Dirichlet Process Mixture Models made Scalable and Effective by means of Massive Distribution

Khadidja Meguelati, Bénédicte Fontez, Nadine Hilgert, Florent Masseglia

Dirichlet Process Mixture Models made Scalable and Effective by means of Massive Distribution

Khadidja Meguelati
Inria and LIRMM
Montpellier, France
khadidja.meguelati@inria.fr

Benedicte Fontez
MISTEA, Montpellier SupAgro, Univ.Montpellier
Montpellier, France
benedicte.fontez@supagro.fr

Nadine Hilgert
MISTEA, INRA, Univ.Montpellier
Montpellier, France
nadine.hilgert@inra.fr

Florent Masseglia
Inria and LIRMM
Montpellier, France
florent.masseglia@inria.fr

ABSTRACT

Clustering with accurate results have become a topic of high interest. Dirichlet Process Mixture (DPM) is a model used for clustering with the advantage of discovering the number of clusters automatically and offering nice properties like, *e.g.*, its potential convergence to the actual clusters in the data. These advantages come at the price of prohibitive response times, which impairs its adoption and makes centralized DPM approaches inefficient. We propose DC-DPM, a parallel clustering solution that gracefully scales to millions of data points while remaining DPM compliant, which is the challenge of distributing this process. Our experiments, on both synthetic and real world data, illustrate the high performance of our approach on millions of data points. The centralized algorithm does not scale and has its limit on 100K data points, where it needs more than 7 hours. In this case, our approach needs less than 30 seconds.

KEYWORDS

Dirichlet Process Mixture Model, Clustering, Parallelism

ACM Reference Format:
Khadidja Meguelati, Benedicte Fontez, Nadine Hilgert, and Florent Masseglia. 2019. Dirichlet Process Mixture Models made Scalable and Effective by means of Massive Distribution. In *The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), April 8–12, 2019, Limassol, Cyprus.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3297280.3297327

1 INTRODUCTION

Clustering, or cluster analysis, is the task of grouping similar data into the same cluster and separating dissimilar data in different clusters. It is a data mining technique and is intensively used for data analytics, with applications to marketing [2], security [13], or sciences like astronomy [21], and many more. Clustering may be used for identification in the new challenge of high throughput plant phenotyping [17], a research field with the purpose of crop improvement in response to present and future demographic and climate scenarios.

SAC '19, April 8–12, 2019, Limassol, Cyprus

ACM ISBN 978-1-4503-5933-7/19/04…$15.00
https://doi.org/10.1145/3297280.3297327

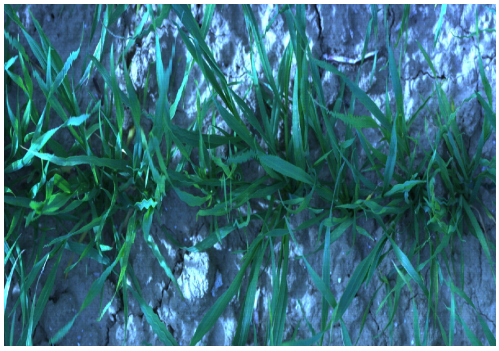


Figure 1: Durum in an experimental field. RGB image.

In this case, data to be considered include data on plants and crop images, like the one illustrated by Figure 1, showing a view of a Durum crop. Automatic identification, from such images, of leaves, soil, and distinguishing plants from foreground, are of high value for experts since they provide the fundamental information used for popular supervised methods in the domain [17]. One of the main difficulties, for clustering, is the fact that we don't know, in advance, the number of clusters to be discovered. To help performing cluster analysis, despite the unknown tackled number of clusters, statistics advocate for:

(1) Setting a number of clustering runs, with varying value of K, and selecting the one that minimizes a goodness of fit criteria. It may be a quadratic risk or the Residual Mean Squared Error of Prediction (RMSEP) [14]. This approach needs the implementation of a cross-validation algorithm [14]. The clustering approach in this case, may be a mixture model with an Expectation-Maximization (EM) algorithm [6], or K-means [14], for instance.
(2) Making a hierarchical clustering and then cut off the tree at a given depth, usually decided by the end-user. Different approaches for pruning with advantages and drawbacks exist, see [14].
(3) Using a Dirichlet Process Mixture (DPM) which automatically detects the number of clusters [8].

In this work, we focus on the DPM approach since it allows estimating the number of clusters and assigning observations to clusters, in the same process. Furthermore, its implementation is quite straightforward in a Bayesian framework. Such properties of DPM make it a very appealing solution for many use-cases. Unfortunately, DPM is

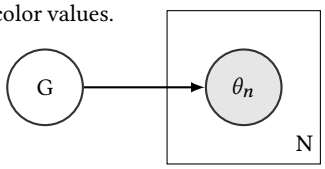
highly time consuming. Consequently, several attempts have been done to make it distributed [16, 25, 26]. However, while being effectively distributed, these approaches usually suffer from convergence issues (imbalanced data distribution on computing nodes) [10, 16, 26] or do not fully benefit from DPM properties [25] (see our discussion in Section 3). Furthermore, making DPM parallel is not straightforward since it must compare each record to the set of existing clusters, a highly repeated number of times. That impairs the global performances of the approach in parallel, since comparing all the records to all the clusters would call for a high number of communications and make the process impracticable. Our goal is to propose a parallel DPM approach that fully exploits parallel architectures for better performances and offers meaningful results. Our main contribution is to keep consistency of clusters among worker nodes, and between the worker and the master nodes with regards to DPM properties. Our motivating example comes from the biology use-case described above, where the processing time on one image may go up to several days in a centralized environment. These performances in response time are the main reason why DPM is not used in the domain. The results are very informative for experts, but the processing times are prohibitive. In this case, parallelization is an appealing solution but it has to guarantee that results remain as informative as the ones targeted by a centralized run. We propose DC-DPM (Distributed Clustering by Dirichlet Process Mixture), a distributed DPM algorithm that allows each node to have a view on the local results of all the other nodes, while avoiding exhaustive data exchanges. The main novelty of our work is to propose a model and its estimation at the master level by exploiting the sufficient statistics from the workers, in a DPM compliant approach. Our solution takes advantage of the computing power of distributed systems by using parallel frameworks such as MapReduce or Spark [27]. Our DC-DPM solution distributes the Dirichlet Process by identifying local clusters on the workers and synchronizing these clusters on the master. These clusters are then communicated as a basis among workers for local clustering consistency. We modified the Dirichlet Process to consider this basis in each worker. By iterating this process we seek global consistency of DPM in a distributed environment. Our experiments, using real and synthetic datasets, illustrate both the high efficiency and linear scalability of our approach. We report significant gains in response time, compared to centralized DPM approaches, with processing times of a few minutes, compared to several days in the centralized case. The paper is organized as follows. In Section 2 we state the problem and give the necessary background on Dirichlet Process Mixture. In Section 3 we discuss related work and in Section 4, we describe the details of our distributed solution for clutering by means of Dirichlet Process Mixture. Section 5 reports the results of our experimental evaluation to verify the efficiency and effectiveness of our approach, and Section 6 concludes.

2 PROBLEM DEFINITION AND BACKGROUND

2.1 Dirichlet Process Mixture Models

A Dirichlet Process (DP) is a probability distribution over distributions. In our use-case, a distribution over the image pixels could be "plant" with probability p_1, and "not plant" with probability p_2, with the property that $p_1+p_2=1$. A DP generates a probability distribution G. We observe a sample $\theta_1,...,\theta_N$ from G. In our use-case, each θ_i is the vector of possible pixel color values.

$$\begin{aligned} \theta_n \mid G &\overset{iid}{\sim} G, n=1,...,N \\ G &\sim DP(\alpha, G_0) \end{aligned}$$



Where G is by construction a discrete probability distribution [22] .

$$G = \sum_{i=1}^{\infty} \pi_i(\mathbf{v}) \delta_{\phi_i}$$

Therefore, observed variables θ_n have a non null probability of having the same value ϕ_i and this allows for clustering. In our use-case, "plant" pixels will have the same color vector ϕ expressing the green value. Clustering is very sensitive to the DP parameters given by the end user. G_0 is a continuous probability distribution from which the $(\phi_i)_{i \in N}$ are initially drawn. In our use-case, G_0 gives the color probability of all possible clusters in the image.

$$\phi_1,...,\phi_i,... \sim G_0$$

α is a scale parameter ($\alpha > 0$) which tunes the probability weights $\pi_i(\mathbf{v})$.

$$V_1,...,V_i,... \sim Beta(1,\alpha) \qquad \pi_i(\mathbf{v}) = v_i \prod_{j=1}^{i-1} (1-v_j)$$

α tunes indirectly the probability mass function for k_N, the number of unique values (namely ϕ_i) in a sample of size N [3].

$$p(k_N) = \left|S_{N,k_N}\right| N! \alpha^{k_N} \frac{\Gamma(\alpha)}{\Gamma(\alpha+N)} \tag{1}$$

where $\left|S_{N,k_N}\right|$ is the unsigned Stirling number of the first kind.

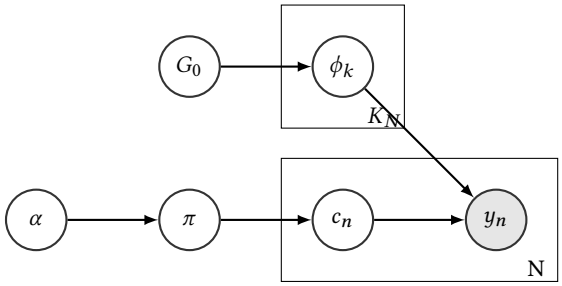
With a Dirichlet Process Mixture we observe the sample $y_1,...,y_N$ from a mixture of distributions $F(\theta_n)$. In our use-case, we assume that colors are observed with a noise distributed according to F. The mixture is controlled by a DP on the parameters θ_n.

$$\begin{aligned} y_n &\sim F(\theta_n), n=1,...,N \\ \theta_n &\sim G \\ G &\sim DP(\alpha_0, G_0) \end{aligned}$$

In a Bayesian framework, the estimation of θ_n is done on the posterior: $P(\theta_1,...,\theta_N \mid y_1,...,y_N)$. Instead of this representation, another parameterization is used to speed up computation of the posterior:

$$P(\phi_{c_1},...,\phi_{c_N} \mid y_1,...,y_N)$$

Where $\theta_n = \phi_{c_n}$, c_n is the cluster label of observation n, and ϕ_{c_n} is the unique value of the θ_n belonging to the same cluster. The Gibbs algorithm [11] samples the cluster labels $c_1,...,c_N$ and next the cluster parameters (here ϕ_c, for all $c \in \{1,...,K\}$ where K designs the number of cluster label values) instead of $\theta_1,...,\theta_N$.



Several versions of a Gibbs sampling are proposed by Neal in [19] to simulate values from the posterior. The principle is to repeat the following loops at least until convergence to the posterior:

(1) Cluster assignment, for $n = 1,\ldots,N$
- Remove observation n from its cluster. Check if the cluster is empty, if yes then remove the cluster and ϕ_{c_n} from the list $\{\phi\}$ of all possible values.
- Draw c_n from

$$P \quad (c_n = c \mid \{c_j\}_{j\neq n}, y_n, \{\phi\}) \propto \begin{cases} \frac{\#(c)}{N-1+\alpha} F(y_n \mid \phi_c) & \text{existing cluster} \\ \frac{\alpha}{N-1+\alpha} \int F(y_n \mid \phi) dG_0(\phi) & \text{new cluster} \end{cases}$$

  Where $\#(c)$ designs the number of observations affected to cluster c (after removing observation n from the sample).
- If c designs a new cluster, need to draw ϕ_c from $P(\phi \mid y_n) \propto F(y_n \mid \phi) G_0(\phi)$
- End loops

(2) Update of $\{\phi\}$,
- draw ϕ_c from the posterior distribution of cluster c, $P(\phi \mid \{y\}_c)$ (which is proportional to the product of the prior G_0 and the likelihood of all observations affected to cluster c).

When distribution F and G_0 are conjugates, ϕ can be integrated out from the Gibbs sampling which becomes time-efficient (no need to update $\{\phi\}$). Then

$$P \quad (c_n = c \mid \{c_j\}_{j\neq n}, y_n, \{\phi\}) \propto \begin{cases} \frac{\#(c)}{N-1+\alpha} \int F(y_n \mid \phi) dP(\phi \mid \{y\}_c) & \text{existing cluster} \\ \frac{\alpha}{N-1+\alpha} \int F(y_n \mid \phi) dG_0(\phi) & \text{new cluster} \end{cases}$$

2.2 Massive Distribution and Spark

Clustering via Dirichlet Process Mixture based on Gibbs Sampling is unable to scale to large datasets due to its high computational costs associated with Bayesian inference. For this reason, we aim to implement a parallel algorithm for DPM clustering in a massively distributed environment called Spark which is a parallel programming framework aiming to efficiently process large datasets. This programming model can perform analytics with in-memory techniques to overcome disk bottlenecks. Similar to MapReduce [4], Spark can be deployed on the Hadoop Distributed File System (HDFS) [23]. Unlike traditional in-memory systems, the main feature of Spark is its distributed memory abstraction, called resilient distributed datasets (*RDD*), that is an efficient and fault-tolerant abstraction for distributing data in a cluster. With RDD, the data can be easily persisted in main memory as well as on the hard drive. Spark is designed to support the execution of iterative algorithms.

To execute a Spark job, we need a master node to coordinate job execution, and some worker nodes to execute a parallel operation. These parallel operations are summarized to two types: (i) Transformations: to create a new RDD from an existing one (*e.g.*, Map, MapToPair, MapPartition, FlatMap); and (ii) Actions: to return a final value to the user (*e.g.*, Reduce, Aggregate or Count).

2.3 Problem Definition

The problem we address is as follows. Given a (potentially big) dataset of *records* find, by means of process performed in parallel, a partition of the dataset into disjoint subsets called clusters, such that:

- Similar records are assigned to the same cluster.
- Dissimilar records are assigned to different clusters.
- The union of the clusters is the original dataset.

3 RELATED WORK

We set our work in the context of parallel clustering. Previous works for distributed algorithms of unsupervised clustering already exist. Ene et al. [7] gave a MapReduce algorithm for the k-center and k-median problems. Both algorithms use Iterative-Sample as a subprocedure to get a substantially smaller subset of points that represents all of the points well. To achieve this, they perform an iterative-Sample. However these algorithms require the number of clusters k to be specified in advance, which is considered as one of the most dificult problems to solve in data clustering. Debatty et al. [5] proposed a MapReduce implementation of G-means which is an iterative algorithm that uses Anderson Darling test to verify if a subset of data follows a Gaussian distribution. However this algorithm overestimates the number of clusters, and then requires a post-processing step to merge clusters.

In our work, we focused on algorithms inspired by the DPM. Lovell et al. [16] and Williamson et al. [26] has suggested an alternative parametrisation for the Dirichlet process in order to derive non-approximate parallel MCMC inference for it, these approaches are criticized by Gal and Ghahramani in [10], these latter showed that the approaches suggested are impractical due to an extremely imbalanced distribution of the data, and gave directions for future research like the development of better approximate parallel inference. The main idea when data is distributed is to perform a DPM in each worker. The issues are then to share information between workers, and to synchronize and update clusters arising from workers at the master level. For synchronization, the main challenge is a problem of identification and of label switching of clusters. In this context we can use a relabelling algorithm like for example the one proposed by Stephens [15] for mixture models. For parallel Latent Dirichlet Allocation (LDA) and Hierarchical Dirichlet Process (HDP), Newman et al. [20] suggested to measure distance between clusters and then proposed a greedy matching. Wang and Lin [25] gave a detailed review of literature and recent advanced in this topic before giving a new proposal. They proposed to use a stepwise hierarchical classification at the master level with half chance for split or merge at each step. They began with a full model considering all clusters from all workers as different components of the model. Their algorithm uses the standard Bayes Factor to compare nested models and choose the best split or merge. In conclusion, at the master level, the proposed algorithms diverge from a DPM-classifier and are not a scalable estimations of a DPM. Moreover, Wang and Lin [25] used a fixed value for the scale parameter (α) in their implementation of the DPM at the workers level. The number of final clusters is related to this value (see equation 1). Authors like Miller and Harrison [18] have demonstrated the inconsistency for the number of components of a DPM model with fixed α value. If the number of components identified at the worker level is underestimated, then the number

of clusters at the master level might be underestimated. The reverse will increase considerably the running time at the master level.

In our approach, we suggest to keep to a DPM algorithm as much as possible, even at the master level, to be close to the good properties of a DPM-classifier, despite the fact that data is distributed. We also suggest a modification of the DPM model to share information among workers. In this way we expect to improve our clustering (better estimation) and suppress label switching. Finally, we do not fix a value to α but allow a different estimation in each worker to add flexibility to our model.

Furthermore [25] is restricted to specific cases where noise in the observations follows the conjugate distribution of the cluster centers distribution. For example, a Gaussian noise imposes a Gaussian distribution of the centers. Therefore, this method is not suited for centers having positive values only.

Our goal is to work on any data, even with exclusively positive centers.

4 DC-DPM: DISTRIBUTED CLUSTERING VIA DPM

In this section, we present a novel parallel clustering approach called DC-DPM, adapted for independent data. Parallelization calls for particular attention to two main issues. The first one is the load balance between computing nodes. In our approach we distribute data evenly across the different nodes, and there is no data exchange between nodes during the processing. The second issue is the cost of communications. In order to be efficient, nodes send and receive as few information as possible by performing many iterations of Gibbs Sampling independently in each worker before synchronizing the global state at the master level and only communicating sufficient statistics between workers and master. The challenge of using sufficient statistics, in a distributed environment, is to remain in the DPM approach at all steps, including the synchronization between the worker and master nodes. The novelty of our approach is to approximate the DPM model even at the master level when local data is replaced by sufficient statistics between iterations.

4.1 Architecture and Distributed Algorithm

Data Distribution

Data is evenly distributed on the computing nodes when the process starts. This is a mere, sequential, distribution, that splits the dataset into equal sized partitions.

Worker

This level handles the innovation parts of DPM (detection of new clusters) and the individual cluster assignment in each worker. The updates of the cluster labels in worker j depend on sample size proportions of the distributed data:

$$P(c_{n,j}=c \mid c_{\neq n,j}, y_{n,j}, \{\phi\}) \propto \begin{cases} \frac{\#(c)_j}{N_j-1+\alpha} F(y_{n,j},\phi_c), c=1,\ldots,K \\ \frac{\alpha}{N_j-1+\alpha} \int F(y_{n,j},\phi) dG_0(\phi) \text{ new} \end{cases}$$

As the clusters are not known at the beginning, we cannot ensure that the sample size proportions of each cluster will be respected in each worker. If the data were uniformly distributed, each cluster would have only, in average, the same weight/proportion in all workers. Therefore we added a modification of the update, as can be seen in Algorithm 1.

Algorithm 1 DPM at worker level

for each data y_n **do**
Draw $c_{n,j}$ from $P(c_{n,j}=c \mid \{c_{l,j}\}_{l\neq n}, y_{n,j}, \{\phi\}, \{w\}, \alpha_j) \propto$

$$\begin{cases} \frac{\#(c)+\alpha_j w_c}{N_j-1+\alpha_j} F(y_{n,j},\phi_c), c=1,\ldots,K \\ \frac{\alpha_j w_u}{N_j-1+\alpha_j} \int F(y_{n,j},\phi) dG_0(\phi) \text{ new} \end{cases}$$

Update of α_j
Draw ϕ_c for new clusters

where the weight w_c is the proportion of observations from cluster c evaluated on the whole dataset and w_u the proportion of non affected observations (awaiting the creation, innovation, discover of their real clusters). Therefore, these parameters are updated at the master level during the synchronization. Now, the scale parameter α_j can be viewed as a tuning parameter between local (worker) and global (master) proportions. Following [9] we use an inverse gamma prior to infer this parameter.

This modification of the update implies a slightly modified DPM in each worker j :

$$\begin{aligned} y_{n,j} &\sim F(\theta_{n,j}) \\ \theta_{n,j} &\sim G_j \\ G_j &\sim DP(\alpha_j, G) \\ \alpha_j &\sim IG(a,b) \\ G &= \sum_{c=1}^{K} w_c \delta_{\phi_c} + w_u G_0, \\ &\quad \text{with } w_u + \sum_{c=1}^{K} w_c = 1 \end{aligned}$$

Master

This level handles the final individual assignment in the master node and therefore the final common number of clusters K. The master gets from each worker the following input: sample size of cluster k in worker j ($n_{j,k}$), cluster parameter values-sufficient statistics, individual predictive value (traditionally/usually the cluster mean value in the worker: $\hat{y}_{n,j} = \bar{y}_{j,c_{n,j}=k}$). At the master level, the observations are assigned by clusters. A cluster corresponds to a set of individuals belonging to the same cluster of the same worker. Each cluster has a representative or individual predictive value which is used to perform the end of the Gibbs sampling at the master level:

$$P(c_{n,j}=c \mid c_{\neq n,j}) \propto \begin{cases} \frac{\#(c)}{N-1+\gamma} F(\hat{y}_{n,j},\phi_c), c=1,\ldots,K \\ \frac{\gamma}{N-1+\gamma} \int F(\hat{y}_{n,j},\phi) dG_0(\phi) \text{ new} \end{cases}$$

Working at an individual level implies a slow Gibbs sampling with poor mixing [12]. So, we suggest an update by clusters. In this view, we denote $z_{j,k}$ the master label of the cluster k in worker j. To take into account the worker information ($\{\phi_k^{\text{worker}j}\}$), we replace the

prior predictive distribution ($\int F(y_{n,j},\phi)dG_0(\phi)$) by a posterior predictive distribution. Eventually, we use the cluster mean value ($\bar{y}_{j,k}$) as an individual predictive value:

$$P(z_{j,k}=c \mid z_{\neq j,k}) \propto \begin{cases} \frac{\#(c)}{N-1+\gamma} F(\bar{y}_{j,k},\phi_c), c=1,\ldots,K \\ \frac{\gamma}{N-1+\gamma} \int F(\bar{y}_{j,k},\phi) dG(\phi \mid \phi_k^{\text{worker}\,j}) \end{cases}$$

The labels $\{c_{n,j}\}$ of all the observations in the cluster k of worker j are then assigned to the master label $z_{j,k}$. Next, the cluster parameters ($\{\phi_c\}_{c=1,\ldots,K}$) are updated from the posterior computed on the whole dataset. We assume that we don't need all the data but only sufficient statistics from all clusters from all workers to compute the posterior. This assumption is straightforward for many distributions, as the exponential family [1]. Last, the synchronization of the workers is done through the definition of G using the updated parameters ($\{\phi_c\}_{c=1,\ldots,K}$) and with weights drawn from a Dirichlet distribution $Dir(n_1,\ldots,n_K,\gamma)$. The end user parameters of this Dirichlet distribution are updated at the master level from the whole dataset. The size n_k is the sum of all observations having label k at the end of the master Gibbs sampling.

$$\begin{aligned} (w_1,\ldots,w_K,w_u) &\sim Dir(n_1,\ldots,n_K,\gamma) \\ \gamma &\sim IG(c,d) \end{aligned}$$

By doing so, we do not have to consider label switching. Clusters are explicitly defined at the master level and parameter values are not updated in the worker. At the worker level, only innovation (creation of new clusters) is implemented. This is summarized by Algorithm 2.

Algorithm 2 DPM at master level

for each (j,k) **do**

Draw $z_{j,k}$ from $P(z_{j,k}=c \mid \{c\}_{\neq j,k},\bar{y}_{j,k},\{\phi\},\gamma) \propto$

$$\begin{cases} \frac{\#(c)}{N-1+\gamma} F(\bar{y}_{j,k},\phi_c), c=1,\ldots,K \\ \frac{\gamma}{N-1+\gamma} \int F(\bar{y}_{j,k},\phi) dG(\phi \mid \phi_k^{\text{worker}\,j}) \end{cases}$$

Update of ϕ and $(w_1,\ldots,w_K,w_u)$

The workflow of our DC-DPM approach is illustrated by Figure 2. It consists in 4 steps:

(1) Identify local new clusters in the workers
(2) Compute and send sufficient statistics and cluster sizes from each worker to the master
(3) Synchronize and estimate cluster labels from sufficient statistics
(4) Send updated cluster parameters and cluster sizes from master to workers

Our first proposition concerns the synchronization and estimation of the DPM. It is done with a Gibbs sampling conditionally on the sufficient statistics instead of the whole dataset/individual observations. Our second proposition is a construction of a shared prior distribution updated at the master level and send to the workers' DPM. This distribution reflects the information/results collected from all workers and synchronized at the master.

The likelihood for one observation y_i from the Exponential family is:

$$F(y_i \mid \eta) = h(y_i) exp\left(\eta^T \psi(y_i) - a(\eta)\right)$$

where :

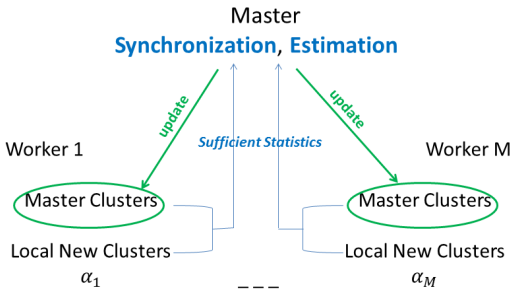


Figure 2: Diagram/workflow of the DC - DPM

- η are the natural parameters and is a function of ϕ.
- $\psi(y_i)$ are sufficient statistics
- $a(\eta)$ is the Log-normalizing factor or Log-partition can be expressed in function of $\phi : a(\eta(\phi))$
- $h(y_i)$ is the base measure

and the likelihood for all the observations is:

$$F(y_1,\ldots,y_n \mid \eta) = \left(\prod_{i=1}^{N} h(y_i)\right) exp\left(\eta^T \left(\sum_{i=1}^{N} \psi(y_i)\right) - N a(\eta)\right)$$

Among all the distributions included in the Exponential Family, we implemented the Normal case for the experiments: $F(. \mid \phi_c) = N(\phi_c,\Sigma_1)$. This choice corresponds to the simple linear model $y_n = \phi_c + \varepsilon_n$ and ε_n is Normally distributed $N(0,\Sigma_1)$. In this case, the sample mean of cluster c, namely $\bar{y}_c$ is a sufficient statistic and the posterior distribution can be conditioned only on its value:

$$P(\phi \mid \{y_n\}_{c_n=c}) = P(\phi \mid \bar{y}_c) \propto F(\bar{y}_c \mid \phi) G_0(\phi)$$

When G_0 is not a conjugate prior (*e.g.*, a normal distribution), the posterior distribution is not a usual one but a value from this posterior can be simulated with a Metropolis Hasting (MH) within Gibbs algorithm.

When variances are known and G_0 is a conjugate prior (normal distribution $N(m,\Sigma_1)$), there is no use of MH algorithm. The posterior is a normal distribution $N(\phi_c^{post} = \Sigma(\#(c)\Sigma_2^{-1}\bar{y}_c + \Sigma_1^{-1}m), \Sigma_c^{post})$ where $\Sigma_c^{post} = (\#(c)\Sigma_2^{-1} + \Sigma_1^{-1})^{-1}$. The predictive posterior is a normal distribution $N(\phi_c^{post}, \Sigma_2 + \Sigma_c^{post})$. In our context, Σ_c^{post} was considered negligible and the mean value ϕ_c^{post} was replaced by an individual drawn from the posterior.

5 EXPERIMENTS

The parallel experimental evaluation was conducted on a cluster of 32 machines, each operated by Linux, with 64 Gigabytes of main memory, Intel Xeon CPU with 8 cores and 250 Gigabytes hard disk. The centralized approach is an implementation DPM in Scala, and was executed on a single machine with the same characteristics.

The distributed algorithm we proposed is an approximation of a classic DPM, we will compare its properties to a centralized DPM implementation, on synthetic data and also in our use-case for digital agronomy. The first step of our process is a distributed K-means that sets the initial state (usually we set K to be one tenth of the dataset size).

Reproducibility : All our experiments are fully reproducible. We make our code and data available at https://github.com/khadidjaM/DC-DPM.

In the rest of this section, we describe the datasets in Section 5.1 and our evaluation criteria in Section 5.2. Then, in Section 5.3, we measure the performances, in response time, of our approach compared to the centralized approach and also by reporting its scalability and speed-up. We evaluate the clusters obtained by DC-DPM in the case of real and synthetic dataset in Section 5.4 and Section 5.5 discusses the results and interest of our work in a real use-case of agronomy.

5.1 Datasets

We carried out our experiments on a real world and a synthetic dataset.

Our synthetic data are generated using a two-steps principle. In the first step we generate cluster centers according to a multivariate normal distribution with the same variance σ_1^2 for all dimensions. In the second step, we generate the data corresponding to each center, by using a multivariate normal distribution parameterized on the center with the same variance σ_2^2 for all dimensions. We generated a first batch of 5 datasets having size 20K, 40, 60, 80K and 100K with $\sigma_1^2 = 1000$ and $\sigma_2^2 = 1$. They represent 10 clusters. We generated a second batch of 5 datasets having size 2M, 4M, 6M, 8M and 10M with $\sigma_1^2 = 100000$ and $\sigma_2^2 = 10$. They represent 100 clusters. This type of generator is widely used in statistics, where methods are evaluated first on synthetic data before being applied on real data.

Our real data correspond to the use-case described in Section 1. The image used to test our algorithm was in RGB format. After preprocessing it contains 1,081,200 data points, described by a vector of 3 values (red, green and blue) belonging to [0,1].

5.2 Clustering Evaluation Criteria

There are two cases for evaluating the results of a clustering algorithm. Either there is a ground truth available, or there is not. In the case of an available ground truth, there are measures allowing to compare the clustering results to the reference, such as ARI, described below, for instance. This is usually exploited for experiments when one wants to check performances in a controlled environment, on synthetic data or labelled real data. In the case where there is no ground-truth (which is the usual case, because we don't know what should be discovered in real world applications of a clustering algorithm) the results may be evaluated by means of relative measures, like RSS, described below, for instance. In our experiments, we chose the following three criteria.

1. The Adjusted Rand Index (ARI): it is the corrected-for-chance version of the Rand Index [24], which is a function that measures the similarity between two data clustering results, for example between the ground truth class assignments (if known) and the clustering algorithm assignments. ARI values are in the range [-1,1] with a best value of 1.

2. The residual sum of squares (RSS): it is a measure of how well the centroids (means) represent the members of their clusters. It is the squared distance of each data from its centroid summed over all vectors. In the univariate case, the RSS value divided by the number of observations gives the value of the Mean Squared Error (MSE), an estimator of the residual variance. In multivariate dataset with independent variables, the RSS value divided by the number of observations gives an estimator of the sum of the variable variances. This sum represents its lower bound and also the best value to be observed in the clustering of synthetic data. To simplify, we give in the following the result of the RSS value divided by the number of data N and the variance. Therefore the lower bound is known and should be equal to the number of variables (for example 2 for our synthetic data).

3. K, the number of discovered clusters.

5.3 Response Time

In this section we measure the clustering time in DC-DPM and compare it to the centralized approach. Figure 3 reports the response times of DC-DPM and the centralized approach on our synthetic data, limited to 100K data points. Actually, the centralized approach does not scale and would take several days for larger datasets.The distributed approach is run on a cluster of 8 nodes. The results reported by Figure 3 are in logarithmic scale. The clustering time increases with the number of data points for all approaches. This time is much lower in the case of DC-DPM, than the centralized approach. On 8 machines (64 cores) and for a dataset of 100K data points, DC-DPM performs the clustering in 24 seconds, while the centralized approach needs more than 7 hours on a single machine.

Figure 4 reports an extended view on the clustering time, only for DC-DPM, and with a dataset having up to 10 million data points from the synthetic dataset. DC-DPM is run on a cluster of 16 machines. The running time increases with the number of data points. Let us note that the centralized approach does not scale and cannot execute on such dataset size. DC-DPM enjoys linear scalability with the dataset size.

Figures 5 and 6 illustrate the parallel speed-up of our approach on 2M data points from the synthetic dataset and on more than 1 million data points obtained after preprocessing the image of our use-case. The results show optimal or near optimal gain. In Figure 6 we observe that the response time for 2 nodes is more than twice the response time for 4 nodes. That is unexpected when measuring a speed-up. However, the response times of our approach are very fast (a few minutes) and do not consider the time it takes for Spark to load-up, before running DC-DPM. The slight difference between an optimal speed-up and the results reported in Figure 6 are due to that loading time.

5.4 Clustering Evaluation

In the following experiments, we evaluate the clustering performance of DC-DPM and compare it to the centralized approach.

Table 1 reports 1) the ARI value computed between the clustering obtained and the ground truth, 2) the RSS value divided by the number of data N and variance (σ_2^2), and 3) the number of clusters, obtained with DC-DPM and the centralized approach on our synthetic data while increasing the dataset size. The DC-DPM is run on a cluster of 8 nodes. DC-DPM performs as well as the centralized approach, there is a small gap in RSS values which is negligible compared to the gained time.

Table 2 reports an extended view on the ARI value, and the RSS value divided by the number of data N and by the variance (σ_2^2), and number of clusters obtained with DC-DPM, while increasing dataset size (up to 10 million data points). DC-DPM is run on a cluster of 16 machines. The performance keeps showing the maximum possible accuracy, even with a large number of data points.

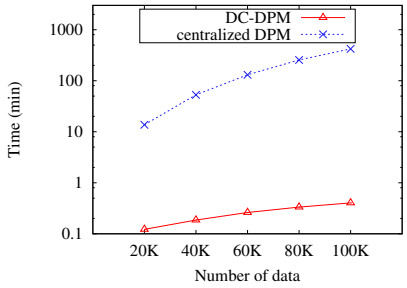


Figure 3: Logarithmic scale. Response time of DC-DPM and centralized DPM as a function of the dataset size.

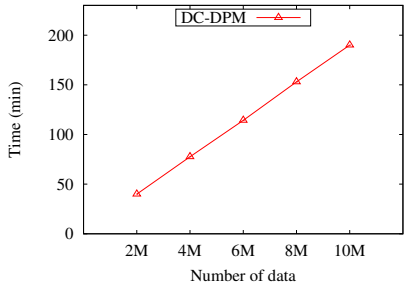


Figure 4: Response time (minutes) of DC-DPM as a function of the dataset size.

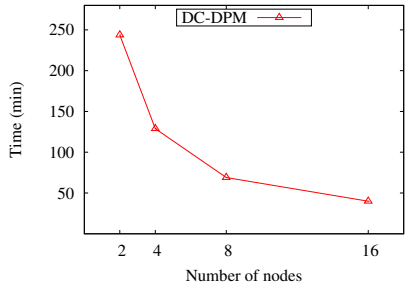


Figure 5: Clustering time as a function of the number of computing nodes on the synthetic data.

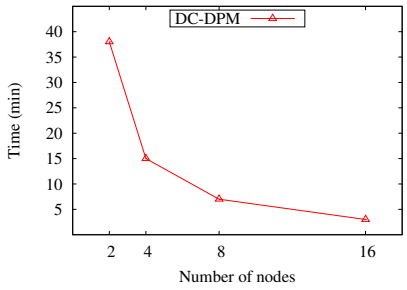


Figure 6: Clustering time as a function of the number of computing nodes on the image of our use-case.

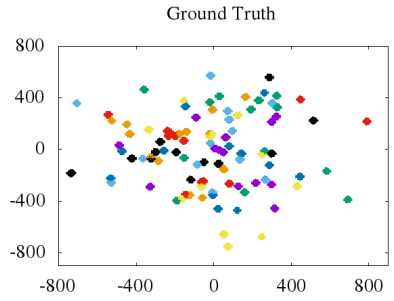


Figure 7: Visual representation of the synthetic dataset.

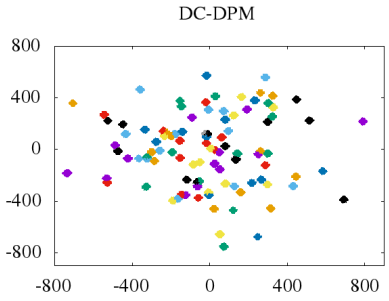


Figure 8: Visual representation of the results obtained by DC-DPM

Table 1: Clustering evaluation criteria obtained with the centralized DPM and with DC-DPM.

	Centralized DPM			DC-DPM		
	ARI	$\frac{RSS}{N \times \sigma_2^2}$	Clusters	ARI	$\frac{RSS}{N \times \sigma_2^2}$	Clusters
20K	1.00	2.01	10	1.00	2.04	10
40K	1.00	2.00	10	1.00	2.03	10
60K	1.00	2.00	10	1.00	2.02	10
80K	1.00	2.00	10	1.00	2.01	10
100K	1.00	2.00	10	1.00	2.02	10

Table 2: Clustering evaluation criteria obtained by DC-DPM .

	ARI	RSS/(N*σ_2^2)	Clusters
2M	1.00	2.00	102
4M	1.00	2.00	100
6M	1.00	2.00	100
8M	1.00	2.02	99
10M	1.00	2.10	101

Figure 7 gives a visual representation of our 4M data points synthetic dataset. Each cluster is assigned to a color. Our goal is to retrieve these clusters. Figure 8 represents the results obtained by our approach on the data of Figure 7, with 16 nodes. Each cluster is assigned to a different color. shows the performance of our approach with almost perfect results where the discovered clusters are the same as the actual ones from the data. This is confirmed by Table 2, line 2.

5.5 Use-case

Phenotyping and precision agriculture use more and more information from sensors and drones, like aerial images, leading to the emerging domain of digital agriculture (see for example http://ec.europa.eu/research/participants/portal/desktop/en/opportunities/h2020/topics/dt-rur-12-2018.html). An important challenge, in this context, is to be able to distinguish clusters of plants: status (normal, hydric stress, disease,...) or species for example. Clustering, applied to images, is a key in this domain.

We want to discover clusters in the image presented in Section 1 and transformed as described in Section 5.1. We set $\sigma_1^2 = 1$ because our data is in the range of [0,1]. For both parameter values of $\sigma_2^2 = 0.01$ and $\sigma_2^2 = 0.0025$, the clusters are extracted in approximately 3 minutes with DC-DPM running in parallel on 16 computing nodes. This is confirmed by Figure 6. The centralized approach does not scale on this data and we could not obtain results.

The number of clusters depends on the value of the variance error. A value of $\sigma_2^2 = 0.01$ gave a rough clustering with only $K = 3$ clusters. Those clusters identified the brightness in the RGB image (see figure 9). A lower value of $\sigma_2^2 = 0.0025$ gave a clustering with $k = 12$ clusters, which is enough to reconstruct the image (see figure 10). Depending on the aim of the clustering, different types of wavelength or data must be used for identification. The accuracy of

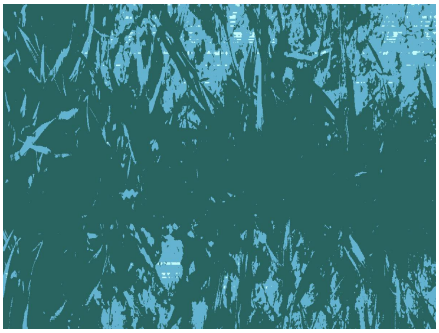


Figure 9: Clustering of the Durum image by DC-DPM with $\sigma^2 = 0.01$.

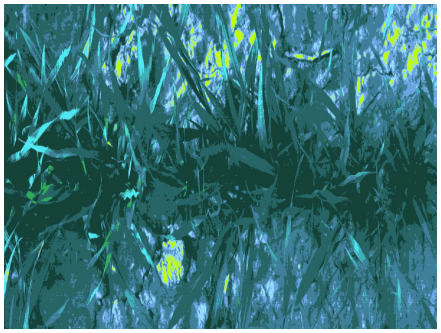


Figure 10: Clustering of the Durum image by DC-DPM with $\sigma_2^2 = 0.0025$.

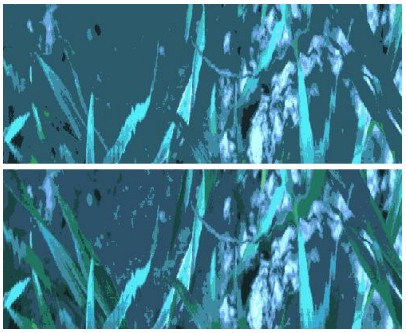


Figure 11: Clustering of a part of the Durum image by DC-DPM (top) and Centralized DPM (bottom).

the clustering (number of clusters) relies on the variance value (σ_2^2). Clustering is used to detect structures in the data (genetic, population, status) before processing. This group detection allows reducing data dimension and bias in further prediction analysis.

DC-DPM was compared to the centralized DPM on a part of the the Durum image in the experimental field. RGB Image with DC-DPM (top, 12 clusters) and centralized DPM (bottom, 17 clusters), $\sigma_2^2 = 0.0025$. The results were quite similar as shown in figure 11. The impact of σ^2 on the number of clusters varies for centralized and distributed approaches and may be adjusted by the end-user.

6 CONCLUSION

We proposed DC-DPM, a novel and efficient parallel solution to perform clustering via DPM on millions of data points. We evaluated the performance of our solution over real world and synthetic datasets. The experimental results illustrate the excellent performance of DC-DPM (*e.g.*, a clustering time of less than 30 seconds for 100K data points, while the centralized algorithm needs several hours). The results also illustrate the high performance of our approach with results that are comparable to the ones of the centralized version. Overall, the experimental results show that by using our parallel techniques, the clustering of very large volumes of data can now be done in small execution times, which are impossible to achieve using the centralized DPM approach. A nice perspective of our approach is to open fundamental research tracks such as DPM clustering on complex data like, *e.g.* time series.

7 ACKNOWLEDGEMENTS

The research leading to these results has received funds from the European Union's Horizon 2020 Framework Programme for Research and Innovation, under grant agreement No. 732051.

REFERENCES

[1] 2010. *Exponential Family.* Wiley-Blackwell, Chapter 18, 93–97. https://doi.org/10.1002/9780470627242.ch18 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470627242.ch18

[2] A. Alamsyah and B. Nurriz. 2017. Monte Carlo simulation and clustering for customer segmentation in business organization. In *2017 3rd International Conference on Science and Technology - Computer (ICST).* 104–109. https://doi.org/10.1109/ICSTC.2017.8011861

[3] Charles E. Antoniak. 1974. Mixtures of Dirichlet Processes with Applications to Bayesian Nonparametric Problems. *Ann. Statist.* 2, 6 (11 1974), 1152–1174. https://doi.org/10.1214/aos/1176342871

[4] J. Dean and S. Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[5] Thibault Debatty, Pietro Michiardi, Wim Mees, and Olivier Thonnard. 2014. Determining the k in k-means with MapReduce.. In *EDBT/ICDT Workshops*. 19–28.

[6] Arthur P Dempster, Nan M Laird, and Donald B Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological)* (1977), 1–38.

[7] Alina Ene, Sungjin Im, and Benjamin Moseley. 2011. Fast clustering using MapReduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 681–689.

[8] Michael D Escobar. 1994. Estimating normal means with a Dirichlet process prior. *J. Amer. Statist. Assoc.* 89, 425 (1994), 268–277.

[9] Michael D Escobar and Mike West. 1995. Bayesian density estimation and inference using mixtures. *Journal of the american statistical association* 90, 430 (1995), 577–588.

[10] Yarin Gal and Zoubin Ghahramani. 2014. Pitfalls in the use of parallel inference for the Dirichlet process. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*. 208–216.

[11] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. 2004. *Bayesian Data Analysis* (2nd ed. ed.). Chapman and Hall/CRC.

[12] Joseph Gonzalez, Yucheng Low, Arthur Gretton, and Carlos Guestrin. 2011. Parallel gibbs sampling: From colored fields to thin junction trees. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 324–332.

[13] Victoria Hodge and Jim Austin. 2004. A Survey of Outlier Detection Methodologies. *Artificial Intelligence Review* 22, 2 (01 Oct 2004), 85–126. https://doi.org/10.1023/B:AIRE.0000045502.10941.a9

[14] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An introduction to statistical learning*. Vol. 112. Springer.

[15] Ajay Jasra, Chris C Holmes, and David A Stephens. 2005. Markov chain Monte Carlo methods and the label switching problem in Bayesian mixture modeling. *Statist. Sci.* (2005), 50–67.

[16] Dan Lovell, Ryan P Adams, and VK Mansingka. 2012. Parallel markov chain monte carlo for dirichlet process mixtures. In *Workshop on Big Learning, NIPS*.

[17] Chuang Ma, Hao Helen Zhang, and Xiangfeng Wang. 2014. Machine learning for Big Data analytics in plants. *Trends in Plant Science* 19, 12 (1 12 2014), 798–808.

[18] Jeffrey W Miller and Matthew T Harrison. 2014. Inconsistency of Pitman-Yor process mixtures for the number of components. *The Journal of Machine Learning Research* 15, 1 (2014), 3333–3370.

[19] Radford M Neal. 2000. Markov chain sampling methods for Dirichlet process mixture models. *Journal of computational and graphical statistics* 9, 2 (2000), 249–265.

[20] David Newman, Arthur Asuncion, Padhraic Smyth, and Max Welling. 2009. Distributed algorithms for topic models. *Journal of Machine Learning Research* 10, Aug (2009), 1801–1828.

[21] Ordovás-Pascual, I. and Sánchez Almeida, J. 2014. A fast version of the k-means classification algorithm for astronomical applications. *Astronomy & Astrophysics* 565 (2014), A53. https://doi.org/10.1051/0004-6361/201423806

[22] Jayaram Sethuraman. 1994. A constructive definition of Dirichlet priors. *Statistica sinica* (1994), 639–650.

[23] J. Shafer, S. Rixner, and A. L. Cox. 2010. The Hadoop distributed filesystem: Balancing portability and performance. In *Int. ISPASS*.

[24] Nguyen Xuan Vinh, Julien Epps, and James Bailey. 2010. Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance. *J. Mach. Learn. Res.* 11 (Dec. 2010), 2837–2854. http://dl.acm.org/citation.cfm?id=1756006.1953024

[25] Ruohui Wang and Dahua Lin. 2017. Scalable Estimation of Dirichlet Process Mixture Models on Distributed Data. In *Proceedings of the 26th International*

Joint Conference on Artificial Intelligence (IJCAI'17). AAAI Press, 4632–4639. http://dl.acm.org/citation.cfm?id=3171837.3171935

[26] Sinead Williamson, Avinava Dubey, and Eric Xing. 2013. Parallel Markov chain Monte Carlo for nonparametric mixture models. In *International Conference on Machine Learning*. 98–106.

[27] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. 2010. Spark: Cluster Computing with Working Sets. In *HotCloud*.