DEEP DIRICHLET PROCESS MIXTURE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper we propose the deep Dirichlet process mixture (DDPM) model, which is an unsupervised method that simultaneously performs clustering and feature learning. As a member of the Bayesian nonparametrics family, the traditional Dirichlet process mixture model is able to adapt the number of its mixture components. However its modelling capacity is restricted since the clustering is performed in the raw feature space, rendering it inapplicable to complex domains like images and texts. Our method alleviates this limitation by using the flow-based generative model, which is a deep invertible neural network, to learn more expressive features. These two seemly orthogonal models are unified by the Monte Carlo expectation-maximization algorithm, and during its iterations Gibbs sampling is used to generate samples from the posterior. This combination allows our method to exploit the mutually beneficial relation between clustering and feature learning. We conducted comparison experiments on four clustering benchmark datasets. The clustering performance of DDPM shows a significant gain over DPM in most cases and is competitive compared to other popular methods. Furthermore, the learned representation of DDPM is shown to be efficient and universal to boost other methods' performance.

1 INTRODUCTION

Clustering is one of the most long-standing and fundamental tasks in computer science. Besides the well-known k-means algorithm (MacQueen (1967)) and Gaussian mixture models (GMMs) (Bishop (2006)), a plethora of methods have been proposed. In those early investigations clustering is performed in the raw feature space, and the models lack the capacity of learning or improving the expressiveness of the representation.

With the recent success of deep neural networks (DNNs), a new line of research termed deep clustering has emerged (Xie et al. (2016); Yang et al. (2016); Jiang et al. (2017); Caron et al. (2018)). These works are based on the intuitive principal that good representation encourages better clustering, and reversely good clustering can lead to better representation. Their methods take advantage of the success network structures, such as convolution neural networks (CNNs) (Krizhevsky et al. (2012)) and variational autoencoders (VAEs) (Kingma & Welling (2014)). Their superior performance demonstrates the benefits of jointly clustering and representation learning.

However, all these methods share the same insufficiency – they consider the number of clusters (i.e. k in k-means) as a hyperparameter that need to be specified by the user. This brings severe restriction to their applications: 1) most of these algorithms are sensitive to the choice of k; 2) users lack the prior knowledge of the number of clusters; 3) this value itself may be time-varying (e.g. k may increase as more data is accumulated); 4) In some scenarios, particularly for large dataset, there is no golden ground truth for this value (Li et al. (2018)).

Dirichlet process mixture (DPM) models, which belong to the Bayesian nonparametrics family, is a popular method that could solve this conundrum (Antoniak (1974)). Its solid mathematical background originates from the Dirichlet process (Ferguson (1973)), which is an infinite generalization of the Dirichlet distribution. DPM is able to model the data with possibly infinite number of mixtures, and the exact value of mixtures is rigorously inferred by the Bayesian principle. Furthermore, DPM also possesses the ability to adapt the number of mixtures as more data arrives. It is thus desirable to combine the strengths of DPM and deep neural networks, which could lead to a clustering method that simultaneously adjusts its number of mixture components and learns better representation.

In this paper we propose the deep Dirichlet process mixture (DDPM) model, which brings the above idea into realization. Our method bridges the standard DPM with the recently proposed flow-based generative models (Dinh et al. (2015); Kingma & Dhariwal (2018)), which is a special kind of invertible deep neural networks that learns better representation through density transformation. The overall clustering process is guided by the Bayesian principle, while the optimization of model parameters is derived from the Monte Carlo expectation-maximization (EM) algorithm (Bishop (2006)). During the iterations, Gibbs sampling is used to obtain samples from the posterior as in the standard DPM literature. DDPM also works as a generative model, so that unseen new samples could be obtained by introducing noises to the learned features.

2 RELATED WORK

Clustering and representation learning are both among the most well-investigated topics in computer science. Besides the well-known k-means (MacQueen (1967)), GMMs (Bishop (2006)) and their variants, the recent success of deep neural networks has inspired a new paradigm called deep clustering. The works of (Yang et al. (2016); Caron et al. (2018)) try to take advantage of the convolution neural networks in computer vision tasks. Their major difference lies in the loss function and the training scheme. Other DNN structures are also exploited. Xie et al. (2016) proposed Deep Embedded Clustering (DEC), which jointly optimizes deep embeddings and performs clustering based on features extracted from deep autoencoders. In (Jiang et al. (2017)) Variational Deep Embedding (VaDE) was introduced, which combines the variational autoencoder with the GMM model. However all these works share the same insufficiency, i.e., the number of the clusters is a hyperparameter that need to be specified by the user. As aforementioned this presents challenges for their applications in real world scenarios, where prior knowledge about the number of clusters is generally unavailable.

Dirichlet process mixture models (Antoniak (1974)), which belong to the Bayesian nonparametrics family, is one of the most popular methods that are capable of inferring the number of clusters automatically. This distinguishing ability is further utilized and strengthened. Teh et al. (2006) proposed Hierarchical Dirichlet processes, which can share mixture components among different clusters. The maximum margin DPM (MMDPM) introduces a discriminate model for clustering, which bridges DPM and the SVM classifier (Chen et al. (2016)). All these works operate in the raw feature space, without the ability to learn more expressive representation.

Naturally one may wonder whether it is possible to simultaneously decide a clustering model's capacity and learn better (possibly nonlinear) features. Ehsan Abbasnejad et al. (2017) proposed to use an infinite mixture of VAEs to model the data. Since the number of effective VAEs may increase as more data arrives, their model can adapt to the complexity of the data. In the paper of (Nalisnick & Smyth (2017)) stick-breaking variational autoencoders (SB-VAEs) were presented, which model the latent variables in VAEs to be infinite dimensional. Dirichlet processes and particularly the stick-breaking construction serve as the cornerstones in their method. Our work distinguishes from theirs in both goal and methodology: their works focus on improving the performance of semi-supervised classification tasks, while our research considers the unsupervised task of clustering where the number of mixtures and better representation need to be jointly learned. Both of their models are based on VAEs while our method utilizes the recently proposed invertible deep neural network, which demonstrates superior performance in various density estimation and computer vision tasks (Dinh et al. (2015); Kingma & Dhariwal (2018)).

3 METHODOLOGY

3.1 OVERVIEW

Consider the input as a set $\mathbf{X} = \{\mathbf{x}_i \in \mathbb{R}^D\}_{i=1}^N$. Our first step is to use a standard dimension reduction technique to extract representative features, so that the following clustering can be performed in the lower dimensional **feature space**. In our work we use the stack autoencoder (Vincent et al. (2010); Xie et al. (2016)) to extract the features as $\mathbf{Y} = \{\mathbf{y}_i = h_e(\mathbf{x}_i) \in \mathbb{R}^d\}_{i=1}^N$ ($d \ll D$), where h_e and h_d denote the encoder and decoder functions respectively such that $h_d(h_e(\mathbf{x})) \approx \mathbf{x}$.

Next the features are transformed by a nonlinear learnable function $f(\mathbf{y};\boldsymbol{\theta})$ into $\mathbf{Z} = \{\mathbf{z}_i = f(\mathbf{y}_i;\boldsymbol{\theta}) \in \mathbb{R}^d\}_{i=1}^N$. Clustering is performed in this **transformed space**. We assume that each $\mathbf{z}_i$ follows an isotropic Gaussian distribution. Suppose that $\mathbf{z}_i$ belongs to the k-th cluster, the likelihood is given by:

$$p(\mathbf{z}_i|\boldsymbol{\mu}_k, \lambda_k) = \mathcal{N}(\mathbf{z}_i|\boldsymbol{\mu}_k, \lambda_k^{-1}\mathbf{I}) = (\frac{\lambda_k}{2\pi})^{\frac{d}{2}} \exp\left(-\frac{\lambda_k}{2}||\mathbf{z}_i - \boldsymbol{\mu}_k||^2\right), \tag{1}$$

where $\boldsymbol{\mu}_k$ and λ_k denote the mean and the precision of the k-th cluster. Note that the isotropic Gaussian assumption does not restrict the model's capacity since the transformation $f(\mathbf{y}_i;\boldsymbol{\theta})$ is nonlinear, which is implemented by a deep neural network in practice. The cluster assignment variables are denoted as $\mathbf{c} = \{c_i \in \{1, ..., K\}\}_{i=1}^N$, where $c_i = k$ indicates that $\mathbf{z}_i$ belongs to the k-th cluster. Here the total number of clusters K is unknown to us and could be arbitrarily large.

From the Bayesian perspective, the task of clustering is equivalent to the inference of

$$p(\mathbf{c}, \{\boldsymbol{\mu}_k\}_{k=1}^K, \{\lambda_k\}_{k=1}^K|\mathbf{Y};\boldsymbol{\theta}, \boldsymbol{\Phi}). \tag{2}$$

Here $\boldsymbol{\Phi}$ is the set of hyperparameters that specify the prior, which will soon be introduced in the next section. We emphasize the challenges of the task: 1) the number of clusters K is unknown; 2) $\boldsymbol{\theta}$ parameterizes a deep neural network which needs to be learned; 3) the cluster information (i.e. $\boldsymbol{\mu}_k$ and λ_k) need to be computed at the same time. In what follows we will see how the proposed method can address all these challenges under a unified framework.

3.2 Model specification

Likelihood in the feature space Eq. (1) describes the likelihood function in the transformed space. Now we derives the likelihood in the feature space (i.e. before the transformation) as follows:

$$\begin{aligned} p(\mathbf{Y}|\mathbf{c}, \{\boldsymbol{\mu}_k\}_{k=1}^K, \{\lambda_k\}_{k=1}^K;\boldsymbol{\theta}, \boldsymbol{\Phi}) &= \prod p(\mathbf{y}_i|\boldsymbol{\mu}_{c_i}, \lambda_{c_i};\boldsymbol{\theta}, \boldsymbol{\Phi}) \\ &= \prod p(\mathbf{z}_i|\boldsymbol{\mu}_{c_i}, \lambda_{c_i})|\det \frac{\partial f(\mathbf{y}_i;\boldsymbol{\theta})}{\partial \mathbf{y}_i}|. \end{aligned} \tag{3}$$

The last term in Eq. (3) is due to the change of variable $\mathbf{z}_i = f(\mathbf{y}_i;\boldsymbol{\theta})$. Recall that $p(\mathbf{z}_i|\boldsymbol{\mu}_{c_i}, \lambda_{c_i})$ is given in Eq (1).

For a general nonlinear function $f(\mathbf{y}_i;\boldsymbol{\theta})$ implemented by a neural network, the Jacobian term in Eq. (3) is analytically intractable. To address this problem we utilize the NICE model (Dinh et al. (2015)), which is a special kind of deep neural network with the appealing property that the determinant of Jacobian can be trivially computed. The basic idea of NICE is that, in each layer it splits the input $\mathbf{x}$ into two parts as $\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2\}$, and defines the output as $\mathbf{y} = \{\mathbf{y}_1, \mathbf{y}_2\}$ where $\mathbf{y}_1 = \mathbf{x}_1$ and $\mathbf{y}_2 = \mathbf{x}_2 + \sigma(\mathbf{x}_1)$. It is easy to verify that for such function the determinant of Jacobian equals one. After stacking multiple such layers, we have a highly nonlinear function whose Jacobian term can be trivially cancelled out. Another useful property of NICE is that the transformation is invertible. Particularly if $\mathbf{z}_i = f(\mathbf{y}_i;\boldsymbol{\theta})$, $\mathbf{y}_i = f^{-1}(\mathbf{z}_i;\boldsymbol{\theta})$ can also be easily computed. Interested readers may refer to (Dinh et al. (2015)) for further details.

Dirichlet process mixture model in the transformed space After the transformation $\mathbf{z}_i = f(\mathbf{y}_i;\boldsymbol{\theta})$, we can now perform clustering in this transformed space. In our problem the number of clusters K is unknown. The Dirichlet process mixture (DPM) models (Antoniak (1974); Neal (2000); Li et al. (2019)) is one of the most popular tools in this situation. Here we present a concise review of DPM models, and in the next subsection we will see how to connect it with NICE.

Given a measurable space $(\Theta, \mathcal{A})$ where $\mathcal{A}$ is a σ-algebra defined on Θ, the Dirichlet process (DP) (Ferguson (1973)) is characterized by a probability measure G_0 on the measure space, and a positive scaling parameter α. A DP is a random probability measure over $(\Theta, \mathcal{A})$, denoted as $G \sim DP(G_0, \alpha)$, such that for any partition $(A_1, ..., A_r)$ of Θ we have

$$(G(A_1), ..., G(A_r)) \sim Dir(\alpha_0 G_0(A_1), ..., \alpha_0 G_0(A_r)),$$

where Dir is the finite-dimensional Dirichlet distribution. In other words, a DP is a "distribution over distribution".

The DPM model is based on DP. Under our formulation, it is defined as

$$G|G_0, \alpha \sim DP(G_0, \alpha) \tag{4}$$
$$\boldsymbol{\mu}_k, \lambda_k|G \sim G(\boldsymbol{\mu}_k, \lambda_k) \tag{5}$$
$$\mathbf{z}_i|\boldsymbol{\mu}_k, \lambda_k \sim p(\mathbf{z}_i|\boldsymbol{\mu}_k, \lambda_k) \tag{6}$$

The likelihood in (6) is given in Eq. (1), i.e. in our work the DPM model is applied in the transformed space $\mathbf{Z} = \{\mathbf{z}_i = f(\mathbf{y}_i; \boldsymbol{\theta})\}_{i=1}^N$. We define the base distribution G_0 as the conjugate prior of the likelihood, which is the normal-gamma distribution (Bishop (2006))[1]:

$$\begin{aligned}\boldsymbol{\mu}_k, \lambda_k|\boldsymbol{\mu}_0, \kappa_0, \alpha_0, \beta_0 &\sim NG(\boldsymbol{\mu}_k, \lambda_k|\boldsymbol{\mu}_0, \kappa_0, \alpha_0, \beta_0) \\ &= \mathcal{N}(\boldsymbol{\mu}_k|\boldsymbol{\mu}_0, (\kappa_0\lambda_k)^{-1}\mathbf{I}) Gamma(\lambda_k|\alpha_0, \beta_0).\end{aligned} \tag{7}$$

A key property of DPM models is that the marginalized conditional distribution of the cluster assignment variable has closed from:

$$p\left(c_i = k \mid \mathbf{c}_{-i}, \alpha\right) = \begin{cases} \dfrac{n_{-i,k}}{N-1+\alpha}, & n_{-i,k} > 0 \\ \dfrac{\alpha}{N-1+\alpha}, & n_{-i,k} = 0 \end{cases} \tag{8}$$

where $\mathbf{c}_{-i} = \mathbf{c} \setminus \{c_i\}$, N is the number of all data points, and $n_{-i,k}$ is the size of the k-th cluster excluding the i-th datum. Intuitively the first formula is the probability of assigning the i-th datum into the k-th existing cluster, while the second formula is the probability of assigning it to a new cluster. The proof of this result is available in many related literature (Görür & Rasmussen (2010); Chen et al. (2016); Li et al. (2019)).

We collect all the hyperparameters in the DPM model as $\boldsymbol{\Phi} = \{\alpha, \boldsymbol{\mu}_0, \kappa_0, \alpha_0, \beta_0\}$, which was used in Eq. (2). To keep the representation succinct we will suppress $\boldsymbol{\Phi}$ in the following discussions, unless it is explicitly needed.

3.3 Parameter estimation with Monte Carlo EM

To combine NICE and DPM models, the key question is how to learn the deep neural network, or in other words how to optimize the parameters $\boldsymbol{\theta}$ in $f(\mathbf{y}_i; \boldsymbol{\theta})$. In this subsection we will address this challenge with the Monte Carlo expectation-maximization (EM) algorithm.

To apply Monte Carlo EM we consider $\mathbf{c}, \{\boldsymbol{\mu}_k\}_{k=1}^K$ and $\{\lambda_k\}_{k=1}^K$ in Eq. (2) as hidden random variables, $\mathbf{Y}$ as the observed variables, and $\boldsymbol{\theta}$ as the set of parameters need to be optimized. To keep the representation succinct, we denote $\mathbf{H} = \{\{\boldsymbol{\mu}_k\}_{k=1}^K, \{\lambda_k\}_{k=1}^K\}$. Following the maximal likelihood principle, the optimal $\boldsymbol{\theta}^*$ is:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} p(\mathbf{Y}|\boldsymbol{\theta}). \tag{9}$$

This can be solved by the well-known expectation-maximization (EM) algorithm (Bishop (2006)), which iterates between the E-step and M-step until converges:

$$\text{E-step: } Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(old)}) = E_{\mathbf{H},\mathbf{c}|\mathbf{Y},\boldsymbol{\theta}^{(old)}}[\log p(\mathbf{H}, \mathbf{c}, \mathbf{Y}|\boldsymbol{\theta})] \tag{10}$$
$$\text{M-step: } \boldsymbol{\theta}^{(new)} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(old)}) \tag{11}$$
$$\boldsymbol{\theta}^{(old)} \leftarrow \boldsymbol{\theta}^{(new)} \tag{12}$$

In the case that the E-step (10) has no closed-form solution, it can be numerically estimated as

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(old)}) \approx \frac{1}{G} \sum_g \log p(\mathbf{H}^{(g)}, \mathbf{c}^{(g)}, \mathbf{Y}|\boldsymbol{\theta}), \tag{13}$$

where $\mathbf{H}^{(g)}, \mathbf{c}^{(g)} \sim p(\mathbf{H}, \mathbf{c}|\mathbf{Y}, \boldsymbol{\theta}^{(old)})$ are i.i.d. samples and G is the sample size. This method is called the Monte Carlo EM algorithm (Bishop (2006)).

[1] In our work we consider $\boldsymbol{\mu}_k$ as a scalar, while in the standard normal-gamma distribution it is a scalar.

As $\boldsymbol{\theta}$ denotes the parameters of a deep neural network, we can use stochastic gradient descent to find the maximal value in Eq. (11):

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \lambda_s \frac{\partial}{\partial \boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(old)}) \tag{14}$$

$$\approx \boldsymbol{\theta}_t + \frac{\lambda_s}{G} \sum_g \frac{\partial}{\partial \boldsymbol{\theta}} \log p(\mathbf{H}^{(g)}, \mathbf{c}^{(g)}, \mathbf{Y}|\boldsymbol{\theta}), \tag{15}$$

where λ_s is the learning rate.

Finally to complete the picture, we need to:

- Present the analytical form of the complete data likelihood $p(\mathbf{H}^{(g)}, \mathbf{c}^{(g)}, \mathbf{Y}|\boldsymbol{\theta})$, and particularly the derivative of the log-likelihood in Eq. (15);
- Develop a method to obtain the samples $\mathbf{H}^{(g)}, \mathbf{c}^{(g)} \sim p(\mathbf{H}, \mathbf{c}|\mathbf{Y}, \boldsymbol{\theta}^{(old)})$.

These two questions will be addressed respectively in the following two subsections.

3.4 The complete data likelihood

With our discussions in the model specification section, the complete data likelihood can be derived as follows:

$$p(\mathbf{H}^{(g)}, \mathbf{c}^{(g)}, \mathbf{Y}|\boldsymbol{\theta}) = p(\mathbf{Y}|\mathbf{H}^{(g)}, \mathbf{c}^{(g)}; \boldsymbol{\theta}) p(\mathbf{H}^{(g)}, \mathbf{c}^{(g)}) \tag{16}$$

$$= p(\mathbf{Z}|\mathbf{H}^{(g)}, \mathbf{c}^{(g)}) p(\mathbf{H}^{(g)}, \mathbf{c}^{(g)}) \prod_i |\det \frac{\partial f(\mathbf{y}_i; \boldsymbol{\theta})}{\partial \mathbf{y}_i}| \tag{17}$$

$$= p(\mathbf{Z}|\mathbf{H}^{(g)}, \mathbf{c}^{(g)}) p(\mathbf{H}^{(g)}) p(\mathbf{c}^{(g)}) \tag{18}$$

$$= \prod_i p(\mathbf{z}_i|\boldsymbol{\mu}_{c_i}^{(g)}, \lambda_{c_i}^{(g)}) \prod_k p(\boldsymbol{\mu}_k^{(g)}, \lambda_k^{(g)}) p(\mathbf{c}^{(g)}), \tag{19}$$

where $p(\mathbf{z}_i|\boldsymbol{\mu}_{c_i}^{(g)}, \lambda_{c_i}^{(g)})$ is given in Eq. (1). As we are interested in optimizing the neural network's parameters $\boldsymbol{\theta}$, and the likelihood only involves $\boldsymbol{\theta}$ through $\mathbf{z}_i = f(\mathbf{y}_i; \boldsymbol{\theta})$, the derivative of the log-likelihood is:

$$\frac{\partial}{\partial \boldsymbol{\theta}} \log p(\mathbf{H}^{(g)}, \mathbf{c}^{(g)}, \mathbf{Y}|\boldsymbol{\theta}) = \sum_i \frac{\partial}{\partial \boldsymbol{\theta}} \log p(\mathbf{z}_i|\boldsymbol{\mu}_{c_i}^{(g)}, \lambda_{c_i}^{(g)}) \tag{20}$$

$$= \sum_i -\lambda_{c_i}^{(g)} (\mathbf{z}_i - \boldsymbol{\mu}_{c_i}^{(g)}) \frac{\partial f(\mathbf{y}_i; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}. \tag{21}$$

Since $f(\mathbf{y}_i; \boldsymbol{\theta})$ is implemented by a DNN, its derivative can be automatically computed by many modern machine learning frameworks like PyTorch (Paszke et al. (2019)).

3.5 Gibbs sampling

Next we consider how to obtain the samples $\mathbf{H}^{(g)}, \mathbf{c}^{(g)} \sim p(\mathbf{H}, \mathbf{c}|\mathbf{Y}, \boldsymbol{\theta}^{(old)}) = p(\mathbf{H}, \mathbf{c}|\mathbf{Z}^{(old)})$, where we define that $\mathbf{Z}^{(old)} = \{\mathbf{z}_i^{(old)} = f(\mathbf{y}_i; \boldsymbol{\theta}^{(old)})\}_{i=1}^N$. This can be achieved by Gibbs sampling, which states that we can sample from the joint distribution by iteratively sampling from the conditional distribution of each variable while keeping others fixed (Bishop (2006)). So in what follows we will derive the conditional distribution of each variable.

Conditional distribution of $\boldsymbol{\mu}_k$ and λ_k:

$$p(\boldsymbol{\mu}_k, \lambda_k|\mathbf{H} \setminus \{\boldsymbol{\mu}_k, \lambda_k\}, \mathbf{c}, \mathbf{Z}^{(old)}) = p(\boldsymbol{\mu}_k, \lambda_k|\mathbf{c}, \mathbf{Z}^{(old)}) = p(\boldsymbol{\mu}_k, \lambda_k|[\mathbf{Z}^{(old)}]_k)$$
$$= NG(\boldsymbol{\mu}_k, \lambda_k|\boldsymbol{\mu}_n, \kappa_n, \alpha_n, \beta_n), \tag{22}$$

where $[\mathbf{Z}^{(old)}]_k = \{\mathbf{z}_i \in \mathbf{Z}^{(old)} | c_i = k\}$ denotes all the latent variables which are assigned to the k-th cluster. So the result is a normal-gamma distribution, with parameters given by:

$$n_k = \#[\mathbf{Z}^{(old)}]_k, \quad \bar{\boldsymbol{z}}_{\boldsymbol{k}} = \frac{1}{n_k} \sum_{\mathbf{z}_i \in [\mathbf{Z}^{(old)}]_k} \mathbf{z}_i, \quad \boldsymbol{\mu}_n = \frac{\kappa_0 \boldsymbol{\mu}_0 + n_k \bar{\boldsymbol{z}}_{\boldsymbol{k}}}{\kappa_0 + n_k}, \quad \kappa_n = \kappa_0 + n_k,$$

$$\alpha_n = \alpha_0 + \frac{n_k d}{2}, \quad \beta_n = \beta_0 + \frac{1}{2} \sum_{\mathbf{z}_i \in [\mathbf{Z}^{(old)}]_k} ||\mathbf{z}_i - \bar{\boldsymbol{z}}_{\boldsymbol{k}}||_2^2 + \frac{\kappa_0 n_k ||\bar{\boldsymbol{z}}_{\boldsymbol{k}} - \boldsymbol{\mu}_0||_2^2}{2(\kappa_0 + n_k)}.$$

Detailed proof of the result can be found in the Appendix.

Conditional distribution of c_i:

• If $n_{-i,k} > 0$ (assign to an existing cluster):

$$\begin{aligned} \log p(c_i = k | \mathbf{c}_{-i}, \mathbf{Z}^{(old)}, \mathbf{H}) &= \log p\left(c_i = k \mid \mathbf{c}_{-i}, \alpha\right) + \log p(\mathbf{z}_i | \boldsymbol{\mu}_k, \lambda_k) + \text{const} \\ &= \log \frac{n_{-i,k}}{N - 1 + \alpha} + \log \mathcal{N}(\mathbf{z}_i | \boldsymbol{\mu}_k, \lambda_k^{-1} \mathbf{I}) + \text{const} \end{aligned} \tag{23}$$

• If $n_{-i,k} = 0$ (assign to a new cluster):

$$\begin{aligned} &\log p(c_i = k | \mathbf{c}_{-i}, \mathbf{Z}^{(old)}, \mathbf{H}) \\ &= \log p(\mathbf{z}_i | \boldsymbol{\mu}_0, \kappa_0, \alpha_0, \beta_0) p\left(c_i = k \mid \mathbf{c}_{-i}, \alpha\right) + \text{const} \\ &= \log \int p(\mathbf{z}_i | \boldsymbol{\mu}, \lambda) NG(\boldsymbol{\mu}, \lambda | \boldsymbol{\mu}_0, \kappa_0, \alpha_0, \beta_0) d\boldsymbol{\mu} d\lambda + \log p\left(c_i = k \mid \mathbf{c}_{-i}, \alpha\right) + \text{const} \\ &= \log \Gamma(\alpha_n') - \log \Gamma(\alpha_0) + \alpha_0 \log \beta_0 - \alpha_n' \log \beta_n' + \\ &\quad \frac{1}{2}(\log \kappa_0 - \log \kappa_n') - \frac{nd}{2} \log 2\pi + \log \frac{\alpha}{N - 1 + \alpha} + \text{const}, \\ &\quad \text{where } \kappa_n' = \kappa_0 + 1, \quad \alpha_n' = \alpha_0 + d/2, \quad \beta_n' = \beta_0 + \frac{\kappa_0 ||\mathbf{z}_i - \boldsymbol{\mu}_0||_2^2}{2(\kappa_0 + 1)}. \end{aligned} \tag{24}$$

Since the normal-gamma distribution is the conjugate prior of the likelihood, the integration in the third line of Eq. (24) is analytically tractable (Murphy (2007)).

Our method is summarized in Algorithm 1 listed above.

4 EXPERIMENTS

4.1 K-AGNOSTIC CLUSTERING METRICS

When the number of clusters K is unknown, the *unsupervised clustering accuracy* (ACC) used by existing K-fixed clustering methods (e.g. Xie et al. (2016)) is no longer suitable to evaluate the clustering performance. Firstly, ACC is only available when the number of clusters matches the ground truth. Second, regardless of that constraint, ACC is still not suitable and may even cause misdirection. The first issue can be addressed by adopting a modified ACC metric we termed ACC*, which is computed by assigning the majority sample label as the cluster label. To illustrate the second problem we can consider an extreme case when each sample constitutes a single cluster, the ACC* is exactly 100% but the algorithm learns nothing from the data. Due to the reasons above we focus on 3 well defined K-agnostic criterion for performance comparison: *clustering F1 score (F score)*, *adjust random index (ARI)* (Steinley (2004)) and *V-measure score (V score)* (Rosenberg & Hirschberg (2007)). The ACC* score is only presented for reference purpose. The detailed definition of the metrics can be found in the Appendix.

4.2 EXPERIMENT SETTINGS

Dataset preprocessing: We evaluate our method on 4 widely used benchmark datasets including MNIST, STL-10, Reuters10K, and HHAR. MNIST is a widely used handwritten digit database. STL-10 is an image recognition dataset, which contains unlabeled data for unsupervised learning.

Algorithm 1 DDPM($\mathbf{X}, h_e(\cdot), \mathbf{\Phi}, \lambda_s$)

Require: Input dataset $\mathbf{X}$; encoder $h_e(\cdot)$; hyperparameters $\mathbf{\Phi}$; learning rate λ_s
Ensure: Cluster parameters $\{\boldsymbol{\mu}_k, \lambda_k\}_{k=1}^K$; cluster assignment vector $\mathbf{c}$.

Initialize neural network's parameters $\boldsymbol{\theta}$
$\mathbf{Y} \leftarrow \{h_e(\mathbf{x}_i) | \mathbf{x}_i \in \mathbf{X}\}_{i=1}^N$.
for *epoch* in {1, ..., EPOCHS} **do**
  $\mathbf{Z}^{(old)} \leftarrow \{f(\mathbf{y}_i; \boldsymbol{\theta})\}_{i=1}^N$
  \\ E-step; iterations for the Gibbs sampling
  for t in {1, ..., GIBBS_STEPS} **do**
    For each k sample $\boldsymbol{\mu}_k, \lambda_k \sim p(\boldsymbol{\mu}_k, \lambda_k | \mathbf{H} \setminus \{\boldsymbol{\mu}_k, \lambda_k\}, \mathbf{c}, \mathbf{Z}^{(old)})$ by Eq. (22)
    For each i sample $c_i \sim p(c_i | \mathbf{c}_{-i}, \mathbf{Z}^{(old)}, \mathbf{H})$ by Eq. (23) and Eq. (24)
  end for
  For each k sample $\boldsymbol{\mu}_k^{(g)}, \lambda_k^{(g)}$, for each i sample $c_i^{(g)}$
  \\ M-step; optimization of the NICE model
  for t in {1, ..., OPT_STEPS} **do**
    Sample a batch $\mathbf{Y}^{(b)} \leftarrow \{\mathbf{y}_i \in \mathbf{Y}\}_{i=1}^B$
    $\nabla_\theta \leftarrow \frac{\partial}{\partial \boldsymbol{\theta}} \log p(\mathbf{H}^{(g)}, \mathbf{c}^{(g)}, \mathbf{Y}^{(b)} | \boldsymbol{\theta})$ (Eq. (21))
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \lambda_s \nabla_\theta$
  end for
end for
Return sampled cluster parameters $\{\boldsymbol{\mu}_k^{(g)}, \lambda_k^{(g)}\}_{k=1}^K$, and the cluster assignment vector $\mathbf{c}^{(g)}$

Table 1: Performance comparison on various datasets. The best performance is bold highlighted. The value of ACC* is only listed for reference. Superscript dp (ddp) means using the final k value of DPM (DDPM). Compared with baselines capable of inferring k (GMeans and DPM), DDPM almost always achieves the best results.

Dataset	Metrics	KMeans^{dp}	KMeans^{ddp}	DEC^{dp}	DEC^{ddp}	GMeans	DPM	DDPM
HAR	F score	0.4831	0.5832	0.5221	0.605	0.1146	0.5894	**0.7047**
	V score	0.5976	0.6493	0.6069	0.6517	0.4358	0.6312	**0.7370**
	ARI	0.4099	0.5117	0.4499	0.5315	0.0904	0.5050	**0.6339**
	ACC*	0.7740	0.7947	0.7906	0.7897	0.8542	0.7325	0.7441
	K	11	8	11	8	234	11	8
MNIST	F score	0.3070	0.2335	0.3604	0.2985	0.1255	0.4639	**0.6388**
	V score	0.6510	0.6253	0.6855	0.6582	0.5314	0.6619	**0.7192**
	ARI	0.2822	0.2132	0.3348	0.2754	0.1126	0.4307	**0.6106**
	ACC*	0.9440	0.9527	0.9628	0.9637	0.9257	0.8909	0.9388
	K	56	81	56	81	440	56	81
STL-10	F score	0.4565	0.4695	0.506	**0.519**	0.2512	0.4315	0.4294
	V score	0.6003	0.5939	0.6469	**0.6607**	0.4830	0.5631	0.5512
	ARI	0.4162	0.4248	0.4438	**0.4564**	0.2140	0.3715	0.3688
	ACC*	0.8058	0.8003	0.6875	0.69	0.7455	0.7188	0.6903
	K	32	31	32	31	115	32	31
REU-10K	F score	0.2268	0.3244	0.241	0.3767	0.0933	0.4126	**0.5333**
	V score	0.3956	0.4332	0.4374	**0.4632**	0.3147	0.3138	0.3743
	ARI	0.1523	0.2282	0.1676	0.2759	0.0581	0.1211	**0.2801**
	ACC*	0.8704	0.8719	0.9029	0.8854	0.8951	0.6902	0.6957
	K	26	17	26	17	252	26	17

Reuters10K is a text classification dataset consists of the TF-IDF features of the word, and HHAR is a sensor signal classification dataset. We flattened the images in MNIST as vectors, and used the pretrained ResNet-50 (He et al. (2016)) to subtract features for STL-10. After the preprocessing, MNIST contains 10 classes of 786-dimensional training samples and 7000 samples for each class; STL-10 contains 10 classes of 2048-dimensional training samples and 1300 samples for each class; Reuters10K contains 4 classes of 2000-dimensional training samples and 10000 samples in total; HHAR contains 10 classes of 561-dimensional training samples and 10200 samples in total.

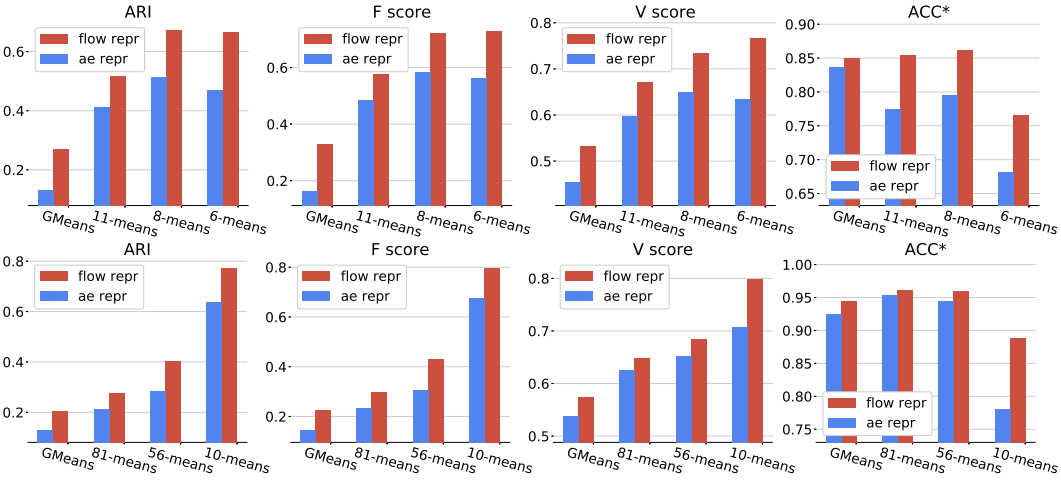

Figure 1: The performance of the presented methods improve significantly using the learned features, demonstrating DDPM's ability of learning better representation. The experiment is conducted on HAR(1st row) and MNIST(2nd row). 81-means means k-means with $k = 81$, etc.

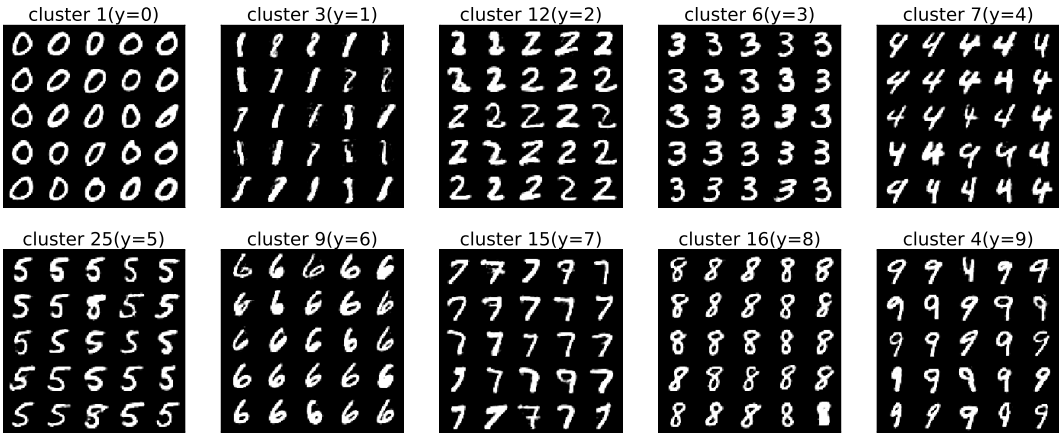

Figure 2: The generated numbers of MNIST from DDPM cluster centers with noise scale 4.5.

Methods and parameters: Considering that there is no deep clustering can adjust the number of clusters K, we consider the original Dirichlet Process Mixture Model as our main baseline. Other popular clustering algorithms includes k-means, GMeans (Zhao et al. (2008)) and DEC. For each dataset we use the same pretrained autoencoder for all methods to ensure a fair comparison. Following the prior work of (Jiang et al. (2017)), we use the network structure of d-500-500-2000-10 for encoder and 10-2000-500-500-d for decoder, where d denotes the dimension of preprocessed input samples. All the layers are fully connected with ReLU activation. Similar to DEC, we perform 10 random restarts when initializing all clustering models and pick the result with the best objective value. We follow Guo et al. (2017) to set hyperparameters for DEC. Hyperparameters of DPM and DDPM are provided in the Appendix. Core code and pretrained models are both available in our code repository *http://github.com/anony/DDPM* (link available in the formal version.)

4.3 Results

Clustering performance: The main comparison results are presented in Table 1, where top 1 scores are highlighted with bold font. ACC* is listed but not considered to be a formal comparison criterion, since it is highly sensitive to K. In our experiment we assume K is unknown in prior, so many

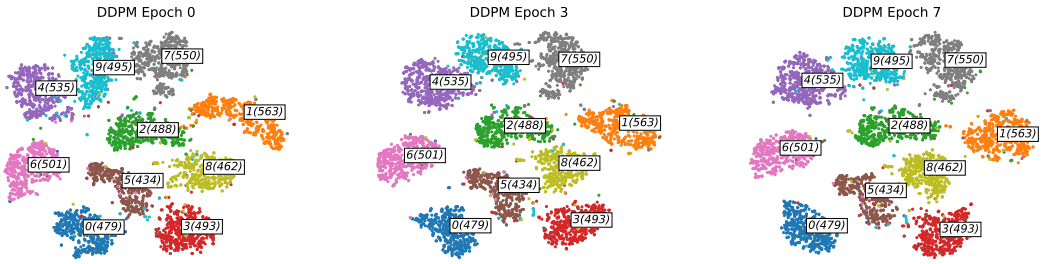

Figure 3: The representation learning process of DDPM on MNIST. See the most evident example: the cluster of number 1 gets denser and more concentrated as the training progresses.

algorithms like k-means and DEC cannot be directly applied. To address this issue we use the final optimized K values of DPM and DDPM for them to make the comparison fair.

DDPM outperforms DPM by a large margin in 3 datasets (HHAR, MNIST and Reuters10K) and performs closely to DPM in STL-10. It indicates DPM can benefit a lot from our iterative learning framework in most situations, while remains indistinguishable to DPM in the worst case. Compared with all the baselines, DDPM clearly outperforms others in 2 datasets (HHAR and MNIST), and becomes slightly suboptimal in 1 dataset (STL-10). Note that the baselines except GMeans and DPM are unable to adjust K, so their better numerical results in STL-10 cannot imply their superiority in our setting.

Representation quality: To demonstrate that DDPM is capable of learning better features, we exam and compare the features before and after being processed by the model. Specifically, we apply the k-means and GMeans clustering algorithms in the feature space (i.e. $\{\mathbf{y}_i = h_e(\mathbf{x}_i) \in \mathbb{R}^d\}_{i=1}^N$) and the transformed space (i.e. $\{\mathbf{z}_i = f(\mathbf{y}_i; \boldsymbol{\theta}) \in \mathbb{R}^d\}_{i=1}^N$). The results are presented in Figure 1. It can be observed that using the features learned by DDPM consistently leads to better performance, demonstrating its feature learning ability. We also visualize the learning process of DDPM on Figure 3. We randomly select 5K samples from the MNIST datasets and visualize the variation of their t-SNE embedding over different epochs. In each cluster, there is a box that shows the ground truth label and the number of samples. We can see the clusters becomes denser and more concentrated as the training progresses, which is potentially beneficial for label discrimination.

DDPM as a generative model: Benefit from the reversibility of the flow model, DDPM can also be utilized as a generative model. We select the largest cluster for each ground truth label, and generate 25 random samples by adding scaled noise to the cluster centers. Specifically for each selected cluster k, we obtain $\hat{\boldsymbol{\mu}} = \boldsymbol{\mu}_k + n\epsilon$, where $\epsilon \sim \mathcal{N}(0, 1)$ is a Gaussian noise and n is the noise scale. Since the flow model is invertible, we can obtain the sample as $\hat{\mathbf{x}} = h_d(f^{-1}(\hat{\boldsymbol{\mu}}; \boldsymbol{\theta}))$. The visualization results are presented in Figure 2. It can be observed that DDPM is capable of generating clear handwritten digits with sharp edges.

5 Conclusion

In this paper we propose the deep Dirichlet process mixture (DDPM) model, which jointly achieves clustering and feature learning in an unsupervised fashion. Our method combines the strengths of Bayesian nonparametrics models and deep neural networks: Based on the Dirichlet process mixture (DPM) models, DDPM inherits the ability in adapting the number of mixture components. This enables our model automatically adjust its capacity without relying on extrinsic prior knowledge or assumptions. The invertible deep neural network component, which is based on the flow-based model, further enables DDPM to learn complex and nonlinear features. Experimental results suggest that DDPM could learn more expressive representation, and achieves better clustering performance compared to other popular methods. As for future work, we would like to improve the training efficiency by employing other inference techniques, such as the variational inference framework (Blei & Jordan (2006)). It is also interesting to extend our method to more sophisticated DPM models like the hierarchical Dirichlet processes (Teh et al. (2006)).

REFERENCES

Charles E Antoniak. Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. *The annals of statistics*, pp. 1152–1174, 1974.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

David M Blei and Michael I Jordan. Variational inference for Dirichlet process mixtures. *Bayesian analysis*, 1(1):121–143, 2006.

Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 132–149, 2018.

Gang Chen, Haiying Zhang, and Caiming Xiong. Maximum margin Dirichlet process mixtures for clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: non-linear independent components estimation. In *Workshop of the 3rd International Conference on Learning Representations (ICLR workshop)*, 2015.

M Ehsan Abbasnejad, Anthony Dick, and Anton van den Hengel. Infinite variational autoencoder for semi-supervised learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5888–5897, 2017.

Thomas S Ferguson. A Bayesian analysis of some nonparametric problems. *The annals of statistics*, pp. 209–230, 1973.

Dilan Görür and Carl Edward Rasmussen. Dirichlet process Gaussian mixture models: Choice of the base distribution. *Journal of Computer Science and Technology*, 25(4):653–664, 2010.

Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. Improved deep embedded clustering with local structure preservation. In *Ijcai*, pp. 1753–1759, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 770–778, 2016.

Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

Diederik P Kingma and Prafulla Dhariwal. Glow: generative flow with invertible $1\times$ 1 convolutions. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NeurIPS)*, pp. 10236–10245, 2018.

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems (NeurIPS)*, 25: 1097–1105, 2012.

Xiaopeng Li, Zhourong Chen, Leonard KM Poon, and Nevin L Zhang. Learning latent superstructures in variational autoencoders for deep multidimensional clustering. In *International Conference on Learning Representations (ICLR)*, 2018.

Yuelin Li, Elizabeth Schofield, and Mithat Gönen. A tutorial on Dirichlet process mixture modeling. *Journal of mathematical psychology*, 91:128–144, 2019.

James MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley symposium on mathematical statistics and probability*, pp. 281–297, 1967.

Kevin P Murphy. Conjugate Bayesian analysis of the Gaussian distribution, 2007.

Eric T. Nalisnick and Padhraic Smyth. Stick-breaking variational autoencoders. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

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

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 8024–8035, 2019.

Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the joint conference on empirical methods in natural language processing and computational natural language learning*, pp. 410–420, 2007.

Douglas Steinley. Properties of the hubert-arable adjusted rand index. *Psychological methods*, 9(3): 386, 2004.

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.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.

Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning (ICML)*, pp. 478–487. PMLR, 2016.

Jianwei Yang, Devi Parikh, and Dhruv Batra. Joint unsupervised learning of deep representations and image clusters. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 5147–5156, 2016.

Zhonghua Zhao, Shanqing Guo, Qiuliang Xu, and Tao Ban. G-means: a clustering algorithm for intrusion detection. In *International Conference on Neural Information Processing (NeurIPS)*, pp. 563–570, 2008.

A Appendix: K-agnostic clustering metrics

Adjust Rand Index: The Adjusted Rand Index (ARI) is an adjust version of the Rand Index (RI) which restrict its value to range (0,1). The Rand Index computes a similarity measure between two clusterings by considering all pairs of samples and counting pairs that are assigned in the same or different clusters in the predicted and true clusterings. Random (uniform) label assignments have a ARI score close to 0.0 and a perfect match gets score 1.0. If C is a ground truth class assignment and K the clustering, let's define a as the number of pairs of elements that are in the same set in C and in the same set in K, define b as the number of pairs of elements that are in different sets in C and in different sets in K. The unadjusted Rand index is then given by $\mathrm{RI} = \frac{a+b}{C_2^N}$, where C_2^N is the total number of possible pairs in the dataset. The Rand index can not guarantee that random label assignments will get a value close to zero (esp. if the number of clusters is in the same order of magnitude as the number of samples). To counter this effect we can discount the expected RI of random labelings by defining the adjusted Rand index as :

$$\mathrm{ARI} = \frac{\mathrm{RI} - E[\mathrm{RI}]}{\max(\mathrm{RI}) - E[\mathrm{RI}]}$$

V-measure Score: The V-measure is defined based on conditional entropy and cares more about the homogeneity $h = 1 - \frac{H(C|K)}{H(C)}$ and completeness $c = 1 - \frac{H(K|C)}{H(K)}$ of clusters, where $H(C)$ is the entropy of the classes and $H(C \mid K)$ is the conditional entropy of the classes given the cluster assignments:

$$H(C) = -\sum_{c=1}^{|C|} \frac{n_c}{n} \cdot \log\left(\frac{n_c}{n}\right), H(C \mid K) = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|} \frac{n_{c,k}}{n} \cdot \log\left(\frac{n_{c,k}}{n_k}\right)$$

The homogeneity encourages each cluster contains only members of a single class and completeness prefers all members of a given class assigned to the same cluster. The V-measure score is then defined by:

$$v = 2 \cdot \frac{h \cdot c}{h + c}$$

Similar to ARI, the score also has a bounded score between 0.0 and 1 where one side means the worst and another the best. To give an intuitive interpretation, clustering with bad V-measure can be qualitatively analyzed in terms of homogeneity and completeness to better feel what "kind" of mistakes is done by the assignment.

Clustering F1 Score: The F1 score for clustering evaluation is just a traditional F1 score calculated based on a pair confusion matrix. Similar to ARI, The pair confusion matrix (Hubert & Arabie (1985)) computes a 2 by 2 similarity matrix between two clusterings by considering all pairs of samples and counting pairs that are assigned into the same or into different clusters under the true and predicted clusterings.

B Appendix: Hyperparameters

Here we provide the tuned parameters for DDPM and DPM on all datasets (shared for them):

MNIST:$\kappa_0 = 0.3, \alpha_0 = 10000, \beta_0 = 250000, \alpha = 0.00001, E_{dpm} = 2, E_{flow} = 1$

HAR:$\kappa_0 = 1, \alpha_0 = 200000, \beta_0 = 10000, \alpha = 0.0001, E_{dpm} = 12, E_{flow} = 3$

REUTERS-10K:$\kappa_0 = 1, \alpha_0 = 5000, \beta_0 = 10000, \alpha = 0.00001, E_{dpm} = 12, E_{flow} = 3$

STL:$\kappa_0 = 1, \alpha_0 = 100000, \beta_0 = 10000, \alpha = 0.00001, E_{dpm} = 10, E_{flow} = 3$

E_{dpm} and E_{flow} denotes optimization epochs of DPM and Flow in every round.

C Appendix: definition and posterior of the normal-gamma distribution

Here we present the definition of the normal-gamma distribution and the derivation of the posterior distribution. Note that in our work we consider the mean variable $\boldsymbol{\mu}$ as a vector, while in the standard normal-gamma distribution it is viewed as a scalar. This causes slight differences in the derivation from the standard literature.

Definition 1 (Normal-gamma)**.** *Random variables $\boldsymbol{\mu} \in \mathbb{R}^D$ and $\lambda \in \mathbb{R}$ follows the normal-gamma distribution, denoted as*

$$(\boldsymbol{\mu}, \lambda) \sim NG(\boldsymbol{\mu}, \lambda | \boldsymbol{\mu}_0, \kappa_0, \alpha_0, \beta_0),$$

if the probability density function is defined as

$$f(\boldsymbol{\mu}, \lambda | \boldsymbol{\mu}_0, \kappa_0, \alpha_0, \beta_0) = \mathcal{N}(\boldsymbol{\mu} | \boldsymbol{\mu}_0, (\kappa_0 \lambda)^{-1}\mathbf{I}) Gamma(\lambda | \alpha_0, \beta_0).$$

Theorem 1 (Posterior of normal-gamma)**.** *After making n observations $\mathcal{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, with each $\mathbf{x}_i \sim \mathcal{N}(\boldsymbol{\mu}, \lambda^{-1}\mathbf{I})$ follows the i.i.d. isotropic normal distribution, the posterior distribution of the parameters $(\boldsymbol{\mu}, \lambda)$ is*

$$(\boldsymbol{\mu}, \lambda) \sim NG(\boldsymbol{\mu}_n, \kappa_n, \alpha_n, \beta_n),$$

where

$$\begin{aligned}
\boldsymbol{\mu}_n &= \frac{\kappa_0 \boldsymbol{\mu}_0 + n\bar{\mathbf{x}}}{\kappa_0 + n}, \\
\kappa_n &= \kappa_0 + n, \\
\alpha_n &= \alpha_0 + nD/2, \\
\beta_n &= \beta_0 + \frac{1}{2}\sum_{i=1}^{n} ||\mathbf{x}_i - \bar{\mathbf{x}}||_2^2 + \frac{\kappa_0 n ||\bar{\mathbf{x}} - \boldsymbol{\mu}_0||_2^2}{2(\kappa_0 + n)}.
\end{aligned}$$

Proof. Likelihood:

$$\begin{aligned}
p(\mathcal{D} \mid \boldsymbol{\mu}, \lambda) &= \frac{1}{(2\pi)^{nD/2}} \lambda^{nD/2} \exp\left(-\frac{\lambda}{2}\sum_{j=1}^{D}\sum_{i=1}^{n}\left(x_{ij} - \mu_j\right)^2\right) \\
&= \frac{1}{(2\pi)^{nD/2}} \lambda^{nD/2} \exp\left(-\frac{\lambda}{2}\sum_{j=1}^{D}\left[n(\mu_j - \bar{x}_j)^2 + \sum_{i=1}^{n}\left(x_{ij} - \bar{x}_j\right)^2\right]\right) \\
&= \frac{1}{(2\pi)^{nD/2}} \lambda^{nD/2} \exp\left(-\frac{\lambda}{2}\left[n||\boldsymbol{\mu} - \bar{\boldsymbol{x}}||^2 + \sum_{i=1}^{n}||\mathbf{x}_i - \bar{\boldsymbol{x}}||^2\right]\right).
\end{aligned}$$

Conjugate prior:

$$\begin{aligned}
NG\left(\boldsymbol{\mu}, \lambda \mid \boldsymbol{\mu}_0, \kappa_0, \alpha_0, \beta_0\right) &\stackrel{\text{def}}{=} \mathcal{N}\left(\boldsymbol{\mu} \mid \boldsymbol{\mu}_0, (\kappa_0\lambda)^{-1} I\right) Gamma\left(\lambda \mid \alpha_0, \text{ rate } = \beta_0\right) \\
&= \frac{1}{Z_{NG}\left(\boldsymbol{\mu}_0, \kappa_0, \alpha_0, \beta_0\right)} \lambda^{\frac{D}{2}} \exp\left(-\frac{\kappa_0\lambda}{2}||\boldsymbol{\mu} - \boldsymbol{\mu}_0||^2\right) \lambda^{\alpha_0 - 1} e^{-\lambda\beta_0} \\
&= \frac{1}{Z_{NG}} \lambda^{\alpha_0 - 1 + \frac{D}{2}} \exp\left(-\frac{\lambda}{2}\left[\kappa_0||\boldsymbol{\mu} - \boldsymbol{\mu}_0||^2 + 2\beta_0\right]\right), \\
Z_{NG}\left(\boldsymbol{\mu}_0, \kappa_0, \alpha_0, \beta_0\right) &= \frac{\Gamma\left(\alpha_0\right)}{\beta_0^{\alpha_0}} \left(\frac{2\pi}{\kappa_0}\right)^{\frac{D}{2}}.
\end{aligned}$$

Posterior:

$$\begin{aligned}
p(\boldsymbol{\mu}, \lambda \mid \mathcal{D}) &\propto NG\left(\boldsymbol{\mu}, \lambda \mid \boldsymbol{\mu}_0, \kappa_0, \alpha_0, \beta_0\right) p(\mathcal{D} \mid \boldsymbol{\mu}, \lambda) \\
&\propto \lambda^{\frac{D}{2}} e^{-\left(\kappa_0\lambda||\boldsymbol{\mu} - \boldsymbol{\mu}_0||^2\right)/2} \lambda^{\alpha_0 - 1} e^{-\beta_0\lambda} \times \lambda^{nD/2} e^{-\frac{\lambda}{2}\sum_{j=1}^{D}\sum_{i=1}^{n}(x_i - \mu)^2} \\
&\propto \lambda^{\frac{D}{2}} \lambda^{\alpha_0 + nD/2 - 1} e^{-\beta_0\lambda} e^{-(\lambda/2)\left[\kappa_0||\boldsymbol{\mu} - \boldsymbol{\mu}_0||^2 + \sum_i ||\mathbf{x}_i - \boldsymbol{\mu}||^2\right]}.
\end{aligned}$$

To proceed, we need the following equations:

$$\sum_{i=1}^{n}||\mathbf{x}_i - \boldsymbol{\mu}||^2 = n||\boldsymbol{\mu} - \bar{\boldsymbol{x}}||^2 + \sum_{i=1}^{n}||\mathbf{x}_i - \bar{\boldsymbol{x}}||^2,$$

$$\kappa_0||\boldsymbol{\mu} - \boldsymbol{\mu}_0||^2 + n||\boldsymbol{\mu} - \bar{\boldsymbol{x}}||^2 = (\kappa_0 + n)\,||\boldsymbol{\mu} - \boldsymbol{\mu}_n||^2 + \frac{\kappa_0 n||\bar{\boldsymbol{x}} - \boldsymbol{\mu}_0||^2}{\kappa_0 + n},$$

where

$$\boldsymbol{\mu}_n = \frac{\kappa_0\boldsymbol{\mu}_0 + n\bar{\boldsymbol{x}}}{\kappa_0 + n}.$$

Hence

$$\begin{aligned}
\kappa_0||\boldsymbol{\mu} - \boldsymbol{\mu}_0||^2 + \sum_i ||\mathbf{x}_i - \boldsymbol{\mu}||^2 &= \kappa_0||\boldsymbol{\mu} - \boldsymbol{\mu}_0||^2 + n||\boldsymbol{\mu} - \bar{\boldsymbol{x}}||^2 + \sum_i ||\mathbf{x}_i - \bar{\boldsymbol{x}}||^2 \\
&= (\kappa_0 + n)\,||\boldsymbol{\mu} - \boldsymbol{\mu}_n||^2 + \frac{\kappa_0 n||\bar{\boldsymbol{x}} - \boldsymbol{\mu}_0||^2}{\kappa_0 + n} + \sum_i ||\mathbf{x}_i - \bar{\boldsymbol{x}}||^2.
\end{aligned}$$

So we have

$$\begin{aligned} p(\boldsymbol{\mu}, \lambda \mid \mathcal{D}) \propto & \quad \lambda^{\frac{D}{2}} e^{-(\lambda/2)(\kappa_0+n)||\boldsymbol{\mu}-\boldsymbol{\mu}_n||^2} \\ & \times \lambda^{\alpha_0+nD/2-1} e^{-\beta_0\lambda} e^{-(\lambda/2)\sum_i ||\mathbf{x}_i-\bar{\boldsymbol{x}}||^2} e^{-(\lambda/2)\frac{\kappa_0 n||\bar{\boldsymbol{x}}-\boldsymbol{\mu}_0||^2}{\kappa_0+n}} \\ & \propto \mathcal{N}\left(\boldsymbol{\mu} \mid \boldsymbol{\mu}_n, ((\kappa_0+n)\lambda)^{-1} I\right) \times Gamma\left(\lambda \mid \alpha_0 + nD/2, \beta_n\right), \end{aligned}$$

where

$$\beta_n = \beta_0 + \frac{1}{2}\sum_{i=1}^{n} ||\mathbf{x}_i - \bar{\boldsymbol{x}}||^2 + \frac{\kappa_0 n||\bar{\boldsymbol{x}} - \boldsymbol{\mu}_0||^2}{2\left(\kappa_0 + n\right)}.$$

□