DPMM.jl

Ekin Akyürek
CSAIL, Massachusetts Institue of Technology
Koç University
akyurek@mit.edu

April 1, 2019

1 Introduction

This repository is a research work on parallel dirichlet process mixture models and clustering on Julia by Ekin Akyürek with supervision of John W. Fischer III.

2 Bacground

2.1 Mixture Models

Mixture models are statistical models for representing a data comes from different probability distributions. A mixture model generally parametrized by mixture(i.e component) parameters and mixture weights which can be considered as prior probability of mixtures in the Bayesian setting. The more weight attached to a mixture, the more data comes from that mixture. In this work, we will mostly use the notation for mixture models which are presented in the document[3]:

Data

N	Number of data vectors
D	Dimension of data vectors
$x_i \in R^D$	The i^th data vector
$X = (x_1, x_2, ..., x_N)$	Set of data vectors
$X_{\setminus i}$	Set of data vectors, without taking x_i into account
X_k	Set of data vectors from mixture component k
$X_{k\setminus i}$	Set of data vectors from mixture component k, without taking x_i into account
N_k	Number of data vectors from mixture component k
$N_{k\setminus i}$	Number of data vectors from mixture component k, without taking x_i into account

Model

K	Number of components in a finite mixture model
z_i	discrete latent state indicating which component observation x_i belongs to
$\mathbf{z} = (z_1, z_2, ..., z_N)$	Latent states for all observations $x_i, ..., x_n$
$\mathbf{z}_{\setminus i}$	Set of all latent states excluding zi
θ_k	Component parameters (e.g. μ_k, Σ_k in a gaussian mixture model)
π_k	Prior probability that data vector x_i will be assigned to mixture component k
$\pi = (\pi_1, \pi_2, ..., \pi_K)$	Prior assignment probability for all K components
β	Hyperparameters of the prior distribution for θ parameters in a bayesian setting

2.2 Dirichlet Process Mixture Models (DPMM)

2.2.1 Dirichlet Processes (DP)

There are many ways to interpret DP. Formally, it is random process whose realizations are probability distributions. The constructive definition of dirichlet process is done by stick-breaking processes. Chineese Restraunt Process(CRP) and Pólya Urn Scheme also leads to DP. All these definitions are related with each other thanks to de Finetti's exchangibility theorem.

DP is parametrized by α concentration parameter and G_0 base distribution. One can show that α controls how similar draws to the G_0. Therefore, G is used to denote samples drawn from DP.

2.2.2 Stick-Breaking Construction

Stick-breaking provides a nice way to draw samples from, but it requires to countably infinite summation. As we stated earlier, draws from DP is itself a distribution which is indeed discrete. Stick-Breaking steps are deliniated in the below.

$$
\begin{aligned}
v_1, v_2, ..., v_i, ... &\sim Beta(1, \alpha) \\
\pi_i &= v_i \prod_{j=1}^{i-1} (1 - v_j) \\
\theta_1, \theta_2, ..., \theta_i, ... &\sim G_0 \\
G &= \sum_{i=1}^{\infty} \pi_i \delta_{\theta_i}
\end{aligned}
$$

Here δ_{θ_i} is a indicator function centered on θ_i, namely $\delta_{\theta_i}(\theta)$ is zero everywhere except $\theta = \theta_i$. Realize that π_i's approaches to zero as i goes to infinity, so it enables to approximate G using a finite summation instead of infinite one.

2.2.3 Chineese Restraunt Process (CRP)

Chineese restraunt process is a dicrete process which is named after the analogy of seating customers to at tables in a restraunt. Let say customers $1 : N$, we will seatch each customer sequentially with the following probabilities where c_k is the number of customers seated k^{th} table and, i is current number of customers seated in the restraunts.

$$
z_i | z_1, ..., z_{i-1} = \begin{cases} P(z_i = k) = \frac{c_k}{i-1+\alpha} & \text{for an existing table} \\ P(z_i = K+1) = \frac{\alpha}{i-1+\alpha} & \text{for opening a new table} \end{cases}
$$

Note that this proceess is independent of the order of customers. This means that the CRP is exchangable[2].

If we assuma a base measure G_0 as a prior for table/cluster parameters and assign each table a probability measure sampled from G_0, the process becomes CRP Mixture Model or Pólya Urn Scheme.

$$
\begin{aligned}
\theta_1 ..., \theta_K, ... &\sim G_0 \\
z_i | z_1, ..., z_{i-1} &= \begin{cases} P(z_i = k) = \frac{c_k}{i-1+\alpha} & \text{for an existing table/cluster} \\ P(z_i = K+1) = \frac{\alpha}{i-1+\alpha} & \text{for opening a new table/cluster} \end{cases} \\
X_i | z_i &\sim f(X_i | \theta_{z_i})
\end{aligned}
$$

Exchangibility allows us to show that above model equals to[1]:

$$
\begin{aligned}
G &\sim DP(\alpha, G_0) \\
\theta_i|G &\sim G \qquad i \in 1, ..., N \\
X_i|\theta_i &\sim f(X_i|\theta_i) \qquad i \in 1, ..., N
\end{aligned}
$$

2.2.4 Formal Definition

We stated that samples from dirichlet process were probability distribution itself. Let say a sample is a probability distribution over S space. Let $\{A_i\}_{i=1}^n$ denote a measurable partition of S. For any measureble partition of S below statement holds (Dir is Dirichlet Distribution):

$$
G \sim DP(\alpha, G_0) \implies (G(A_1), G(A_2), ..., G(A_N)) \sim Dir(\alpha G_0(A_1), ..., \alpha G_0(A_N))
$$

This defines Dirichlet Process, however doesn't allow to draw samples from unlike CRP or Pólya Urn.

2.2.5 Infinite Mixture Models

As we show in CRP section, Diricihlet process is very good prior on cluster parameters because we can model infinite number of clusters. Note that every finite data realization of DPMM is a finite mixture model with Dirichlet distribution as shown in formal definition. One can use finite stick-breaking process to construct DP, however it has approximation errors. On the other hand, CRP Mixture Model provides exact way to sampling or do inference on the mixture data. Finally, these two different representations are summarized by below graphical model representations. In the following section we will discuss how to do inference on DPMM.

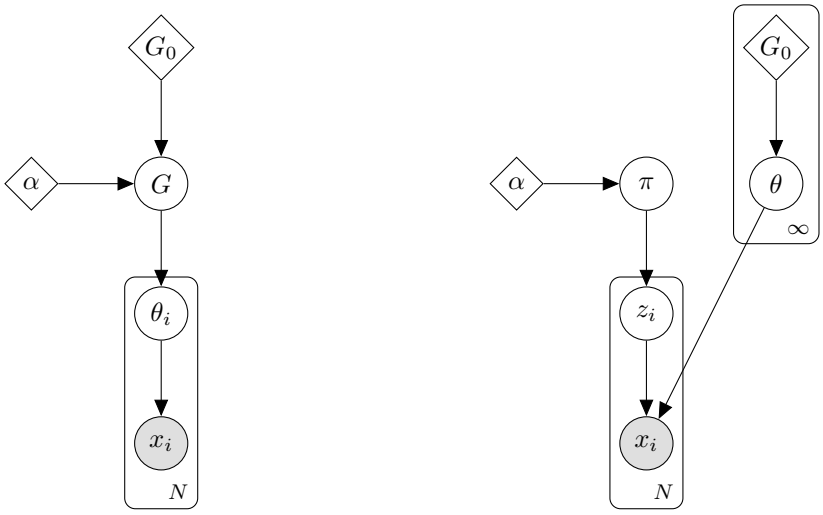


Figure 1: Two equivalent graphical representation of DPMM

2.3 Inference in DPMM

Inference problem that we interested is to obtain cluster assignments for each data point given base distribution G_0 and α parameter. We will use conjuagte prior which assure that posterior probability of cluster parameters are in same distribution family with the prior. Let's investigate conjugate priors for Gaussian and Multinomial distributions.

2.3.1 Gaussian Conjugate: Normal-Inverse-Wishart Distribution

Multivariate Gaussian Distribution generally parametrized by mean vector μ and covariance matrix Σ. Normal-Inverse-Wishart(NIW) Distribution[4] is the prior distrubution for Gaussian with unkown μ and and unknown Σ.

Normal-Inverse-Wishart is parametrized by:

$\boldsymbol{\mu}_0$	prior mean for μ
$\boldsymbol{\Psi}_0$	proportional to prior mean Σ
λ_0	how strongly we believe the μ_0
ν_0	how strongly we believe the Ψ_0

NIW has seperable pdf function with Gaussian and Inverse-Wishart($\mathcal{W}^{-1}$)

$$p(\boldsymbol{\mu}, \boldsymbol{\Sigma}|\boldsymbol{\mu}_0, \lambda_0, \boldsymbol{\Psi}_0, \nu_0) = \mathcal{N}\left(\boldsymbol{\mu}\middle|\boldsymbol{\mu}_0, \frac{1}{\lambda}\boldsymbol{\Sigma}\right)\mathcal{W}^{-1}(\boldsymbol{\Sigma}|\boldsymbol{\Psi}_0, \nu_0)$$

The $\mathcal{W}^{-1}$ includes `inverse` because the inverse of the sampled matrices has Wishart($\mathcal{W}$) distrubition. To understand further and to learn how to sample from Wishart distribution see the reference for the Bartlett decomposition method [6].

If we use sample mean($\hat{\mu}$) and sample covariance($\hat{\Sigma}$) matrix as sufficent statistics, posterior parameters for the Normal-Inverse-Wishart distribution is given by the below equations[3]:

$$\begin{aligned}
p(\boldsymbol{\mu}, \boldsymbol{\Sigma}|\mathcal{X}) &= NIW(\boldsymbol{\mu}, \boldsymbol{\Sigma}|\boldsymbol{\mu}_N, \boldsymbol{\Psi}_N, \lambda_N, \nu_N) \\
\boldsymbol{\mu}_N &= \lambda_0\boldsymbol{\mu}_0 + N\hat{\mu} \\
\lambda_N &= \lambda_0 + N \\
\nu_N &= \nu_0 + N \\
\boldsymbol{\Psi}_N &= \Psi_0 + \hat{\Sigma} + \frac{\lambda_0 N}{\lambda_0 + N}(\hat{\mu} - \boldsymbol{\mu}_0)(\hat{\mu} - \boldsymbol{\mu}_0)^T
\end{aligned}$$

It is also possible to store $t = N\hat{x}$ and $T = \sum_{i=1}^{N} xx^T$ as sufficient statistics, which is computationally adventagous, the equations become:

$$\begin{aligned}
\boldsymbol{\mu}_N &= \lambda_0\boldsymbol{\mu}_0 + t \\
\lambda_N &= \lambda_0 + N \\
\nu_N &= \nu_0 + N \\
\boldsymbol{\Psi}_N &= \Psi_0 + T + \lambda_0\boldsymbol{\mu}_0\boldsymbol{\mu}_0^T - \lambda_N\boldsymbol{\mu}_N\boldsymbol{\mu}_N^T
\end{aligned}$$

Low-Rank Updates for Ψ_N In the program we store Ψ_N cholesky factorized form to sample efficently. However, we need to calculate cholesk factorization,which is $\mathcal{O}(n^3)$, everytime if we remove or add single data point. For that case, there is a faster algorithm to compute updated cholesky factorization from the previous cholesky factorization[5]. We utilize Rank-1 update and Rank-1 downdate methods to speed up the collapsed gibbs sampler.

2.3.2 Multinomial Conjugate: Dirichlet Distribution

In discrete data setting, multinomial probabilty distribution is used for likelihood of data. Dirichlet distribution is the conjugate prior for multinomial distribution.

Dirichlet distribution is parametrized by $\vec{\alpha}$ where $\vec{\alpha}_i$ corresponds to pseudocount of multinomial event i.

The likelihood of the probabilities of a multinomial distribution given by Dirichlet prior is:

$$f(p_1, \ldots, p_K; \vec{\alpha}_1, \ldots, \vec{\alpha}_K) = \frac{1}{\mathrm{B}(\boldsymbol{\alpha})} \prod_{i=1}^{K} p_i^{\vec{\alpha_i}-1}$$

Sufficient statistics t for Dirichlet prior and multinomial distribution is the count vector of events $t = \vec{c}$ where:

$$\vec{c_i} = \sum_{j=1}^{N} \mathcal{X}_{ik}$$

Then, posterior distribution of Dirichlet is given by:

$$\vec{\alpha}' = \vec{\alpha} + \vec{c}$$

2.3.3 Collapsed Gibbs Sampler

When using conjugate priors for cluster parameters, marginal likelihood of data can be obtained by analytically integrating out the cluster parameters θ_i's. For the Gaussian parameters the marginal likelihood:

$$p(\mathcal{X}) = \int_{\mu} \int_{\Sigma} p(\mathcal{X}, \mu, \Sigma) p(\mu, \Sigma) d\mu d\Sigma$$

The likelihod of a new data x^* is given by:

$$p(x^*|\mathcal{X}) = \frac{p(x^*, \mathcal{X})}{p(\mathcal{X})}$$

It turns out that the posterior likelihood (posterior-predictive) of new data has Multivariate Student's t-distribution[3] for NIW-Gaussian pair:

$$p(x^*|\mathcal{X}) = \mathcal{T}(x^*|\boldsymbol{\mu}_N, \frac{(\lambda_N + 1)}{\lambda_N(\nu_N - D + 1)} \boldsymbol{\Psi}_N, \nu_n - D + 1)$$

The posterior predictive likelihood of Dirichlet-Multinomial pair has rather complicated function:

$$p(x^*|\mathcal{X}) = \frac{\Gamma(n+1)}{\prod_{j=1}^{K} \Gamma(x_j^* + 1)} \frac{\Gamma(\sum_{j=1}^{K} \alpha_j')}{\prod_{j=1}^{K} \Gamma(\alpha_j')} \frac{\prod_{j=1}^{K} \Gamma(x_j^* + \alpha_j')}{\Gamma(n + \sum_{j=1}^{K} \alpha_j')}$$

The collapsed inference in DP Mixture Models can be done by using CRP procedure. The difference is that instead of using $f(x|\theta_{z_i})$ likelihood, we use $\mathcal{T}(x|\beta, \mathcal{X}_{k\setminus i})$ posterior-predictive and we never sample θ's explicitly [3].

$$z_i|\boldsymbol{z}_{\setminus i} \sim \begin{cases} P(z_i = k) = \mathcal{T}(x|\beta, \mathcal{X}_{k\setminus i}) \frac{N_{k\setminus i}}{N-1+\alpha} & \text{for an existing table/cluster} \\ P(z_i = K+1) = \mathcal{T}(x|\beta) \frac{\alpha}{N-1+\alpha} & \text{for a new table/cluster} \end{cases}$$

Pseudocode [3] of collapsed gibbs sampler is presented in Algorithm 1 for Gaussian DPMMs, and the multinomial one can easily be obtained by changing posterior-predictive in the code.

Remarks

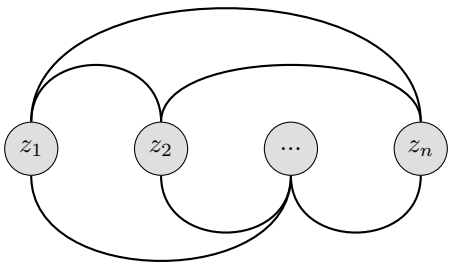
- Collapsed sampler is hard to parallelize because in each data point we need update cluster statistics, and we don't know which cluster's statistic will change and what they will be beforehand.
- Collapsed sampler has some issues when it starts with under-estimated number of cluster. Closer clusters tend to form a single cluster in that situation.

Algorithm 1 Collapsed Gibbs sampler for an infinite Gaussian mixture model.

1: Choose an initial $\mathbf{z}$.
2: **for** M iterations **do** ▷ Gibbs Sampling Iterations
3: **for** i = 1 to N **do**
4: Remove $\mathbf{x}_i's$ statistics from component z_i ▷ Old assignment for $\mathbf{x}_i$
5: **for** k = 1 to K **do** ▷ For all existing clusters
6: Calculate $P(z_i = k|\mathbf{z}_{\setminus i}, \alpha) = \frac{N_{k\setminus i}}{N+\alpha-1}$ as in CRP
7: Calculate $p(\mathbf{x}_i|\mathcal{X}_{k\setminus i}, \beta)$ using $\mathcal{T}$ distribution above
8: Calculate $P(z_i = k|\mathbf{z}_{\setminus i}, \mathcal{X}, \alpha, \beta) \propto P(z_i = k|\mathbf{z}_{\setminus i}, \alpha)p(\mathbf{x}_i|\mathcal{X}_{k\setminus i}, \beta)$
9: **end for**
10: Calculate $P(z_i = K + 1|\mathbf{z}_{\setminus i}, \alpha) = \frac{\alpha}{N+\alpha-1}$ as in CRP ▷ Consider opening a new cluster
11: Calculate $p(\mathbf{x}_i|\beta)$ using $\mathcal{T}$ distribution above
12: Calculate $P(z_i = K + 1|\mathbf{z}_{\setminus i}, \mathcal{X}, \alpha, \beta) \propto P(z_i = K + 1|\mathbf{z}_{\setminus i}, \alpha)p(\mathbf{x}_i|\beta)$
13: Sample k_{new} from $P(z_i = k|\mathbf{z}_{\setminus i}, \mathcal{X}, \alpha, \beta)$ after normalizing
14: Add z_i's statistics to the component $z_i = k_{new}$. ▷ New assignment for $\mathbf{x}_i$
15: If any component is empty, remove it and decrease K
16: **end for**
17: **end for**

2.3.4 Quasi-Collapsed Gibbs Sampler

Note that in collapsed sampling every latent variable z_i is connected with each other in the un-directed representation of DP.



So, it is not possible to sample z_i's in parallel for an exact sampling. However, the affect of change in z_i is negligible when $N \to \infty$. Therefore, we can break above connections for an approximate collapsed sampling algorithm which is parallelizable. It is also corresponds to deleting 4^{th} and 14^{th} lines, and calculation posterior-predictive distribution $\mathcal{T}$ before the iteration in Algorithm 1. So, in this version of algorithm, instead of updating posterior-predictive for each data point, we update it only in the beginning of an iteration. The quasi-collapsed sampler algorithm is presented in Algorithm 2 and sampling scheme is presented in the equation:

$$z_i|\boldsymbol{z} \sim \begin{cases} P(z_i = k) = \mathcal{T}(x|\beta, \mathcal{X}_k)\frac{N_k}{N+\alpha} & \text{for an existing table/cluster} \\ P(z_i = K + 1) = \mathcal{T}(x|\beta)\frac{\alpha}{N+\alpha} & \text{for a new table/cluster} \end{cases}$$

Algorithm 2 Quasi-Collapsed Gibbs sampler for an infinite Gaussian mixture model.

Choose an initial $\mathbf{z}$.
for M iterations **do** ▷ Gibbs Sampling Iterations
  Update posterior-predictive $\mathcal{T}$ with $\mathbf{z}$
  for i = 1 to N **do**
    for k = 1 to K **do** ▷ For all existing clusters
      Calculate $P(z_i = k|\mathbf{z}, \alpha) = \frac{N_k}{N+\alpha}$ as in CRP
      Calculate $p(\mathbf{x}_i|\mathcal{X}_k, \beta)$ using posterior $\mathcal{T}$ distribution with old assignments $\mathbf{z}$
      Calculate $P(z_i = k|\mathbf{z}, \mathcal{X}, \alpha, \beta) \propto P(z_i = k|\mathbf{z}, \alpha)p(\mathbf{x}_i|\mathcal{X}_k, \beta)$
    end for
    Calculate $P(z_i = K + 1|\mathbf{z}, \alpha) = \frac{\alpha}{N+\alpha}$ as in CRP ▷ Consider opening a new cluster
    Calculate $p(\mathbf{x}_i|\beta)$ using $\mathcal{T}$ distribution above
    Calculate $P(z_i = K + 1|\mathbf{z}, \mathcal{X}, \alpha, \beta) \propto P(z_i = K + 1|\mathbf{z}, \alpha)p(\mathbf{x}_i|\beta)$
    Sample k_{new} from $P(z_i = k|\mathbf{z}, \mathcal{X}, \alpha, \beta)$ after normalizing
  end for
  If any component is empty, remove it and decrease K
end for

2.3.5 Direct Gibbs Sampler

Collapsed and Quasi-Collapsed algorithms are using CRP procedure to construct DP. However, there is another approach which we sample mixture weights and mixture parameters directly. There is an infinite mixtures however in DP, but we can combine mixtures for un-instantiated clusters together. So, π_{K+1} is sum of all un-instantiated cluster weights and θ_{K+1} is sampled from base distribution for a new cluster. Direct sampler scheme is presented in the below:

$$\pi_1..., \pi_K, \pi_{K+1} \sim Dir\big(\frac{N_1}{N+\alpha}, ..., \frac{N_K}{N+\alpha}, \frac{\alpha}{N+\alpha}\big)$$
$$\theta_1..., \theta_K \sim G_0(\theta|\boldsymbol{z}, \beta)$$
$$z_i|\boldsymbol{z} \sim \begin{cases} P(z_i = k) = f(X_i|\theta_k)\pi_k & \text{for an existing cluster} \\ P(z_i = K + 1) = f(X_i|\theta_{K+1})\pi_{K+1} & \text{for a new cluster} \end{cases}$$

Algorithm 3 Direct Gibbs sampler for an infinite Gaussian mixture model.

Choose an initial $\mathbf{z}$.
for M iterations **do** ▷ Gibbs Sampling Iterations
  for k = 1 to K **do** ▷ For all existing clusters
    Sample $\theta_k \sim NIW(\theta|\beta, \mathcal{X}_k)$ ▷ if $N_k \neq 0$
  end for
  Sample $\pi_{1:K+1} \sim Dir\big(\frac{N_1}{N+\alpha}, ..., \frac{N_K}{N+\alpha}, \frac{\alpha}{N+\alpha}\big)$
  for i = 1 to N **do**
    for k = 1 to K **do** ▷ For all existing clusters
      Calculate $P(z_i = k|\mathbf{z}, \alpha) = \pi_k$ as in CRP
      Calculate $p(\mathbf{x}_i|\theta_k)$ using Gaussian distribution
      Calculate $P(z_i = k|\mathbf{z}, \mathcal{X}, \alpha, \beta) \propto P(z_i = k|\mathbf{z}, \alpha)p(\mathbf{x}_i|\theta_k)$
    end for
    Calculate $P(z_i = K + 1|\mathbf{z}, \alpha) = \frac{\alpha}{N+\alpha}$ as in CRP ▷ Consider opening a new cluster
    Calculate $p(\mathbf{x}_i|\theta_{K+1})$ using Gaussian distribution
    Calculate $P(z_i = K + 1|\mathbf{z}, \mathcal{X}, \alpha, \beta) \propto P(z_i = K + 1|\mathbf{z}, \alpha)p(\mathbf{x}_i|\theta_{K+1})$
    $z_i \leftarrow k_{new}$ from $P(z_i = k|\mathbf{z}, \mathcal{X}, \alpha, \beta)$ after normalizing
  end for
end for

Parallelization As it can be seen in pseudocode, direct sampler allows to sample each z_i asynchronously. This provide very good parallelization opportunity in large datasets. Results for parallel experiments discussed in Results section.

2.3.6 Quasi-Direct Gibbs Sampler

When $N \to \infty$, we can approximate $\pi_{1:K+1}$ with their proportions:

$$\pi_i = \frac{N_i}{N + \alpha}$$

So, we can get faster algorithm which skips sampling from Dirichlet process:

$$\theta_1 ..., \theta_K \sim G_0(\theta|\boldsymbol{z}, \beta)$$
$$z_i|\boldsymbol{z}_{\setminus i} \sim \begin{cases} P(z_i = k) = f(X_i|\theta_k)\frac{N_k}{N+\alpha} & \text{for an existing cluster} \\ P(z_i = K+1) = f(X_i|\theta)\frac{\alpha}{N+\alpha} & \text{for a new cluster} \end{cases}$$

We iteratively repeat above steps. Pseudocode of quasi-direct gibbs sampler presented in Algorithm 4 for Gaussian DPMMs, and multinomial example can easily be obtained by changing posterior distribution in the pseudocode.

Algorithm 4 Quasi-Direct Gibbs sampler for an infinite Gaussian mixture model.

Choose an initial $\mathbf{z}$.
for M iterations **do** ▷ Gibbs Sampling Iterations
  for k = 1 to K **do** ▷ For all existing clusters
    Sample $\theta_k \sim NIW(\theta|\beta, \mathcal{X}_k)$ ▷ if $N_k \neq 0$
  end for
  for i = 1 to N **do**
    for k = 1 to K **do** ▷ For all existing clusters
      Calculate $P(z_i = k|\mathbf{z}, \alpha) = \frac{N_k}{N+\alpha}$ as in CRP
      Calculate $p(\mathbf{x}_i|\theta_k)$ using Gaussian distribution
      Calculate $P(z_i = k|\mathbf{z}, \mathcal{X}, \alpha, \beta) \propto P(z_i = k|\mathbf{z}, \alpha)p(\mathbf{x}_i|\theta_k)$
    end for
    Calculate $P(z_i = K+1|\mathbf{z}, \alpha) = \frac{\alpha}{N+\alpha}$ as in CRP ▷ Consider opening a new cluster
    Calculate $p(\mathbf{x}_i|\theta^*)$ using Gaussian distribution
    Calculate $P(z_i = K+1|\mathbf{z}, \mathcal{X}, \alpha, \beta) \propto P(z_i = K+1|\mathbf{z}, \alpha)p(\mathbf{x}_i|\theta_{K+1})$
    $z_i \leftarrow k_{new}$ from $P(z_i = k|\mathbf{z}, \mathcal{X}, \alpha, \beta)$ after normalizing
  end for
end for

Parallelization As it can be seen in pseudocode, quasi-direct sampler allows to sample each z_i asynchronously. This provide very good parallelization opportunity in large datasets. Results for parallel experiments discussed in Results section.

Remarks

- Quasi-Direct sampler is easy to parallelize as discussed in above. Even without parallelizing, quasi-direct sampler faster than collapsed sampler.

3 Code Structure

3.1 Data

We represent data `X` in matrix form where each column is i.i.d. multidimensional data.

3.2 Clusters

Representation of clusters changes but functionality remains same. So, I implemented two different cluster types under `AbstractCluster` type which has functionality that we want from clusters.

Collapsed Cluster

```
struct CollapsedCluster{Pre<:Distribution, Pri<:Distribution} <: AbstractCluster
    n::Int
    predictive::Pre
    prior::Pri
end
```

Direct Cluster

```
struct DirectCluster{P<:Distribution, Pri<:Distribution} <: AbstractCluster
    n::Int
    sampled::P
    prior::Pri
end
```

3.2.1 Functionalities

Constructors:

Each cluster needs a constructor function which takes sufficient statistics of data or data itself* .

```
CollapsedCluster(m::DPGMM) = CollapsedCluster(m, suffstats(m))

CollapsedCluster(m::DPGMM,X::AbstractArray) = CollapsedCluster(m, suffstats(m,X))

CollapsedCluster(m::DPGMM,s::DPGMMStats) =
    CollapsedCluster(s.n, posterior_predictive(m,s), m.θprior)
```

```
DirectCluster(m::DPGMM) = DirectCluster(m, suffstats(m))

DirectCluster(m::DPGMM,X::AbstractArray) = DirectCluster(m, suffstats(m,X))

DirectCluster(m::DPGMM,s::DPGMMStats)
    = DirectCluster(s.n, rand(posterior(m,s)), m.θprior)
```

Likelihood:

Each cluster needs to define its likelihod function which is defined by overloading `pdf`* :

```
pdf(m::DirectCluster,x)     = pdf(m.sampled,x)
pdf(m::CollapsedCluster,x) = pdf(m.predictive,x)
```

Update & Downdate:

Clusters may need to define update and downdate functionality for data changes. Especially for the Collapsed Cluster we need a fast update and downdate functionality for a single data point changes. We will overload − and + operations for that purposes.

```
-(c::CollapsedCluster{V,P},x::AbstractVector) where {V<:Distribution,P<:Distribution}
= CollapsedCluster{V,P}(c.n-1, downdate_predictive(c.prior,c.predictive,x,c.n), c.prior)

+(c::CollapsedCluster{V,P},x::AbstractVector) where {V<:Distribution,P<:Distribution}
= CollapsedCluster{V,P}(c.n+1, update_predictive(c.prior,c.predictive,x,c.n), c.prior)
```

3.3 Sufficent Statistics

For Gaussian family we will use sufficient statistics as $t = N\hat{x}$ and $T = \sum_{i=1}^{N} xx^T$. We define sufficient statistics type `DPGMMStats` for this purpose:

```
struct DPGMMStats{T<:Real}
    nμ::Vector{T} # t
    S::Matrix{T}  # T
    n::Int
end
```

3.3.1 Constructors

Sufficient statistics need constructors with single-data points, muliple data points or zero initialization* .

```
suffstats(m::DPGMM{T,D}) where {T<:Real,D} =
    DPGMMStats(zeros(T,D),zeros(T,D,D),0)

suffstats(m::DPGMM{T},X::AbstractMatrix{T}) where T<:Real =
    DPGMMStats(vec(sum(X,dims=2)),X*X',size(X,2))

suffstats(m::DPGMM{T},x::AbstractVector{T}) where T<:Real =
    DPGMMStats(x,x*x',1)
```

3.3.2 Update & Downdate:

Sufficient statistics need to be updated if new data points added or some data points removed from a cluster. `updatestats` and `downdatestats` functions are defined* for this purposes:

```
function updatestats(m::DPGMMStats{T},x::AbstractVector{T}) where T<:Real
    m.nμ .+= x
    m.S  .+= x*x'
    DPGMMStats{T}(m.nμ,m.S,m.n+1)
end

function updatestats(m::DPGMMStats{T},X::AbstractMatrix{T}) where T<:Real
    m.nμ .+= vec(sum(X,dims=2))
    m.S  .+= X*X'
    DPGMMStats{T}(m.nμ,m.S,m.n+size(X,2))
end

function downdatestats(m::DPGMMStats{T},x::AbstractVector{T}) where T<:Real
    m.nμ .-= x
    m.S  .-= x*x'
    DPGMMStats{T}(m.nμ,m.S,m.n-1)
end

function downdatestats(m::DPGMMStats{T},X::AbstractMatrix{T}) where T<:Real
    m.nμ .-= vec(sum(X,dims=2))
    m.S  .-= X*X'
    DPGMMStats{T}(m.nμ,m.S,m.n-size(X,2))
end
```

3.4 Distributions and Posteriors

We use Distributions.jl package for fundamental distrubitions. Adding new distributions to Distributions.jl is quite easy.

3.4.1 Distributions

Normal-Inverse-Wishart:

Because Normal-Inverse-Wishart (NIW) distribution is just a combination $\mathcal{W}^{-1}$ and Gaussian, it is not defined in Distribution.jl. So, I used slight different version of NIW defined in ConjugatePriors.jl. The paremetrization of NIW is exactly same with our notation in the documentation thanks to `unicodee` support of Julia.

```
struct NormalInverseWishart{T<:Real,S<:AbstractPDMat} <: ContinuousUnivariateDistribution
    μ::Vector{T}
    λ::T
    Ψ::S
    ν::T
    function NormalInverseWishart{T}(μ::Vector{T}, λ::T, Ψ::AbstractPDMat{T}, ν::T) where T<:Real
        new{T,typeof(Ψ)}(μ, λ, Ψ, ν)
    end
end
```

Sampling function of `NIW` as a combination of $\mathcal{W}^{-1}$ and Gaussian:

```
function rand(niw::NormalInverseWishart)
    Σ   = rand(InverseWishart(niw.ν, niw.Ψ))
    μ   = rand(MvNormal(niw.μ, Σ ./ niw.λ))
    return MvNormal(μ, Σ)
end
```

3.4.2 Posteriors

We defined a private function `_posterior` that does the posterior parameter updates according to equations given in section 3.1. For NIW the `_posterior` is given in below:

```
function _posterior(m::NormalInverseWishart{V},T::DPGMMStats{V}) where V<:Real
    λn   = m.λ + T.n
    νn   = m.ν + T.n
    μn   = (m.λ * m.μ + T.nμ)/λn
    Ψn   = m.Ψ + T.S + m.λ * (m.μ * m.μ') - λn * (μn * μn')
    return (μn,λn,Ψn,νn)
end
```

For the users we exports `posterior`, `posterior_predictive` functions which are calling `_posterior` inside.

Posterior:

Posterior function should return the posterior distribution by doing parameters updates according to given sufficient statistics. For NIW the `posterior` is given by:

```
posterior(m::NormalInverseWishart{V},T::DPGMMStats{V}) where V<:Real =
    T.n!=0 ? NormalInverseWishart(_posterior(m,T)...) : m
```

Posterior predictive:

Posterior predictive function should return the marginalized likelihood for new data after updating parameters with given sufficient statistics. For NIW the `posterior_predictive` is given by:

```
function posterior_predictive(m::NormalInverseWishart{T}) where T<:Real
    df = m.ν-length(m.μ)+1
    MvTDist(df, m.μ, ((m.λ+1)/(m.λ*df)) * m.Ψ)
end

function posterior_predictive(m::DPGMM{V,D},T::DPGMMStats{V}) where {V<:Real,D}
    if T.n!=0
        (μn,λn,Ψn,νn) = _posterior(m,T)
        df = νn-D+1
        MvTDist(df, μn, ((λn+1)/(λn*df)) * Ψn)
    else
        posterior_predictive(m)
    end
end
```

3.4.3 Single data point updates:

In collapsed cluster algorithm we need single data points updates to posterior predictive. We might do this by constructing a new `suffstats`, however there is a faster way for NIW updates as explained in section 3.1. We implemented fast single data points updates to posterior predictive distribution:

```
function downdate_predictive(p::NormalInverseWishart, m::MvTDist, x::AbstractVector{V},
n::Int) where {V<:Real}
    λ   = p.λ+n # prior needed for λ
    dfn = m.df-1
    λn  = λ - 1
    μn  = (λ * m.μ - x)/λn
    MvTDist(dfn, μn, PDMat(((λn+1)/(λn*dfn)) *
    lowrankdowndate(((λ*m.df)/(λ+1))*m.Σ.chol,sqrt(λ/λn) * (x-m.μ))))
end

function update_predictive(p::NormalInverseWishart, m::MvTDist, x::AbstractVector{V},
n::Int) where {V<:Real}
    λ   = p.λ+n
    dfn = m.df+1
    λn  = λ + 1
    μn  = (λ * m.μ + x)/λn
    MvTDist(dfn, μn, PDMat(((λn+1)/(λn*dfn))*
    lowrankupdate(((λ*m.df)/(λ+1))*m.Σ.chol, sqrt(λ/λn) * (x-m.μ))))
end
```

4 Algorithms

The code for the algorithms are intended to be very similar to what is presented in pseudocodes. Julia's multiple dispatch and type system made it easier.

4.1 Collapsed Gibbs Sampler

The main code for collapsed gibbs sampler is presented in below* where:

- `model`: DPMM model which stores prior distribution and α
- `labels`: Initial labels for data
- `X` : Data
- `cluster0`: Empty cluster representing a new cluster

```
clusters = Clusters(model,X,labels) # current clusters
cluster0 = CollapsedCluster(model)  # empty cluster
for t in 1:T
    for i=1:N
        x, z = X[:,i], labels[i]
        clusters[z] -= x # remove xi's statistics
```

```
        isempty(clusters[z]) && delete!(clusters,z)
        probs     = CRPprobs(model,clusters,cluster0,x) # CRP probabilities
        znew      =~ Categorical(probs,NoArgCheck()) # new label
        labels[i] = place_x!(model,clusters,znew,x)
    end
end
return labels
```

`CRPprobs` calculates CRP probabilities:

```
function CRPprobs(model::DPGMM{V}, clusters::Dict, cluster0::AbstractCluster,
x::AbstractVector) where V<:Real
    probs = Array{V,1}(undef,length(clusters)+1)
    for (j,c) in enumerate(values(clusters))
        probs[j] = c(x)
    end
    probs[end] = model.α*pdf(cluster0,x)
    return probs/sum(probs)
end
```

`place_x!` add seats `x` to its new cluster by handling sufficient statistics:

```
function place_x!(model::DPGMM,clusters::Dict,knew::Int,xi::AbstractVector)
    cks = collect(keys(clusters))
    if knew > length(clusters)
        ck = maximum(cks)+1
        clusters[ck] = CollapsedCluster(model,xi)
    else
        ck = cks[knew]
        clusters[ck] += xi # add xi's statistics to new cluster
    end
    return ck
end
```

4.2 Quasi-Collapsed Gibbs Sampler

Quasi-Collapsed sampler algorithm is differs collapsed sample by deleting single-data point updates to clusters, and adding `CollapsedCluster` calculation before each iteration.

```
clusters = CollapsedClusters(model,X,labels) # current clusters
cluster0 = CollapsedCluster(model)  # empty cluster
for t in 1:T
    for i=1:N
        x =  X[:,i]
        probs     = CRPprobs(model,clusters,cluster0,x) # CRP Probabilities
        znew      =~ Categorical(probs,NoArgCheck()) # new label
        labels[i] = label_x(clusters,znew)
    end
    clusters = CollapsedClusters(model,X,labels) # TODO handle empty clusters
end
return labels
```

`label_x` assign the label according to `znew`:

```
function label_x(clusters::Dict,knew::Int)
    cks = collect(keys(clusters))
    if knew > length(clusters)
        return maximum(cks)+1
    else
        return cks[knew]
    end
end
```

4.3 Direct Gibbs Sampler

In direct sampler we will use `DirectCluster` which keeps sampled cluster parameters. We will sample mixture weights in the beginning of each iteration:

```
clusters = DirectClusters(model,X,labels) # current clusters
cluster0 = DirectCluster(model)  # empty cluster
for t in 1:T
    πs  = mixture_πs(model,clusters) # unnormalized weights
    for i=1:N
        x =  X[:,i]
        probs     = ClusterProbs(πs,clusters,cluster0,x) # cluster likelihoods
        znew      =~ Categorical(probs,NoArgCheck()) # new label
        labels[i] = label_x(clusters,znew)
    end
    clusters = DirectClusters(model,X,labels) # TODO handle empty clusters
end
return labels
```

`ClusterProbs` calculates likelihood according to given $\pi's$ which are sampled from Dirichlet prior:

```
function ClusterProbs(πs::AbstractVector{V}, clusters::Dict, cluster0::AbstractCluster,
x::AbstractVector{V}) where V<:Real
    probs = Array{V,1}(undef,length(clusters)+1)
    for (j,c) in enumerate(values(clusters))
        @inbounds probs[j] = πs[j]*pdf(c,x)
    end
    probs[end] = πs[end]*pdf(cluster0,x)
    return probs/sum(probs)
end
```

```
function mixture_πs(model::DPGMM{V}, clusters::Dict) where V<:Real
    return ~Dirichlet([(c.n for c in values(clusters))...;model.α])
end
```

4.4 Quasi-Direct Gibbs Sampler

The main code for quasi-direct gibbs sampler is presented in below. It differs from direct gibbs sampler by skipping mixture weight sampling. It instead use CRP weights to as described in Algorithm 4.

```
clusters = DirectClusters(model,X,labels) # current clusters
cluster0 = DirectCluster(model)  # empty cluster
for t in 1:T
    record!(observables,labels,t)
    for i=1:N
        x =  X[:,i]
        probs     = CRPprobs(model,clusters,cluster0,x) # CRP Probabilities
        znew      =~ Categorical(probs,NoArgCheck()) # new label
        labels[i] = label_x(clusters,znew)
    end
    clusters = DirectClusters(model,X,labels) # TODO handle empty clusters
end
return labels
```

5 Parallel Algorithms

5.1 Data Sharing

As explained in the previous sections, quasi-direct gibbs sampler is parallelizable throughout the data points. Julia's `Distributed` and `SharedArrays` modules are used for the implementation. Our aim

is minimizing communication between `main` and `worker` nodes during iterations. For this purposes, we share $\mathcal{X}$ and hyperparameters in the beginning with all the `worker` nodes.

`@everywhere` allows us to do this efficiently:

```
# Initialize worker nodes
addprocs(O[:ncpu])
@everywhere begin
    using Pkg; Pkg.activate(".")
    using DPMM, SharedArrays
end

# Send data to worker nodes
@everywhere begin
    const α     = $O[:alpha]
    const X     = $data
    const D     = $O[:D]
    const N     = $O[:N]
    const model = $dpmm
    const empty_cluster = $cluster0
end
```

Each worker needs to calculate probabilities for some of the data points and then assign labels to them. `SharedArrays` module helps us to create a shared label array every worker has acces to different indices of the array. `localindices` function helps eachworker to find the range of indices they are responsible for. Therefore, we created a `SharedArray` with initial cluster labels:

```
shared_labels = SharedArray(plabels)
```

5.2 Quasi-Collapsed Gibbs Sampler

We define a quasi-collapsed gibbs sampler iteration function with specific range of indices:

```
function quasi_collapsed_parallel!(model, X, range, labels, clusters, empty_cluster)
    for i in range
        probs      = CRPprobs(model,clusters,empty_cluster,X[:,i]) # chinese restraunt process probabilities
        znew       =~ Categorical(probs,NoArgCheck()) # new label
        labels[i]  = label_x(clusters,znew)
    end
end
```

We also need a caller function for each worker which call `quasi_collapsed_parallel!` with its range:

```
@inline quasi_collapsed_gibbs_parallel!(labels, clusters) =

    quasi_collapsed_parallel!(Main.model,Main.X,localindices(labels),labels,clusters,Main.empty_cluster)
```

The only thing remains to be shared in each iteration is `clusters`. Clusters are updated posterior parameters, so we need update them in main node, then send each worker. We will use `remotecall` utility of julia to call `quasi_collapsed_gibbs_parallel!` while sending `clusters` at the same time. So, this what main body of the parallel direct gibbs sampler:

```
function quasi_collapsed_gibbs_parallel!(model, X, clusters, labels::SharedArray; observables=nothing, T=10)
    for t=1:T
        record!(observables,labels,t)
        @sync begin
            for p in procs(labels)
                @async remotecall_wait(quasi_collapsed_gibbs_parallel!,p,labels,clusters)
            end
        end
        clusters = CollapsedClusters(model,X,labels)
    end
end
```

5.3 Direct Gibbs Sampler

We define a direct gibbs sampler iteration function with specific range of indices:

```
function direct_parallel!(πs, X, range, labels, clusters, empty_cluster)
    for i in range
        probs      = ClusterProbs(πs,clusters,empty_cluster,X[:,i]) # chinese restraunt process probabilities
        znew       =~ Categorical(probs,NoArgCheck()) # new label
        labels[i]  = label_x(clusters,znew)
    end
end
```

We also need a caller function for each worker which call `direct_parallel!` with its range:

```
@inline direct_gibbs_parallel!(labels, clusters, πs) =

    direct_parallel!(πs,Main.X,localindices(labels),labels,clusters,Main.empty_cluster)
```

The only thing remains to be shared in each iteration is `clusters`. Clusters are sampled θ parameters, so we need sample them in main node, then send each worker. We will use `remotecall` utility of julia to call `direct_gibbs_parallel!` while sending `clusters` at the same time. So, this what main body of the parallel direct gibbs sampler:

```
function direct_gibbs_parallel!(model, X, clusters, labels::SharedArray; observables=nothing, T=10)
    for t=1:T
        record!(observables,labels,t)
        πs = mixture_πs(model,clusters) # unnormalized weights
        @sync begin
            for p in procs(labels)
                @async remotecall_wait(direct_gibbs_parallel!,p,labels,clusters,πs)
            end
        end
        clusters = DirectClusters(model,X,labels)
    end
end
```

5.4 Quasi-Direct Gibbs Sampler

We define a quasi-direct gibbs sampler iteration function with specific range of indices:

```
function quasidirect_parallel!(model, X, range, labels::SharedArray, clusters, empty_cluster)
    for i in range
        probs      = CRPprobs(model,clusters,empty_cluster,X[:,i])
        znew       =~ Categorical(probs,NoArgCheck())
        labels[i]  = label_x(clusters,znew)
    end
end
```

We also need a caller function for each worker which call `quasidirect_parallel!` with its range:

```
function quasidirect_gibbs_parallel!(labels, clusters)
    quasidirect_parallel(model,X,localindices(labels),labels,clusters,empty_cluster)
end
```

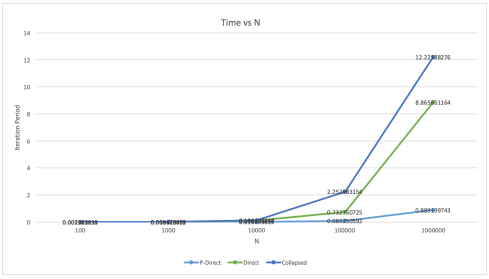
The only thing remains to be shared in each iteration is `clusters`. Clusters are sampled θ parameters, so we need sample them in main node, then send each worker. We will use `remotecall` utility of julia to call `quasidirect_gibbs_parallel!` while sending `clusters` at the same time. So, this what main body of the parallel quasi-direct gibbs sampler:

```
function quasidirect_gibbs_parallel!(model, X, clusters, labels::SharedArray; T=10)
```

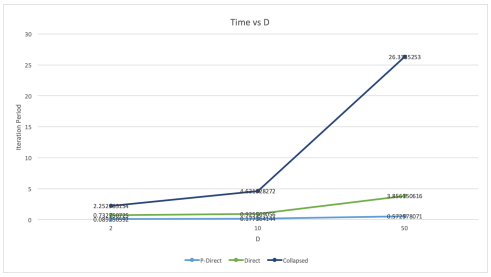
```
    for t=1:T
        @sync begin
            for p in procs(labels)
                @async remotecall_wait(quasidirect_gibbs_parallel!,p,labels,clusters)
            end
        end
        clusters = DirectClusters(model,X,labels)
    end
end
```

6 Results

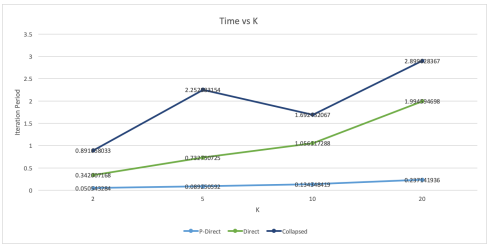
6.1 Time Analysis



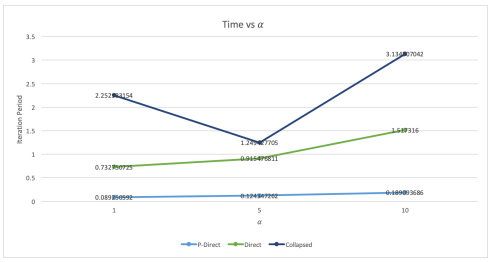
(a) $K = 5, \alpha = 1.0, D = 2$



(b) $N = 100.000, K = 5, \alpha = 1.0$



(c) $N = 100.000, \alpha = 1.0, D = 2$



(d) $N = 100.000, K = 4, D = 2$

Figure 2: Timing benchmarks of the algorithms

6.2 Convergence Analysis

References

[1] D. Blackwell, J. B. MacQueen, et al., Ferguson distributions via pólya urn schemes, The annals of statistics, 1 (1973), pp. 353–355.

[2] B. Fang, Introduction to dirichlet process, 2016.

[3] H. Kamper, Gibbs sampling for fitting finite and infinite gaussian mixture models, 2013.

[4] K. P. Murphy, Conjugate bayesian analysis of the gaussian distribution, def, 1 (2007), p. 16.

[5] M. Seeger, Low rank updates for the cholesky decomposition, (2004).

[6] J. Wishart, The generalised product moment distribution in samples from a normal multivariate population, Biometrika, 20 (1928), pp. 32–52.