3. Mean shift clustering

The Mean Shift algorithm and its variants consist in two major steps as described in Algorithm 1. The first important step (the density gradient ascent) is generally the most computationally intensive. This gradient ascent can be computed in different ways, such as with kernel functions or as we propose in this paper, nearest neighbors. The second step is the cluster labeling phase where we use the result from the first step to assign cluster labels to the original data points.

Algorithm 1 Mean Shift

Input: Candidate points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$
Output: Cluster labels $\{\tilde{c}(\boldsymbol{x}_1), \ldots, \tilde{c}(\boldsymbol{x}_m)\}$
Step 1: Density gradient ascent;
Step 2: Cluster labeling;

3.1. Density gradient ascent

The Mean Shift method for a d-dimensional point $\boldsymbol{x}$, generates a sequence of points $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots\}$ which follows the gradient density ascent paths [11] 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 $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$ is a random sample drawn from a common density f and the k nearest neighbors of $\boldsymbol{x}$ are $k\text{-nn}(\boldsymbol{x}) = \{\boldsymbol{X}_i : \|\boldsymbol{x} - \boldsymbol{X}_i\| \leq \delta_{(k)}(\boldsymbol{x})\}$ with $\delta_{(k)}(\boldsymbol{x})$ is the kth nearest neighbor distance, and $\boldsymbol{x}_0 = \boldsymbol{x}$. Equation (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 neighbors for the next iterate $\boldsymbol{x}_{j+1}$. The gradient ascent paths towards the local modes produced by Eq. (1) form the basis

of Algorithm 2, our nearest neighbor Mean Shift gradient ascent (NNGA).

The inputs to the NNGA are the data sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$ and the candidate points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$, which we wish to cluster (these can be the same as $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$, but this is not required); and the tuning parameters: the number of nearest neighbors k, the tolerance under which subsequent iterations in the Mean Shift update are considered to be convergent ε_1, the maximum number of iterations $j_{\max}$.

Algorithm 2 NNGA — Nearest Neighbor Gradient Ascent with exact k-nn

Input: $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n\}, \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}, k, \varepsilon_1, j_{\max}$
Output: $\{\boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_m^*\}$
1: Compute similarity matrix of Euclidean pairwise distances between $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n\}$ and $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$, and sort each row;
2: **for** $\ell := 1$ to m **do**
3: $j := 0$; $\boldsymbol{x}_{\ell,0} := \boldsymbol{x}_\ell$;
/* Search for k-nn based on similarity matrix */
4: $\boldsymbol{x}_{\ell,1} :=$ mean of k-nn$(\boldsymbol{x}_{\ell,0})$;
5: **while** $\|\boldsymbol{x}_{\ell,j+1}, \boldsymbol{x}_{\ell,j}\| > \varepsilon_1$ **or** $j < j_{\max}$ **do**
6: $j := j + 1$;
7: $\boldsymbol{x}_{\ell,j+1} :=$ mean of k-nn$(\boldsymbol{x}_{\ell,j})$;
8: $\boldsymbol{x}_\ell^* := \boldsymbol{x}_{\ell,j}$;

The classical version of the NNGA introduced in Algorithm 2 requires, for each candidate point that we compute, the distance to all other data points, from which the mean of the k nearest neighbors is set to be the current prototype. The algorithm associates this prototype with the original candidate points. We repeat this step until the prototype moves less than a threshold ε_1 or whenever the algorithm has reached $j_{\max}$ iterations.

The complexity for the exact nearest neighbors search of a single point is $n\log(n)$. Applied to every data point multiple times, this complexity increases to $n^2 j_{\max}\log(n)$, preventing its application on Big datasets.

4. Scalable algorithms for mean shift

4.1. Approximate nearest neighbors search for density gradient ascent

One promising algorithmic complexity reduction approach relies on computing approximate nearest neighbors rather than exact neighbors. Among the techniques that can be used, Locality Sensitive Hashing (LSH), introduced in [9,17], is a probabilistic method based on a random scalar projection of multivariate data point $\boldsymbol{x}$ defined below:

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

where $\boldsymbol{Z} \sim N(0, \mathbf{I}_d)$ is a standard d-variate normal random variable and $U \sim \text{Unif}(0, v)$ is a uniform random variable on $[0, v)$, $v > 0$. The LSH is parametrized by the number of buckets M_1 in the hash table. In our context, we propose to set $v = 1$, and 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_1 partition intervals of width $w = (L_{(n)} - L_{(1)})/M_1$ where $I_j = [L_{(1)} + w(j-1), L_{(1)} + wj]$, $\forall j \in \{j = 1, \ldots, M_1\}$. The hash value of $\boldsymbol{x}$ is the index of the interval in which $L(\boldsymbol{x}; 1)$ falls

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

where $\mathbf{1}\{\cdot\}$ is the indicator function. To search for approximate nearest neighbors, the reservoir of potential nearest neighbors 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 neighbors of $\boldsymbol{x}$ are the k nearest neighbors only drawn from the reduced reservoir $R(\boldsymbol{x})$ defined as below:

$$k\text{-}\widetilde{\text{nn}}(\boldsymbol{x}) = \{\boldsymbol{X}_i \in R(\boldsymbol{x}) : \|\boldsymbol{x} - \boldsymbol{X}_i\| \leq \delta_{(k)}(\boldsymbol{x})\},$$

where $\delta_{(k)}(\boldsymbol{x})$ is the kth nearest neighbor distance to $\boldsymbol{x}$. The approximation error in the nearest neighbors to $\boldsymbol{x}$ induced by searching in $R(\boldsymbol{x})$ rather than the full dataset is probabilistically controlled [25]. Our improvement to the classical LSH is based on the following observations:

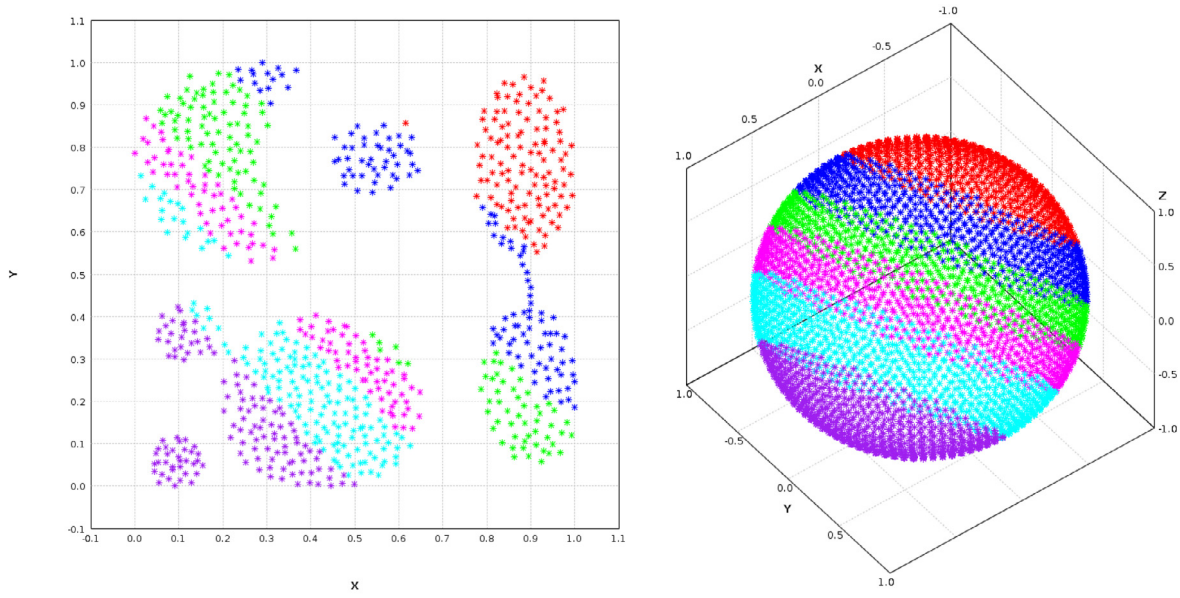
- it takes into account the properties of the local data space more accurately. Rather than looking exclusively in the bucket where the prototype lies, we also look in the adjacent buckets on both sides. The main advantage of this is to take into account the case where a prototype is at the border of a bucket and so some of its k nearest neighbors mostly likely fall into the adjacent buckets. This is especially important for high values of k. It is computationally more expensive but the cost can be controlled for a fixed bucket size. The memory cost is increased by a factor of 3 per partition due to copying the adjacent layers into the active one.
  The LSH method partitions the data space into buckets of approximately k nearest neighbors, which are delimited by parallel hyperplanes. In practice, the LSH controls the number of neighboring buckets to two, except for the edge buckets which have only one neighbor bucket. This is in contrast to cell based buckets, where the number of neighbor buckets increases exponentially with the number of dimensions. Fig. 2 illustrates the LSH buckets of approximate nearest neighbors on 2D (Aggregation) and 3D (GolfBall) data examples: the orientation of hyperplanes depends on the random projections utilized to construct the buckets.
- it adds the possibility of allowing the prototype to change buckets during its gradient ascent. In this case, we look for its k_2 nearest neighbors in order to place the prototype within the most representative bucket using a majority voting process. Thus a prototype can pass through multiple buckets before converging to its final position, as illustrated in Fig. 3.

Algorithm 3 describes the NNGA$^+$, an approximate nearest neighbor search using LSH with the hash function H. The inputs are the data sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$, the candidate points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$, and the tuning parameters: the number of nearest neighbors k_1 and the number of buckets in the hash table M_1. In line 1, the hash table is created by applying the LSH to the data values $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$. In lines 2–6, for each candidate point $\boldsymbol{x}_\ell$, the approximate k_1 nearest neighbors k-$\widetilde{\text{nn}}(\boldsymbol{x}_\ell)$ are computed from within the reservoir $R(\boldsymbol{x}_\ell)$.

Algorithm 3 NNGA$^+$ – Approximate Nearest Neighbors Gradient Ascent with LSH and adjacent buckets

Input: $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n\}, \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}, k_1, M_1$
Output: $\{k\text{-}\widetilde{\text{nn}}(\boldsymbol{x}_1), \ldots, k\text{-}\widetilde{\text{nn}}(\boldsymbol{x}_m)\}$
/* Create hash table with M_1 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: $R(\boldsymbol{x}_\ell) := \{\boldsymbol{X}_i : H_i = H(\boldsymbol{x}_\ell), i \in \{1, \ldots, n\}\}$
4: **while** card$(R(\boldsymbol{x}_\ell)) < k_1$ **do**
5: $R(\boldsymbol{x}_\ell) := R(\boldsymbol{x}_\ell) \cup$ neighbor bucket;
6: $k\text{-}\widetilde{\text{nn}}(\boldsymbol{x}_\ell) :=$ k-nn from $R(\boldsymbol{x}_\ell)$ to $\boldsymbol{x}_\ell$;

Fig. 4 illustrates the effect of including adjacent buckets in NNGA$^+$ versus nearest neighbor gradient ascent without adjacent



(a) Aggregation with $M_1 = 6$ buckets. (b) GolfBall with $M_1 = 6$ buckets.

Fig. 2. LSH buckets for the Aggregation and GolfBall datasets.

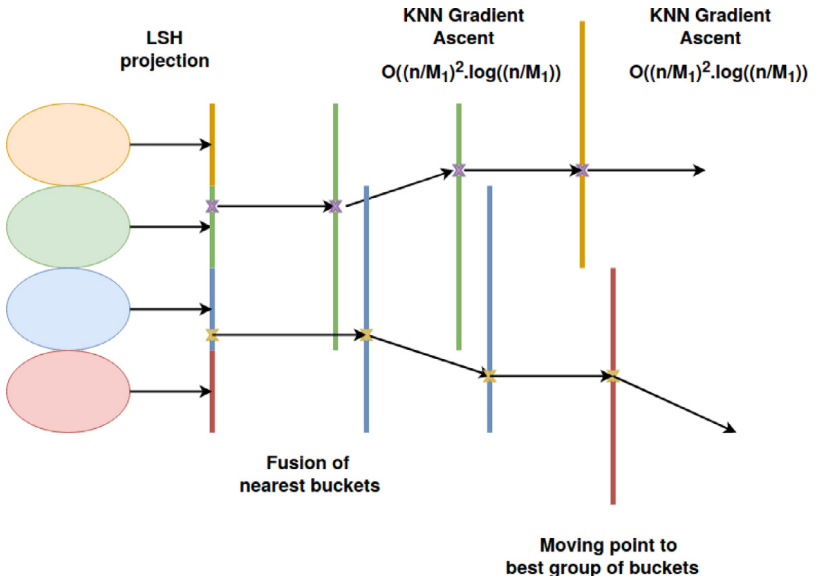


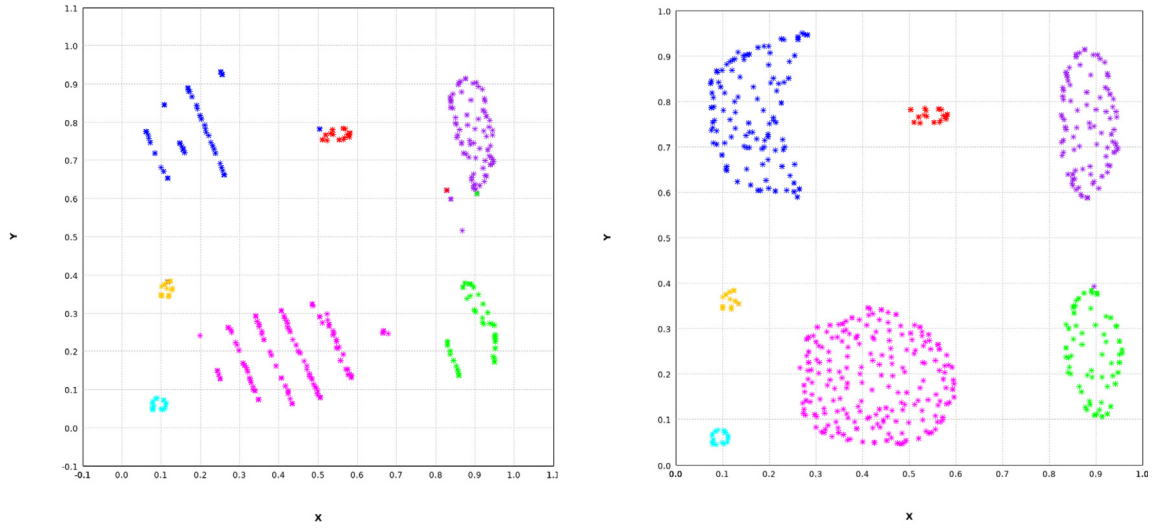
Fig. 3. Passage of a prototype through different LSH buckets during the gradient ascent; n is the number of data points and M_1 is the number of buckets.

buckets on the Aggregation dataset. For $k_1 = 20$ without adjacent buckets, in Fig. 4a, then we observe that the data are artificially forced to follow the hyperplanes which delimit the different buckets. Fig. 4b shows our algorithmic improvement with adding one layer of adjacent buckets where the underlying structure of data is maintained.

Whilst the use of the LSH to reduce the complexity of kernel Mean Shift clustering was already proposed in [7], these authors did not quantify the reduction in complexity. The complexity of our NNGA$^+$ is reduced to $O((\frac{n}{M_1})^2 \log(\frac{n}{M_1}))$ per bucket with M_1 buckets, and so the total complexity is $O((\frac{n}{M_1})^2 \log(\frac{n}{M_1}))$ for all buckets. Because of this segmentation of the original data space into M_1 sub-spaces, the complexity is inversely proportional to the number of buckets. The trade-off is that the data points in each bucket have to be sufficiently representative of the local properties of the original space. Thus the number of buckets M_1 is a crucial tuning parameter. Despite this, there are no optimal methods for selecting the number of buckets [15]. Consequently, we will examine empirical choices of the number of buckets in the sequel.

4.2. Cluster labeling: ε-proximity

NNGA$^+$ carries out the gradient ascent on the data points until they have converged to their prototypes. The question is then how to assign cluster labels to the data points. A first solution is to assign the same cluster to all points sharing the same prototype. As observed in Fig. 4(b), even with a prior good choice of k_1, there are (possibly) hundreds of generated prototypes, so assigning a label to each point according to its closest prototype is not effective because it can generate too many clusters. Applying



(a) Aggregation after NNGA$^+$ with number of nearest neighbors $k_1 = 20$, without adjacent buckets.

(b) Aggregation after NNGA$^+$ with number of nearest neighbors $k_1 = 20$, with adjacent buckets.

Fig. 4. NNGA$^+$ without and with adjacent buckets for the Aggregation dataset.

the density gradient ascent NNGA$^+$ leads to a converged dataset with increased inter-cluster distances and decreased intra-cluster distances as compared to the original dataset. In order to further exploit this property, we propose a new proximity-based approach where points which are under a threshold $\varepsilon > 0$ from each other are considered to belong to the same cluster.

Algorithm 4 illustrates ε-proximity cluster labeling. It consists in exploring the similarity matrix **S** which is defined as a map whose objects IDs are the keys and whose pairs (object IDs, distances) are the values. We initialize the process by taking the first object of **S** and cluster with it every point whose distance is less than ε. We then apply this exploration process by iteratively adding the k_2 nearest neighbors of these added points until this process terminates. During the process we remove the explored points from **S** to avoid repeated calculations. Once the first cluster is generated, we take another object from outside this first cluster from the reduced similarity matrix **S** and repeat the above cluster formation, until all objects are assigned to a cluster label.

In order to apply this algorithm, we have to build the similarity matrix which has a $O(n^2)$ time complexity, preventing any Big Data application. A scalable version consists of applying this algorithm in each LSH partition, and merging each bucket its right or left adjacent bucket to maintain the bucket order.

Once this step is completed, we apply a *MapPartitions* procedure to check if two clusters of two different buckets share at least 1 pair of points which are less than ε apart, then these two clusters are considered to form a single cluster. We obtain a dataset which chains common clusters between partitions: all chained clusters are assigned with the same label by generating an undirected graph where each connected subgraph represents a cluster. The search for connected components in a graph is a common problem which can be solved in linear time in the number of vertices.

It is important to bound the number of data points in each bucket because the scalable version of ε-proximity clustering and the check for cluster merging between two buckets have quadratic complexity in this size. Empirically we advise to set the number of buckets M_1 in order to have around 500 to 2000 data points in each bucket.

A notable problem still remains with the choice of the main tuning parameter ε. We set it to be the average of distance from each point to their k nearest neighbors. We compute it as an approximate value in using LSH procedure in order to maintain the scalability property.

Algorithm 4 ε-proximity labeling

Input: $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$, **S**, ε
Output: $\{\tilde{c}(\boldsymbol{x}_1), \ldots, \tilde{c}(\boldsymbol{x}_m)\}$
1: `needToVisit` ← Set(**S**.head)
2: $c_{ID} \leftarrow 0$
3: `clusters` ← Map.empty[Int, Set[Int]]
4: `clusters` += (c_{ID}, `needToVisit`)
5: **while S** has elements **do**
/* p_c is the current point of `needToVisit` */
6: **for each** p_c **in** `needToVisit` **do**
/* Add all points under ε to needToVisit */
7: `needToVisitUpdated` ← $\{p \in \mathbf{S}, \text{dist}(p, p_c) \leq \varepsilon\}$
/* Remove explored point from the similarity matrix */
8: **S** -= p_c
/* Update points to explore on next iterations */
9: `needToVisit` ← `needToVisitUpdated`
10: **if** `neetToVisit` is empty **then**
11: c_{ID} += 1
12: `needToVisit` ← Set(**S**.head)
/* Create a new entry in the clusters map */
13: `clusters` += (c_{ID}, `needToVisit`)
14: **else**
/* Add new points to cluster c_{ID} */
15: `clusters`(c_{ID}) += `needToVisit`

5. Numerical experiments

Our experiments are carried out on the Grid'5000 testbed which is the French national testbed for computer science research. We use a dedicated Spark Linux image optimized for Grid'5000 where Apache Spark is deployed on top of Spark in

Table 1
Experimental datasets. n is the dataset size, d is the data dimension, N is the number of clusters.

Dataset	n	d	N
R15	600	2	15
Aggregation	788	2	7
Sizes5	1000	2	4
EngyTime	4096	2	2
Banana	4811	2	2
S3	5000	2	15
Disk6000	6000	2	2
Unbalance	6500	2	8
DS1	9153	2	14
Hepta	212	3	7
Hyperplane	100 000	10	5
CovType10	581 012	10	7
ScalabiltyDS	140 000 000	10	–

Standalone mode. Apache Spark is a fast general purpose distributed computing system based on a master–slaves architecture. Only the deployment of the image is automatized. We manually reserve the nodes and provide the Spark cluster with our code to execute the different experiments on a 2×8 core Intel Xeon E5-2630v3 CPUs and 128 Gb RAM setup.

A key concept in Spark is the resilient distributed dataset (RDD) which is a read-only collection of objects partitioned across a group of machines which can be rebuilt if necessary from the hierarchy of previous RDD operations. Most of the *Map* and *Reduce* operations will be performed on RDDs even if other pure Scala *Map* and *Reduce* operations are executed inside each Spark partition. We implement our algorithm in Scala because it is the Spark's native language and thus allows for good performance.

We use a range from 2 to high dimensional datasets with different sizes, as summarized in Table 1 [12,26]. To ensure the comparability of the results across these different datasets, all algorithms are carried out on the normalized version of the datasets: $x_i = (x_i - x_i^{\min})/(x_i^{\max} - x_i^{\min})$ where x_i is the ith component of $\boldsymbol{x}$, and $x_i^{\min}$, $x_i^{\max}$ are respectively the ith marginal minimum and maximum values. We repeat each experiment ten times for robustness.

5.1. Comparison of NNGA$^+$ with k-means and DBScan

In this section, we compare NNGA$^+$ (Algorithm 3) with ε-proximity labeling (Algorithm 4) to k-means [3] and DBScan [16]. Table 2 shows the optimal combinations of tuning parameters used in the comparison between the gradient ascent (NNGA$^+$) with ε-proximity, k-means [3], DBScan [16] and ε-proximity. These optimal combinations are obtained by grid searches over a range of values of each parameter.

To evaluate the quality of the clustering, we use both the Normalized Mutual Information (NMI) [8] and the Rand index [22], as displayed in Table 3. The bold values are the highest value within each row. The value of each measure lies between 0 and 1. A higher value indicates better clustering results. The conclusions from both of these indices are similar for all datasets, except for the large datasets Hyperplan $n = 100\,000$ and CovType10 $n = 581\,012$. NNGA$^+$ with ε-proximity achieves the highest clustering accuracies for 7 out of the 12 experimental datasets. For those datasets where it is not the best performing, it is close behind, with the only possible exception for Disk6000, whose nested clusters are difficult for NNGA$^+$ to detect accurately.

Table 3
NMI and Rand clustering quality indices for the cluster labeling on the experimental datasets for NNGA$^+$ with ε-proximity labeling, k-means and DBScan. The bold entries indicate the best results within each row, and $\pm$ entries are the standard deviations over 10 trials.

Dataset		NNGA$^+$ with ε-proximity	k-means	DBScan
Aggregation	NMI	0.97 ± 0.02	0.83 ± 0.02	$\mathbf{0.98 \pm 0.00}$
	Rand	0.98 ± 0.02	0.91 ± 0.01	$\mathbf{0.99 \pm 0.00}$
Banana	NMI	$\mathbf{1.00 \pm 0.00}$	0.31 ± 0.00	$\mathbf{1.00 \pm 0.00}$
	Rand	$\mathbf{1.00 \pm 0.00}$	0.70 ± 0.00	$\mathbf{1.00 \pm 0.00}$
Disk6000	NMI	0.32 ± 0.00	0.00 ± 0.00	$\mathbf{1.00 \pm 0.00}$
	Rand	0.34 ± 0.00	0.50 ± 0.00	$\mathbf{1.00 \pm 0.00}$
DS1	NMI	0.03 ± 0.01	0.75 ± 0.01	$\mathbf{0.94 \pm 0.00}$
	Rand	0.80 ± 0.01	0.86 ± 0.00	$\mathbf{0.98 \pm 0.00}$
EngyTime	NMI	0.85 ± 0.08	$\mathbf{0.98 \pm 0.00}$	0.75 ± 0.00
	Rand	0.93 ± 0.05	$\mathbf{1.00 \pm 0.00}$	0.90 ± 0.00
Hepta	NMI	0.97 ± 0.04	$\mathbf{0.98 \pm 0.03}$	0.83 ± 0.00
	Rand	0.98 ± 0.04	$\mathbf{0.99 \pm 0.02}$	0.94 ± 0.00
R15	NMI	0.91 ± 0.02	0.96 ± 0.02	$\mathbf{0.99 \pm 0.00}$
	Rand	0.98 ± 0.00	0.99 ± 0.01	$\mathbf{1.00 \pm 0.00}$
S3	NMI	0.74 ± 0.00	$\mathbf{0.78 \pm 0.01}$	0.42 ± 0.00
	Rand	$\mathbf{0.96 \pm 0.00}$	$\mathbf{0.96 \pm 0.00}$	0.52 ± 0.00
Sizes5	NMI	$\mathbf{0.89 \pm 0.01}$	0.81 ± 0.12	0.80 ± 0.00
	Rand	$\mathbf{0.97 \pm 0.00}$	0.88 ± 0.13	$\mathbf{0.97 \pm 0.00}$
Unbalance	NMI	0.98 ± 0.01	0.94 ± 0.05	$\mathbf{0.99 \pm 0.00}$
	Rand	$\mathbf{1.00 \pm 0.00}$	0.97 ± 0.03	$\mathbf{1.00 \pm 0.00}$
Hyperplane	NMI	$\mathbf{0.04 \pm 0.00}$	0.01 ± 0.00	One cluster only
	Rand	0.31 ± 0.00	$\mathbf{0.62 \pm 0.00}$	
CovType10	NMI	$\mathbf{0.09 \pm 0.01}$	0.07 ± 0.006	Dataset is too massive
	Rand	0.56 ± 0.04	$\mathbf{0.59 \pm 0.00}$	

Table 2
Optimal tuning parameter choices for clustering comparisons. k_1 is the number of nearest neighbors in the gradient ascent, M_1 is the number of buckets in the LSH, ε_1 is the distance threshold for the ε-proximity labeling, bold values are manually settled ε_1, normal ones correspond to approximate average value of its "ε_1 value" k nearest neighbors distance, for NNGA$^+$ with ε-proximity; k is the number of clusters for k-means; ε is the radius of the hypersphere and *minPts* is the minimum of number of data points for DBScan.

Dataset	NNGA$^+$ with ε-proximity	k-means	DBScan
Aggregation	$k_1 = 50, M_1 = 8, \varepsilon_1 = 30$	$k = 7$	$\varepsilon = 0.05, minPts = 8$
Banana	$k_1 = 40, M_1 = 8, \boldsymbol{\varepsilon_1 = 0.1}$	$k = 2$	$\varepsilon = 0.02, minPts = 3$
Disk6000	$k_1 = 100, M_1 = 8, \boldsymbol{\varepsilon_1 = 0.02}$	$k = 7$	$\varepsilon = 0.02, minPts = 4$
DS1	$k_1 = 50, M_1 = 8, \varepsilon_1 = 30$	$k = 14$	$\varepsilon = 0.03, minPts = 25$
EngyTime	$k_1 = 200, M_1 = 8, \varepsilon_1 = 50$	$k = 2$	$\varepsilon = 0.1, minPts = 100$
Hepta	$k_1 = 20, M_1 = 4, \varepsilon_1 = 10$	$k = 7$	$\varepsilon = 0.1, minPts = 10$
R15	$k_1 = 20, M_1 = 8, \varepsilon_1 = 5$	$k = 15$	$\varepsilon = 0.05, minPts = 25$
S3	$k_1 = 40, M_1 = 8, \varepsilon_1 = 15$	$k = 15$	$\varepsilon = 0.05, minPts = 50$
Sizes5	$k_1 = 20, M_1 = 8, \varepsilon_1 = 5$	$k = 4$	$\varepsilon = 0.08, minPts = 8$
Unbalance	$k_1 = 40, M_1 = 8, \varepsilon_1 = 80$	$k = 8$	$\varepsilon = 0.05, minPts = 20$
Hyperplane	$k_1 = 50, M_1 = 100, \varepsilon_1 = 5$	$k = 5$	$\varepsilon = 0.05, minPts = 8$
CovType10	$k_1 = 50, M_1 = 500, \varepsilon_1 = 20$	$k = 5$	$\varepsilon = 0.05, minPts = 8$

Table 4
NMI and Rand clustering quality indices and execution times for NNGA$^+$ with ε-proximity labeling for the DS1 dataset. k_1 is the number of nearest neighbors in the gradient ascent, M_1 is the number of buckets in the LSH, ε_1 is the distance threshold for the ε-proximity labeling. The bold entries indicate the best accuracy results, and $\pm$ entries are the standard deviations over 10 trials.

DS1	$M_1 = 12, \varepsilon_1 = 30$	$k_1 = 10$	$k_1 = 50$	$k_1 = 200$
NMI		0.0078 ± 0.0003	0.0315 ± 0.0016	0.0206 ± 0.0021
Rand		0.7268 ± 0.0146	0.8190 ± 0.0007	0.8129 ± 0.0029
Execution time (s)		81.1 ± 2.3	84.2 ± 6.5	76.5 ± 2.9
	$k_1 = 50, \varepsilon_1 = 30$	$M_1 = 8$	$M_1 = 12$	$M_2 = 20$
NMI		0.0240 ± 0.0016	0.0315 ± 0.0016	0.0183 ± 0.0020
Rand		0.8039 ± 0.0024	0.8190 ± 0.0007	0.7932 ± 0.0041
Execution time (s)		133.4 ± 4.2	84.2 ± 6.5	43.9 ± 2.0
	$k_1 = 50, M_1 = 12$	$\varepsilon_1 = 5$	$\varepsilon_1 = 30$	$\varepsilon_2 = 100$
NMI		0.1085 ± 0.0017	0.0315 ± 0.0016	0.0034 ± 0.0002
Rand		0.8283 ± 0.0001	0.8190 ± 0.0007	0.7078 ± 0.0006
Execution time (s)		78.3 ± 3.6	84.2 ± 6.5	76.5 ± 2.7

5.2. Effect of tuning parameters on clustering accuracy and execution time

Table 4 illustrates clustering quality results with NMI, Rand and time duration exclusively on DS1 dataset. We present three rows, for three parameters ε_1, M_1, and k_1, on which for each of them we fix two parameters and change the third one.

As expected from the algorithm design the only parameter which influence strongly experiences duration is the number of buckets M_1 used for the LSH. The larger M_1 is the faster one algorithm run becomes. It is due to quadratic complexity operations that remain in each buckets as well as for gradient ascent that for local ε-proximity. Increasing the buckets number will decrease number of elements per bucket and then greatly decrease needed computation time per bucket. Concerning link with accuracy scores, they stay relatively stable for Rand and NMI.

We observe that specifics combinations of parameters of ε_1 and k_1 perform better than others in Table 4. Higher values of k_1 will result fusion of closest clusters into bigger ones when smaller values will smooth clusters shapes. ε-proximity clustering algorithm will precisely gives results depending of how main parameter ε_1 is settled. Smaller ε_1 values will provide much more cluster because less points tend to be closer from each others with a small ε_1 value. At the opposite bigger ε_1 values will generate less clusters with more points. It is decisive on some indices as the NMI. Too many discovered clusters compared to ground truth classes number will drastically reduce this index score even if there is a belonging logic with original classes, fortunately the Rand index suggest that there is a consistency with the original labeling.

5.3. NNGA$^+$ as a data-shrinkage method

As shown in Fig. 5, our version of the nearest neighbors gradient ascent NNGA$^+$ results in shrinking the data points towards their local modes. DS1 after NNGA$^+$ (with $k_1 = 50$ nearest neighbors) presents a more compact version than its original version, maintaining the underlying data structures whilst increasing empty space between clusters; likewise for the Hepta dataset.

Being a data shrinkage method, this implies that we can apply NNGA$^+$ before we apply other clustering algorithms. Fig. 6(a) shows an application of k-means on Sizes5 dataset with $k = 4$ clusters without NNGA$^+$. Even if the number of clusters in the k-means on the left is the correct number, these four clusters do not match so closely the original clusters (NMI=0.81, Rand=0.88). Fig. 6(b) shows the k-means cluster labeling results applied on NNGA$^+$ output with $k_1 = 20$. We observe that these clusters are more similar to the original clusters (NMI = 0.89, Rand = 0.95) than with k-means only.

DBScan cluster labeling collates points more efficiently after NNGA$^+$ is applied than without NNGA$^+$, as shown in Fig. 7. NNGA$^+$ is an efficient way to attach noisy points to their closest cluster. Furthermore, NNGA$^+$ facilitates more robust choices for the DBScan parameters, since it increases the local density which improves the detection of smaller clusters. The NMI and Rand increase after the NNGA$^+$ data shrinkage.

5.4. Evaluation of scalability

5.4.1. Density gradient ascent

As the data are distributed over all buckets or Spark partitions, this allows for efficient computations in the local environment of a data point. Fig. 8(a) shows that the execution time gradually decreases as the number of slaves increases for a dataset of fixed size. In Fig. 8(b), for a fixed number of slaves, if we maintain the constant number of elements per bucket, the execution time grows linearly with the size of the dataset. This indicates that the number of nearest neighbors k_1 needs to be constrained. The number of layers p indicates the adjacent buckets in the LSH which can be searched to find nearest neighbors: $p = 0$ means that no adjacent buckets, $p = 1$ means the immediately adjacent buckets to the left and right, $p = 2$ means that second order adjacent buckets further away etc.

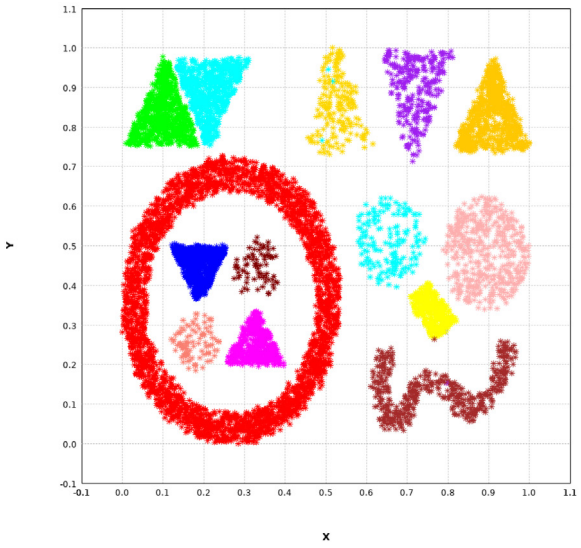
Since we have an $O((\frac{n}{M_1})^2 \log(\frac{n}{M_1}))$ complexity per bucket for NNGA$^+$, a suitable value for the number of buckets M_1 is to keep the $\frac{n}{M_1}$ ratio approximately equal to a constant C. Then the time complexity of NNGA$^+$ reduces to $O(nC \log(C))$. The scalability is demonstrated by the decrease in execution time with the number of slaves and a linear increase of execution time with the dataset size, reaching 140 million data points (the ScalabilityDS dataset) which is infeasible for the original quadratic algorithm.

5.4.2. LSH buckets and neighbor layers

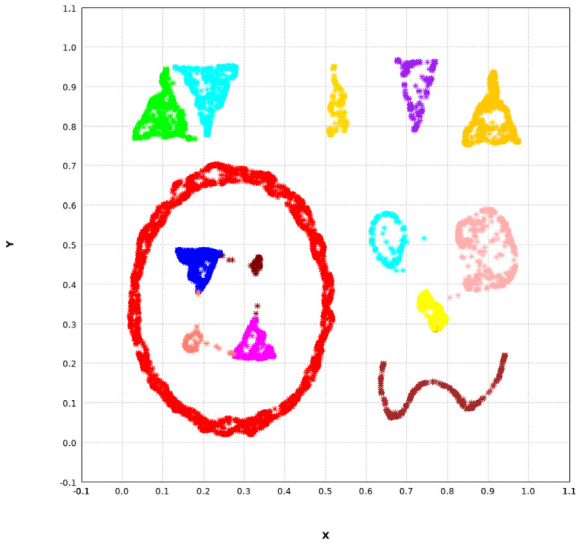
For a fixed number of neighbor layers, we observe in Fig. 9(a) that the execution time rapidly decreases and then slows down to reach a plateau, as the number of buckets increases. The observed plateau is due to the quadratic complexity of the NNGA$^+$: more buckets leads to fewer data points within each bucket and so the execution times can quickly reach the minimal plateau after a sufficiently large number of buckets. We also studied the influence of the number of neighbors layers p on the execution time. Whilst NNGA$^+$ has quadratic time complexity in each bucket, if we select an appropriate number of nearest neighbors k_1, then we are able to control the execution time of NNGA$^+$ to be linear with respect to the number of neighbors layers p, as illustrated in Fig. 9(b).

5.4.3. ε-proximity cluster labeling

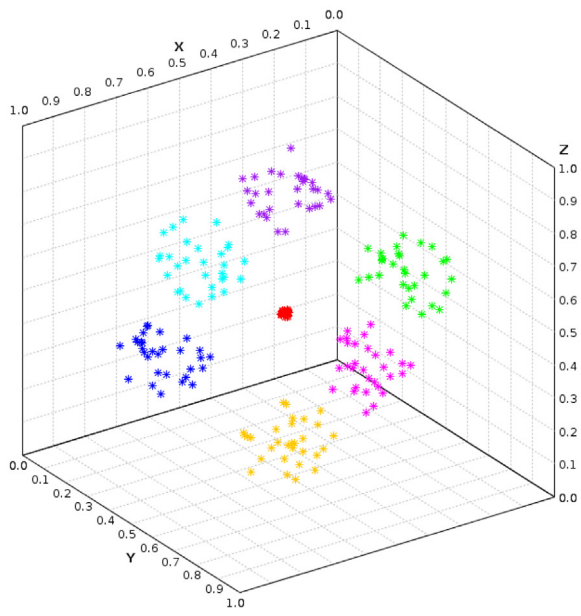
Concerning ε-proximity cluster labeling, similar remarks as for the gradient ascent apply here. As Fig. 10 follows same decrease in the execution time as for the gradient ascent as a function of the number of slaves and data points we can be confident in the scalability of our approach.



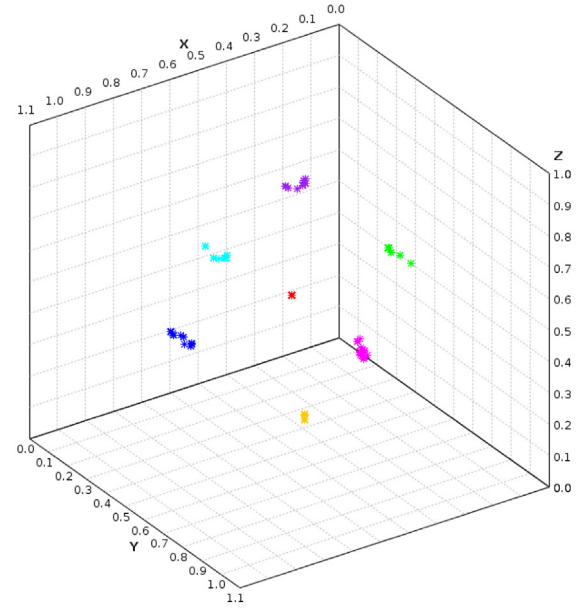
(a) DS1



(b) DS1 after NNGA$^+$ with number of nearest neighbors $k_1 = 50$



(c) Hepta



(d) Hepta after NNGA$^+$ with number of nearest neighbors $k_1 = 20$

Fig. 5. Data shrinkage after nearest neighbor gradient ascent NNGA$^+$ on the DS1 and Hepta datasets.

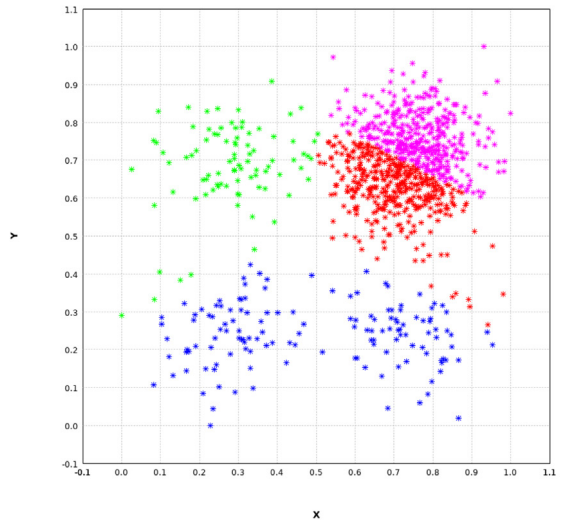
5.5. Software

In order to facilitate further experiments and reproducible research, we provide our contributions through an open source API which will contain NNGA$^+$, the ε-proximity cluster labeling, the traditional Mean-shift, and k-means implemented in Spark/Scala and the API documentation at https://github.com/Clustering4Ever/Clustering4Ever.

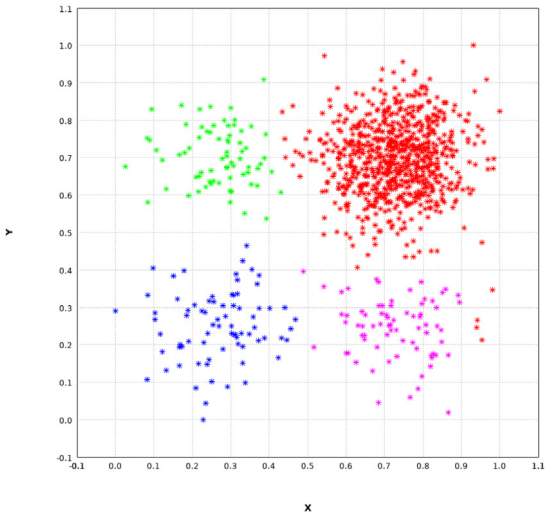
Moreover Spark Notebooks of our Mean Shit proposition are available at https://github.com/Spark-clustering-notebook/Clustering4Ever-Notebooks/tree/master/SparkNotebooks/0.9.4. The first one exposes simple usage of the algorithm and the second is the one used to obtain Table 4.

6. Conclusion

In this paper, we have introduced multiple improvements to the standard nearest neighbors gradient ascent used in Mean Shift algorithm. The first series of improvements are based on new usages of Locality Sensitivity Hashing for approximate nearest neighbors during the nearest neighbors gradient ascent (NNGA$^+$),

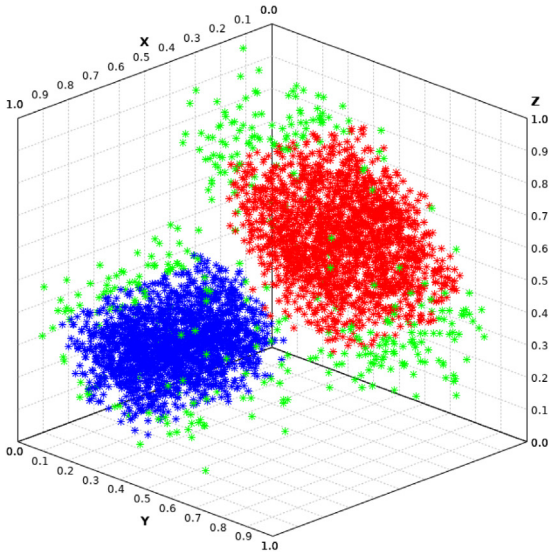


(a) Sizes5 after k-means ($k = 4$)

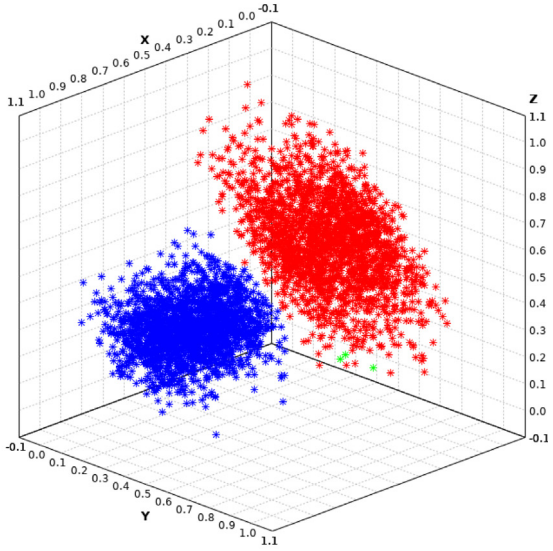


(b) Sizes5 after k-means ($k = 4$) with NNGA$^+$ (number of nearest neighbors $k_1 = 20$)

Fig. 6. k-means cluster labeling on the Sizes5 dataset without and with NNGA$^+$ data shrinkage.



(a) EngyTime after DBScan with parameters #minPts=22, #ε=0.07

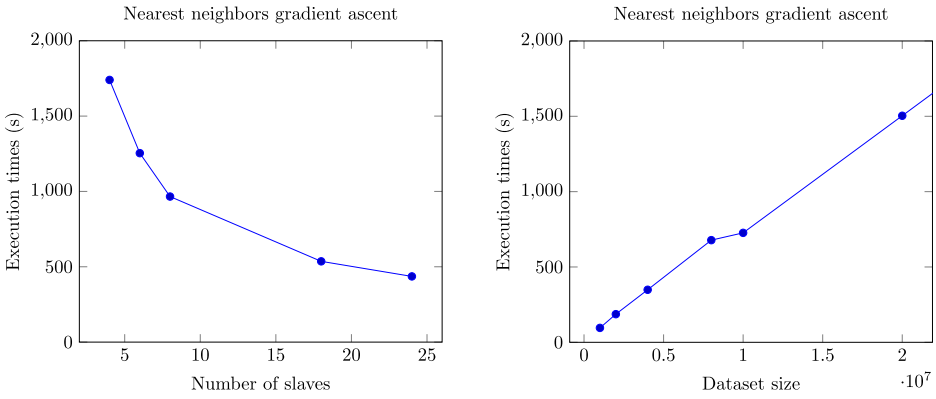


(b) EngyTime after DBScan with parameters #minPts=22, #ε=0.07 after NNGA$^+$ (number of nearest neighbors $k_1 = 200$)

Fig. 7. DBSCan cluster labeling on the Sizes5 dataset without and with NNGA$^+$ data shrinkage.

and the cluster labeling (ε-proximity). The second one is an efficient and scalable distributed implementation in the Spark/Scala ecosystem. We demonstrated that these improvements greatly decrease the execution time whilst maintaining a suitable quality of clustering. As a side benefit, we also show that using our NNGA$^+$ algorithm, as a pre-processing step in other clustering methods, can improve clustering quality. These improvements open the opportunity to apply Mean Shift related methods for Big Data clustering.

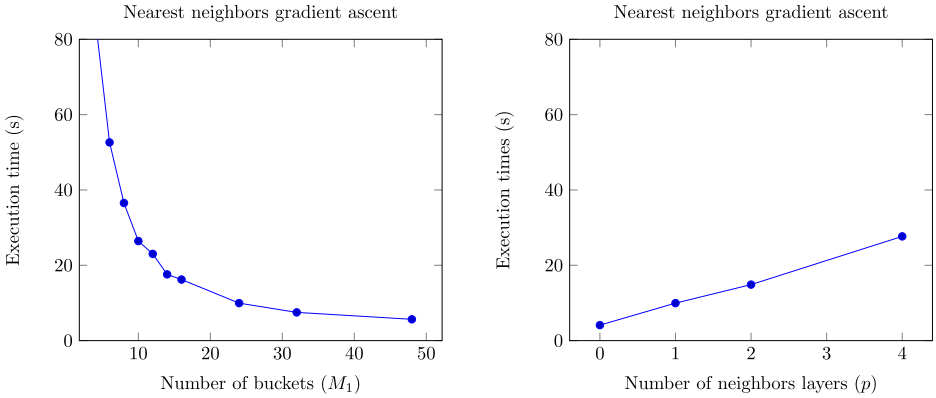
Optimal choices of the most important tuning parameters for our proposed methods for distributed clustering, namely the number of nearest neighbors for the density gradient ascent, the number of buckets for the LSH, and the threshold for ε-proximity cluster labeling will be a subject of further investigations. Future work includes exploring the utility of NNGA$^+$ for clustering algorithms with very high dimensional datasets as this is one of the most important advantages of nearest neighbor methods over kernel and other dense methods, different dissimilarity measure on the ε-proximity clustering and different hashing methods [19, 21] other than the simple hashing we have utilized.



(a) Dataset size $n = 500000$, #buckets $M_1 = 500$, #slaves varies, #layers $p = 1$.

(b) Dataset size n varies, #buckets $M_1 = n/1000$, #slaves $= 18$, #layers $p = 1$.

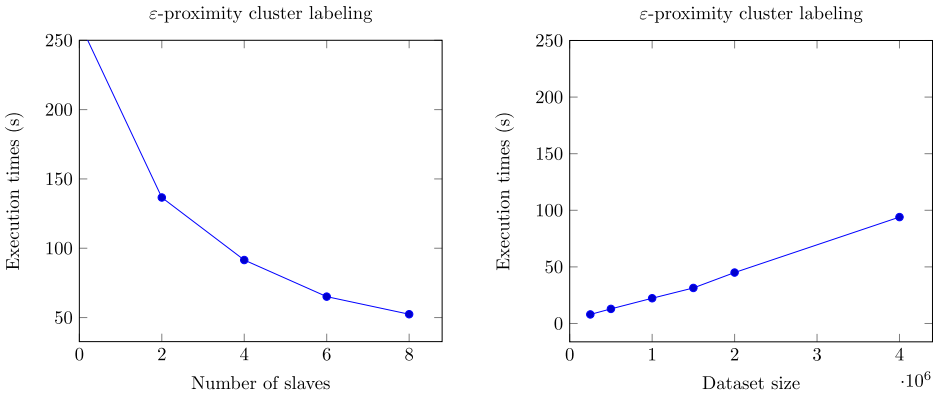
Fig. 8. (a) Execution times for NNGA$^+$ with respect to the number of slaves. (b) Execution times for NNGA$^+$ with respect to the dataset size n.



(a) Dataset size $n = 50000$, #buckets varies, #layers $p = 1$.

(b) Dataset size $n = 50000$, #buckets $M_1 = 24$, #layers p varies.

Fig. 9. (a) Execution times for NNGA$^+$ with respect to the number of buckets (M_1). (b) Execution times for NNGA$^+$ with respect to the number of adjacent layers (p).



(a) Dataset size $n = 1000000$, #buckets $M_1 = 1000$, #slaves varies, #layers $p = 1$.

(b) Dataset size n varies, #buckets $M_1 = n/1000$, #slaves $= 8$, #layers $p = 1$.

Fig. 10. (a) Execution times for ε-proximity cluster labeling with respect to the number of slaves. (b) Execution times for ε-proximity cluster labeling with respect to the dataset size n.

Declaration of competing interest

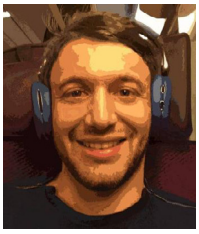
No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.jpdc.2019.07.015.

Acknowledgments

The experiments presented in this paper were carried out using the Grid'5000 testbed hosted by Inria, and supported by CNRS, France, RENATER, several universities, and other organizations https://www.grid5000.fr.

References

[1] C.C. Aggarwal, C.K. Reddy, Data Clustering: Algorithms and Applications, first ed., Chapman & Hall/CRC, 2013.
[2] C.C. Aggarwal, C.K. Reddy, Data Clustering: Algorithms and Applications, CRC Press, 2014.
[3] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, S. Vassilvitskii, Scalable K-means++, Proc. VLDB Endow. 5 (2012) 622–633.
[4] G. Beck, T. Duong, H. Azzag, M. Lebbah, Distributed mean shift clustering with approximate nearest neighbours, in: Proceedings of the 2016 International Joint Conference on Neural Networks (IJCNN), 2016, pp. 3110–3115.
[5] C.M. Bishop, Pattern Recognition and Machine Learning, Springer-Verlag, New York, 2006.
[6] Y. Cheng, Mean shift, mode seeking, and clustering, IEEE Trans. Pattern Anal. Mach. Intell. 17 (1995) 790–799.
[7] G.Z.Y. Cui, K. Cao, F. Zhang, An adaptive mean shift algorithm based on LSH, Procedia Eng. (2011) 265–269.
[8] L. Danon, A. Dí az Guilera, J. Duch, A. Arenas, Comparing community structure identification, J. Stat. Mech.: Theory E 2005 (2005) P09008.
[9] P.I.M. Datar, N. Immorlica, V.S. Mirrokni, Locality-sensitive hashing scheme based on p-stable distributions, in: Proceedings of the 20th Annual Symposium on Computational Geometry, 2004, pp. 253–262.
[10] J. Dean, S. Ghemawat, MapReduce: Simplified data processing on large clusters, Commun. ACM 51 (2008) 107–113.
[11] T. Duong, G. Beck, H. Azzag, M. Lebbah, Nearest neighbour estimators of density derivatives, with application to mean shift clustering, Pattern Recognit. Lett. 80 (2016) 224–230.
[12] P. Fränti, Clustering basic benchmark, http://cs.uef.fi/sipu/datasets, 2015.
[13] K. Fukunaga, L. Hostetler, Optimization of *k*-nearest-neighbor density estimates, IEEE Trans. Inform. Theory 19 (1973) 320–326.
[14] K. Fukunaga, L. Hostetler, The estimation of the gradient of a density function, with applications in pattern recognition, IEEE Trans. Inform. Theory 21 (1975) 32–40.
[15] P.I.S. Har-Peled, R. Motwani, Approximate nearest neighbor: Towards removing the curse of dimensionality, Theory Comput. (2012) 321–350.
[16] Y. He, H. Tan, W. Luo, S. Feng, J. Fan, MR-DBSCAN: a scalable MapReduce-based DBSCAN algorithm for heavily skewed data, Front. Comput. Sci. 8 (2014) 83–99.
[17] P. Indyk, R. Motwani, Approximate nearest neighbors: Towards removing the curse of dimensionality. in: Proceedings of the 30th Annual ACM Symposium on Theory of Computing, 1998, pp. 604–613.
[18] E. Januzaj, H.-P. Kriegel, M. Pfeifle, Dbdc: Density based distributed clustering, in: Advances in Database Technology-EDBT 2004, Springer, 2004, pp. 88–105.
[19] Jingdong H. T. Shen, J. Song, J. Ji, Hashing for similarity search: A survey, 2014, arXiv:1408.2927.
[20] Message Passing Interface Forum, MPI: A message-passing interface standard, Tech. rep., University of Tennessee, Knoxville, USA, 1994.
[21] A. Morvan, A. Souloumiac, K. Choromanski, C. Gouy-Pailler, J. Atif, On the needs for rotations in hypercubic quantization hashing, 2018, arXiv:1802.03936.
[22] J.M. Santos, M.J. Embrechts, On the use of the adjusted rand index as a metric for evaluating supervised classification, in: C. Alippi, M.M. Polycarpou, C.G. Panayiotou, G. Ellinas (Eds.), Proceedings of the 19th International Conference on Artificial Neural Networks (ICANN), Springer, 2009, pp. 175–184.
[23] T. Sarazin, H. Azzag, M. Lebbah, SOM clustering using spark-mapreduce, in: 2014 IEEE International Parallel & Distributed Processing Symposium Workshops, Phoenix, AZ, USA, May 19–23, 2014, 2014, pp. 1727–1734.
[24] D. Schugk, A. Kummert, C. Nunn, Adaptation of the Mean Shift Tracking Algorithm To Monochrome Vision Systems for Pedestrian Tracking Based on HoG-Features, Tech. rep., SAE International, 2014.
[25] M. Slaney, M. Casey, Locality-sensitive hashing for finding nearest neighbors, in: IEEE Signal Proc. Mag. 2008, pp. 128–131.
[26] A. Ultsch, Clustering with SOM: U*C, in: Proceedings of the Workshop on Self-Organizing Maps, 2005, pp. 75–82.
[27] A. Vedaldi, S. Soatto, Quick shift and kernel methods for mode seeking, in: D. Forsyth, P. Torr, A. Zisserman (Eds.), Proceedings Part IV of the 10th European Conference on Computer Vision 2008, Marseille, France, 2008, pp. 705–718.
[28] K.-L. Wu, M.-S. Yang, Mean shift-based clustering, Pattern Recognit. 40 (2007) 3035–3052.
[29] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M.J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in: Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25–27, 2012, 2012, pp. 15–28.
[30] W. Zhao, H. Ma, Q. He, Parallel k-means clustering based on mapreduce, in: Cloud Comput., Springer, 2009, pp. 674–679.



Gael Beck is currently Ph.D researcher at LIPN at Paris 13 university and Data scientist at Kameloon. He received his engineer degree in 2015 and his M.S. degree in Dauphine university, France in 2015. His research interests are in the areas of Scalable machine learning and data mining with emphasis in clustering and visualization. Find more information on https://github.com/beckgael.

Tarn Duong began his undergraduate studies (1995–1998) at the University of Western Australia, Perth, with majors in statistics and applied mathematics, and a minor in computer science. Upon graduation with First Class honors, he was employed at the Australian Bureau of Statistics in the Canberra and Sydney offices (1998–2000). After this period in the Australian public service, Dr DUONG undertook his doctoral studies in computational statistics (2001–2004) at the University of Western Australia. He analyzed the theoretical properties of optimal smoothing algorithms for non-parametric estimation of probability density functions and implemented them in an R package suitable for large scale data sets. Find more information on http://www.mvstat.net/tduong/.

Mustapha Lebbah is currently Associate Professor at the University of Paris 13 and a member of Machine learning Team A3, LIPN. His main researches are centered on machine learning (Unsupervised learning, mixture model, cluster analysis, scalable machine learning big data and data science). Graduated from USTO University where he received his engineer diploma in 1998. Thereafter, he gained an MSC (DEA) in Artificial Intelligence from the Paris 13 University in 1999. In 2003, after three year in RENAULT RD, he received his PhD degree in Computer Science from the University of Versailles. He received the "Habilitation Diriger des Recherches" (accreditation to lead research) degree in Computer Science from Paris 13 University in 2012. He is a member of the french group in "complex data mining", and Secretary for the French Classification Society since November 2012. Mustapha Lebbah is qualified by the French National Council of Universities in Computer Science (section CNU 27) and Statistic (Section CNU 26). Since 2009 he obtained French Science Excellence Award (prime d'encadrement doctoral et de recherche(PEDR). Find more information on https://sites.google.com/site/lebbah/.

Hanane Azzag is currently associate professor at the University of Paris 13 (France) and a member of machine learning team A3 in LIPN Laboratory. Her main research is in biomimetic algorithms, machine learning and visual data mining. Graduated from USTHB University where she received her engineer diploma in 2001. Thereafter, in 2002 she gained an MSC (DEA) in Artificial Intelligence from Tours University. In 2005, after three years Tours Lab, she received her PhD degree in Computer Science from the University of Tours.

Christophe Cérin has been a professor of computer science at the University of Paris 13, France since 2005. In 2015, he was involved with an infrastructure project related to big data and high performance computing for e-sciences for USPC (Université Sorbonne Paris Cité). At Paris13, he chairs the board for the cluster computing facility available to all campus scientists and he chaired (2011–2016) the 'Expert Committee' in charge of recruiting and mentoring full time junior and senior professors in computer science. His current industrial experience includes serving as local chair for the Wolphin project (Alterway, Gandhi, Objectif Libre and Paris 6). His recent industrial experience was for the Wendelin and Resilience projects related to Cloud and Big-Data. He is also active with RSU (industrial demonstrator for sustainable cities) and with Qarnot Computing. His research focuses on High Performance Computing, including Grid and Cloud Computing and he develops middleware, algorithms, tools and methods for distributed systems. Find more information on http://lipn.univ-paris13.fr/cerin/.