Inférence distribuée pour les modèles de mélange de processus de Dirichlet dans l'apprentissage fédéré

Reda Khoufache [1] & Mustapha Lebbah [2] & Hanene Azzag [3] & Etienne Goffinet [4] & Djamel Bouchaffra[5]

[1] *DAVID, UVSQ, Université Paris-Saclay, Versailles. Email: reda.khoufache@uvsq.fr*
[2] *DAVID, UVSQ, Université Paris-Saclay, Versailles. Email: mustapha.lebbah@uvsq.fr*
[3] *LIPN, Université Sorbonne Paris Nord, France. Email: azzag@univ-paris13.fr*
[4] *Technology Innovation Institute, Abu Dhabi, UAE. Email: etienne.goffinet@tii.ae*
[5] *CDTA, Algérie. Email: djamel.bouchaffra@gmail.com*

Résumé. Cet article a déjà été publié dans [16]. Il présente une méthode d'inférence distribuée (DisCGS) pour les modèles de mélange de processus de Dirichlet (DPMM). Le DPMM est beaucoup utilisé pour résoudre les problèmes de (*clustering*), offrant l'avantage d'estimer automatiquement le nombre de *clusters* durant l'inférence via la modélisation bayésienne non paramétrique. Cependant, leur processus d'inférence est considérablement lent lorsque le nombre d'observations est grand. Notre approche, basée sur l'échantillonneur de Gibbs qui est une méthode de Monte-Carlo par chaînes Markov (MCMC), est conçue pour être exécutée sur un environnement distribué, notamment dans le contexte de l'apprentissage fédéré horizontal. La méthode DisCGS a montré des performances remarquables. Par exemple, pour 100 000 observations, notre approche atteint 100 itérations en seulement 3 minutes, soit un facteur de réduction du temps d'exécution de 200 par rapport à l'algorithme centralisé qui nécessite environ 12 heures. Le code source est accessible publiquement sur `https://github.com/redakhoufache/DisCGS`.

Mots-clés. Apprentissage fédéré, Calcul distribué, Modèles de mélange de processus de Dirichlet, Monté-Carlo par chaînes de Markov, Modélisation bayésienne non-paramétrique

Abstract. This paper has already been published in [16]. It introduces DisCGS, a distributed Markov Chain Monte Carlo (MCMC) inference method for Dirichlet Process Mixture Models (DPMMs) using sufficient statistics. Our method uses collapsed Gibbs sampling and is designed to work on distributed data across independent and heterogeneous machines, making it suitable for horizontal federated learning. Our approach demonstrates promising results and remarkable scalability. For instance, the centralized algorithm requires approximately 12 hours to complete 100 iterations while our approach achieves the same number of iterations in just 3 minutes, reducing the execution time by a factor of 200 without compromising clustering performance. The source code is publicly available at `https://github.com/redakhoufache/DisCGS`.

Keywords. Federated learning, Distributed computing, Dirichlet process mixture models, Markov Chain Monte Carlo, Bayesian non-parametric

1 Introduction

Les modèles de mélange de processus de Dirichlet (DPMM) représentent une extension des modèles de mélange vers une approche bayésienne non paramétrique. Ces modèles probabilistes génératifs supposent que les observations suivent une loi de mélange avec un nombre infini de composantes, dont les paramètres sont aléatoires et suivent une loi a priori. Ils sont largement utilisés pour traiter les problèmes de *clustering*, en particulier lorsque le nombre de *clusters* est inconnu car ils ont la capacité de l'estimer durant le processus d'inférence. L'algorithme de Gibbs [15] est une méthode MCMC qui infère les paramètres du DPMM en échantillonnant la variable latente de chaque observation selon la loi a posteriori. Bien que le DPMM offre l'avantage de découvrir de nouvelles structures et d'estimer automatiquement le nombre de *clusters*, l'étape d'inférence devient considérablement lente et ne passe pas à l'échelle lorsque le nombre d'observations est très grand. Ce qui limite son usage pour le traitement des données massives et son applicabilité dans des cas d'usages réels. Le calcul distribué consiste à distribuer les données sur des *workers*, ce qui permet d'effectuer des opérations parallèles, accélérant ainsi les calculs. L'apprentissage fédéré (FL), introduit dans [10], a révolutionné l'apprentissage distribué à grande échelle en entraînant les modèles directement sur des dispositifs locaux, que nous appelons "workers". L'une des caractéristiques du FL est que les données peuvent avoir des distributions différentes d'un *worker* à l'autre. Cette hétérogénéité nécessite des méthodes de *clustering* robustes. Par ailleurs, la nature décentralisée du FL pose des défis statistiques et informatiques, et la complexité des problèmes de *clustering* augmente considérablement dans ce contexte.

Les problèmes de *clustering* dans le contexte fédéré ont suscité un grand intérêt récemment. [11] a introduit une approche fédérée du K-Means, [19] a proposé un algorithme Fuzzy c-Means fédéré, tandis que [9] a présenté une méthode de *clustering* fédérée basée sur des modèles probabilistes. Plusieurs approches parallèles et distribuées d'inférence du DPMM ont été proposées dans la littérature. [12] a introduit un reparamétrage du processus de Dirichlet pour l'apprentissage des *clusters*, tandis que [22] a intégré des variables auxiliaires pour permettre la parallélisation. Cependant, [5] a montré que ces approches sont impraticables en raison de distributions déséquilibrées. Une autre méthode MCMC parallélisée a été présentée dans [1], combinant un échantillonneur de Gibbs restreint avec des propositions de division/fusion pour garantir l'ergodicité. Dans [3], cette approche a été étendue à un environnement distribué. [6] a introduit un algorithme d'inférence distribuée pour le DPMM basé sur l'échantillonnage par tranche. Une méthode d'estimation distribuée pour le DPMM est présentée dans [21], qui crée de nouvelles composantes localement au niveau des *workers*, suivies d'un schéma de consolidation probabiliste au niveau du *master*. [13] propose une approche d'inférence distribuée basée sur la méthode MCMC, utilisant un échantillonneur de Gibbs au niveau des *workers* pour découvrir de nouveaux *clusters*, et un autre échantillonneur sur les moyennes de chaque *cluster* au niveau du *master* pour l'agrégation des résultats. Cette méthode suppose des variances connues, limitant ainsi son applicabilité.

Dans cet article, nous présentons une nouvelle approche distribuée, DisCGS, basée sur les méthodes MCMC pour l'inférence du DPMM, en utilisant des statistiques suffisantes. Il est important de noter que notre approche se concentre sur un algorithme d'inférence spécifique pour le DPMM, le Gibbs Sampler proposé dans [15].

2 Modèle de mélange de processus de Dirichlet

Soient n et d deux entiers naturels, $\mathbf{x} = (x_1, \cdots, x_n)^T \in \mathbb{R}^{n \times d}$ le jeu de données, où $(\cdot)^T$ est l'opérateur de transposition. Soit $\mathbf{z} = (z_1, \cdots, z_n)$ le vecteur d'appartenance (la partition), où z_i est une variable latente telle que $z_i = k$ signifie que l'observation x_i appartient au *cluster* k. le DPMM suppose que les observations sont générées suivant ce modèle :

$$\begin{aligned} x_i \mid \{z_i = k, \theta_k\} &\overset{\text{i.i.d.}}{\sim} f(x_i, \theta_k), \forall i \in \{1, \cdots, n\} \\ \theta_k &\overset{\text{i.i.d.}}{\sim} G_0, \forall k \in \{1, 2, \cdots\} \\ z_i &\overset{\text{i.i.d.}}{\sim} \text{Mult}(\pi), \forall i \in \{1, \cdots, n\} \\ \pi &\sim SB(\alpha), \end{aligned}$$

Sous cette hypothèse, une observation x_i est générée en échantillonnant d'abord z_i suivant la loi multinoulli de paramètre $\pi = (\pi_k)_{k=1}^{\infty}$. Ensuite, x_i est générée suivant $f(x_i, \theta_{z_i})$, où $f(\cdot, \theta_k)$ est la densité associée au *cluster* k dont le paramètre θ_k suit une loi a priori G_0 (distribution de base); dans le cas Gaussien multivarié, nous avons $\theta_k = (\mu_k, \Sigma_k)$ avec $\mu_k \in \mathbb{R}^d$ et $\Sigma_k \in \mathbb{R}^{d \times d}$ une matrice symétrique semi-définie positive. Le vecteur π suit le processus du *Stick-Breaking* [17] de paramètre de concentration $\alpha > 0$. Dans ce qui suit, nous supposons que f est une distribution gaussienne multivariée et que G_0 est l'a priori conjugué Normal Inverse Wishart [14] (NIW) de paramètres $(\mu_0, \kappa_0, \Psi_0, \nu_0)$. Nous allons noter $\Theta = \{\theta_k,\ k > 1\}$ l'ensemble des paramètres et $\Omega = (\alpha, G_0)$ l'ensemble des hyperparamètres.

Inférence. L'algorithme de Gibbs [15] permet d'inférer les paramètres via la méthode de Monte-Carlo. Il alterne entre la mise à jour de la partition $\mathbf{z}$ et celle des paramètres associés à chaque *cluster*. Pendant la première étape, l'échantillonneur de Gibbs simule la loi jointe a posteriori $\text{p}(\mathbf{z} \mid \mathbf{x}, \Theta, \Omega)$ en tirant chaque z_i suivant la loi conditionnelle $\text{p}(z_i \mid \mathbf{x}, \mathbf{z}_{-i}, \Theta, \Omega)$, où $\mathbf{z}_{-i} = \{z_l,\ l \neq i\}$. Durant la seconde étape, les paramètres associés au *cluster* k sont tirés selon la loi a posteriori $\text{p}(\theta_k | \mathbf{x}_k, G_0)$, avec $\mathbf{x}_k = \{x_i,\ z_i = k\}$ le contenu du *cluster* k. Le calcul de cette distribution peut se faire analytiquement comme G_0 est la loi a priori conjuguée pour la densité f. Également, il est possible d'intégrer les paramètres θ_k, ce qui permet de sauter l'étape de mise à jour des paramètres, ainsi seuls les variables latentes z_i sont mises à jour. Ceci nous amène à une variante de l'algorithme de Gibbs qui correspond au 3 ème algorithme proposé dans [15].

3 La méthode proposée

L'objectif de notre approche consiste à distribuer le processus d'inférence sans compromettre les performances de l'estimateur initial. Pour ce faire, après avoir distribué uniformément les données entre les *workers*, notre algorithme alterne entre des phases d'inférence aux niveaux *master* et *worker*.

3.1 DisCGS au niveau *worker*

Soit $\mathbf{x}^j = \{x_1^j, \cdots, x_{n^j}^j\}$ l'ensemble d'observations assignées au j-ième *worker*, n^j le cardinal de $\mathbf{x}^j$, et $\mathbf{z}^j = (z_1^j, \cdots z_{n^j}^j)$ la partition locale où z_i^j est une variable latente locale de sorte que $z_i^j = k$ signifie que l'observation x_i^j (assignée au *worker* j) appartient au *cluster* local k. Les variables latentes locales sont mises à jour en utilisant la variante de l'échantillonneur de Gibbs introduit dans la section 2. Chaque z_i^j est échantillonné selon $\mathrm{p}(z_i^j \mid \mathbf{z}_{-i}^j, \mathbf{x}^j, \Omega) \propto$

$$\begin{cases} n_k^j \mathrm{p}(x_i^j \mid z_i^j = k, \mathbf{x}_k^j, G_0), & \textit{cluster} \text{ existant } k, \quad (1) \\ \alpha \mathrm{p}(x_i^j \mid \Omega), & \text{nouveau } \textit{cluster}, \quad (2) \end{cases}$$

où n_k^j et $\mathbf{x}_k^j$ sont respectivement la taille et le contenu du *cluster* k du *worker* j. Sous l'hypothèse de conjugaison, les distributions prédictives a posteriori et a priori (équations 1 et 2 respectivement) sont calculées analytiquement [14]. Après avoir mis à jour la partition, nous calculons les statistiques suffisantes [18] associées à chaque *cluster*. Dans le cas Gaussien multivarié, les statistiques suffisantes (T_k^j, S_k^j) pour un *cluster* $\mathbf{x}_k^j$ sont données par [7]:

$$T_k^j = \frac{1}{n_k^j} \sum_{x \in \mathbf{x}_k^j} x \in \mathbb{R}^d, \quad (3)$$

$$S_k^j = \sum_{x \in \mathbf{x}_k^j} (x - T_k^j)(x - T_k^j)^T \in \mathbb{R}^{d \times d}. \quad (4)$$

Enfin, les statistiques suffisantes et les tailles de chaque *cluster* sont envoyées au *master*.

3.2 DisCGS au niveau *master*

Le *master* reçoit de chaque *worker* les tailles et les statistiques suffisantes associées à chaque *cluster*. L'objectif est d'estimer le vecteur d'appartenance global $\mathbf{z} = (z_1, \cdots, z_n)$ et de mettre à jour les hyperparamètres associés à chaque *cluster*. À ce niveau, au lieu d'assigner les observations une par une à leur *cluster* global, nous assignons un groupe d'observations qui partagent déjà le même *cluster* local (c'est-à-dire au niveau du *worker*) à un *cluster* global au niveau du *master*. Ainsi, les observations assignées au même *cluster* global auront la même étiquette. Nous échantillonnons la variable latente globale z_h^j du *cluster* $\mathbf{x}_h^j$ (*cluster* local h du *worker* j) suivant $\mathrm{p}(z_h^j \mid \mathbf{z}_{-h}^{-j}, \mathbf{x}, \Omega) \propto$

$$\begin{cases} n_k \mathrm{p}(\mathbf{x}_h^j \mid z_h^j = k, \mathbf{x}_k, G_0), & \textit{cluster} \text{ existant } k, \quad (5) \\ \alpha \mathrm{p}(\mathbf{x}_h^j \mid G_0), & \text{nouveau } \textit{cluster}. \quad (6) \end{cases}$$

Les distributions prédictives a posteriori et a priori jointes (équations 5 et 6 respectivement) sont calculées analytiquement en utilisant uniquement les statistiques suffisantes, c'est-à-dire sans avoir accès au contenu du *cluster* $\mathbf{x}_h^j$. En effet, nous avons:

$$\mathrm{p}\left(\mathbf{x}_h^j \mid \Omega\right) = \pi^{-n_h^j \frac{d}{2}} \cdot \frac{\kappa_0^{d/2}}{\left(\kappa_h^j\right)^{d/2}} \cdot \frac{\Gamma_d\left(\nu_h^j/2\right)}{\Gamma_d\left(\nu_0/2\right)} \cdot \frac{|\Psi_0|^{\nu_0/2}}{\left|\Psi_h^j\right|^{\nu_h^j/2}}$$

où $|\cdot|$ est le déterminant, et les hyperparamètres $(\mu_h^j, \kappa_h^j, \Psi_h^j, \nu_h^j)$ sont obtenus comme suit:

$$\mu_h^j = \frac{\kappa_0 \mu_0 + n_h^j T_h^j}{\kappa_h^j}, \quad \kappa_h^j = \kappa_0 + n_h^j, \quad \nu_h^j = \nu_0 + n_h^j,$$
$$\Psi_h^j = \Psi_0 + S_h^j + \frac{\kappa_0 n_h^j}{\kappa_h^j} \left(\mu_0 - T_h^j\right) \left(\mu_0 - T_h^j\right)^T,$$

où T_h^j et S_h^j sont les statistiques suffisantes renvoyées par les *workers*. Par ailleurs, on a:

$$\mathrm{p}(\mathbf{x}_h^j \mid z_h^j = k, \mathbf{x}_k, G_0) = \pi^{\frac{-d n_h^j}{2}} \cdot \frac{\kappa_k^{d/2}}{\left(\kappa_h^j\right)^{d/2}} \cdot \frac{\Gamma_d\left(\nu_h^j/2\right)}{\Gamma_d\left(\nu_k/2\right)} \cdot \frac{|\Psi_k|^{\nu_k/2}}{\left|\Psi_h^j\right|^{\nu_h^j/2}}$$

où les paramètres de la loi a posteriori $(\mu_k, \kappa_k, \Psi_k, \nu_k)$ associés au *cluster* global k, sont mis à jour à partir de la loi a priori ainsi:

$$\mu_k = \frac{\kappa_0 \mu_0 + n_k T_k}{\kappa_k}, \quad \kappa_k = \kappa_0 + n_k, \quad \nu_k = \nu_0 + n_k,$$
$$\Psi_k = \Psi_0 + S_k + \frac{\kappa_0 n_k}{\kappa_k} \left(\mu_0 - T_k\right) \left(\mu_0 - T_k\right)^T,$$

avec T_k et S_k, les statistiques suffisantes agrégées lorsque des *clusters* locaux sont assignées au même *cluster* global k sont calculées comme suit:

$$T_k = \frac{1}{n_k} \sum_{j,h \mid \mathbf{z_h^j = k}} n_h^j \cdot T_h^j,$$
$$S_k = \sum_{j,h \mid \mathbf{z_h^j = k}} S_h^j + \sum_{j,h \mid \mathbf{z_h^j = k}} \left(n_j^h \cdot T_h^j \cdot T_h^{j^T}\right) - n_k \cdot T_k \cdot T_k^T.$$

Le pseudo-code du processus d'inférence au niveau *worker* et *master* sont donnés dans [16].

3.3 L'échantillonneur de Gibbs dans l'apprentissage fédéré

Dans le contexte fédéré, les observations se retrouvent sur différents composants (*workers*). Cela correspond à la décomposition horizontale des données. Chaque *worker* est initialisé avec le même modèle global. Ensuite, chaque *worker* met à jour son modèle local en utilisant ses données privées via l'échantillonneur de Gibbs détaillé dans la section 3.1; cette étape permet de découvrir les *clusters* locaux et d'inférer le DPMM local. Ensuite, les statistiques suffisantes et les tailles associées à chaque *cluster* local sont calculées et transmises au serveur (*master*). Ce dernier procède à la mise à jour du modèle global et à l'estimation de la partition globale sans avoir accès aux données. Ce processus est réalisé en utilisant l'échantillonneur de Gibbs décrit dans la section 3.2. Le modèle à jour est ensuite partagé avec chaque composant. Ce processus itératif se poursuit en alternant ces deux étapes jusqu'à ce que le modèle global soit estimé.

Jeu de données	n	d	K	Description
Synthetic 10K	10000	2	6	Jeu de donnnées synthetique généré selon des composantes gaussiennes de dimension 2.
Mnist	70000	8	10	Chiffres écrits à la main, Lecun et al.
Fashion mnist	70000	8	10	Images d'articles Zalando [23].
Balanced	131600	8	47	Chiffres et lettres écrits à la main [2].
Digits	280000	8	10	Chiffres écrits à la main [2].
UrbanGB	360177	3	469	Coordonnées (longitude et latitude) d'accidents de la route [4].

Table 1: Description des jeux de données utilisés pour évaluer les performances *clustering* de notre approche. n est le nombre d'observations, d la dimension, K le nombre de *clusters*.

4 Expériences

Pour évaluer notre approche, nous présentons trois expériences sur des jeux de données synthétiques et réels: d'abord un *benchmark* comparant les performances de *clustering* et la vitesse de convergence, puis une comparaison des temps d'exécution de l'algorithme centralisé et distribué, et enfin, une expérience qui évalue le passage à l'échelle de notre méthode.

4.1 Implémentation et environnement distribué

Dans ce qui suit, nous utilisons la loi a priori non informative pour les algorithmes CGS et DisCGS. Ainsi, nous posons μ_0 et la matrice de précision Ψ_0 égaux au vecteur moyen et à la matrice de covariance des données. κ_0 et ν_0 représentent notre confiance en μ_0 et Ψ_0, et sont fixés à leurs plus petites valeurs, qui sont 1 et $d+1$. L'état initial est une partition à un *cluster*. Nous avons exécuté les algorithmes en utilisant la machine Neowise (1 CPU AMD EPYC 7642, 48 cores/CPU) hébergée par le *cluster* grid5000 [1].

4.2 Performances de *clustering*

Dans cette expérience, nous comparons DisCGS avec deux algorithmes distribués: M-R [6] et SubC [3], ainsi qu'avec deux algorithmes parallélisés: Kmeans et GMM. Toutes les exécutions sont effectuées en distribuant les données sur 32 cœurs. Nous utilisons 6 jeux de données décrits dans le tableau 1. Les jeux de données d'images sont encodés en un vecteur de dimension 8 en utilisant un auto-encodeur variationnel. Pour évaluer les performances de *clustering*, nous calculons les trois métriques de *clustering*; l'indice de Rand ajusté (ARI) [8], l'information mutuelle normalisée (NMI) [20], et l'*accuracy* (ACC) [24].

La table 2 présente la moyenne et l'écart type des trois métriques obtenues par chaque méthode sur 10 essais. Les résultats montrent que notre méthode proposée surpasse les autres méthodes ou obtient le deuxième meilleur score sur presque tous les jeux de données. Il est important de noter que dans cette expérience, nous nous concentrons uniquement sur la comparaison des performances de *clustering*; nous ne comparons pas les temps d'exécution car les autres approches proposent une inférence différente de l'échantillonneur de Gibbs.

[1] `https://www.grid5000.fr/w/Grid5000:Home`

Jeu de données		DisCGS	M-R	SubC	GMM	Kmeans
Synthetic 10K	ARI	**0.80** ∓ 0.06	0.10 ∓ 0.01	0.38 ∓ 0.07	0.80 ∓ 0.07	0.75 ∓ 0.03
	NMI	0.86 ∓ 0.04	0.12 ∓ 0.01	0.61 ∓ 0.08	**0.88** ∓ 0.04	0.85 ∓ 0.01
	ACC	**0.88** ∓ 0.06	0.29 ∓ 0.01	0.38 ∓ 0.08	0.85 ∓ 0.08	0.76 ∓ 0.05
Mnist	ARI	**0.72** ∓ 0.01	0.20 ∓ 0.07	0.66 ∓ 0.04	0.39 ∓ 0.02	0.26 ∓ 0.01
	NMI	0.74 ∓ 0.00	0.38 ∓ 0.08	**0.79** ∓ 0.01	0.69 ∓ 0.01	0.62 ∓ 0.00
	ACC	**0.79** ∓ 0.01	0.30 ∓ 0.06	0.71 ∓ 0.03	0.38 ∓ 0.02	0.23 ∓ 0.01
Fashion-Mnist	ARI	**0.45** ∓ 0.02	0.35 ∓ 0.02	0.40 ∓ 0.02	0.41 ∓ 0.02	0.37 ∓ 0.02
	NMI	**0.60** ∓ 0.01	0.54 ∓ 0.01	**0.60** ∓ 0.01	0.59 ∓ 0.02	0.57 ∓ 0.01
	ACC	**0.55** ∓ 0.02	0.45 ∓ 0.03	0.48 ∓ 0.01	0.52 ∓ 0.04	0.48 ∓ 0.01
Balanced	ARI	**0.35** ∓ 0.00	0.02 ∓ 0.01	0.05 ∓ 0.02	0.35 ∓ 0.01	0.22 ∓ 0.00
	NMI	0.59 ∓ 0.00	0.17 ∓ 0.06	0.31 ∓ 0.04	**0.63** ∓ 0.00	0.54 ∓ 0.00
	ACC	**0.46** ∓ 0.01	0.07 ∓ 0.02	0.10 ∓ 0.02	0.44 ∓ 0.02	0.30 ∓ 0.01
Digits	ARI	0.55 ∓ 0.01	0.28 ∓ 0.14	**0.74** ∓ 0.05	0.46 ∓ 0.02	0.36 ∓ 0.01
	NMI	0.71 ∓ 0.01	0.51 ∓ 0.14	**0.79** ∓ 0.02	0.71 ∓ 0.01	0.63 ∓ 0.00
	ACC	0.58 ∓ 0.00	0.38 ∓ 0.09	**0.81** ∓ 0.05	0.44 ∓ 0.03	0.34 ∓ 0.01
UrbanGB	ARI	0.63 ∓ 0.01	0.12 ∓ 0.05	0.09 ∓ 0.00	**0.67** ∓ 0.05	0.49 ∓ 0.06
	NMI	0.68 ∓ 0.01	0.21 ∓ 0.07	0.24 ∓ 0.00	0.77 ∓ 0.01	**0.81** ∓ 0.01
	ACC	0.45 ∓ 0.01	0.29 ∓ 0.02	0.29 ∓ 0.00	0.53 ∓ 0.02	**0.54** ∓ 0.02

Table 2: La moyenne et l'écart type des trois métriques ARI, NMI et ACC, sur 10 exécutions. Le meilleur résultat dans chaque ligne est marqué en gras, et le deuxième meilleur est souligné.

4.3 Convergence

Maintenant, nous comparons les performances des modèles qui partagent les mêmes hypothèses: CGS et SubC sur les bases de données Synthetic 100K, Fashion-mnist et Balanced. La Figure 1 illustre la log-vraisemblance et le score ARI à chaque itération. Nous observons que notre algorithme converge presque à la même vitesse que l'algorithme centralisé CGS et est beaucoup plus rapide que SubC. Cela est dû au fait que notre algorithme est capable de découvrir de nouveaux *clusters* au niveau des *workers*. Or, dans SubC, le nombre de *clusters* est fixe à ce niveau, et les nouvelles composantes ne sont découvertes qu'au niveau du *master*. Ainsi, plus d'itérations sont nécessaires pour générer suffisamment de composantes qui représentent les données. Dans l'ensemble, notre algorithme converge rapidement et maintient un score ARI stable au fil des itérations. Alors que le CGS et SubC peuvent dégrader leur score comme on peut l'observer sur le jeu de données Fashion-Mnist.

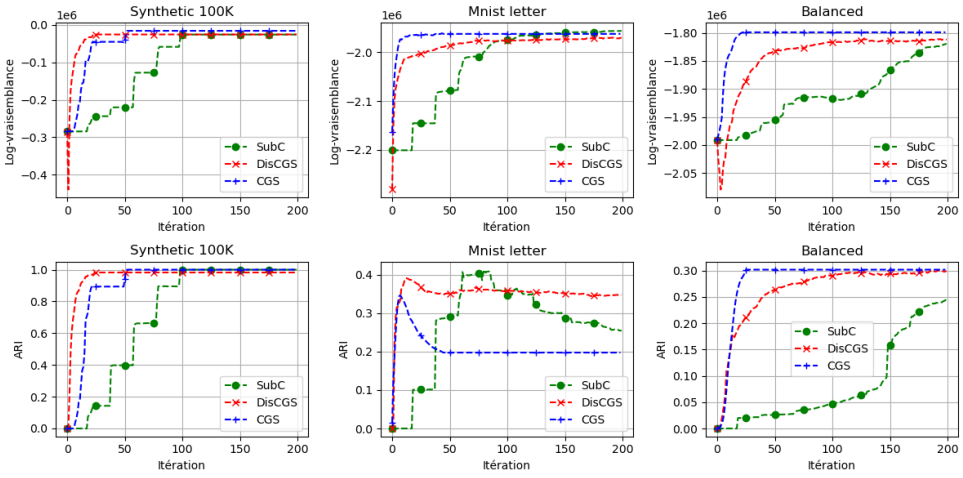

Figure 1: La log-vraisemblance et le score ARI à chaque itération.

4.4 Comparaison de la méthode distribuée et centralisée

Dans cette expérience, nous comparons le temps d'exécution et les performances de *clustering* de l'algorithme distribué (DisCGS) et centralisé (CGS). Nous exécutons les deux algorithmes sur des jeux de données synthétiques de tailles différentes (de $n = 20$K à $n = 100$K) générés à partir de $K = 10$ composantes gaussiennes de dimension 2. Les résultats présentés dans le tableau 3 montrent le gain significatif du temps d'exécution de l'approche distribuée sans que celle-ci compromette les performances de *clustering*. Par exemple, le temps de calcul est divisé par 200 sur le jeu de données de taille 100k. De plus, notre méthode obtient des scores plus élevés dans 80% des cas.

Nombre d'observations	ARI		NMI		ACC		Temps d'exécution	
	Dis.	Cen.	Dis.	Cen.	Dis.	Cen.	Dis.	Cen.
20K	**0.99**	0.89	**0.99**	0.96	**0.99**	0.89	**66.53**	1704.21
40K	0.96	**0.99**	0.97	**0.99**	**0.99**	0.97	**99.45**	10898.28
60K	**0.91**	0.89	0.92	**0.96**	**0.92**	0.89	**135.76**	25738.46
80K	**0.94**	0.89	**0.96**	**0.96**	**0.91**	0.89	**201.59**	27492.08
100K	**0.91**	0.89	**0.94**	0.89	**0.91**	0.89	**207.58**	44688.31

Table 3: Métriques de *clustering* et temps d'exécution obtenus par l'inférence distribuée (Dis.) et centralisée (Cen.) sur des jeux de données synthétiques de différentes tailles.

4.5 Passage à l'échelle

Dans cette expérience, nous utilisons $n = 10^6$ points de données générés à partir de $K = 10$ composantes gaussiennes bidimensionnelles. Nous exécutons notre inférence distribuée plusieurs fois en augmentant le nombre de cœurs de 8 à 48. La Figure 2 représente le temps d'exécution en fonction du nombre de cœurs. Nous observons que le temps d'exécution diminue significativement lorsque le nombre de cœurs augmente, montrant que notre algorithme passe efficacement à l'échelle lorsque le nombre de coeurs augmente.

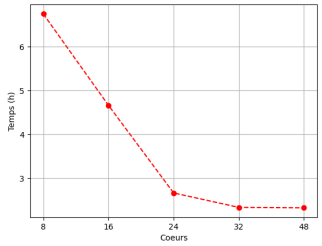

Figure 2: Temps d'exécution (h) en fonction du nombre de cœurs de DisCGS.

5 Conclusion et perspectives

Cet article introduit une nouvelle méthode MCMC distribuée pour les DPMM, spécialement conçue pour des données distribuées sur des machines indépendantes et hétérogènes, adaptée aux scénarios d'apprentissage horizontal fédéré. Les résultats expérimentaux sont prometteurs, démontrant une réduction significative du temps d'inférence tout en maintenant des performances satisfaisantes. Nos recherches en cours exploitent cette approche pour distribuerles modèles à blocs latents non paramétriques.

6 Remerciement

Projet soutenu par la Région Île-de-France dans le cadre du DIM AI4IDF. Je remercie Grid5000 pour avoir fourni les ressources de calculs nécessaires et la start-up HephIA pour l'échange sur les algorithmes distribués.

Bibliographie

[1] J. Chang and J. W. Fisher III. Parallel sampling of dp mixture models using sub-cluster splits. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[2] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik. Emnist: Extending mnist to handwritten letters. In *International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926, 2017.

[3] O. Dinari, A. Yu, O. Freifeld, and J. W. Fisher III. Distributed mcmc inference in dirichlet process mixture models using julia. In *19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 518–525, 2019.

[4] D. Dua and C. Graff. UCI machine learning repository.

[5] Y. Gal and Z. Ghahramani. Pitfalls in the use of parallel inference for the dirichlet process. In E. P. Xing and T. Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 208–216, Bejing, China, 22–24 Jun 2014. PMLR.

[6] H. Ge, Y. Chen, M. Wan, and Z. Ghahramani. Distributed inference for dirichlet process mixture models. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2276–2284, Lille, France, 07–09 Jul 2015.

[7] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian data analysis*. 1995.

[8] L. Hubert and P. Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[9] C. Jichan, K. Lee, and R. Kannan. Federated unsupervised clustering with generative models. *In AAAI 2022 International Workshop on Trustable, Verifiable and Auditable Federated Learning*, 2022.

[10] J. Konečný, H. B. McMahan, D. Ramage, and P. Richtárik. Federated optimization: Distributed machine learning for on-device intelligence, 2016.

[11] H. H. Kumar, K. V R, and M. K. Nair. Federated k-means clustering: A novel edge ai based approach for privacy preservation. In *IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pages 52–56, 2020.

[12] D. Lovell, J. Malmaud, R. Adams, and V. Mansinghka. Clustercluster: Parallel markov chain monte carlo for dirichlet process mixtures. In *In Workshop on Big Learning, NIPS, 2012*.

[13] K. Meguelati, B. Fontez, N. Hilgert, and F. Masseglia. Dirichlet process mixture models made scalable and effective by means of massive distribution. pages 502–509, 04 2019.

[14] K. P. Murphy. Conjugate bayesian analysis of the gaussian distribution. *def*, 1(2σ2):16, 2007.

[15] R. M. Neal. Markov chain sampling methods for dirichlet process mixture models. *Journal of Computational and Graphical Statistics*, 9(2):249–265, 2000.

[16] K. Reda, L. Mustapha, A. Hanene, G. Etienne, and B. Djamel. Distributed collapsed gibbs sampler for dirichlet process mixture models in federated learning. In *Proceedings of the 2024 SIAM International Conference on Data Mining (SDM)*, 2024.

[17] J. Sethuraman. A constructive definition of dirichlet priors. *Statistica Sinica*, 4(2):639–650, 1994.

[18] S. Silvey. *Statistical Inference*. CRC Press, 2017.

[19] M. Stallmann and A. Wilbik. Towards federated clustering: A federated fuzzy c-means algorithm (FFCM). *CoRR*, abs/2201.07316, 2022.

[20] A. Strehl and J. Ghosh. Cluster ensembles - a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, 01 2002.

[21] R. Wang and D. Lin. Scalable estimation of dirichlet process mixture models on distributed data. In *International Joint Conference on Artificial Intelligence*, 2017.

[22] S. Williamson, A. Dubey, and E. Xing. Parallel Markov chain Monte Carlo for nonparametric mixture models. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 98–106, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[23] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[24] Y. Yang, D. Xu, F. Nie, S. Yan, and Y. Zhuang. Image clustering using local discriminant models and global integration. *IEEE Transactions on Image Processing*, 19(10):2761–2773, 2010.