Massively Distributed Clustering via Dirichlet Process Mixture

Khadidja Meguelati

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITE DE MONTPELLIER

En Informatique

École doctorale : Information, Structures, Systèmes

Unité de recherche LIRMM, UMR 5506

Clustering Massivement Distribuée via Mélange de Processus de Dirichlet

Présentée par Khadidja Meguelati

Le 13/03/2020

Sous la direction de Florent Masseglia, Nadine Hilgert et Bénédicte Fontez

Devant le jury composé de

Christophe Biernacki, Professeur, INRIA, Université Lille1	Rapporteur
Mustapha Lebbah, MCF HDR, Université Sorbonne Paris Nord, LIPN	Rapporteur
Sandra Bringay, Professeur, LIRMM, Université Montpellier 3	Examinatrice
Pierre Pudlo, Professeur, Université Aix Marseille, I2M	Examinateur
Nadine Hilgert, DR INRAE, UMR MISTEA, Montpellier	Co-directrice
Florent Masseglia, DR INRIA, LIRMM, Université Montpellier	Co-directeur
Bénédicte Fontez, MCF Montpellier SupAgro, UMR MISTEA	Invitée (Co-encadrante)

UNIVERSITÉ DE MONTPELLIER

Résumé

La classification non supervisée (ou clustering) a pour objectif d'identifier des classes pertinentes dans les données. elle est largement utilisée dans de nombreuses applications telles que le marketing, la reconnaissance de patterns, l'analyse de données et le traitement d'images. Déterminer le nombre optimal de clusters dans un ensemble de données est un défi fondamental qui a ouvert de nombreuses directions de recherche. De multiples méthodes sont alors proposées pour résoudre ce problème.

Le Mélange de Processus de Dirichlet (DPM) est utilisé pour le clustering car il permet de définir automatiquement le nombre de classes, mais les temps de calculs qu'il implique sont généralement trop importants, nuisant à son adoption et rendant inefficaces ses versions centralisées.

Dans cette thèse, nous visons le problème de la parallélisation du mélange de processus de Dirichlet pour améliorer ces performances en exploitant des environnements massivement distribués. En effet, d'après la littérature, l'algorithme de DPM distribué fait appel à de nombreux problèmes tels que : l'équilibre de charge entre les nœuds de calcul, les coûts de communication, et le plein bénéfice de propriétés du DPM.

Dans cette thèse, nous proposons deux nouvelles approches pour le clustering parallèle via DPM. Tout d'abord, nous proposons DC-DPM (Cluste-

ring Distribué via mélange de processus de Dirichlet), une version parallélisée, qui permet le clustering de millions de points de données, ce qui représente un vrai défi. Nos expérimentations, tant sur des données synthétiques que réelles, illustrent la performance de notre approche. Comparativement, l'algorithme centralisé ne passe pas à l'échelle. Son temps de réponse est de plus de 7 heures sur des données de 100K points, quand notre approche prend moins de 30 secondes.

Dans un deuxième temps, nous nous intéressons au problème de dimensionnalité de données qui devient un défi important avec les obstacles numériques et théoriques dans ce cas. Nous proposons HD4C (Clustering de Dirichlet Distribué pour des Données de Haute Dimension), une solution de clustering parallèle qui s'adresse à la dimensionnalité par deux moyens. Premièrement, elle s'adapte à des données massives en exploitant les architectures distribuées. Deuxièmement, elle effectue le clustering de données de haute dimension telles que les séries temporelles (en fonction du temps), les données hyperspectrales (en fonction de la longueur d'onde), etc. Nous avons réalisé des expériences exhaustives sur des jeux de données synthétiques et réels pour confirmer l'efficacité de notre solution.

Titre en français

Clustering Massivement Distribué via Mélange de Processus de Dirichlet

Mots-clés

- Modèle de mélange de processus de Dirichlet
- Classification non supervisée
- Parallélisme
- Processus aléatoire gaussien
- Espace de Hilbert à noyau reproduisant

Abstract

Clustering with accurate results has become a topic of high interest, it is broadly used in many applications such as market research, pattern recognition, data analysis, and image processing. Determining the optimal number of clusters in a dataset is a fundamental issue that opened many directions for research. Multiple methods are then proposed to tackle this bottleneck.

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.

In this thesis, we focus on the problem of parallelizing Dirichlet process mixture to improve performances by exploiting massively distributed environments. Indeed, from the literature, distributing DPM algorithm calls for many issues such as: load balance between computing nodes, communication costs, and the full benefit from DPM properties.

In this thesis, we propose two novel approaches for parallel DPM clustering. First, we propose DC-DPM (Distributed Clustering via Dirichlet Process Mixture), a parallel clustering solution that enables clustering of

millions of data points while remaining DPM compliant. 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.

The second problem we address in this thesis is the high dimensionality of data. In this case, it becomes an important challenge with numerical and theoretical pitfalls. We propose HD4C (High Dimensional Data Distributed Dirichlet Clustering), a distributed clustering solution that addresses the curse of dimensionality by two means. First it gracefully scales to massive datasets by distributed computing. Second, it performs clustering of high dimensional data such as time series (as a function of time), hyperspectral data (as a function of wavelength) etc. Exhaustive experiments are carried out over synthetic and real world datasets to confirm the efficiency of our solution.

Title in English

Massively Distributed Clustering via Dirichlet Process Mixture

Keywords

- Dirichlet Process Mixture Model
- Clustering
- Parallelism
- Gaussian random process
- Reproducing Kernel Hilbert Space

Équipes de Recherche

Zenith Team, Inria & LIRMM

Laboratoire

LIRMM - Laboratoire d'Informatique, Robotique et Micro-électronique de Montpellier

Adresse

Université Montpellier
Bâtiment 5
CC 05 018
Campus St Priest - 860 rue St Priest
34095 Montpellier cedex 5
FRANCE

GAMMA, UMR MISTEA, INRAE & Montpellier SupAgro

Laboratoire

UMR MISTEA - Mathématiques, Informatique et STatistique pour l'Environnement et l'Agronomie

Adresse

INRAE, Montpellier SupAgro
UMR MISTEA
2, place Pierre Viala
34060 Montpellier Cedex 2
FRANCE

Résumé Étendu

Introduction

La classification non supervisée ou le clustering est la tâche de regrouper un ensemble d'objets de telle sorte que les objets d'un même groupe, appelé cluster, soient plus similaires les uns aux autres qu'à ceux des autres groupes. C'est une tâche principale de data mining, et une technique classique en analyse statistique des données, utilisée dans de nombreux domaines avec des applications au marketing [4, 82], sécurité [33], analyse de texte (document) [80], ou des sciences comme la biologie [26], l'astronomie [59], et bien d'autres.

Un des principaux défis, pour le clustering, est le fait que le nombre de clusters n'est généralement pas connu a priori. C'est la caractéristique essentielle des problèmes d'apprentissage non supervisé. Cependant, il existe des solutions pour effectuer du clustering, malgré le nombre inconnu de clusters :

1. Définir un certain nombre d'exécutions de clustering, avec une valeur variable de K, et sélectionner celle qui minimise un critère de qualité d'ajustement. Il peut s'agir d'un risque quadratique ou de l'erreur quadratique moyenne résiduelle de prédiction (RMSEP) [35]. Cette approche nécessite l'implémentation d'un algorithme de validation croisée [35]. Dans ce cas, l'approche de clustering peut être un modèle de mélange avec un algorithme d'Espérance-Maximisation (EM) [15], ou K-means [35], par exemple.

2. Faire un clustering hiérarchique puis couper l'arbre à une profondeur donnée, généralement décidée par l'utilisateur final. Différentes approches avec des avantages et des inconvénients existent, voir [35].

3. Utiliser un Mélange de Processus de Dirichlet (DPM) qui détecte automatiquement le nombre de clusters [19].

Dans cette thèse, nous nous concentrons sur l'approche DPM parce qu'elle permet d'estimer le nombre de clusters et d'assigner les observations aux clusters, dans le même processus. Ces propriétés de DPM en font une solution très intéressante pour de nombreux cas d'utilisation.

Cependant, le DPM est très coûteux en temps. Par conséquent, plusieurs tentatives ont été faites pour le rendre distribué [43, 86, 84]. Tout en étant efficacement distribuées, ces approches souffrent généralement de problèmes de convergence (distribution déséquilibrée des données sur les nœuds de calcul) [43, 86, 25] ou ne bénéficient pas pleinement des propriétés de DPM [84]. De plus, rendre le DPM parallèle n'est pas simple car il doit comparer les probabilités d'assigner chaque donnée à l'ensemble des clusters existants, un nombre de fois très répété. Cela affecte les performances globales de l'approche en parallèle, parce que comparer toutes les données à tous les clusters appellerait un nombre élevé de communications et rendrait le processus impraticable.

État de l'art

Processus de Dirichlet

Un processus de Dirichlet (DP) est un processus stochastique utilisé dans les modèles bayésiens non paramétriques de données. Il s'agit d'une distribution de probabilités sur des distributions, c'est-à-dire que chaque tirage d'un processus de Dirichlet est lui-même une distribution.

Un DP génère une distribution de probabilité G. On observe un échantillon $\theta_1, \ldots, \theta_N$ à partir de G.

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

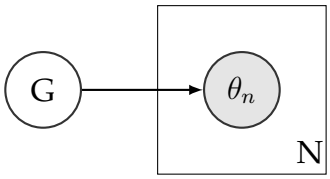

FIGURE 1 – *Echantillonnage de* $\theta_n, n = 1, \ldots, N$.

où G est par construction une distribution de probabilité discrète[74] :

$$G(\theta_n) = \sum_{k=1}^{\infty} \pi_k \delta_{\phi_k}(\theta_n)$$

avec π_k la probabilité d'avoir la valeur ϕ_k et δ est le symbole de la fonction delta de Dirac.

Par conséquent, les variables observées θ_n ont une probabilité non nulle d'avoir la même valeur ϕ_k et cela permet de faire du clustering. Le clustering est très sensible aux paramètres DP donnés par l'utilisateur final. G_0 est

une distribution de probabilité continue à partir de laquelle les $(\phi_k)_{k\in N}$ sont initialement tirés :

$$\phi_1, \ldots, \phi_k, \ldots \sim G_0$$

α est un paramètre d'échelle ($\alpha > 0$) qui ajuste les poids de probabilité π_k. Les poids π_k sont construits en utilisant la représentation des bâtons cassés (Stick-Breaking), où :

$$\begin{aligned} v_1, \ldots, v_k &\sim Beta(1, \alpha) \\ \pi_k(\mathbf{v}) &= v_k \prod_{i=1}^{k-1}(1 - v_i). \end{aligned}$$

Cette séquence de nombres $\pi_k(\mathbf{v})$ suit une distribution appelée Stick-Breaking, et on note $\pi \sim GEM(\alpha)$, elle tire son nom des noms de leurs auteurs Griffiths, Engen et McCloskey [67].

α accorde indirectement la fonction de masse pour k_N, le nombre de valeurs uniques (ϕ_i) dans un échantillon de taille N [6].

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

où $|S_{N,k_N}|$ est le nombre de Stirling de première espèce non signé.

Mélange de Processus de Dirichlet

Avec un Mélange de Processus de Dirichlet (DPM) nous observons l'échantillon $y_1, \ldots, y_N$ d'un mélange de distributions $F(\theta_n)$. Le mélange est contrôlé par un DP sur les paramètres θ_n.

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

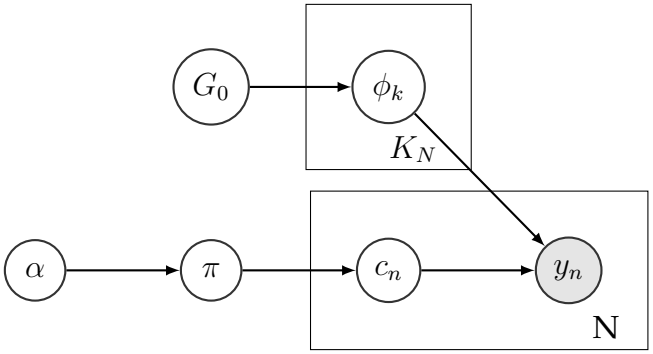

FIGURE 2 – *Graphe acyclique dirigé du DPM basé sur le Processus du Restaurant Chinois (CRP)*

Dans un cadre bayésien, l'estimation de θ_n se fait sur la distribution a posteriori : $P(\theta_1, \ldots, \theta_N \mid y_1, \ldots, y_N)$. A la place de cette représentation, un autre paramétrage est utilisé pour accélérer le calcul de la distribution a posteriori:

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

où $\theta_n = \phi_{c_n}$, c_n est le label du cluster de l'observation n, et ϕ_{c_n} est la valeur unique de θ_n appartenants au même cluster.

Processus du Restaurant Chinois

Le Processus du Restaurant Chinois (CRP) [5] est une métaphore utilisée pour voir le clustering de DPM de manière plus explicite. Dans cette métaphore, nous considérons un restaurant chinois avec un nombre infini de tables, chacune d'entre elles pouvant accueillir un nombre infini de clients servis avec le même plat. Le premier client entre dans le restaurant et s'assoit à la première table (c_1 = 1) et commande un plat ϕ_1. Le deuxième client entre et décide soit de s'asseoir avec le premier client (c_2 = 1) et commande

le même plat ϕ_1, soit de s'asseoir seul à une nouvelle table(c_2 = 2) et commande un nouveau plat ϕ_2. En général, le $n + 1^{er}$ client rejoint une table k déjà occupée avec une probabilité proportionnelle au nombre n_k de clients déjà assis là, ou s'assoit à une nouvelle table avec une probabilité proportionnelle à α. En identifiant les clients avec $y_1, \ldots, y_n$ et les tables comme des clusters, après que n clients se soient assis, les tables définissent un clustering de $y_1, \ldots, y_n$.

Echantillonnage de Gibbs

L'utilisation des modèles de mélange de processus de Dirichlet est devenue réalisable sur le plan de calcul avec le développement des méthodes de chaînes de Markov pour l'échantillonnage à partir de la distribution a posteriori des paramètres des distributions de composants et/ou des associations des composants du mélange avec les observations [56]. L'algorithme de Gibbs [27] échantillonne les labels de clusters $c_1, \ldots, c_N$ et ensuite les paramètres de clusters (ici ϕ_c, pour tous les $c \in \{1, \ldots, K\}$ où K forme le nombre de valeurs de labels au lieu de $\theta_1, \ldots, \theta_N$).

Plusieurs versions de l'échantillonnage de Gibbs (Gibbs Sampling) sont proposées par Neal dans [56] pour simuler des valeurs à partir de distribution a posteriori. Le principe est de répéter les boucles suivantes au moins jusqu'à la convergence vers la distribution a posteriori :

1. Affectation d'observations aux clusters, pour $n = 1, \ldots, N$

    - Retirez l'observation y_n de son cluster. Vérifier si le cluster est vide, si oui alors supprimer le cluster et ϕ_{c_n} de la liste $\{\phi\}$ de toutes les valeurs possibles.
    - Tirez c_n de :

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

où $\#(c)$ forme le nombre d'observations affectées au cluster c (après avoir retiré l'observation y_n de l'échantillon).

- Si c forme un nouveau cluster, tirer ϕ_c de $P(\phi \mid y_n) \propto F(y_n \mid \phi) G_0(\phi)$

2. Mise à jour de$\{\phi\}$,

   - tirer ϕ_c de la distribution postérieure du cluster c, $P(\phi \mid \{y\}_c)$ (qui est proportionnelle au produit de la G_0 antérieure et de la probabilité que toutes les observations soient affectées au cluster c).

Lorsque les distributions F et G_0 sont conjuguées, ϕ peut être intégré à partir de l'échantillonneur de Gibbs qui devient efficace en temps de calculs (pas besoin de mettre à jour $\{\phi\}$). Alors :

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

Clustering de DPM dans des environnements massivement distribués

Bien que le mélange de processus de Dirichlet ait l'avantage de découvrir automatiquement le nombre de clusters et d'assigner les données aux clusters dans le même processus, il souffre de temps de réponse prohibitifs, ce qui nuit à l'adoption de ses approches centralisées. Une solution prometteuse consiste à exploiter les systèmes distribués, tels que MapReduce [13] ou Spark [88], pour passer à l'échelle sur des données massives.

L'inférence pour les modèles qui utilisent le processus de Dirichlet peut être faite en utilisant les techniques de Monte Carlo par chaînes de Markov dans lesquelles une chaîne de Markov est construite pour tirer des échantillons à partir de la distribution a posteriori. Ces techniques sont bien connues pour leur longue durée de fonctionnement puisque le parcours de la chaîne devrait en théorie converger vers sa distribution stationnaire avant que les échantillons produits puissent être utilisés. Le processus de convergence est souvent lent car il dépend des propriétés de mélange de l'échantillonneur alors que le temps prolongé de burn-in et la variance illimitée empêchent d'exécuter simultanément plusieurs chaînes indépendantes de manière naïve [25].

Ainsi, de nombreuses solutions distribuées ont été proposées au fil des ans. Lovell et al. [43, 44] et Williamson et al. [86] ont suggéré une paramétrisation alternative pour le processus de Dirichlet afin d'en déduire une inférence MCMC parallèle non-approximative. Ces approches sont critiquées par Gal et Ghahramani dans [25]. Ces derniers ont montré que les approches proposées sont irréalisables en raison d'une distribution extrêmement déséquilibrée des données. Ils donnent des orientations pour les recherches futures comme le développement d'une meilleure inférence parallèle approximative.

L'idée principale, lorsque les données sont distribuées, est d'effectuer un DPM dans chaque worker (unité de calcul dans la distribution). Il s'agit ensuite de partager l'information entre les workers, et de synchroniser et de mettre à jour, au niveau du master, les clusters provenant des workers. Pour la synchronisation, le défi principal est le problème d'identification et de commutation des labels de clusters. Dans ce contexte, nous pouvons utiliser un algorithme de relabeling comme par exemple celui proposé par Stephens [36, 79] pour les modèles de mélange. Pour l'allocation de Dirichlet latente (LDA) parallèle et le processus de Dirichlet hiérarchique (HDP), Newman et al. [57] ont suggéré de mesurer la distance entre les clusters et ont ensuite proposé un greedy matching.

Wang et Lin [84] ont fait une revue détaillée de la littérature et des avan-

cées récentes sur ce sujet avant de donner une nouvelle proposition. Ils ont proposé d'utiliser une classification hiérarchique par étapes au niveau du master avec une demi chance de division ou de fusion à chaque étape. Ils ont commencé avec un modèle complet en considérant tous les clusters de tous les workers comme différentes composantes du modèle. Leur algorithme utilise le facteur de Bayes standard [39] pour comparer les modèles imbriqués et choisir la meilleure division ou fusion. Comme la dimension du modèle est variable, ils ont implémenté un algorithme de saut réversible [29]. En conclusion, au niveau du master, les algorithmes proposés divergent d'un classifieur de DPM et ne sont pas des estimations évolutives d'un DPM. De plus, Wang et Lin [84] ont utilisé une valeur fixe pour le paramètre d'échelle (α) dans leur implémentation du DPM au niveau des workers. Le nombre final de clusters est lié à cette valeur (voir l'équation 1). Des auteurs comme Miller et Harrison [54, 55] ont démontré l'inconsistance pour le nombre de composantes d'un modèle DPM avec une valeur fixe de α. Si le nombre de composantes identifiées au niveau des workers est sous-estimé, alors le nombre de clusters au niveau du master pourrait être sous-estimé. L'inverse augmentera considérablement le temps d'exécution au niveau du master. De plus, pour [84], ce temps de parcours dépend du taux d'acceptation du déplacement (division ou fusion) du saut réversible.

Dans notre travail, nous suggérons de s'en tenir autant que possible à un algorithme DPM, même au niveau du master, pour être proche des bonnes propriétés d'un classifieur DPM, malgré le fait que les données sont distribuées. Nous suggérons également une modification du modèle DPM pour partager l'information entre les workers. De cette façon, nous espérons améliorer notre classification (meilleure estimation) et supprimer la commutation de labels. Enfin, nous ne fixons pas la valeur de α mais nous permettons une estimation différente pour chaque worker afin d'ajouter de la flexibilité à notre modèle.

De plus, [84] est limité à des cas spécifiques où le bruit dans les observations suit la distribution conjuguée de la distribution des centres de clusters. Par exemple, un bruit gaussien impose une distribution gaussienne des centres. Par conséquent, cette méthode ne convient pas aux centres ayant

uniquement des valeurs positives. Notre but est de travailler sur n'importe quelle donnée, même avec des centres exclusivement positifs.

Contributions

L'objectif de cette thèse est de proposer des approches parallèles de DPM qui exploitent pleinement les architectures parallèles pour de meilleures performances et offrent des résultats significatifs. Notre but principal est de maintenir la consistance des clusters entre les noeuds workers, et entre les noeuds workers et master en ce qui concerne les propriétés DPM. Nos contributions principales sont les suivantes :

Modèle de mélange de processus de Dirichlet rendu efficace grâce à la distribution massive

Dans ce travail [48], nous proposons DC-DPM (Clustering Distribué via Mélange de Processus de Dirichlet), un algorithme distribué de DPM qui permet à chaque nœud d'avoir une vue sur les résultats locaux de tous les autres nœuds, tout en évitant les échanges exhaustifs de données. La nouveauté principale de notre travail est de proposer un modèle et son estimation au niveau du master en exploitant les statistiques suffisantes des workers, dans une approche conforme au DPM. Notre solution tire parti de la puissance de calcul des systèmes distribués en utilisant des frameworks parallèles tels que MapReduce [13] ou Spark [88]. Notre solution DC-DPM distribue le Processus de Dirichlet en identifiant les clusters locaux sur les workers et en synchronisant ces clusters sur le master. Ces clusters sont ensuite communiqués comme base entre les workers pour une consistance locale de clustering. Nous modifions le Processus de Dirichlet pour prendre en compte cette base dans chaque worker. En itérant ce processus, nous recherchons la consistance globale du DPM dans un environnement distribué. Nos expériences, utilisant des jeux de données réels et synthétiques, illus-

trent à la fois la grande efficacité et la scalabilité linéaire de notre approche. Nous constatons des gains significatifs en termes de temps de réponse, par rapport aux approches centralisées de DPM, avec des temps de traitement de quelques minutes, contre plusieurs jours dans le cas centralisé.

Clustering de données de haute dimensionnalité par modèle de processus de Dirichlet distribué

Dans ce travail [49, 51], nous proposons HD4C (Clustering de Dirichlet Distribué pour des Données de Haute Dimension), une nouvelle approche de clustering parallèle adaptée aux données de haute dimension et basée sur notre première contribution DC-DPM. En fait, DC-DPM est une solution proposée à ce problème lorsque les données sont multivariées. Dans le cas de données ou de signaux à haute dimension (dimension infinie), le calcul matriciel n'est plus possible (pas d'inverse de matrices par exemple, pas de produit matriciel). Il faut remplacer un produit matriciel par un produit interne dans un espace de fonctions adéquat et trouver la mesure adéquate. Ce produit interne est obligatoire pour calculer la vraisemblance et la distribution a posteriori. Pour ce faire, HD4C utilise les propriétés des espaces de Hilbert à noyau reproduisant (RKHS) (utilisées par exemple dans l'approche SVM "machine à vecteurs de support") qui sont très populaires dans l'apprentissage automatique grâce au « théorème du représentant qui a simplifié un problème empirique de minimisation du risque à dimension infinie en un problème à dimension finie où la solution est incluse dans le span linéaire de la fonction du noyau évaluée aux points d'apprentissage » [53]. Nous supposons que la variable aléatoire d'intérêt prend ses valeurs dans un espace de dimension infinie. Par conséquent, les données à haute dimension seront considérées comme des trajectoires d'un processus aléatoire. Notre travail se concentre sur le processus aléatoire gaussien pour « sa capacité à éviter les hypothèses paramétriques simples et à intégrer beaucoup de structures » [71]. De plus, de nombreux calculs sont facilités dans le cadre gaussien. Dans notre approche, nous définissons les données comme un processus gaussien autocorrélé appelé Ornstein-Uhlenbeck (OU) et nous

utilisons le même algorithme que dans DC-DPM. Nous évaluons notre proposition en utilisant des jeux de données réels et synthétiques et les résultats confirment la haute performance de notre approche.

Publications

- Khadidja Meguelati, Bénédicte Fontez, Nadine Hilgert, Florent Masseglia. Dirichlet Process Mixture Models made Scalable and Effective by means of Massive Distribution. ACM/SIGAPP SAC: Symposium on Applied Computing, Apr 2019, Limassol, Cyprus.

- Khadidja Meguelati, Bénédicte Fontez, Nadine Hilgert, Florent Masseglia. Massively Distributed Dirichlet Process Mixture Models. INFORSID: INFormatique des ORganisations et Systèmes d'Information et de Décision, Jun 2019, Université Paris-Dauphine, France.

- Khadidja Meguelati, Bénédicte Fontez, Nadine Hilgert, Florent Masseglia. Dirichlet Process Mixture Models made Scalable and Effective by means of Massive Distribution. BDA (Bases de Données Avancées): Conférence sur la Gestion de Données - Principes, Technologies et Applications, Oct 2019, Lyon, France.

- Khadidja Meguelati, Bénédicte Fontez, Nadine Hilgert, Florent Masseglia. High Dimensional Data Clustering by means of Distributed Dirichlet Process Mixture Models. IEEE International Conference on Big Data (IEEE BigData), Dec 2019, Los-Angeles, United States.

- Khadidja Meguelati, Bénédicte Fontez, Nadine Hilgert, Florent Masseglia, Isabelle Sanchez. Massively Distributed Clustering via Dirichlet Process Mixture. ECML PKDD: European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, Sep 2020, Ghent, Belgium.

Organisation de la thèse

Cette thèse est divisée en deux grands chapitres de contribution précédés d'un chapitre présentant le contexte nécessaire.

Dans le chapitre 2, nous faisons le point sur l'état de l'art. Il est divisé en trois sections principales : Dans la section 2.2, nous donnons un aperçu général des techniques principales de clustering dans l'environnement centralisé. En particulier, nous présentons quatre méthodes : le clustering hiérarchique, K-means, le clustering basé sur la densité et le clustering basé sur le modèle. La section 2.3 présente les modèles de mélange de processus de Dirichlet (DPMM), elle détaille quelques notions de processus de Dirichlet et discute l'algorithme de Gibbs sampling qui permet d'effectuer un clustering de DPM. La section 2.4 sera dédiée à l'introduction de quelques environnements distribués et de quelques solutions parallèles existante pour le clustering.

Le chapitre 3 est consacré à l'étude et à la résolution du problème des temps prohibitifs de réponse qui nuit à l'adoption du clustering par DPM et rend inefficaces ses approches centralisées. Ce chapitre commence par la motivation et l'aperçu de la contribution dans la section 3.2. Dans la section 3.3, nous proposons notre algorithme DC-DPM et nous expliquons en détail son principe. Dans la section 3.4, nous validons notre proposition à travers différentes expérimentations en utilisant des données réelles et synthétiques. Finalement, dans la section 3.5, nous concluons notre travail.

Dans le chapitre 4, nous traitons le problème de la haute dimensionnalité. Dans la section 4.2, nous présentons le contexte et donnons un aperçu de notre travail. Le contexte nécessaire des espaces de Hilbert à noyau reproduisant (RKHS) est indiqué dans la section 4.3. Dans la section 4.4, nous proposons HD4C, notre solution distribuée pour le clustering de données de haute dimension. Dans la section 4.5, nous évaluons notre approche en réalisant diverses expérimentations sur des données réelles et synthétiques. Enfin, nous résumons nos travaux dans la section 4.6

Cette thèse se termine par un chapitre de conclusion (chapitre 5) qui résume nos contributions et indique des orientations futures de la recherche dans ce domaine.

Contents

List of Figures

List of Tables

I

Introduction

1.1 Context

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group called a cluster are more similar to each other than to those in other groups. It is a main task of data mining, and a common technique for statistical data analysis, used in many fields with applications to marketing [4, 82], security [33], text (document) analysis [80], or sciences like biology [26], astronomy [59], and many more.

One of the main challenges, for clustering, is the fact that the number of clusters is typically not a priori known. That is basically the characteristic of unsupervised learning problems. However, there are some solutions that can be used to help performing cluster analysis, despite the unknown tackled number of clusters :

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) [35]. This approach needs the implementation of a cross-validation algorithm [35]. The clustering approach in this case, may be a mixture model with an Expectation-Maximization (EM) algorithm [15], or K-means [35], 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 [35].

3. Using a Dirichlet Process Mixture (DPM) which automatically detects the number of clusters [19].

In this thesis, 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.

However, DPM is highly time consuming. Consequently, several attempts have been made to make it distributed [43, 86, 84]. However, while being effectively distributed, these approaches usually suffer from convergence issues (imbalanced data distribution on computing nodes) [43, 86, 25] or do not fully benefit from DPM properties [84] (see our discussion in Section 3.2). 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.

1.2 Contributions

The objective of this thesis is to propose parallel DPM approaches that fully exploit parallel architectures for better performances and offer meaningful results. Our main goal is to keep consistency of clusters among worker

nodes, and between the worker and the master nodes with regards to DPM properties. Our main contributions are as follows:

- **Dirichlet Process Mixture Models made Scalable and Effective by means of Massive Distribution.** In this work [48], we propose DC-DPM (Distributed Clustering via Dirichlet Process Mixtures), 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 [13] or Spark [88]. 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 modify 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.

- **High Dimensional Data Clustering by means of Distributed Dirichlet Process Mixture Models.** In this work [49, 51], we propose HD4C (High Dimensional Data Distributed Dirichlet Clustering), a novel parallel clustering approach adapted for high dimensional data and based on our first contribution DC-DPM. Actually, DC-DPM is a solution proposed to this issue when data is multivariate. In the case of high dimensional data or signals (infinite dimension), matrix computation is no more feasible (no inverse matrix for example, no matrix product). We need to replace a matrix product by an inner product in an adequate space of functions and to find the adequate measure. This

inner product is mandatory to compute the likelihood and the posterior. To do that, HD4C uses the properties of the Reproducible Kernel Hilbert Spaces (RKHS) (used for example in the Support Vector Machine approach) that are very popular in machine learning thanks to « the representer theorem which simplified an infinite dimensional empirical risk minimization problem into a finite dimensional problem where the solution is included in the linear span of the kernel function evaluated at the training points » [53]. We assume that the random variable of interest takes its values in a space of infinite dimension. Therefore, high dimensional data will be seen as trajectories of a random process. Our work focuses on Gaussian random process because of « its ability to avoid simple parametric assumptions and still build in a lot of structure » [71]. In addition many calculations are facilitated in the Gaussian framework. In our approach, we define data as an autocorrelated Gaussian process called Ornstein-Uhlenbeck (OU) and we use the same algorithm as in DC-DPM. We evaluate our proposal using real and synthetic datasets and the results confirm the high performance of our approach.

1.3 Publications

The results of this thesis have been presented in the following papers:

- Khadidja Meguelati, Bénédicte Fontez, Nadine Hilgert, Florent Masseglia. Dirichlet Process Mixture Models made Scalable and Effective by means of Massive Distribution. ACM/SIGAPP SAC: Symposium on Applied Computing, Apr 2019, Limassol, Cyprus.

- Khadidja Meguelati, Bénédicte Fontez, Nadine Hilgert, Florent Masseglia. Massively Distributed Dirichlet Process Mixture Models. INFORSID: INFormatique des ORganisations et Systèmes d'Information et de Décision, Jun 2019, Université Paris-Dauphine, France.

- Khadidja Meguelati, Bénédicte Fontez, Nadine Hilgert, Florent Masseglia. Dirichlet Process Mixture Models made Scalable and Effective by means of Massive Distribution. BDA (Bases de Données Avancées): Conférence sur la Gestion de Données - Principes, Technologies et Applications, Oct 2019, Lyon, France.

- Khadidja Meguelati, Bénédicte Fontez, Nadine Hilgert, Florent Masseglia. High Dimensional Data Clustering by means of Distributed Dirichlet Process Mixture Models. IEEE International Conference on Big Data (IEEE BigData), Dec 2019, Los-Angeles, United States.

- Khadidja Meguelati, Bénédicte Fontez, Nadine Hilgert, Florent Masseglia, Isabelle Sanchez. Massively Distributed Clustering via Dirichlet Process Mixture. ECML PKDD: European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, Sep 2020, Ghent, Belgium.

1.4 Thesis Organisation

This thesis is divided into tow main contribution chapters preceded by a chapter introducing the necessary background.

In chapter 2, we review the state of the art. It is divided into three main sections: In section 2.2, we give a general overview of the main clustering techniques in the centralized environment. In particular, we deal with four methods: Hierarchical clustering, K-means, Density-Based clustering and Model-based clustering. Section 2.3 introduces the Dirichlet Process Mixture Models (DPMM), it details some notions of Dirichlet process and discusses the Gibbs Sampling algorithm that allows performing DPM clustering. The section 2.4 will be dedicated to introduce multiple parallel processing frameworks and some existing distributed clustering solutions.

Chapter 3 is devoted to studying and solving the problem of the prohibitive response times that impairs the adoption of DPM clustering and

makes centralized approaches inefficient. This chapter starts with the motivation and overview of the proposal in section 3.2. In section 3.3, we propose our algorithm DC-DPM and we thoroughly explain its clustering principle. In section 3.4, we validate our proposal through different experiments using real-world and synthetic datasets. Eventually, in section 3.5, we conclude our work.

In chapter 4, we deal with the problem of high dimensionality. In section 4.2 we present the context and give an overview of our work. The necessary background of Reproducible Kernel Hilbert Spaces (RKHS) is stated in section 4.3. In section 4.4, we propose HD4C, our distributed solution for high dimensional data clustering. In section 4.5, we evaluate our approach by carrying out various experiments on real-world and synthetic datasets. Finally, we summarize our work in section 4.6

This thesis ends with a concluding chapter (chapter 5) that summarizes our contributions and points out future research directions in this field.

II

State of the Art

2.1 Introduction

Clustering, or cluster analysis, is the task of grouping similar data into the same cluster and separating dissimilar data in different clusters. In this chapter, we introduce the basics and the necessary background of this thesis. First, we present some objectives, interests and common techniques of clustering. In particular, we introduce the Dirichlet Process Mixture Models (DPMM) by describing the notion of Dirichlet Process and discussing the algorithm of Gibbs Samplig which performs the clustering by DPM.

Second, we investigate and detail multiple parallel processing frameworks and the existing distributed clustering algorithms focusing on parallel DPM solutions.

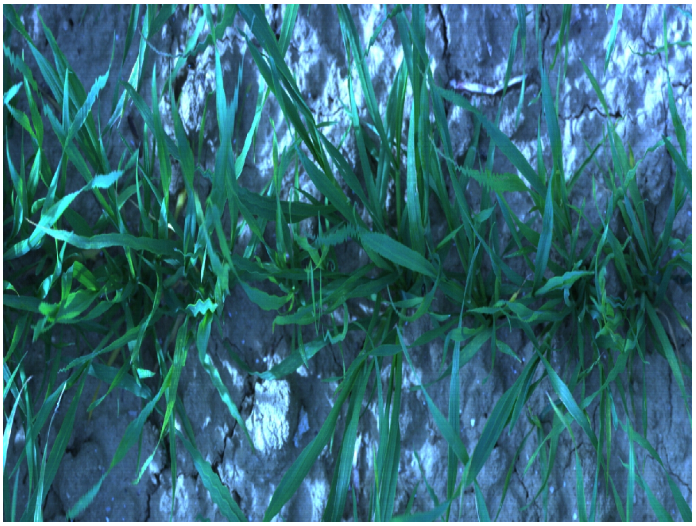

Figure 2.1 – *Durum in an experimental field. RGB image.*

2.2 Clustering

2.2.1 Objectives and Interests

Clustering is a data mining technique intensively used for data analytics, with applications many fields as mentioned in the introduction. In biology, for example, clustering may be applied to Image processing (Magnetic Resonance Imaging (MRI) [12]), detection of population structure-Genetic diversity [69] and even in dynamic systems [24].

Clustering is also used for identification in the new challenge of high throughput plant phenotyping [77], a research field with the purpose of crop improvement in response to present and future demographic and climate scenarios. In this case, data to be considered include data on plants and crop images, like the one illustrated by Figure 2.1, 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 [77, 45].

2.2.2 Common Techniques

Classical clustering techniques can be separated into six categories: partitioning, hierarchical, density-based, grid-based, model based and multi-step methods [3]. In the following, we describe the most popular algorithms: hierarchical clustering [40], k-means [46], density based clustering [41], and model-based clustering [76].

Hierarchical clustering

Hierarchical clustering [11, 40] is one of the oldest clustering methods, but it is still well-established.These methods create a tree of clusters from the given data represented in the form of a dendrogram (see figure 2.2) , at the bottom of the hierarchy is the thinnest partition, with only one observation per class, while at the top of the hierarchy is the coarsest partition for which all observations are in the same class.

We distinguish two different versions of this algorithm: a bottom-up approach, called Hierarchical Agglomerative Clustering (HAC), and a top-down procedure, named Divise Hierarchical Clustering (DHC). The first one (HAC) is used more frequently [31], it initially assigns each data instance to its own cluster and successively merges clusters until the reach of one class regrouping all data. The second algorithm (DHC) starts with one initial cluster containing all elements and proceeds by successively splitting the clusters in two until each element has its own cluster. In both cases, hierarchical clustering requires the ability to calculate, at each step, a distance between classes, called a link, based on a measure of dissimilarity between observations.

Once these distances have been chosen, the principle of hierarchical bottom-up clustering is simple. A partition of n classes each containing a single observation is formed. The algorithm begins by calculating the dissimilarity matrix, where the element d_{ij} is the distance between the observations Y_i and Y_j . The algorithm then forms a class by aggregating the two closest

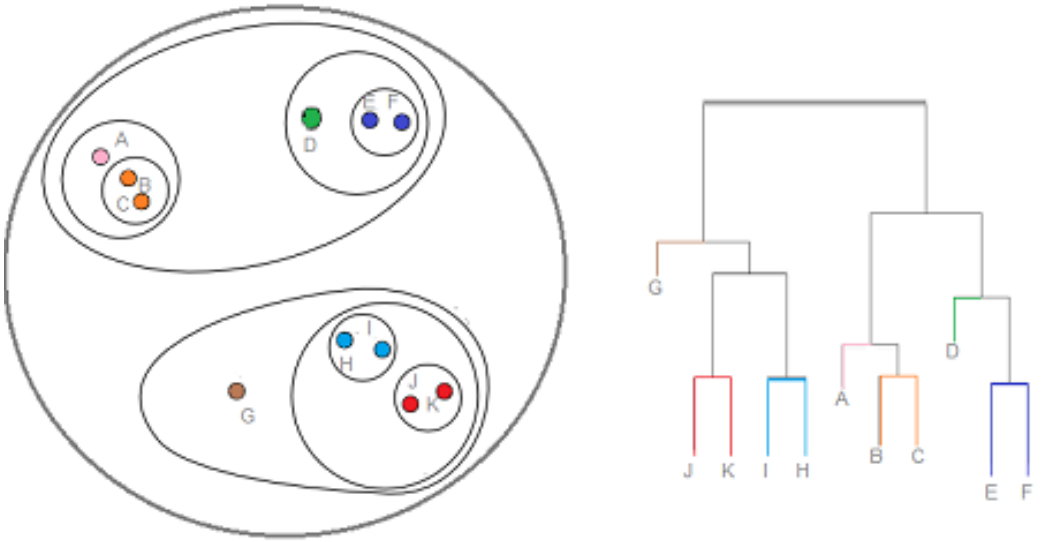

Figure 2.2 – *Hierarchical Clustering dendrogram example*

observations. The distance between this new class and the other classes is determined by the link. This process is then repeated until only one class is obtained. A similar algorithmic method is applied in the case of a hierarchical top-down clustering.

Hierarchical clustering has several benefits . The first one is that the hierarchy can be cut at any level to create a different partitioning of the collection. Second, the hierarchy can be used to navigate the data and it is useful to visualise the inherent structure of the dataset. Further, Hierarchical clustering is a good method to evaluate the performance of distance measure between data instances and being able to choose distances according to the nature of the data. On the other side, it is easy to verify that a hierarchical clustering is very sensitive to this choice. In addition, hierarchical clustering algorithms do not scale for large data sizes, due to their high complexity, and if we want to add an observation to the dataset to be classified, it is necessary to repeat the algorithm from the beginning. Some contributions are proposed to perform an approximation of the hierarchical clustering that improves time and space complexity, in order to be able to scale to large datasets [9].

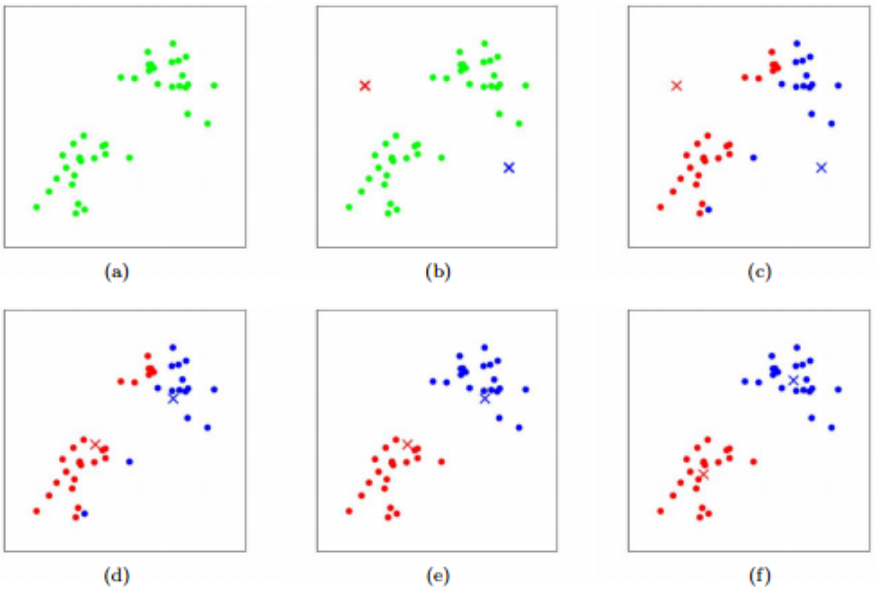

Figure 2.3 – *K-means Clustering example*

K-means

Initially proposed in 1957 by Lloyd [42], then adopted in 1967 by MacQueen [46], the K-means algorithm is an iterative method that, whatever the initial configuration, converges towards a solution. It consists in grouping the observations by minimizing the distance between each observation and the centre of its cluster, called the centroid.

Given a number of K clusters, the first step is to randomly select K centroids. Then, each observation is assigned to the centroid to which it is closest in terms of Euclidean distance. Each centroid is then recalculated using the data assigned to it. These steps are repeated until a convergence criteria is reached. In practice, the algorithm is repeated until the assignments of observations to clusters no longer change (see figure 2.3).

The K-means algorithm is easy to implement and runs quickly, making it a very popular unsupervised classification algorithm. However, it requires the number of clusters K to be specified in advance, which is considered as one of the most difficult problems to solve in data clustering, that's why many approaches are introduced to tackle this drawback [14]. Finally, the

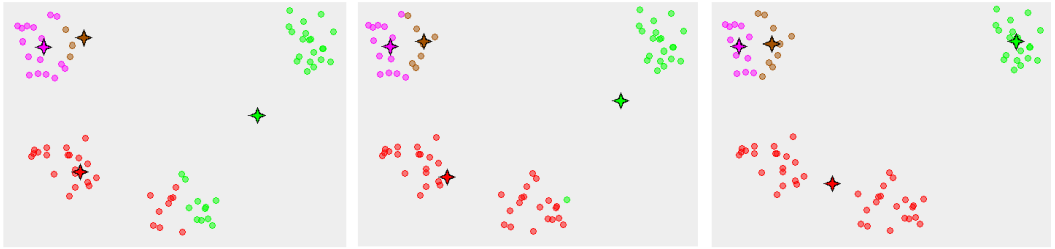

Figure 2.4 – *K-means converges to a local minimum*

K-means algorithm converges to a local minimum as shown in figure 2.4, and can find wrong clusters when they are nested within each other.

Density-based clustering

Density-based clustering methods assume that clusters appear as dense regions in a metric space. These methods search for highly dense regions in the dataset and consider them as separate clusters. A relatively well-known density-based clustering algorithm called Density Based Spatial Clustering of Applications with Noise (DBSCAN) was introduced by Ester et al. [21], which assumes that clusters appear in concentrated regions and is designed to find clusters of arbitrary shape. An interesting property of this algorithm is that it inherently copes with noise in the dataset, by declaring dense regions as clusters and regions of low-density as noise. This approach requires the user to define two parameters: a minimum distance d and a minimum number of neighbours n. Correspondingly a point p requires at least n neighbours in the radius of d in order to form a cluster.

DBSCAN is not only useful as a pure clustering algorithm but also for the detection of noise. Unfortunately it does not perform well on sets of varying density and in high-dimensional space where the data is often sparse [22].

Model-based clustering

Model-based clustering attempts to recover the original model from a set of data. This approach assumes a model for each cluster a parametric distri-

bution, and finds the best fit of data to that model[3]. In detail, it presumes that there are some fixed centroids, chosen at random, and then some individual noise is added to them with a probability distribution. The model that is recovered from the generated data defines clusters [76].

An example of this kind of clustering is the Gaussian mixture model [65] which represents a composite distribution whereby points are drawn from one of k Gaussian sub-distributions, each with its own probability.

In general, model-based clustering has two drawbacks: first, it needs to set parameters and it is based on user assumptions which may be false and consequently the result clusters would be inaccurate. Second, it has a slow processing time on large datasets [3].

2.3 Dirichlet Process Mixture Models

One of the main difficulties, for clustering, is the fact that we don't know, in advance, the number of clusters to be discovered. In order to help performing cluster analysis, despite the unknown tackled number of clusters, statistics advocate for some solutions as mentioned in the introduction. In this thesis, we focus on the DPM approach as it has the advantage of detecting the number of clusters automatically and assigning observations to clusters, in the same process.

In this section, we give the necessary background on Dirichlet Process Mixture Models illustrated by an example comes from a biology use-case.

2.3.1 Dirichlet Process

A Dirichlet Process (DP) is a stochastic process used in Bayesian non-parametric models of data. It is a probability distribution over distributions, i.e. each draw from a Dirichlet process is itself a distribution. 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 (figure 2.5). We observe a sample $\theta_1, \ldots, \theta_N$ from G. In our use-case, each θ_n is the vector of possible pixel color values.

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

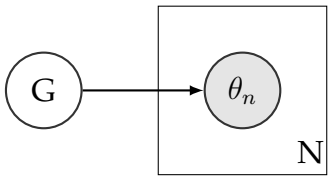

Figure 2.5 – *Sampling of* $\theta_n, n = 1, \ldots, N$.

The probability G is by construction a discrete probability distribution [74]:

$$G(\theta_n) = \sum_{k=1}^{\infty} \pi_k \delta_{\phi_k}(\theta_n),$$

where π_k is the probability of having value ϕ_k and δ is the symbol for the Dirac delta function.

Therefore, observed variables θ_n have a non null probability of having the same value ϕ_k and this allows for clustering. In our use-case of a plant image (figure 2.1), "plant" pixel parameter θ_n will have the same color value ϕ_k 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 $(\phi_k)_{k \in N}$ are initially drawn. In our use-case, G_0 gives the color probability of all possible clusters in the image.

$$\phi_1, \ldots, \phi_k, \ldots \sim G_0$$

Figure 2.6 gives an example of a DP distributed G with a base distribution G_0 and a concentration parameter α equal to 10, where G_0 is a Gaussian

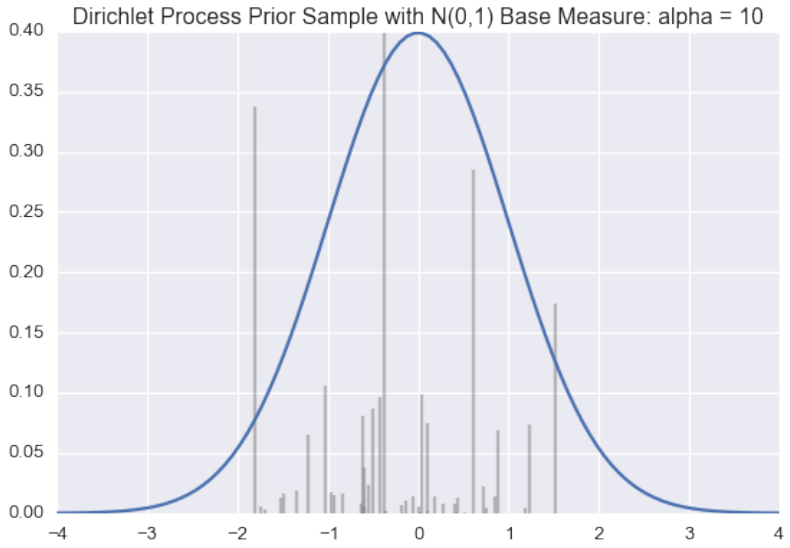

Figure 2.6 – *Dirichlet Process Sample with Gaussian base distribution*

distribution with zero mean and a variance equal to 1.

α is a scale parameter ($\alpha > 0$) which tunes the probability weights π_k. The weights π_k are constructed using the Stick-Breaking representation of Figure 2.7, where:

$$\begin{aligned} v_1, \ldots, v_k &\sim Beta(1, \alpha) \\ \pi_k(\mathbf{v}) &= v_k \prod_{i=1}^{k-1} (1 - v_i) \end{aligned}$$

This sequence of numbers $\pi_k(\mathbf{v})$ follows a distribution called Stick-Breaking, and we note $\pi \sim GEM(\alpha)$, it takes its name from the names of their authors Griffiths, Engen and McCloskey [67].

α tunes indirectly the probability mass function for k_N, the number of unique values (namely ϕ_i) in a sample of size N [6].

$$p(k_N) = |S_{N,k_N}| \; N! \, \alpha^{k_N} \frac{\Gamma(\alpha)}{\Gamma(\alpha + N)} \tag{2.1}$$

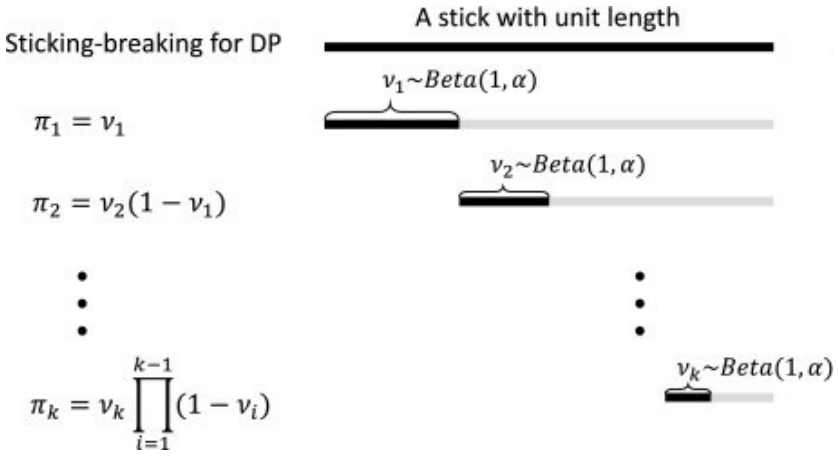

Figure 2.7 – *Stick-Breaking illustration*

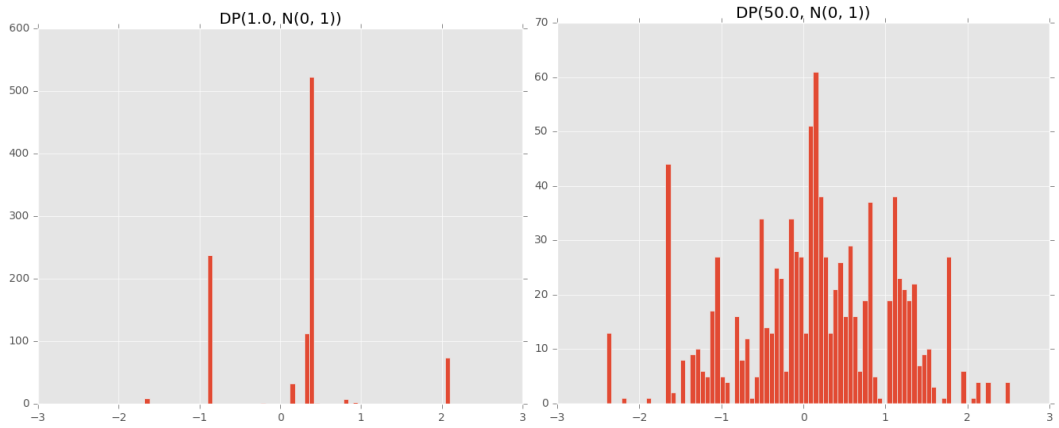

Figure 2.8 – *Concentration parameter's role in DP*

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

Figure 2.8 shows the important role of the concentration parameter α, where two samples are performed from a DP with the same base distribution and different concentration parameters. The larger the α, the smaller the variance, and the DP will concentrate more of its mass around the mean.

2.3.2 Dirichlet Process Mixture

With a Dirichlet Process Mixture we observe the sample $y_1, \ldots, y_N$ 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.

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

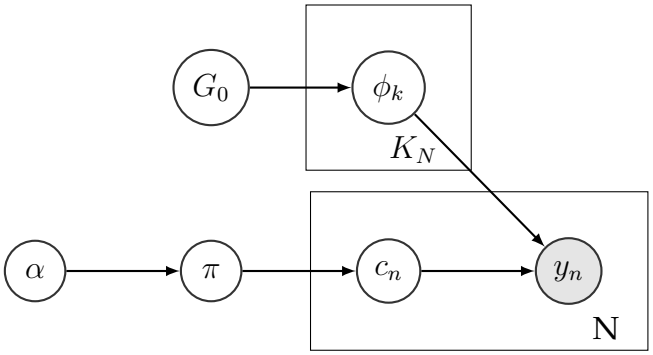

Figure 2.9 – *Directed Acyclic Graph of the Dirichlet Process Mixture (DPM) based on the Chinese Restaurant Process (CRP)*

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

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

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.

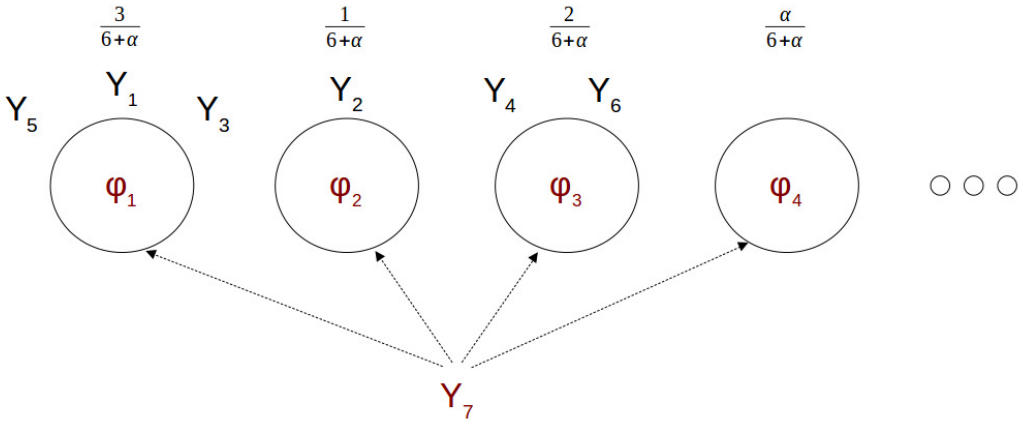

Figure 2.10 – *Chinese Restaurant Process*

Chinese Restaurant Process

The Chinese Restaurant Process (CRP) [5] is a metaphor used in order to see the DPM clustering more explicitly. In this metaphor, we consider a Chinese restaurant with an infinite number of tables, each of which can seat an infinite number of customers served the same dish. The first customer enters the restaurant and sits at the first table (c_1 = 1) and orders a dish ϕ_1. The second customer enters and decides either to sit with the first customer (c_2 = 1) and orders the same dish ϕ_1, or by himself at a new table(c_2 = 2) and orders a new dish ϕ_2. In general, the $n + 1^{st}$ customer either joins an already occupied table k with probability proportional to the number n_k of customers already sitting there, or sits at a new table with a probability proportional to α. Identifying customers with $y_1, \ldots, y_n$ and tables as clusters, after n customers have sat down, the tables define a clustering of $y_1, \ldots, y_n$.

2.3.3 Gibbs Sampling

Use of Dirichlet process mixture models has become computationally feasible with the development of Markov chain methods for sampling from the posterior distribution of the parameters of the component distributions and/or the associations of mixture components with observations [56]. The

Gibbs algorithm [27] samples the cluster labels $c_1, \ldots, c_N$ and next the cluster parameters (here ϕ_c, for all $c \in \{1, \ldots, K\}$ where K designs the number of cluster label values instead of $\theta_1, \ldots, \theta_N$).

Several versions of Gibbs sampling are proposed by Neal in [56] 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 y_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 assigned to cluster c (after removing observation y_n from the sample).
   - If c designs a new cluster, draw ϕ_c from $P(\phi \mid y_n) \propto F(y_n \mid \phi) G_0(\phi)$

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 assigned 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:

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

2.4 Massively Distributed DPM Clustering

Although Dirichlet Process Mixture has the advantage of discovering the number of clusters automatically and assigning data to clusters in the same process, it suffers from the prohibitive response times, which impairs the adoption of its centralized approaches. A promising solution is to exploit parallel frameworks, such as MapReduce [13] or Spark [88], to gracefully scale to large datasets.

In this section, we first introduce multiple parallel processing frameworks widely used in big data, and then present some parallel clustering solutions.

2.4.1 Parallel Frameworks

Recently, more and more parallel processing techniques and frameworks are coming out, and they are implemented and used in many areas, such as government, healthcare, bank, weather, transportation, social media, and education. In the following, we present the most popular frameworks : MapReduce[13] and Spark[88].

MapReduce

MapReduce is one of the most popular solutions for big data processing [8], in particular due to its automatic management of parallel execution in computing clusters. Initially proposed in [13], it was popularized

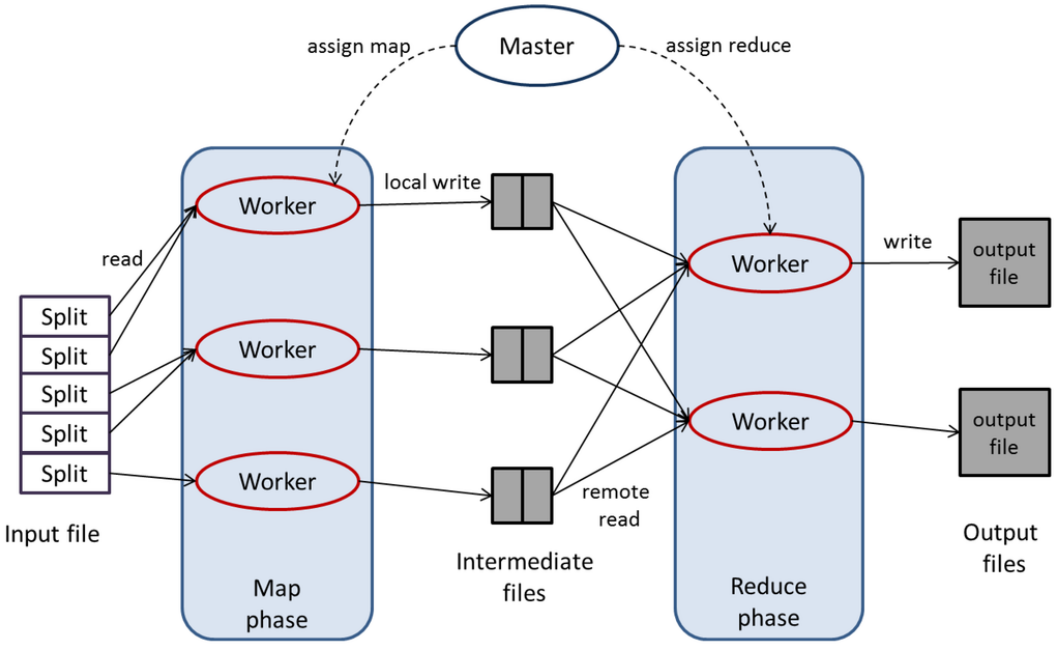

Figure 2.11 – *MapReduce architecture*

by Hadoop [85], an open source implementation. There are two functions in MapReduce program: map function and reduce function. Both map and reduce functions are written by user.

The idea behind MapReduce is simple and elegant, each job is executed in two main phases. In the first phase, the Map function is used to accept the input data which is generally in the form of file or directory and is stored in the Hadoop Distributed File System (HDFS) [75], and produce a set of intermediate results (key, value), and then send the results to reduce function. In the second phase, reduce function will accept the results and merge them together to output file.

In order to execute a MapReduce job, we need a master node that coordinates the job execution and some worker nodes to execute the map and reduce tasks. Figure 2.11 shows the MapReduce programming workflow. The user submit a MapReduce job to the master node. Input data are portioned into multiple data splits. Each split is processed by a map task in a given worker node which writes on its disc (local write). And then, results of all map tasks will be redistributed and shuffled, in this process each key

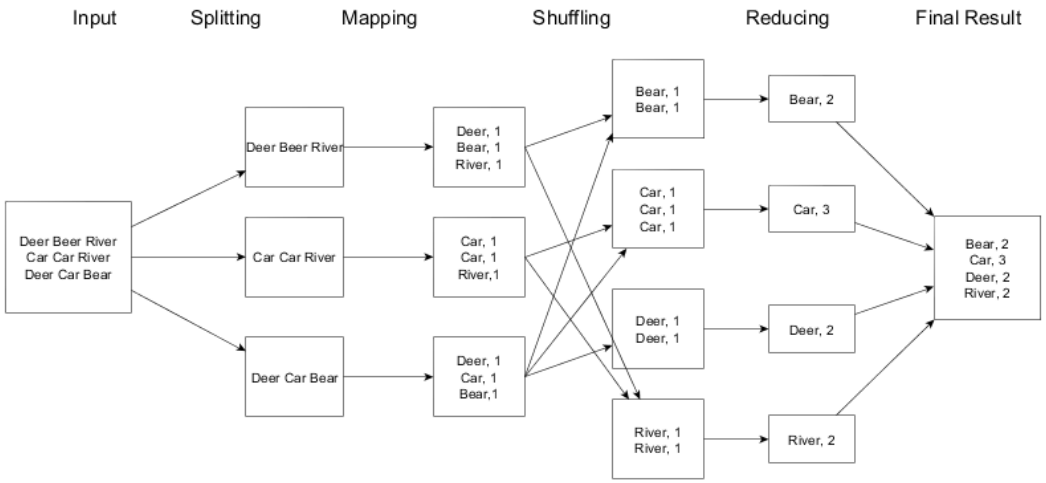

Figure 2.12 – *The overall MapReduce word count process*

will be associated with its list of values. Those shuffled results are sent to reduce function through processing and stored into the result file.

The authors of MapReduce introduce an example that counts occurrences of every word from large datasets [13]. The map function emits each word plus an associated count of occurrences. Then, the reduce function sums together all counts emitted for a particular word (see figure 2.12).

MapReduce contains a lot of pitfalls like for example, when dealing with an algorithm or an application that applies to iterative jobs, every MapReduce job has to reload the data from disk. This causes massive delay [88] and implies that those algorithms or applications cannot efficiently run using MapReduce.

Spark

Apache Spark [88] is an open-source computing cluster framework that was initially developed by a research group from University of California, Berkeley, to deal with the problems that can not be handled by MapReduce. Spark introduces multi-stage in-memory primitives that overcome disk bottlenecks and provide performance up to 100 times faster for certain applications (see figure 2.13). In addition, Spark extends the MapReduce

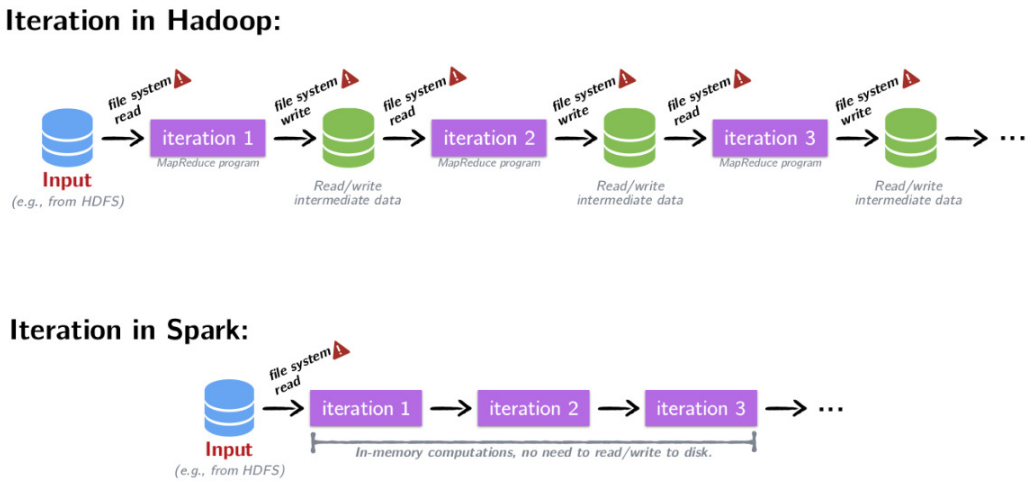

Figure 2.13 – *Spark Vs Hadoop/MapReduce*

model to efficiently support more types of computations, including interactive queries and stream processing. Spark is implemented in Scala [58], a statically typed high-level programming language for the Java Virtual Machine (JVM).

The main feature of Spark is its distributed memory abstraction, called Resilient Distributed Datasets (RDD) [87] and parallel operations used to handle it. Resilient Distributed Dataset is a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Spark lets programmers construct RDDs in four ways:

- From a file system, such as Hadoop Distributed File System.
- By parallelizing a Scala collection.
- By transforming an existing RDD.
- By changing the persistence of an existing RDD.

Two types of parallel operations can be performed on RDDs: transformations and actions. Transformations are operations on RDDs that return a new RDD, such as map and filter. Actions are operations that return a result

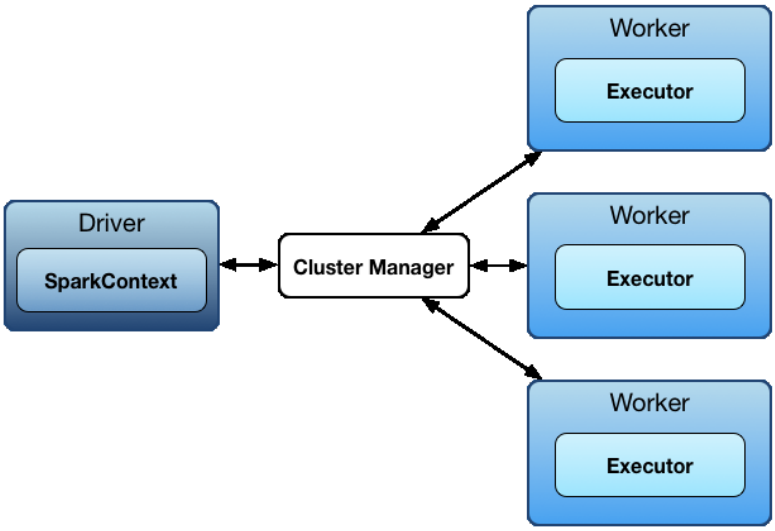

Figure 2.14 – *Spark Architecture*

to the driver program or write it to storage, and kick off a computation, such as reduce or count [38].

In distributed mode, Spark uses a master/slave architecture with one central coordinator and many distributed workers as shown in figure 2.14. The central coordinator is called the driver, it communicates with a potentially large number of distributed workers called executors. The driver runs in its own Java process and each executor is a separate Java process. A driver and its executors are together termed a Spark application.

A Spark application is launched on a set of machines using an external service called a cluster manager. Spark is packaged with a built-in cluster manager called the Standalone cluster manager. It also works with Hadoop YARN and Apache Mesos, two popular open source cluster managers [38].

2.4.2 Parallel Clustering

We set this thesis in the context of parallel clustering. Previous works for distributed algorithms of unsupervised clustering already exist. Ene et al. [18] gave a MapReduce algorithms for the k-center and k-median prob-

lems. Both algorithms use Iterative-Sample as a sub-procedure 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 difficult problems to solve in data clustering.

In [34] an efficient Earth Mover's Distance similarity joins using MapReduce is proposed. The similarity join retrieves all the pairs of objects from two datasets such that the similarity between the two objects in every pair is beyond a certain threshold. The similarity measure has a large influence on the effectiveness of the operation. The Earth Mover's Distance (EMD) is an attractive measure for applications such as probabilistic data mining. However, It has the problem of complexity ; in their experiments, the EMD's computation time was about 25000 times of the euclidean distance's on the same histograms. Huang et al. [34] used MapReduce to tackle this problem.

Debatty et al. [14] proposed a MapReduce implementation of G-means [30] which is an iterative algorithm that uses Anderson Darling test to verify if a subset of data follows a Gaussian distribution, it starts with a small number of clusters and increases the number of centers, to estimate k with a computation cost that is proportional to k, but this algorithm overestimates the number of clusters, thus it requires a post-processing step to merge clusters.

Parallel Clustering with DPM

Inference for models that use the Dirichlet process can be done using Markov chain Monte Carlo techniques in which a Markov chain is constructed to draw samples from the posterior. These techniques are well known for their long running time since the walk along the chain should in theory converge to its stationary distribution before the samples produced can be used. The convergence process is often slow as it depends on the mixing properties of the sampler while prolonged burn-in time and unbounded variance inhibit running multiple independent chains concur-

rently in a naive way [25].

Thus, many approximate distributed samplers have been suggested over the years [43, 86, 84]. However, in practice, these approaches usually suffer from convergence issues (imbalanced data distribution on computing nodes) [43, 86, 25] or do not fully benefit from DPM properties [84] (see our discussion in Section 3.2).

2.5 Conclusion

In this chapter, we have discussed the state of the art about different categories of clustering focusing on Dirichlet Process Mixtures. The main limitation is the prohibitive response time.

In this thesis, we carry out extensive theoretical and practical studies and 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.

In the next chapter, we will discuss the problem of DPM Clustering in a distributed environment. Then, we will introduce DC-DPM, our parallel solution.

III

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

3.1 Introduction

Clustering with accurate results have became 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 converging to the actual clusters in the data. However DPM is highly time consuming. In this chapter, we propose DC-DPM [48, 50, 47], a parallel clustering solution that gracefully scales to millions of data points while remaining DPM compliant, which is the challenge of distributing this process.

This chapter is organized as follows. In Section 3.2 we present the context and give an overview of our work. In Section 3.3, we describe the details of our distributed solution for clustering by means of Dirichlet Process Mixture. Section 3.4 reports the results of our experimental evaluation to verify

the efficiency and effectiveness of our approach, and Section 3.5 concludes.

3.2 Motivation and Overview of the Proposal

In the past few years, advances in hardware and software technologies have made it possible to the users of information systems to produce large amounts of data. With such complex and massive datasets, we need to improve the performance of data mining techniques, such as clustering.

In this thesis, we focused on algorithms inspired by the DPM. Lovell et al. [43, 44] and Williamson et al. [86] 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 [25]. This 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 [36, 79] for mixture models. For parallel Latent Dirichlet Allocation (LDA) and Hierarchical Dirichlet Process (HDP), Newman et al. [57] suggested to measure distance between clusters and then proposed a greedy matching.

Wang and Lin [84] 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 [39] to compare nested models and

choose the best split or merge. Since the model dimension is varying, they have implemented a reversible jump algorithm [29]. 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 [84] 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 2.1). Authors like Miller and Harrison [54, 55] 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 addition, for [84] this running time depends on the acceptance rate of the move (split or merge) of the reversible jump.

In this work, 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 [84] 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.

3.3 DC-DPM: Distributed Clustering via DPM

In this section, we present a novel parallel clustering approach called DC-DPM (Distributed Clustering via Dirichlet Process Mixtures), 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.

3.3.1 Architecture and Distributed Algorithm

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.

The general workflow of our DC-DPM approach is illustrated by Figure 3.1. 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

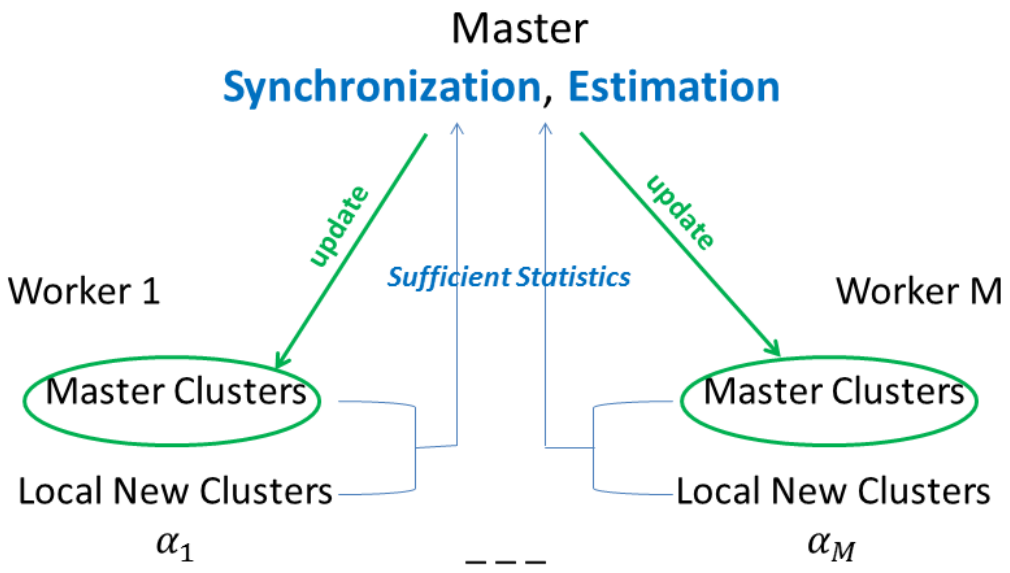

Figure 3.1 – *Diagram/workflow of the DC - DPM*

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.

Therefore we replace the Chinese Restaurant Process by a Food Courts Process illustrated in Figure 3.2. Observations (or clients) are distributed on different workers (courts) has a probability of being assigned to a cluster (table) proportional to the size of the cluster and to the likelihood (accordance between table dish and client taste) taking into consideration the information sent by the master about clusters in the other workers (Display of information about the occupancy of the tables in the others courts). Each cluster (dish/table) with at least one data (client) still exists in all workers (courts).

The interactions between the master and the worker nodes are detailed below describing tasks excuted at each level.

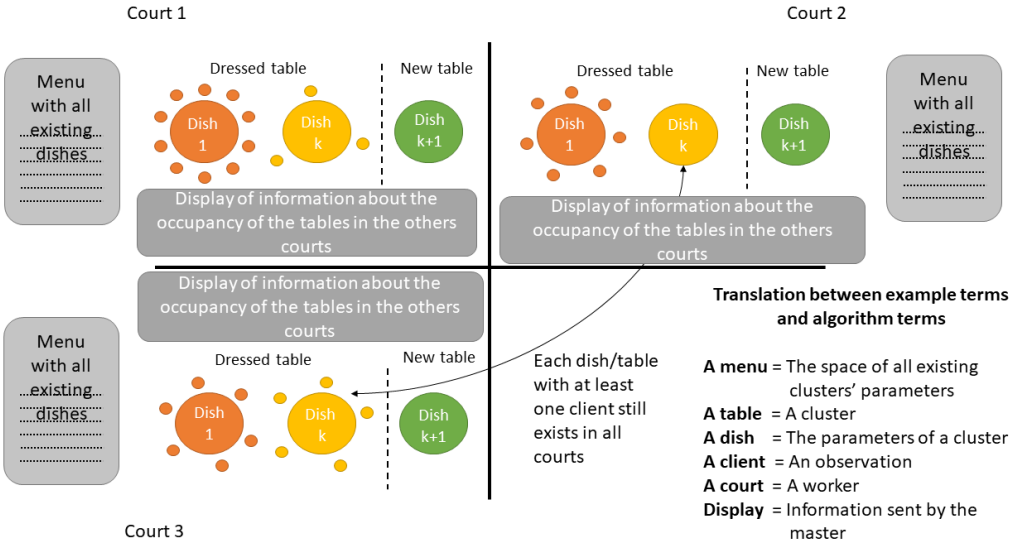

Figure 3.2 – *Food courts process*

Worker Level

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 :

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

were 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.

When a new cluster is created, we draw $b \sim beta(1, \gamma)$ and set $w_c^{new} = bw_u$ and $w_u^{new} = (1-b)w_u$. We can understand b as follows : When a new cluster is instantiated, it is instantiated from G_0 by choosing an atom in G_0 with probability given by its weight b. Using the fact that the sequence of stick-breaking weights is a size-biased permutation of the weights in a draw from a DP [66], the weight b corresponding to the chosen atom in G_0 will have the same distribution as the first stick-breaking weight, that is, $beta(1, \gamma)$ [80].

Now, the scale parameter α_j can be viewed as a tuning parameter between local (worker) and global (master) proportions. Following [20] we use an inverse gamma prior to infer this parameter.

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

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

Algorithm 1 summarizes the DPM at the worker level.

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

Master Level

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:

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

Working at an individual level implies a slow Gibbs sampling with poor mixing [28]. 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:

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

The labels $\{c_{n,j}\}$ of all the observations $y_{n,j}$ 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].

At the master, we use also an inverse gamma prior to infer the scale parameter γ as at the worker level.

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

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

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

3.3.2 The Exponential Distribution Family

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

- η is the vector of 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, it can be expressed as a function of ϕ : $a(\eta(\phi))$
- $h(y_i)$ 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) [32] 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.

3.4 Performance Evaluation

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 project is written in Scala on top of Apache Spark [88]. Spark is deployed on top of Hadoop Distributed File System (HDFS) [75] in order to efficiently read input data, as well as to store final results, and thus to overcome the bottleneck of centralized data storing. The intermediate results are stored in a distributed memory instead of stable storage (Disk) and make the system faster. Figure 3.3 illustrates the basic architecture of DC-DPM in Spark.

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,

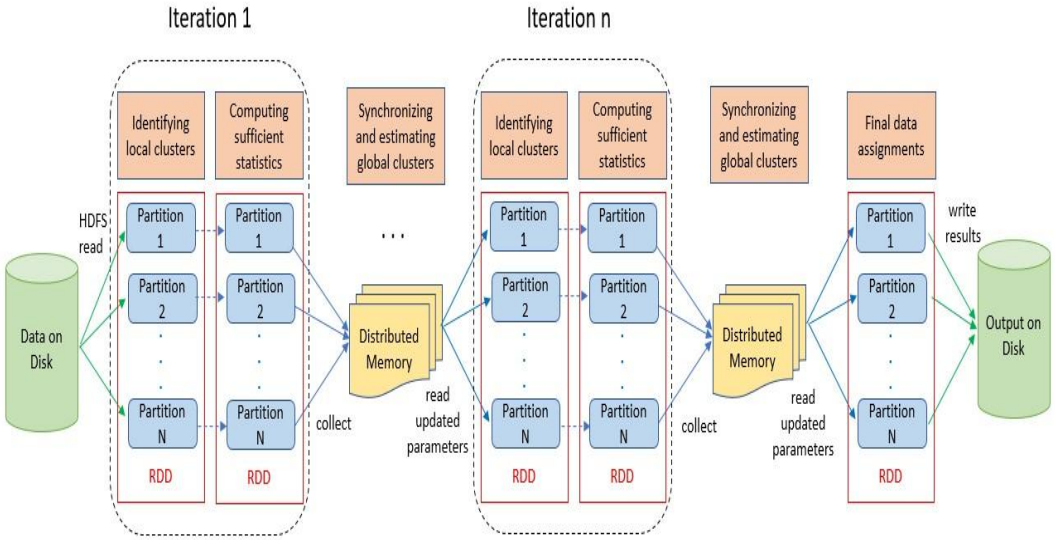

Figure 3.3 – *Architecture of DC-DPM in Spark*

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 3.4.1 and our evaluation criteria in Section 3.4.2. Then, in Section3.4.3, 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 3.4.4 and Section 3.4.5 discusses the results and interest of our work in a real use-case of agronomy.

3.4.1 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 of the figure 2.1 described in Section 2.2.1. The image used to test our algorithm was in RGB format. After pre-processing it contains 1,081,200 data points, described by a vector of 3 values (red, green and blue) belonging to $[0, 1]$.

3.4.2 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:

- The Adjusted Rand Index (ARI), see [83]: it is the corrected-for-chance version of the Rand Index [70], 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.

- 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).
- K, the number of discovered clusters.

3.4.3 Response Time

In this section we measure the clustering time in DC-DPM and compare it to the centralized approach. Figure 3.4 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 results reported by Figure 3.4 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.

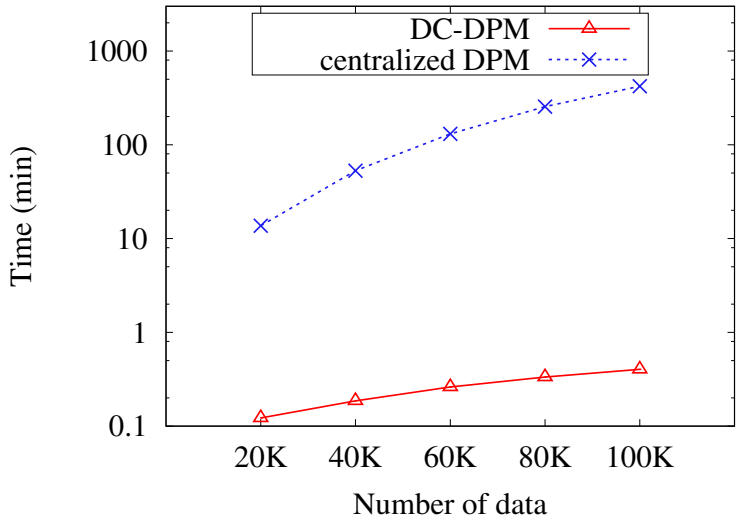

Figure 3.4 – *Logarithmic scale. Response time (minutes) of the centralized and the distributed DPM approaches as a function of dataset size. The distributed approach is run on a cluster of 8 nodes. With 20K to 100K data points from the synthetic dataset. The centralized approach needs more than 7 hours and our distributed approach needs 24 seconds*

Figure 3.5 reports an extended view on the clustering time, only for DC-DPM, and with a dataset having up to 10 million data points. 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 3.6 and 3.7 illustrate the parallel speed-up of our approach on the synthetic dataset and on the dataset obtained after preprocessing the image of our use-case. The results show optimal or near optimal gain. In Figure 3.7 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 3.7 are due to that loading time.

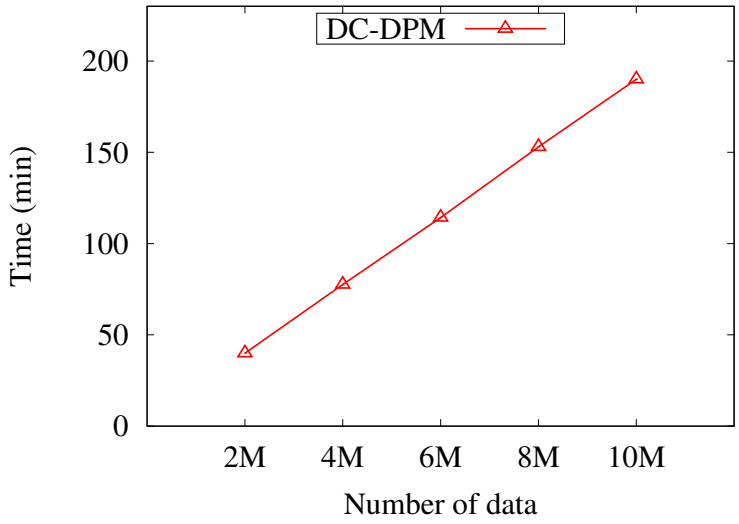

Figure 3.5 – *Response time (minutes) of DC-DPM as a function of dataset size. DC-DPM is run on a cluster of 16 machines. With 10 million data points from the synthetic dataset.*

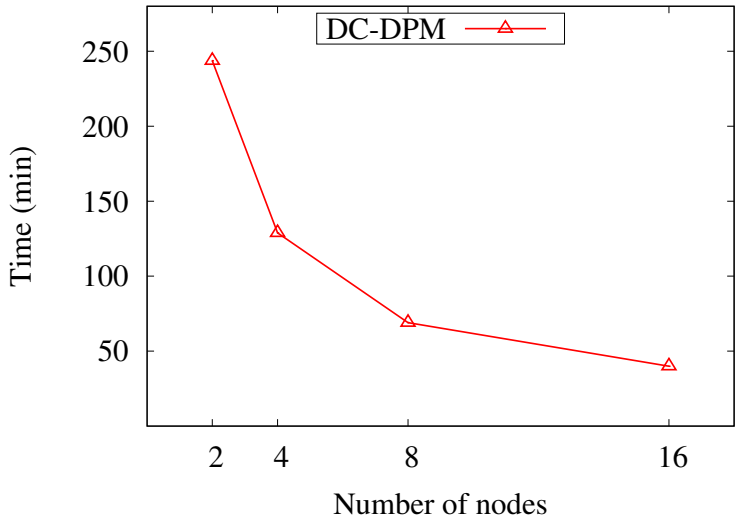

Figure 3.6 – *Clustering time as a function of the number of computing nodes on synthetic data. DC-DPM has a near optimal speed-up. With 2M data points from the synthetic dataset.*

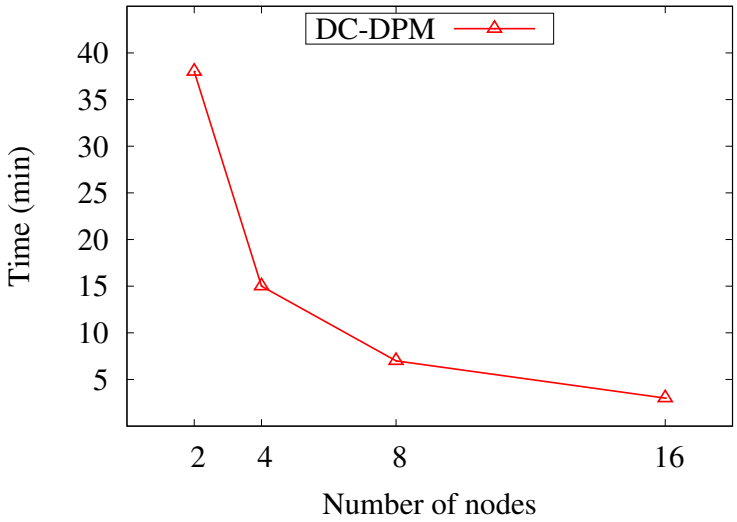

Figure 3.7 – *Clustering time as a function of the number of computing nodes on the image of our use-case. DC-DPM has an optimal speed-up. The image represents more than 1 million data points.*

3.4.4 Clustering Evaluation

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

Table 3.1 reports the ARI value computed between the clustering obtained and the ground truth, the RSS value divided by the number of data N and variance (σ_2^2), and the number of clusters of DC-DPM and of the centralized approach on our synthetic data. 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 3.2 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 number for DC-DPM, with increasing dataset size (up to 10 million data points). The performance keeps showing the maximum possible accuracy, even with a large number of data points.

Figure 3.8 gives a visual representation of our 4M data points synthetic dataset. Each cluster is assigned a color. Our goal is to retrieve these clus-

Table 3.1 – *ARI, RSS divided by the number of data N and the variance (σ_2^2), and number of Clusters obtained with the centralized DPM and with DC-DPM, on increasing dataset size. The DC-DPM is run on a cluster of 8 nodes.*

	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 3.2 – *ARI, RSS divided by the number of data N and by the variance σ_2^2, and number of clusters for DC-DPM on increasing dataset size. DC-DPM is run on a cluster of 16 machines.*

	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

ters. Figure 3.9 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 3.2, line 2.

3.4.5 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, dis-

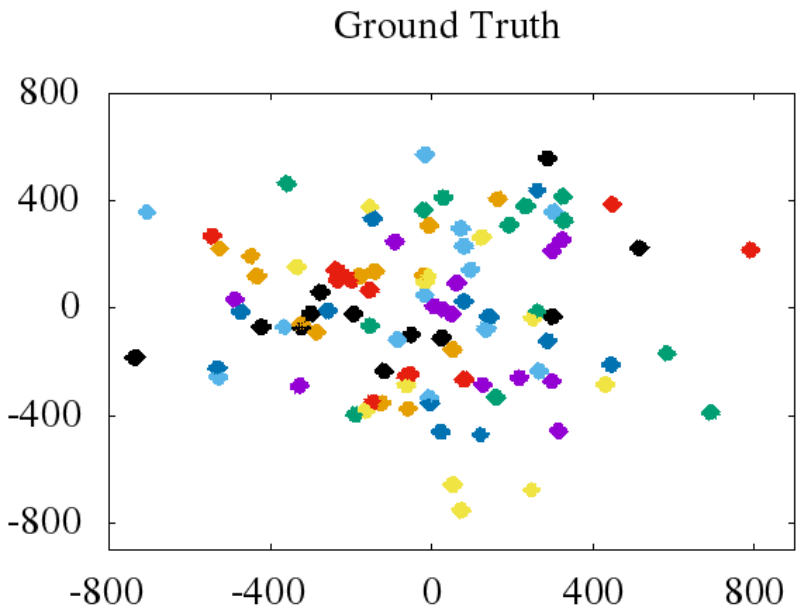

Figure 3.8 – *Visual representation of our synthetic dataset with 4 millions data points on 100 clusters. Each cluster is assigned a different color.*

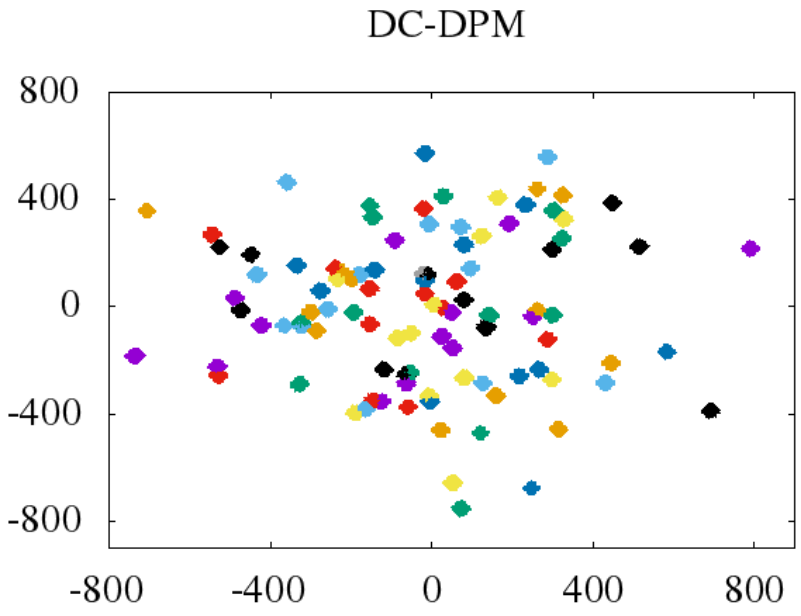

Figure 3.9 – *Visual representation of the results obtained by our approach on the data of Figure 3.8, with 16 nodes. Each cluster is assigned a different color.*

ease,...) 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 2.1 (figure 2.1) and transformed as described in Section 4.5.1. 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 3.7. 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 3.10). 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 3.11). Depending on the aim of the clustering, different types of wavelength or data must be used for identification. The accuracy of 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 RGB image. The results were quite similar as shown in figure 3.12.

There are very powerful supervised methods for classifying structures or features present in images [78], such as deep learning methods for example. Our unsupervised DPM clustering approach to image processing does not compete with these methods. On the contrary, it can be seen as a complementary method that facilitates the image labelling step of the learning dataset, a step that is always challenging and necessary in supervised classification.

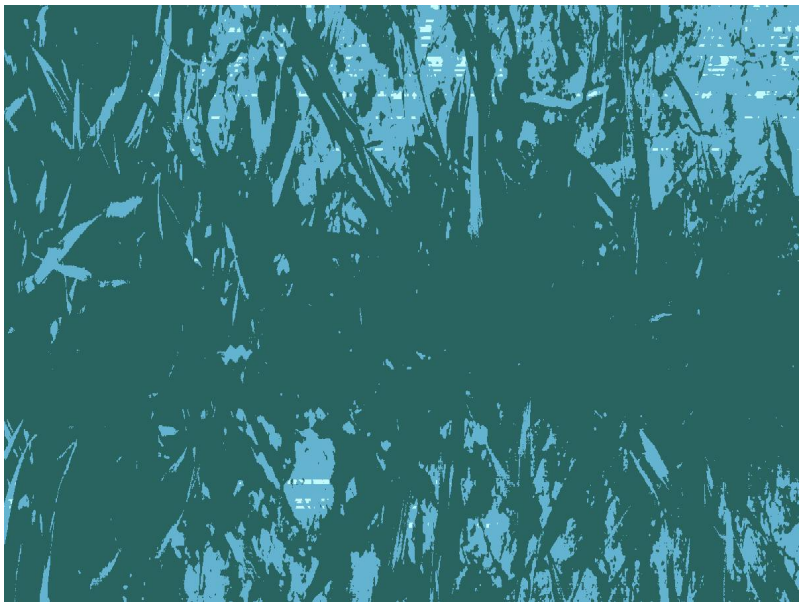

Figure 3.10 – *Clustering of the Durum image in the experimental field. RGB image with* $\sigma^2 = 0.01$*, resulting in 3 clusters.*

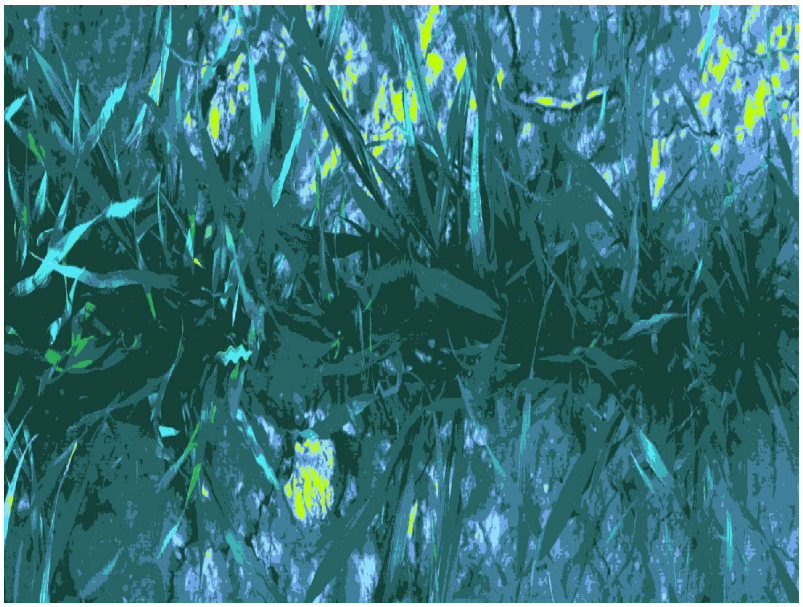

Figure 3.11 – *Clustering of the Durum image in the experimental field. RGB Image with* $\sigma_2^2 = 0.0025$*.*

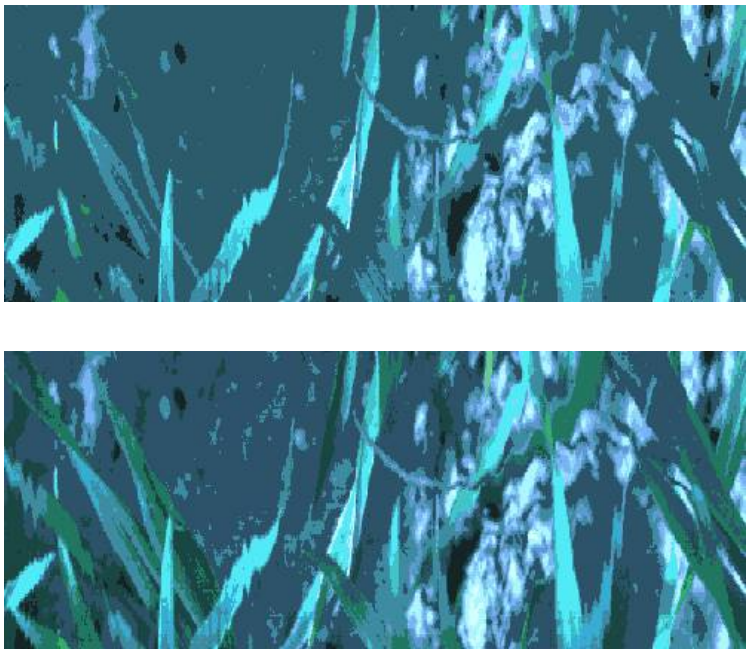

Figure 3.12 – *Clustering of a part of the Durum image in the experimental field. RGB Image with DC-DPM (top, 12 clusters) and centralized DPM (bottom, 17 clusters),* $\sigma_2^2 = 0.0025$*. The impact of* σ^2 *on the number of clusters varies for centralized and distributed approaches and may be adjusted by the end-user.*

3.5 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.

In the following chapter, we will open a fundamental research track which is clustering on high dimensional data like, *e.g.* time series.

IV

High Dimensional Data Clustering by means of Distributed Dirichlet Process Mixture Models

4.1 Introduction

Clustering may be used for identification in the new challenge of digital agriculture, where large amounts of complex data are collected: for example in herd monitoring, animal activity is monitored using a collar-mounted accelerometer, as illustrated in figure 4.1.

Unfortunately, in this case of high dimensional data, DPM relies on matrix computations. These computations are no more feasible by the several distributed approaches presented in the previous chapter (see the discussion in Section 4.2).

In this chapter, we propose HD4C (High Dimensional Data Distributed

Figure 4.1 – *An accelerometer mounted on a sheep's collar.*

Dirichlet Clustering) [49, 51], a novel parallel clustering approach adapted for high dimensional data, based on a distributed algorithm for Dirichlet Process Mixture. HD4C takes advantage of the properties of Reproducible Kernel Hilbert Spaces (RKHS) to allow clustering on the whole data (the whole signal or curve or time series) [37]. Other approaches that use feature selection and/or dimensionality reduction (like PCA or SVM) are often inappropriate because clusters generally lie in different subspaces [68].

This chapter is organized as follows. In Section 4.2, we give the motivation and overview of our work. The necessary background of Reproducible Kernel Hilbert Spaces (RKHS) is stated in section 4.3. Our distributed solution for high dimensional data clustering by means of Dirichlet Process Mixture is detailed in Section 4.4. The efficiency and effectiveness of our approach are illustrated in Section 4.5 through an experimental evaluation. Finally, the conclusion is in Section 4.6.

4.2 Motivation and Overview of the Proposal

There is a significant research on clustering of big high dimensional data. Some efforts have focused on making the similarity measures faster, like, *e.g.*, Zhu et al. [89] who introduced a novel data-adaptive approximation to DTW which can be quickly computed. Other studies suggest to make the main clustering algorithms scalable by means of massive distribution.

In this thesis, we focus on algorithms inspired by the DPM. DC-DPM is a solution proposed to this issue when data is multivariate. In the case of high dimensional data or signals (infinite dimension), matrix computation is no more feasible (no inverse for example, no matrix product). The definition of densities with respect to the usual Lebesgue measure is not available anymore. Therefore a new metric/measure must be found in order to compute a likelihood.

A first attempt to work with this kind of data is to reduce their dimensionality, by sub sampling the observations or projecting them into sub spaces like the one defined by a truncated basis of B-splines [2] or a truncated basis of kernel principal component analysis [23]. Multivariate analysis, like SVM, k-means or DC-DPM, can then be applied.

A better approach is to continue working in infinite dimension to keep all information on the data. To compute a distributed DPM for high dimensional data or signals, we need to replace a matrix product by an inner product in an adequate space of functions and to find the adequate measure to compute the likelihood and the posterior. To do that, we used the properties of the Reproducible Kernel Hilbert Spaces (RKHS), as in [37].

RKHS (used for example in the Support Vector Machine approach) are very popular in machine learning thanks to "the representer theorem which simplified an infinite dimensional empirical risk minimization problem into a finite dimensional problem where the solution is included in the linear span of the kernel function evaluated at the training points" [53].

Our goal is to propose a parallel DPM approach for high dimensional

data clustering based on the DC-DPM algorithm [48].

4.3 RKHS of Gaussian Process and DPM

We assume that the random variable of interest takes its values in a space of infinite dimension. Therefore, high dimensional data will be seen as trajectories of a random process Y: $Y = (Y(t))_{t\in[0,T]}$, where t stands for the general index of the Y function, t can be for example a time index in case of time series or a wavelength index in case of spectrum. In order to guarantee the existence of necessary conditional probabilities in the DPM algorithm, we will assume that the trajectories belong to the space of the integrable square functions ($L^2([0,T])$) on $[0,T]$ (from [17]). Our work focuses on Gaussian random process because most of the random process can be approximated by a Gaussian process. In addition many calculations are facilitated in the Gaussian framework. For example, [73] stated that using Gaussian process for machine learning "turn out to be much more accurate than for parametric models of equal flexibility (such as multilayer perceptrons)".

A Gaussian process $GP(m, K)$ is entirely defined by its mean function $m(t)$ and its covariance function $K(s,t)$, for all $t, s \in [0,T]$. The main idea behind the clustering with Gaussian Process is to use results from signal processing where the data is the sum of two Gaussian processes, namely a signal (a trajectory m_i issued from a $GP(m_0, K_0)$) and a noise (ε_i issued from a $GP(0, K)$):

$$Y_i = m_i + \varepsilon_i.$$

We assume that the signal is smoother than the noise in order to be able to detect it. To extract the signals and cluster them, we use the following DPM:

$$\begin{aligned} Y_i \mid m_i, K &\sim GP(m_i, K)\,,\ i = 1, \ldots, N \\ m_i &\sim G \\ G \mid m_0, K_0 &\sim DP(\alpha, GP(m_0, K_0)) \end{aligned}$$

DPM will create clusters of m_i where for all observations in cluster c, $m_i = \phi_c$. To run the DPM with algorithm 8 from Neal [56, 48], we need to define a posterior distribution $GP(m^*, K^*)$ for ϕ_c and the likelihood process $dGP(m_i, K)/dGP(0, K)$ for Y_i. From [73], the Reproducing Kernel Hilbert Space with reproducing kernel K, denoted H_K "will turn out to contain expected values of m_i conditioned on a finite amount of information, thus the posterior mean function m^* we are interested in".

Moreover, there exists a duality between a Gaussian process $GP(m, K)$ and H_K. H_K is a space of real functions defined on $[0, T]$ which verifies the following property: $\forall t \in [0, T], \forall f \in H_K$, $f(t) = (f, K(., t))_K$, where $(., .)_K$ is the inner product of H_K. From [62], we define the random variable $(Y, f)_K$ like a stochastic integral. The properties of H_K allow to define the likelihood process [63, 64]:

$$(Y, K(, t))_K = Y(t) \tag{4.1}$$

$$f, g \in H_K,\ (f, g)_K = E[(Y, f)_K (Y, g)_K] \tag{4.2}$$

$$m_i \in H_K,\ \frac{dGP(m_i, K)}{dGP(0, K)}(Y_i) = e^{(Y_i, m_i)_K - \frac{1}{2}(m_i, m_i)_K} \tag{4.3}$$

To ensure that $m_i \in H_K$, we must choose carefully the covariance function K_0, because the differentiability of m_i up to a given order (and therefore the smoothness of m_i) can be controlled via the covariance function.

Finally, following [16, 81, 37], the posterior distribution for the signal of a cluster c is a Gaussian process, namely $\phi_c \mid (Y_i)_{c_i=c} \sim GP(m^*, K^*)$ with:

$$m^*(t) = m_0(t) + (K_0(., t), (\bar{Y}_c - m_0))_{K/n_c + K_0} \tag{4.4}$$

$$K^*(s, t) = K_0(s, t) - (K_0(., s), K_0(., t))_{K/n_c + K_0} \tag{4.5}$$

where the covariance functions K and K_0 are weakly continuous functions on $[0, T] \times [0, T]$; n_c and $\bar{Y}_c$ are respectively the number of observations and the mean function $\left(\bar{Y}_c = \frac{1}{n_c} \sum_{c_i=c} Y_i\right)$ in cluster c.

When K is non singular and weakly continuous, usual matrix approxi-

mations of the inner product results from [63]:

$$\begin{aligned}\lim_{L\to\infty} {}^t f^{(L)} {K^{(L)}}^{-1} g^{(L)} &= (f,g)_K \\ \lim_{L\to\infty} {}^t Y_i^{(L)} {K^{(L)}}^{-1} g^{(L)} &= (Y_i,g)_K\end{aligned}$$

where $(t^l)_{l=1\ldots L}$ is dense in $[0,T]$ and $f^{(L)} = (f(t^1),\ldots,f(t^L))$, $g^{(L)} = (g(t^1),\ldots,g(t^L))$ and $K^{(L)}$ is a $L \times L$ matrix whose elements are $K(t^l,t^j)$ for $1 \leq l,j \leq L$. Oya et al. [61] proposed a generalised numerical approach to estimate the inner product in H_k. In our approach (Section IV), we use a known analytical form for the inner product, which avoids matrix product or inversion and thus allows to escape the curse of dimensionality.

4.4 HD4C : High Dimensional Data Distributed Dirichlet Clustering

Working in infinite dimension (functional data) allows to use information on the trajectories but also on their derivatives, which may reveal key information for the data clustering (see [10]). Indeed an Hilbert space (like the RKHS) is a space of integrable square functions ($L^2([0,T])$) on $[0,T]$, it is a special case of a Sobolev space. It means that a RKHS is a vector space of functions equipped with a norm that is a combination of L^p-norms of the function itself and its derivatives up to a given order. The given order is conditioned by the differentiability of the trajectories and therefore by the covariance function K of the random process Y.

In our experiments, we defined $Y_i \mid \theta_i = m_i, K$ as an autocorrelated Gaussian process called Ornstein-Uhlenbeck (OU) whose covariance function is defined as follows:

$$K(s,t) = \frac{\sigma^2}{2\,\beta} e^{-\beta|s-t|}, \tag{4.6}$$

where σ and β are two positive real.

Therefore, from [7], H_K is a space of differentiable functions in $[0, T]$ with the scalar product (defining the norm):

$$(f, g)_K = \frac{1}{\sigma^2} \int_0^T \Big(f'(t)g'(t) + \beta^2 f(t)g(t) \Big) dt + \frac{\beta}{\sigma^2} \Big(f(0)g(0) + f(T)g(T) \Big). \tag{4.7}$$

To ensure that $m_i \in H_K$, we used the prior $G = GP(m_0, K_0)$, where

$$K_0(s, t) = \frac{\sigma_0^2}{2\,\beta_0} e^{-\beta_0 (s-t)^2}.$$

This covariance gives very smooth trajectories (infinitely differentiable).

Other choice of covariance functions are possible for non smooth observations (like a Wiener process). Defining the covariance function K on the observations is equivalent to defining the kernel covariance K of the RKHS H_K. Defining a kernel K requires defining an inner product in H_K, which is equivalent to defining a metric, a distance between two observations $d(i, j) = (m_i - m_j, m_i - m_j)_K$. This led us to use a Sobolev metric for high dimensional Gaussian data (ie a distance between trajectories and their derivatives for OU Gaussian data) instead of the usual euclidean distance $\int_0^T (m_i(t) - m_j(t))^2 dt$ or the Mahalanobis distance for multivariate Gaussian data.

Implementing this algorithm requires:

- The set of indexes used for computing the integrals in the inner product equation (4.7); for example in time series, it could be the observation time steps or not.
- An interpolation of the observations (if needed) to simplify the computation of the inner product. This interpolation can be used to adapt the observations to the covariance function K.
- Computation of the densities at the master and at the worker level,

from equation (4.3). This requires estimating the hyperparameters β and σ. To avoid overly complex modelling, we have chosen to fix them empirically. As the Y_i curves are generated from Gaussian processes with covariance function K in (4.6), the parameters β and σ were determined from the empirical estimation of the intra-class variance-covariance matrix of the curves discretized in a few points.

We provide below more specific details:

Worker level

In the Gaussian process framework, the likelihood process is defined with respect to the Gaussian measure from $GP(0, K)$. Using [64] we have

$$F(y_i, \phi_c) = e^{(y_i,\phi_c)_K - \frac{1}{2}(\phi_c,\phi_c)_K}.$$

As the density of the predictive prior cannot be expressed with respect to the same Gaussian measure ($GP(0, K)$) than the likelihood, we approximated the integral in the MCMC algorithm, as suggested in algorithm 8 of [56], by drawing m realisations of ϕ_c.

To improve the variety of new candidate values of ϕ_c^{new}, we modified the original algorithm according to the following: $\phi_c^{new}(t) = m_0(t) + \zeta(t)$, where $\zeta(t)$ is a trajectory simulated from $GP(m_0, K)$ and $m_0(t)$ is randomly simulated from a truncated polynomial basis (the basis order is also randomly chosen).

Following [20], we used an inverse Gamma prior to infer the parameter α_j.

The following algorithm 3 summarizes the worker level.

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), with

Algorithm 3 DPM at worker j

for each data y_i **do**
Draw m values ϕ_c^{new}
Draw individual cluster label c_i from
$P(c_i = c \mid \{c_l\}_{l \neq i}, y_i, \{\phi\}, \{w\}, \alpha_j) \propto$

$$\begin{cases} \frac{\#(c)+\alpha_j\, w_c}{N_j - 1 + \alpha_j}\, e^{(y_i, \phi_c)_K - \frac{1}{2}(\phi_c, \phi_c)_K}, c = 1, \ldots, C \\ \frac{1}{m}\frac{\alpha_j\, w_u}{N_j - 1 + \alpha_j}\, e^{(y_i, \phi_c^{new})_K - \frac{1}{2}(\phi_c^{new}, \phi_c^{new})_K},\ c = 1, \ldots, m \end{cases}$$

Update of α_j

$w_u + \sum_{c=1}^{C} w_c = 1$. Therefore, these parameters are updated at the master level during the synchronization.

Master level

Instead of drawing new values ϕ_c^{new}, the proposed algorithm reuses the center values of the clusters received from the workers, namely $\phi_k^{worker_j}$.

The approximation of ϕ is updated by computing the posterior mean in each cluster, equation (4.4), to which we add a noise drawn from a $GP(0, K/n_c)$.

Following [48], we use a Dirichlet prior to infer $(w_1, \ldots, w_K, w_u)$.

The master lever is outlined in algorithm 4.

Algorithm 4 DPM at master level

for each cluster k from worker j **do**
Draw cluster label $z_{j,k}$ from
$P(z_{j,k} = c \mid \{c\}_{\neq j,k}, \bar{y}_{j,k}, \{\phi\}, \gamma) \propto$

$$\begin{cases} \frac{\#(c)}{N - \#(c_{j,k}) + \gamma} e^{(\bar{y}_{j,k}, \phi_c)_K - \frac{1}{2}(\phi_c, \phi_c)_K}, c = 1, \ldots, C \\ \frac{\gamma}{N - \#(c_{j,k}) + \gamma} e^{(\bar{y}_{j,k}, \phi_k^{\text{worker}j})_K - \frac{1}{2}(\phi_k^{\text{worker}j}, \phi_k^{\text{worker}j})_K} \end{cases}$$

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

4.5 Performance Evaluation

The parallel experimental evaluation was conducted on a computing 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 project is written in Scala on top of Apache Spark [88] withe the same architecture as DC-DPM illustrated in figure 3.3.

We compared our approach to K-means, which is one of the most commonly used clustering algorithms. We used an implementation available at Spark's machine learning library (MLlib) [52].

The first step of HD4C 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/HD4C`.

In the rest of this section, we describe the datasets in Section 4.5.1 and our evaluation criteria in Section 4.5.2. Then, in Section 4.5.3, we measure the performances, in response time, of our approach by reporting its scalability and speed-up. We evaluate the clusters obtained by HD4C in the case of real and synthetic dataset in Section 4.5.4.

4.5.1 Datasets

We carried out our experiments on two real world datasets and many synthetic datasets.

Our synthetic data are generated using a two-steps principle. In the first step we generate four cluster centers according to the following polynomials :

$$\begin{cases} s_1(t) = & 0.11t^3 - 0.16t^2 + 0.55t \\ s_2(t) = & -0.75t^4 + 1.49t^3 - 0.91t^2 + 0.17t \\ s_3(t) = & 3.91t^5 - 9.77t^4 + 0.854t^3 - 3.05t^2 + 0.37t \\ s_4(t) = & -20.09t^6 + 60.26t^5 - 68.22t^4 + 36t^3 \\ & -8.71t^2 + 0.76t \end{cases}$$

In the second step, we generate the data corresponding to each center, by using a Gaussian process of mean s_i and a covariance given by an Ornstein-Uhlenbeckh process parametrized by $\beta = 10$ and $\sigma = 2.5$. We generated independently a batch of 5 datasets having size 200K, 400, 600, 800K and 1M time series of 100 points, the latter dataset is about 2 Gigabytes. Figures 4.2 and 4.3 give a visual representation of our synthetic dataset. Each cluster is assigned a color and represented by 10 time series. This type of generator is widely used in statistics, where methods are evaluated first on synthetic data before being applied on real data.

The first real world dataset corresponds to more than five thousands accelerometer time series which have been measured by sensor on 13 sheep (as in figure 4.1). Each time series is made of 500 observation times and has been visually assigned to one of six activities (STANDING-GRAZING, STANDING-EATING BRUSH, STANDING-RUMINATING, WALKING, RUNNING, STANDING-IMMOBILE). Accelerometers captured 3-axial acceleration at a constant rate of 100Hz. The sensor signals were pre-processed and for each activity of interest, sampled in fixed-width of 5 seconds (500 values / a time series). Each of the three axial acceleration gives a different information for the zoologist, so HD4C clustering was performed by axis (horizontals (x and y) and vertical (z)). The objective was to discover the underlying structures of each axis and then to link these structures to sheep activities. Figures 4.4 and 4.5 represent one axis of the accelerometers dataset. Each label of activity is assigned a color and represented by 5 time series.

The second real dataset corresponds to more than 4K spectrum of 680

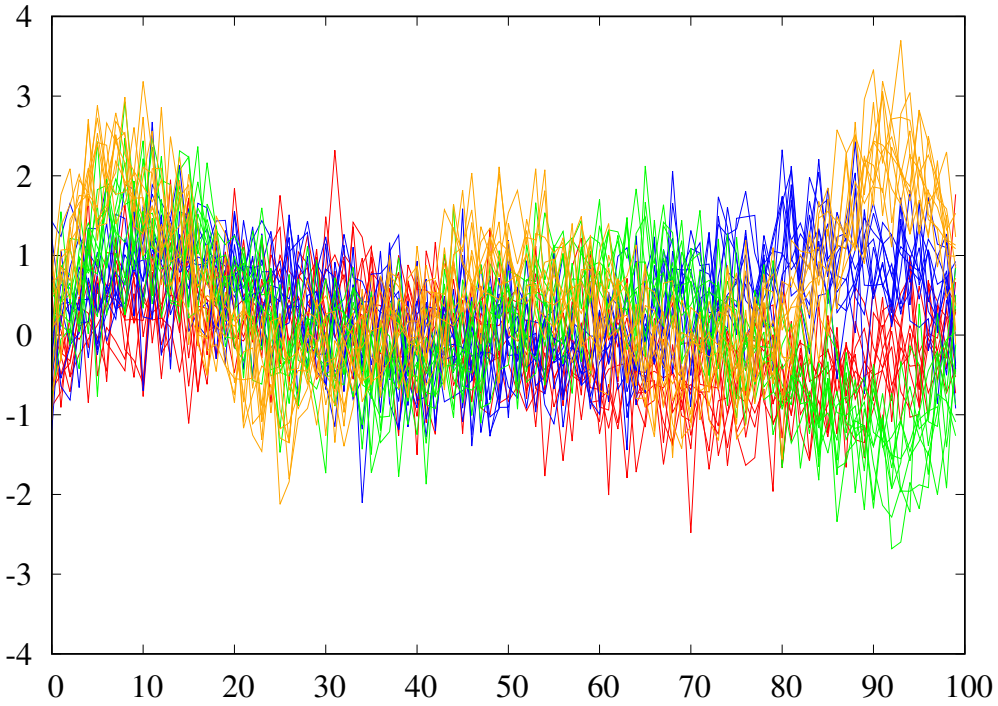

Figure 4.2 – *Visual representation of the synthetic dataset clusters.*

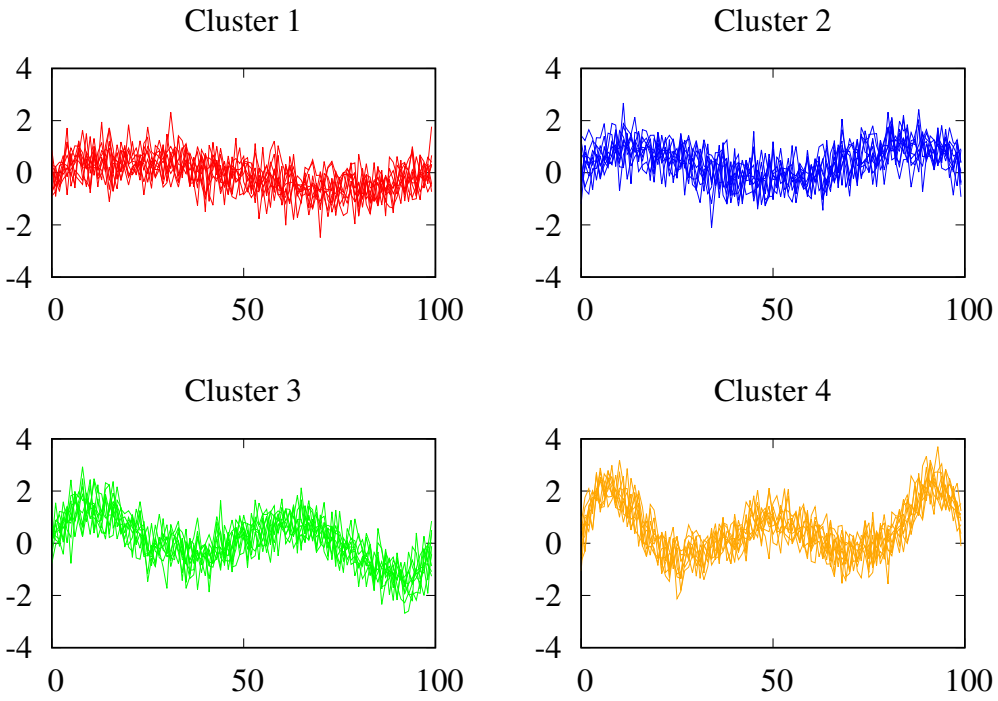

Figure 4.3 – *Visual representation of the synthetic dataset with separated clusters.*

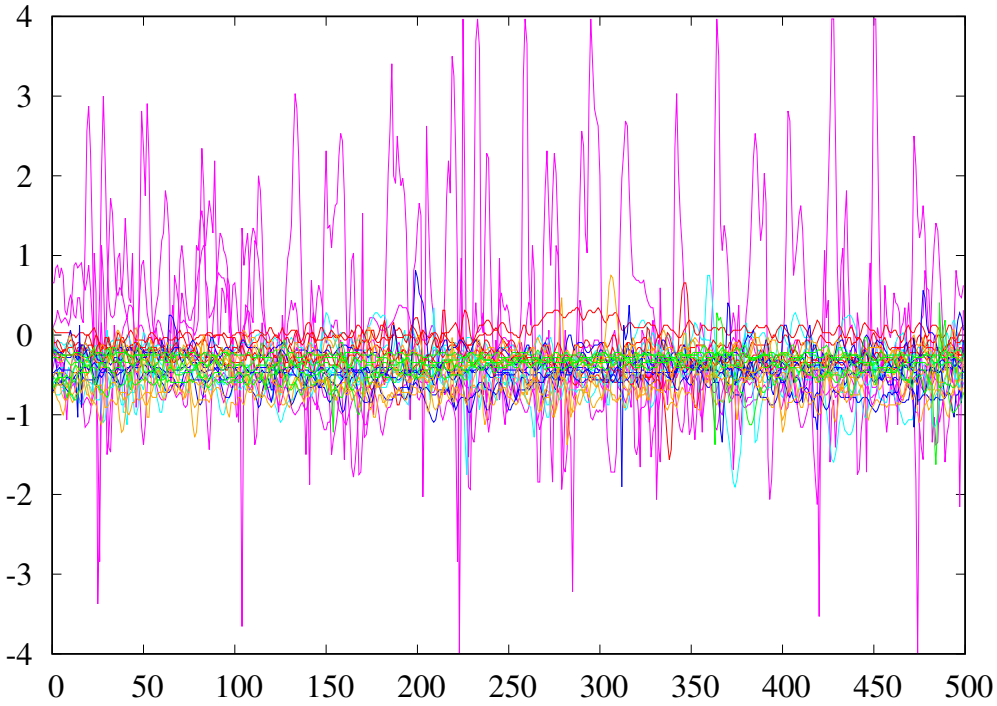

Figure 4.4 – *One axis visual representation of labeled accelerometers data*

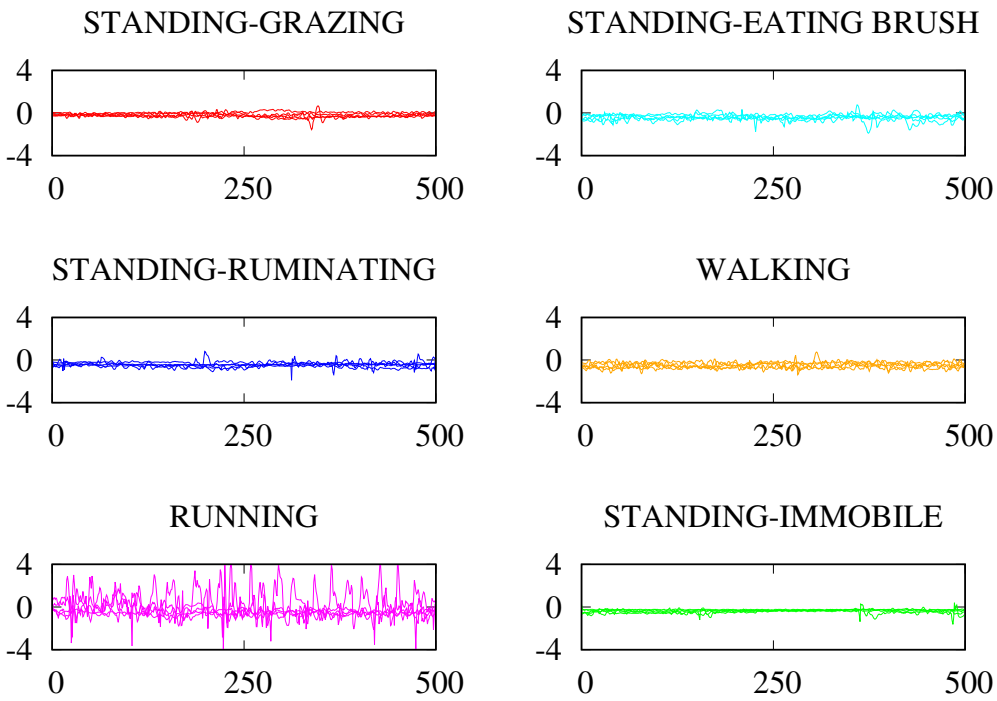

Figure 4.5 – *Separated clusters of one axis accelerometers data*

dimensions representing a protein rate measured on 10 different products: rapeseed (CLZ), corn gluten (CNG), sun flower seed (SFG), grass silage (EHH), full fat soya (FFS), wheat (FRG), sun flower seed (SFG), animal feed (ANF), soya meal (TTS), maize (PEE), milk powder and whey (MPW). Figure 4.6 gives a visual representation of the spectral data. Each product is assigned a color and represented by 50 spectrum.

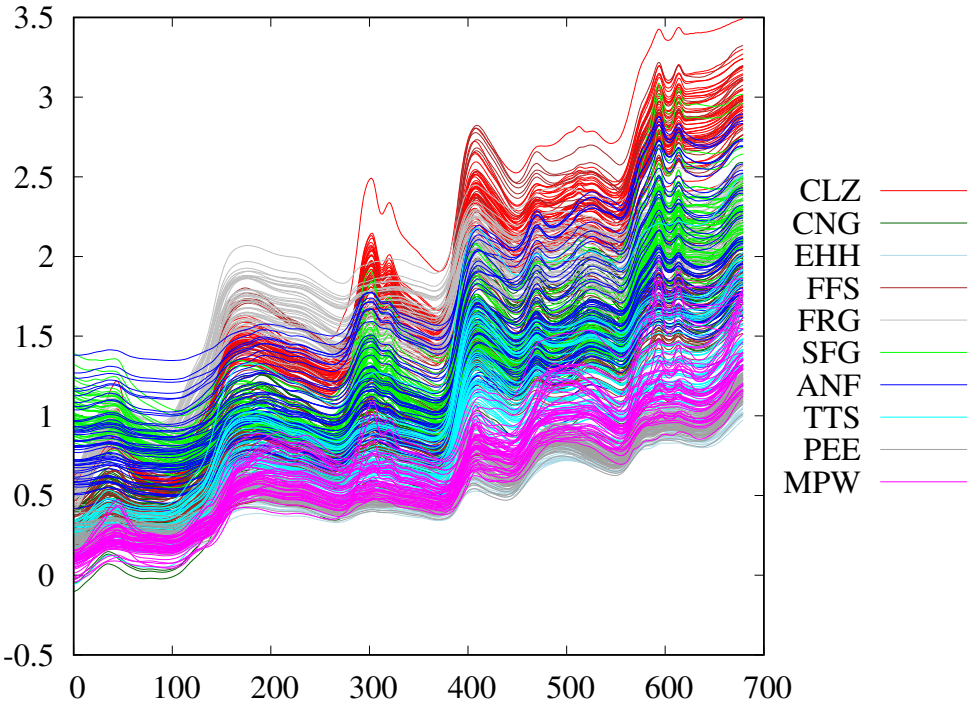

Figure 4.6 – *Visual representation of the spectral dataset*

4.5.2 Clustering Evaluation Criteria

In our experiments, we chose the following criteria already described in the previous chapter (see section 3.4.2) for evaluating the results of our clustering approach HD4C.

1. The Adjusted Rand Index (ARI).

2. K, the number of discovered clusters.

4.5.3 Response Time

In this section we measure the clustering time in HD4C. Figure 4.7 reports the response times on our synthetic data, HD4C is run on a computing cluster of 16 nodes. The clustering time increases with the number of data, our approach benefits from linear scalability with the dataset size. For a dataset of 200K data points, HD4C performs the clustering in about 12 minutes, while a centralized approach does not scale and cannot execute on such dataset size, it needs several days on a single machine.

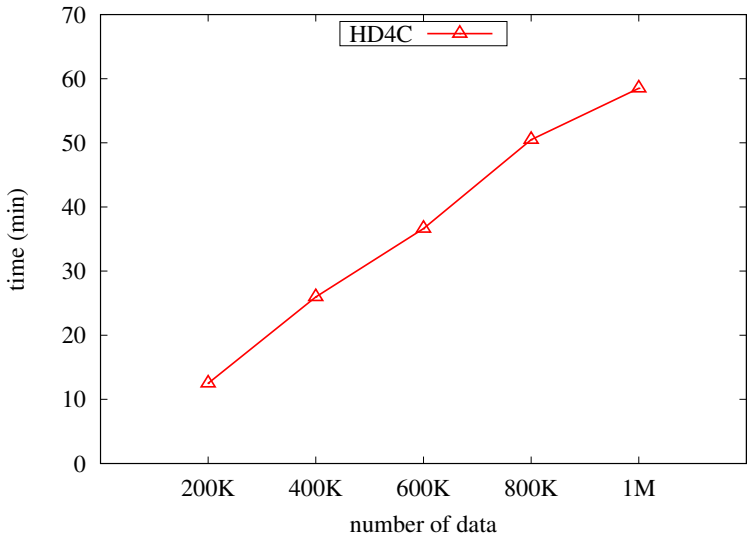

Figure 4.7 – *Response time (minutes) of HD4C as a function of the dataset size.*

Figures 4.8, 4.9 and 4.10 illustrate the parallel speed-up of our approach on 200K time series from the synthetic dataset, on accelerometers data from the first real world dataset, and on spectrum from the second real dataset. The results show optimal or near optimal gain. On the accelerometers dataset there is not a big difference between 8 and 16 nodes because this dataset is not big, and distributing it on 8 or 16 nodes is super fast at workers level while the synchronisation at the master level takes almost the same time, an other reason is that the computing nodes do not have the same performances and some of them finish and wait the other nodes that are slower.

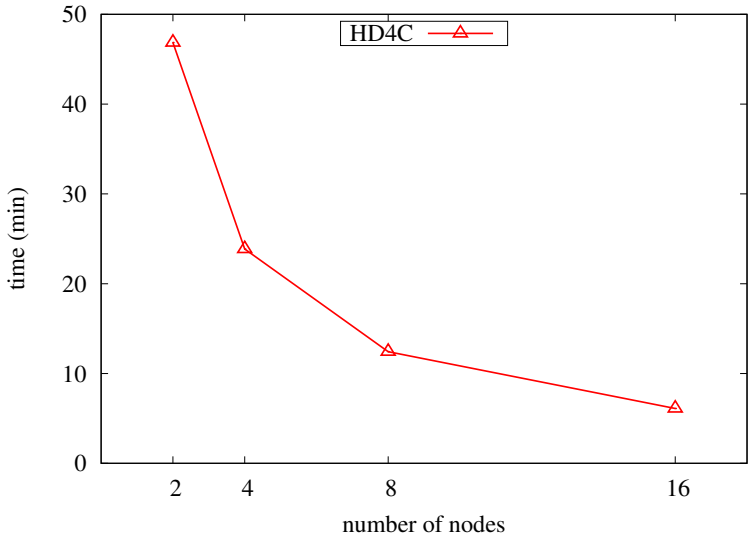

Figure 4.8 – *Clustering time as a function of the number of computing nodes on the synthetic data.*

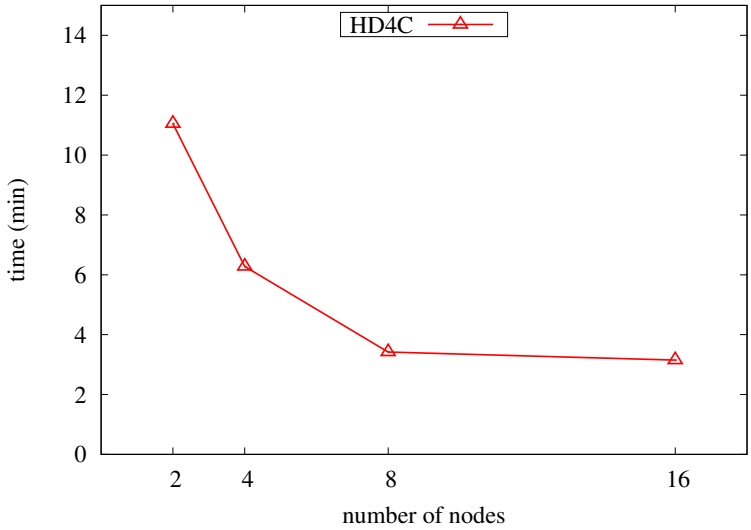

Figure 4.9 – *Clustering time as a function of the number of computing nodes on the accelerometers data.*

4.5.4 Clustering Evaluation

In the following experiments, we evaluate the clustering performance of HD4C and compare it to the K-means approach.

Table 4.1 reports the ARI value computed between the clustering ob-

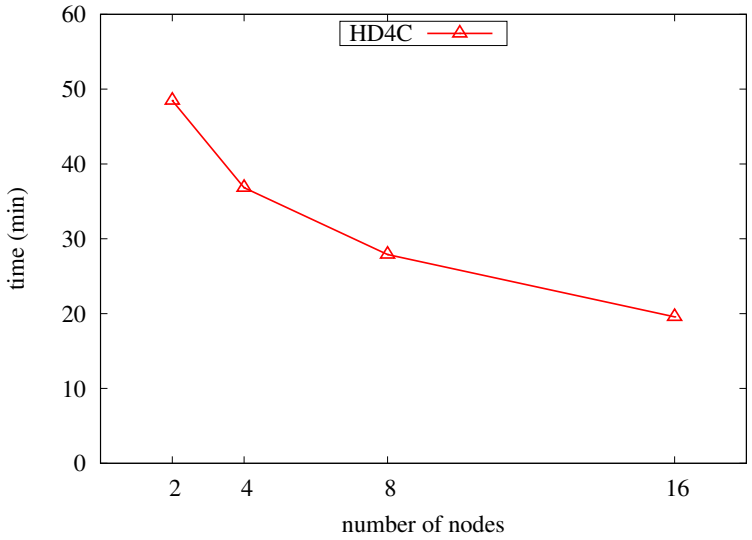

Figure 4.10 – *Clustering time as a function of the number of computing nodes on the spectral data.*

tained and the ground truth, the estimated values of parameters $\hat{\sigma}$ and $\hat{\beta}$, and the number of clusters, obtained with HD4C on our synthetic data while increasing the dataset size. The HD4C is run on a cluster of 16 nodes. HD4C performs well, the ARI values are almost equal to 1 (best value), the number of discovered clusters is equal to the real number of clusters, the estimated values of $\hat{\sigma}$ and $\hat{\beta}$ are close to the parameters used for simulating the data. Note also that the estimated ratio $\frac{\hat{\sigma}^2}{2\hat{\beta}}$ converges to the true simulated ratio $\frac{\sigma^2}{2\beta}$, which corresponds to the variance on the diagonal of K in (4.6).

Table 4.1 – *Clustering evaluation criteria obtained with HD4C (synthetic data).*

	ARI	$\hat{\sigma}$	$\hat{\beta}$	$\hat{\sigma}^2/2\hat{\beta}$	**Clusters**
200K	1.00	2.57	10.59	0.31	4
400K	1.00	2.13	7.25	0.31	4
600K	0.99	2.15	7.44	0.31	4
800K	1.00	2.28	8.30	0.31	4
1M	0.99	2.13	7.25	0.31	4

Figure 4.11 reports the Adjusted Rand Index values obtained by performing K-means approach on 200K time series from the synthetic dataset

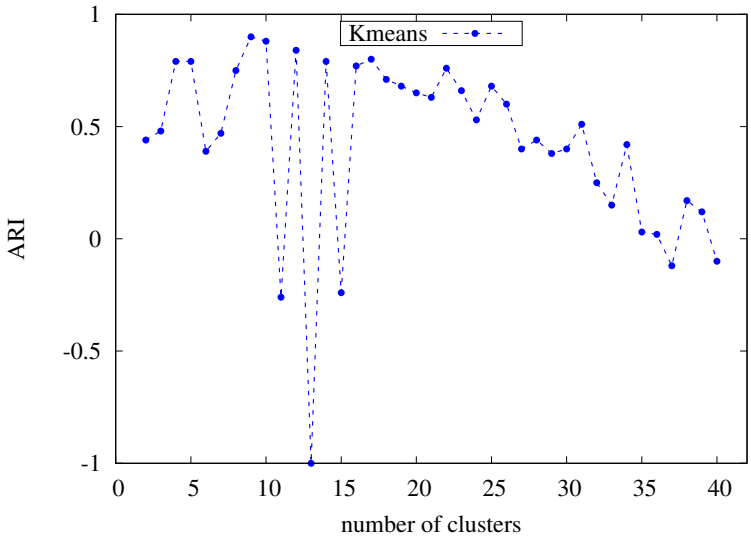

Figure 4.11 – *ARI values of K-means as a function of the number of clusters.*

as a function of the number of clusters, it is run on two nodes (16 workers). The K-means approach does not reach the best value 1, the peak of these values is 0.90 but with 9 clusters which is not the real number in the ground truth, while with the real number of clusters (4 clusters) the ARI value is 0.79.

K-means suffers from the convergence to a local minimum which may produce "wrong" results, as illustrated for example in table 4.2. This table shows the results of K-means performed on 600K time series of the synthetic dataset with the right number of clusters (4 clusters, each containing 150K data) and run on 16 nodes. Each line of table 4.2 represents one cluster obtained by K-means and reports the number of data obtained in each cluster: the cluster 2 obtained by K-means regroups the two real clusters 1 and 3 , while the real cluster 2 is divided between clusters 1 and 3 discovered by K-means.

By comparison, when applying HD4C on the same dataset, the right number of clusters is discovered and all the data except a few ones are affected to the true clusters, as presented in table 4.3.

Repeating the clustering on accelerometers data many times by HD4C

Table 4.2 – *Example of K-means convergence to a local minimum.*

	Ground truth			
	1	2	3	4
1	0	75945	0	0
2	150000	0	150000	0
3	0	74055	0	0
4	0	0	0	150000

Table 4.3 – *Number of data obtained by HD4C in each cluster compared to the ground truth.*

	Ground truth			
	1	2	3	4
1	0	149989	0	0
2	150000	0	1	0
3	0	11	149999	0
4	0	0	0	150000

and K-means, we obtained the ARI values showed on table 4.4. Our approach performs better than the K-means approach, the average value obtained by HD4C is 0.50 which is a good value regarding the shapes of data in clusters: STANDING-GRAZING, STANDING-EATING BRUSH, STANDING-RUMINATING, WALKING. The true labels have been visually assigned by experts, by observing the three axes at the same time. It is difficult to label them by only analysing one axis at a time (see figure 4.5). HD4C is not intended to cluster multidimensional time series.

Table 4.4 also represents the ARI values obtained with the real world datasets both for HD4C and K-means. K-means was processed with the number of clusters found by HD4C. Each time we repeat the HD4C clustering we find a number close to the number of labels given by the experts.

Table 4.4 – *Clustering evaluation criteria obtained with HD4C and K-means on real datasets.*

	HD4C		**K-means**
	ARI	**Clusters**	**ARI**
Accelerometers	0.50	8	0.11
Spectrums	0.34	9	0.32

4.6 Conclusion

We proposed HD4C, a novel and efficient parallel solution to perform clustering via DPM on large amount of infinite dimensional data. These infinite dimensional data include lengthy time series or spectral data for example. We evaluated the performance of our solution over real world and synthetic datasets. The experimental results illustrate the high performance of HD4C with results that are comparable to K-means, one of the most commonly used clustering algorithms. Overall, the experimental results show that by using our parallel techniques, the DPM clustering of very large volumes of high dimensional data can now be done, which is impossible to achieve using the multivariate DPM approach.

V

Conclusion

This thesis was carried out in the context of parallel clustering in massively distributed environments. We have focused on the Dirichlet Process Mixture (DPM) clustering since it enables the discovery of clusters number automatically and the attribution of data to clusters in the same process. Our aim was to improve and accelerate the DPM algorithm which suffers from the high computational costs that impairs the benefit of its advantages.

In this chapter, we summarize and discuss the main contributions made in this thesis. Then we give some research directions for future work.

5.1 Contributions

This thesis included the following main contributions related to clustering via Dirichlet process mixture in massively distributed environments.

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

In this contribution, our main challenge was the parallelization of Dirichlet Process Mixture (DPM) clustering algorithm since it must calculate for each data, the probability of assigning it to each cluster, a highly repeated number of times, that requires a global view of all dataset and all existing clusters in different nodes. Such parallelization calls for particular attention to three main issues: *i)* the load balance between computing nodes, *ii)* the cost of communication, *iii)* the full benefit from DPM properties. To this end, we proposed in this thesis DC-DPM (Distributed Clustering via Dirichlet Process Mixtures), our solution for DPM clustering that can be performed on millions of data points while remaining DPM compliant. We have extensively evaluated our algorithm using very large real-world and synthetic datasets, the results confirm the high performance in comparison with 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 made in small execution times, which is impossible to achieve using the centralized DPM approach.

5.1.2 High Dimensional Data Clustering by means of Distributed Dirichlet Process Mixture Models

In this contribution, we opened a fundamental research track which is clustering of high dimensional data such as time series (as a function of time) or hyperspectral data (as a function of wavelength). In fact, DC-DPM solution is dedicated to multivariate data clustering, it needs to perform some matrix computations like inverse matrix and matrix product for example. These computations are no more feasible in the case of high dimensional data. An existing solution is the dimensionality reduction, but this

technique may lead to data loss, or we may not know how many principal component to keep in practice. Thus our main challenge was to adapt the DPM clustering algorithm to high dimensional data by keeping all information on the data in order to avoid dimensionality reduction drawbacks and to keep all the properties of the DPM. For this reason, we proposed HD4C (High Dimensional Data Distributed Dirichlet Clustering), an efficient parallel approach for DPM clustering on large amount of infinite dimensional data. We evaluated effectiveness and the capabilities of HD4C algorithm by carrying out extensive various experiments over real-world and synthetic datasets. The results have shown an outstanding performance of our parallel technique, it enables the DPM clustering of high dimensional data, which is impossible to achieve using the multivariate DPM approach.

5.2 Directions for Future Work

The results achieved in this thesis keep the door open for several possible extensions and improvements. First, our contributions could be enriched with extensions to more general data types and use cases. Second, in order to accelerate the running time, we could consider implementations using GPU (Graphics Processing Unit) computing. In the following, we develop these directions of research.

- **Generalizing HD4C:** As mentioned previously, our work focuses on Gaussian random Process, data are defined as an autocorrelated process called Ornstein-Uhlenbeck for which the covariance function is defined. We could envisage the addition of a hierarchy that allows taking into account different data covariates, where each data is associated with a functional covariate. It is therefore a question of constructing a functional linear model in which both data and its covariate are functions.

Let's remember that the implementation of the DPM algorithm requires: *i)* the numerical calculation of densities. *ii)* the simulation according to the posterior distribution of each cluster parameter knowing its assigned data. Both requirements imply the calculation of the scalar product of a stochastic process. We could propose a new generalized numerical version to evaluate the scalar product, as in [60]. Then the same algorithm as HD4C could be used for implementation.

- **Extending to multidimensional data:** Several exciting new technologies have been developed in different fields such as digital agriculture, earth science, electric industry, biomedical and life sciences, producing a huge amounts of multidimensional data. In general, such datasets consists of different measured variables (dimensions) which can contribute to a single observation, like for example accelerometer data described in section 4.5.1. A good perspective is to adapt our work to deal with such datasets.

- **Implementing DPM with non conjugate prior:** In the DC-DPM approach, we have implemented DPM clustering algorithm when the prior is conjugated (Normal Distribution), the posterior expectation then can be estimated simply. Other distributions are not conjugated, such as log-normal distribution that is important in the description of natural phenomena for example in biology, medicine, chemistry and finance. In this case, sampling from the posterior will usually be hard. According to Neal [56], the best way of handling non-conjugate priors is by using Metropolis-Hasting algorithm [32] to update the cluster labels using the conditional prior as the proposal distribution.

- **Using GPU computing:** Since GPU computing becomes more and more practical and popular, we may benefit of GPU parallel processing capabilities to improve the DPM clustering. A Graphics Processing Unit is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate processing. It is originally made

to handle computation only for computer graphics. In the last few years, parallel GPU computing [72] has begun making computational inroads against the CPU, and it has found its way into several fields outside the image rendering and processing such as data analytics. A GPU program comprises two parts: a host part that runs on the CPU and one or more kernels that are run by thousands of threads in parallel on the GPU. Typically, the CPU portion of the program is used to set up the parameters and data for the computation, while the kernel portion performs the actual computation. This architecture allows implementing our contributions affecting workers tasks to threads in GPU and master tasks to CPU.

Bibliography

[1] *Exponential Family*, chapter 18, pages 93–97. Wiley-Blackwell, 2010. 35

[2] C. Abraham, P. A. Cornillon, E. Matzner-Løber, and N. Molinari. Unsupervised curve clustering using b-splines. *Scandinavian Journal of Statistics*, 30(3):581–595, 2003. 51

[3] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. Time-series clustering–a decade review. *Information Systems*, 53:16–38, 2015. 9, 13

[4] A. Alamsyah and B. Nurriz. Monte carlo simulation and clustering for customer segmentation in business organization. In *2017 3rd International Conference on Science and Technology - Computer (ICST)*, pages 104–109, July 2017. vii, 1

[5] David J Aldous. Exchangeability and related topics. In *École d'Été de Probabilités de Saint-Flour XIII—1983*, pages 1–198. Springer, 1985. xi, 18

[6] Charles E. Antoniak. Mixtures of dirichlet processes with applications to bayesian nonparametric problems. *Ann. Statist.*, 2(6):1152–1174, 11 1974. x, 15

[7] Alain Berlinet and Christine Thomas-Agnan. *Reproducing kernel Hilbert spaces in probability and statistics*. Springer Science & Business Media, 2011. 55

[8] Christian Bizer, Peter Boncz, Michael L Brodie, and Orri Erling. The meaningful use of big data: four perspectives–four challenges. *ACM Sigmod Record*, 40(4):56–60, 2012. 20

[9] Michael Cochez and Hao Mou. Twister tries: Approximate hierarchical agglomerative clustering for average distance in linear time. In *Proceedings of the 2015 ACM SIGMOD international conference on Management of data*, pages 505–517. ACM, 2015. 10

[10] Norma Coffey and John Hinde. Analyzing time-course microarray data using functional data analysis-a review. *Statistical Applications in Genetics and Molecular Biology*, 10(1), 2011. 54

[11] Richard M Cormack. A review of classification. *Journal of the Royal Statistical Society: Series A (General)*, 134(3):321–353, 1971. 9

[12] Adelino R. Ferreira da Silva. A dirichlet process mixture model for brain mri tissue classification. *Medical Image Analysis*, 11(2):169 – 182, 2007. 8

[13] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. xiii, xvi, 3, 20, 22

[14] Thibault Debatty, Pietro Michiardi, Wim Mees, and Olivier Thonnard. Determining the k in k-means with mapreduce. 2014. 11, 25

[15] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977. viii, 2

[16] Michael F Driscoll. The signal-noise problem—a solution for the case that signal and noise are gaussian and independent. *Journal of Applied Probability*, 12(1):183–187, 1975. 53

[17] Richard M Dudley. Real analysis and probability. wadsworth & brooks. *Cole, Pacific Groves, California*, 1989. 52

[18] Alina Ene, Sungjin Im, and Benjamin Moseley. Fast clustering using mapreduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 681–689. ACM, 2011. 24

[19] Michael D Escobar. Estimating normal means with a dirichlet process prior. *Journal of the American Statistical Association*, 89(425):268–277, 1994. viii, 2

[20] Michael D Escobar and Mike West. Bayesian density estimation and inference using mixtures. *Journal of the american statistical association*, 90(430):577–588, 1995. 33, 56

[21] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996. 12

[22] BS Everitt, S Landau, M Leese, and D Stahl. Cluster analysis, wiley. *Chichester, UK*, 2011. 12

[23] Mathieu Fauvel, Jocelyn Chanussot, and Jón Atli Benediktsson. Kernel principal component analysis for the classification of hyperspectral remote sensing data over urban areas. *EURASIP J. Adv. Signal Process*, 2009:11:1–11:14, January 2009. 51

[24] Emily Fox, Erik B. Sudderth, Michael I. Jordan, and Alan S. Willsky. Nonparametric bayesian learning of switching linear dynamical systems. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 457–464. Curran Associates, Inc., 2009. 8

[25] Yarin Gal and Zoubin Ghahramani. Pitfalls in the use of parallel inference for the dirichlet process. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 208–216, 2014. viii, xiv, 2, 26, 28

[26] H. Gao, X. Wang, and H. Huang. New robust clustering model for identifying cancer genome landscapes. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 151–160, Dec 2016. vii, 1

[27] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, 2nd ed. edition, 2004. xii, 19

[28] Joseph Gonzalez, Yucheng Low, Arthur Gretton, and Carlos Guestrin. Parallel gibbs sampling: From colored fields to thin junction trees. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 324–332, 2011. 34

[29] Peter J Green. Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika*, 82(4):711–732, 1995. xv, 29

[30] Greg Hamerly and Charles Elkan. Learning the k in k-means. In *Advances in neural information processing systems*, pages 281–288, 2004. 25

[31] Pierre Hansen and Brigitte Jaumard. Cluster analysis and mathematical programming. *Mathematical programming*, 79(1-3):191–215, 1997. 9

[32] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. 1970. 37, 72

[33] Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, Oct 2004. vii, 1

[34] Jin Huang, Rui Zhang, Rajkumar Buyya, and Jian Chen. Melody-join: Efficient earth mover's distance similarity joins using mapreduce. In *2014 IEEE 30th International Conference on Data Engineering*, pages 808–819. IEEE, 2014. 25

[35] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013. viii, 2

[36] Ajay Jasra, Chris C Holmes, and David A Stephens. Markov chain monte carlo methods and the label switching problem in bayesian mixture modeling. *Statistical Science*, pages 50–67, 2005. xiv, 28

[37] Damien Juery, Christophe Abraham, and Bénédicte Fontez. Classification bayésienne non supervisée de données fonctionnelles. *Journal de la Société Française de Statistique*, 155(2):185–201, 2014. 50, 51, 53

[38] Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia. *Learning spark: lightning-fast big data analysis*. " O'Reilly Media, Inc.", 2015. 24

[39] Robert E Kass and Adrian E Raftery. Bayes factors. *Journal of the american statistical association*, 90(430):773–795, 1995. xv, 28

[40] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. John Wiley & Sons, 2009. 9

[41] Hans-Peter Kriegel, Peer Kröger, Jörg Sander, and Arthur Zimek. Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):231–240, 2011. 9

[42] SP Lloyd. Least square quantization in pcm. bell telephone laboratories paper. published in journal much later: Lloyd, sp: Least squares quantization in pcm. *IEEE Trans. Inform. Theor.(1957/1982)*, 18, 1957. 11

[43] Dan Lovell, Ryan P Adams, and VK Mansingka. Parallel markov chain monte carlo for dirichlet process mixtures. In *Workshop on Big Learning, NIPS*, 2012. viii, xiv, 2, 26, 28

[44] Dan Lovell, Jonathan Malmaud, Ryan P Adams, and Vikash K Mansinghka. Clustercluster: parallel markov chain monte carlo for dirichlet process mixtures. *arXiv preprint arXiv:1304.2302*, 2013. xiv, 28

[45] Chuang Ma, Hao Helen Zhang, and Xiangfeng Wang. Machine learning for big data analytics in plants. *Trends in Plant Science*, 19(12):798–808, 12 2014. 8

[46] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967. 9, 11

[47] Khadidja Meguelati, Fontez Benedicte, Nadine Hilgert, and Florent Masseglia. Dirichlet process mixture models made scalable and effective by meansof massive distribution. In *Gestion de données - principes, technologies et applications (BDA)*, October 2019. 27

[48] Khadidja Meguelati, Bénédicte Fontez, Nadine Hilgert, and Florent Masseglia. Dirichlet process mixture models made scalable and effec-

tive by means of massive distribution. In *SAC: Symposium on Applied Computing*, Limassol, Cyprus, April 2019. xvi, 3, 27, 52, 53, 57

[49] Khadidja Meguelati, Bénédicte Fontez, Nadine Hilgert, and Florent Masseglia. High Dimensional Data Clustering by means of Distributed Dirichlet Process Mixture Models. In *IEEE International Conference on Big Data (IEEE BigData)*, Los-Angeles, United States, December 2019. xvii, 3, 50

[50] Khadidja Meguelati, Benedicte Fontez, Nadine Hilgert, and Florent Masseglia. Massively distributed dirichlet process mixture models. In *Actes du XXXVIIème Congrès INFORSID, Paris, France, June 11-14, 2019*, pages 221–222, 2019. 27

[51] Khadidja Meguelati, Bénédicte Fontez, Nadine Hilgert, Florent Masseglia, and Isabelle Sanchez. Massively Distributed Clustering via Dirichlet Process Mixture. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, Ghent, Belgium, September 2020. xvii, 3, 50

[52] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016. 58

[53] Karl Øyvind Mikalsen, Filippo Maria Bianchi, Cristina Soguero-Ruiz, and Robert Jenssen. Time series cluster kernel for learning similarities between multivariate time series with missing data. *Pattern Recognition*, 76:569–581, 2018. xvii, 4, 51

[54] Jeffrey W Miller and Matthew T Harrison. A simple example of dirichlet process mixture inconsistency for the number of components. In *Advances in neural information processing systems*, pages 199–206, 2013. xv, 29

[55] Jeffrey W Miller and Matthew T Harrison. Inconsistency of pitman-yor process mixtures for the number of components. *The Journal of Machine Learning Research*, 15(1):3333–3370, 2014. xv, 29

[56] Radford M Neal. Markov chain sampling methods for dirichlet process mixture models. *Journal of computational and graphical statistics*, 9(2):249–265, 2000. xii, 18, 19, 53, 56, 72

[57] David Newman, Arthur Asuncion, Padhraic Smyth, and Max Welling. Distributed algorithms for topic models. *Journal of Machine Learning Research*, 10(Aug):1801–1828, 2009. xiv, 28

[58] Martin Odersky, Lex Spoon, and Bill Venners. *Programming in scala*. Artima Inc, 2008. 23

[59] Ordovás-Pascual, I. and Sánchez Almeida, J. A fast version of the k-means classification algorithm for astronomical applications. *Astronomy & Astrophysics*, 565:A53, 2014. vii, 1

[60] Antonia Oya, Jesús Navarro-Moreno, and Juan Carlos Ruiz-Molina. Numerical evaluation of reproducing kernel hilbert space inner products. *IEEE Transactions on signal processing*, 57(3):1227–1233, 2008. 72

[61] Antonia Oya, Jesús Navarro-Moreno, and Juan Carlos Ruiz-Molina. Numerical evaluation of reproducing kernel hilbert space inner products. *IEEE Transactions on signal processing*, 57(3):1227–1233, 2009. 54

[62] E. Parzen. Regression analysis of continuous parameter time series. In *Int. ISPASS*, 2010. 53

[63] Emanuel Parzen. Statistical inference on time series by hilbert space methods, i. Technical report, STANFORD UNIV CA APPLIED MATHEMATICS AND STATISTICS LABS, 1959. 53, 54

[64] Emanuel Parzen. Probability density functionals and reproducing kernel hilbert spaces. In *Proceedings of the Symposium on Time Series Analysis*, volume 196, pages 155–169. Wiley, New York, 1963. 53, 56

[65] Karl Pearson. Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society of London. A*, 185:71–110, 1894. 13

[66] Jim Pitman. Random discrete distributions invariant under size-biased permutation. *Advances in Applied Probability*, 28(2):525–539, 1996. 33

[67] Jim Pitman et al. Combinatorial stochastic processes. Technical report, Technical Report 621, Dept. Statistics, UC Berkeley, 2002. Lecture notes for . . . , 2002. x, 15

[68] Saurabh Prasad and Lori Mann Bruce. Limitations of principal components analysis for hyperspectral target recognition. *IEEE Geoscience and Remote Sensing Letters*, 5(4):625–629, 2008. 50

[69] Jonathan K. Pritchard, Matthew Stephens, and Peter Donnelly. Inference of population structure using multilocus genotype data. *Genetics*, 155(2):945–959, 2000. 8

[70] W.M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971. 39

[71] C.E. Rasmussen and C.K.I. Williams. *Gaussian Processes for Machine Learning*. Massachusetts Institute of Technology, 2006. xvii, 4

[72] Jason Sanders and Edward Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010. 73

[73] MATTHIAS SEEGER. Gaussian processes for machine learning. *International Journal of Neural Systems*, 14(02):69–106, 2004. PMID: 15112367. 52, 53

[74] Jayaram Sethuraman. A constructive definition of dirichlet priors. *Statistica sinica*, pages 639–650, 1994. ix, 14

[75] Jeffrey Shafer, Scott Rixner, and Alan L Cox. The hadoop distributed filesystem: Balancing portability and performance. In *2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*, pages 122–133. IEEE, 2010. 21, 37

[76] Jude W Shavlik, Thomas Dietterich, and Thomas Glen Dietterich. *Readings in machine learning*. Morgan Kaufmann, 1990. 9, 13

[77] Arti Singh, Baskar Ganapathysubramanian, Asheesh Kumar Singh, and Soumik Sarkar. Machine learning for high-throughput stress phenotyping in plants. *Trends in Plant Science*, 21(2):110–124, 2016. 8

[78] Arti Singh, Baskar Ganapathysubramanian, Asheesh Kumar Singh, and Soumik Sarkar. Machine learning for high-throughput stress phenotyping in plants. *Trends in plant science*, 21(2):110–124, 2016. 46

[79] Matthew Stephens. Dealing with label switching in mixture models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 62(4):795–809, 2000. xiv, 28

[80] Yee Whye Teh, Michael I Jordan, Matthew J Beal, and David M Blei. Hierarchical dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581, 2006. vii, 1, 33

[81] Aad W van der Vaart, J Harry van Zanten, et al. Reproducing kernel hilbert spaces of gaussian priors. In *Pushing the limits of contemporary statistics: contributions in honor of Jayanta K. Ghosh*, pages 200–222. Institute of Mathematical Statistics, 2008. 53

[82] Vasilis Verroios, Panagiotis Papadimitriou, Ramesh Johari, and Hector Garcia-Molina. Client clustering for hiring modeling in work marketplaces. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 2187–2196, New York, NY, USA, 2015. ACM. vii, 1

[83] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *J. Mach. Learn. Res.*, 11:2837–2854, December 2010. 39

[84] Ruohui Wang and Dahua Lin. Scalable estimation of dirichlet process mixture models on distributed data. *arXiv preprint arXiv:1709.06304*, 2017. viii, xiv, xv, 2, 26, 28, 29

[85] Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012. 21

[86] Sinead Williamson, Avinava Dubey, and Eric Xing. Parallel markov chain monte carlo for nonparametric mixture models. In *International Conference on Machine Learning*, pages 98–106, 2013. viii, xiv, 2, 26, 28

[87] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012. 23

[88] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010. xiii, xvi, 3, 20, 22, 37, 58

[89] Qiang Zhu, Gustavo Batista, Thanawin Rakthanmanon, and Eamonn Keogh. A novel approximation to dynamic time warping allows anytime clustering of massive time series datasets. In *Proceedings of the 2012 SIAM international conference on data mining*, pages 999–1010. SIAM, 2012. 51