Università degli Studi di Padova
Dipartimento di Scienze Statistiche

Corso di Laurea Magistrale in
Scienze Statistiche

Bayesian Nonparametric clustering for high dimensional data via infinite factorizations

Relatore: Prof. Antonio Canale
Dipartimento di Scienze Statistiche

Laureando: Lorenzo Cifelli
Matricola N 2018900

Anno Accademico 2021/2022

Contents

Introduction

With the increasing availability of cheap storage and the growing use of data streams, we are witnessing a huge increase in data volume and also in the number of attributes that are collected for each data point. While this implies a vast amount of information that may help to provide better understanding of the phenomena under study, it also involves many technical challenging analysis problems. One of these is the so called curse of dimensionality which affects all standard statistical methods.

Concerning data clustering, a common strategy to escape from the curse of dimensionality is to project data into a reduced dimension space. In line with this idea, Chandra et al. (2020) recently proposed a Bayesian nonparametric model-based clustering that overcomes the high-dimensional problem by performing clustering in a reduced dimension space of latent factor obtained by a factorization of the sample space. Motivated by the criticism on the definition of the latent space dimension before estimation, we propose a method that learns it from the data. This is done by exploiting an infinite factorization and an appropriate prior distribution. In this way, the posterior estimation of the latent space dimension occurs along with posterior distribution computation, and relieves the analyst from fixing or estimating that dimension before algorithm computation.

Chapter 1 introduces the main theoretical and computational characteristic of Bayesian nonparametric statistics, with a particular slant concerning clustering and an investigation on clusters posterior inference methodologies. In Chapter 2, the proposed Bayesian nonparametric clustering method based on infinite factorization is presented and a Gibbs sampling algorithm is developed for posterior computation. Special attention is given to the choice of the prior distribution. Chapter 3 discusses some numerical consideration to develop an efficient software implementation for the estimation algorithm. In Chapter 4 the performance and criticism of the proposed model are assessed under different scenarios through a simulation study. Finally, in Chapter 5 the proposed model is tested and compared with other methodologies in a real application related to Single-Cell RNA-Seq data clustering.

Chapter 1

Bayesian nonparametric statistics

Clustering methods aim to separate a heterogeneous collection of items into homogeneous subsets and are an important tool in scientific investigations in many different domains. Model-based clustering attempts to provide a measure of uncertainty to cluster assignments by considering data as coming from a mixture model. Formally, for the observation $y = (y_1, \ldots, y_p)^T$ the density function is given by

$$f(y;\theta) = \sum_{k=1}^{K} \pi_h \mathcal{K}(y;\theta_k^*) \tag{1.1}$$

where K is the number of clusters, $\pi = (\pi_1, \ldots, \pi_K)^T$ are the probability weights, $\mathcal{K}(\cdot;\cdot)$ is a density function parameterized by the vector of parameters $\theta \in \Theta$. A common choice for the kernel function is the multivariate Gaussian distribution. The assignment of a subject to a cluster is made by taking the cluster with the highest value $\pi_k \mathcal{K}(y;\theta_k^*)$.

One of the main challenges of clustering data is estimating the number of clusters K. A common solution is to fit several models, with a different number of clusters, and then select the one that is the best according to a comparison metric. This metric usually is characterized by two terms: the first one measure how well the model fits the data, while the second one is a penalized term favouring simpler models.

Bayesian nonparametric methods provide a solution by a different approach: rather than fit different models that vary in complexity, Bayesian nonparametric methods fit a single model that can adapt its complexity to the data. In other words, the number of clusters is part of the posterior distribution (Hjort et al., 2010). In this context, "nonparametric" indicates a model with an infinite number of parameters.

1.1 Infinite mixture models

In the Bayesian nonparametric framework, the model in 1.1 becomes an infinite mixture of distributions

$$y \sim f, \quad f(y;\theta) = \sum_{k=1}^{\infty} \pi_k \mathcal{K}(y;\theta_k). \tag{1.2}$$

Since this representation assumes an infinite number of parameters, the main issue is how to define a suitable prior distribution for them. A possible choice is provided by the stick-breaking process (Sethuraman, 1994) defined as follows:

$$\theta_k^* \overset{\text{iid}}{\sim} P_0, \quad V_k \overset{\text{ind}}{\sim} Beta(1,\alpha), \quad \pi_k = V_k \prod_{l<k}(1-V_l) \tag{1.3}$$

for $k = 1, \ldots, \infty$,with P_0 a base probability measure and $\alpha > 0$. The name stick-breaking derives from the metaphoric way to see the process. Consider a stick of unitary length, first break it at the location $V_1 \sim Beta(1,\alpha)$ and assign the probability V_1 to a random point $\theta_1^* \sim P_0$. The remaining probability mass $1 - V_1$ is split at the random location V_2 and the probability mass $V_2(1-V_1)$ is assigned to a second random point $\theta_2^* \sim P_0$. The process continues until the whole probability mass is assigned.

The model in 1.2 can be written in a more elegant form:

$$f(y;\theta) = \int_{\Theta} \mathcal{K}(y;\theta)\mathrm{d}P, \quad P = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k^*}, \tag{1.4}$$

where δ_* is the Dirac measure.

The random mixing measure P is distributed as a Dirichlet Process (DP) (Ferguson, 1973) which can be interpreted as the extension in infinite-dimensional parameters space of the Dirichlet distribution. As a stochastic process, it is a collection of random elements in a functional space which in this case is a probability measures space. In other words, the DP is a probability distribution over probability distributions.

To give a formal definition, we say that the distribution P defined in a measurable space Ω follow a DP with base distribution P_0 and concentration parameter $\alpha > 0$ if for every measurable finite partition of space Ω denoted $\{A_j\}_{i=1}^{J}$ we have that

$$(P(A_1), \ldots, P(A_J)) \sim Dir(\alpha P_0(A_1) \ldots, P_0(A_J)).$$

The infinite mixture model in 1.2 can be equivalently written in a hierarchical

form:

$$\begin{aligned} Y_i|\theta_i &\sim \mathcal{K}(Y_i;\theta_i), \qquad i = 1,\dots,n \\ \theta_i|P &\overset{\text{iid}}{\sim} P, \\ P &\sim DP(P_0,\alpha). \end{aligned} \tag{1.5}$$

In this formulation, the notation is slightly different. Despite the parameters θs being indicized by i, the discrete distribution P implies that some of them are repeated. In this way, the DP involves automatic clustering in the data as already seen with the stick-breaking representation.

The first two moments of the distribution can help to understand the role of the base distribution P_0 and of the parameter α. Let $P \sim DP(P_0,\alpha)$ and $A \subset \Omega$, we have that

$$\mathbb{E}(P(A)) = P_0(A), \quad \text{Var}(P(A)) = \frac{P_0(A)(1-P_0(A))}{1+\alpha}.$$

The above expressions show that the base distribution is the mean of the process while the parameters α can be interpreted as precision parameters. Indeed, as α increases, the variance goes to zero and therefore the distribution is more concentrated around P_0. However, it is worth pointing out that $\alpha \to \infty$ implies $P(A) \to P_0(A)$ and not that $P \to P_0$, since the support of the DP is the space of the discrete probability measure while P_0 is a continuous probability function. Figure 1.1 shows the realizations for different values of the precision parameter α.

The parameter α plays also an important role in the definition of the number clusters. Considering the stick-breaking representation, for a small value of α the $Beta(1,\alpha)$ density function of the random variables V_j is concentrated on values close to 1, and therefore most of the probability mass is assigned to the first atoms θ_js sampled. This implies that the mixture distribution is represented by only a few clusters. Formally, Antoniak (1974) showed that the expected value for the number of clusters K_n is

$$\mathbb{E}(K_n) = \sum_{i=1}^{n} \frac{\alpha}{\alpha+i-1} = \alpha(\psi(\alpha+n) - \psi(\alpha)),$$

where $\psi(\cdot)$ is the digamma function, and, for $n \to \infty$, converges to $\alpha \log(\frac{n}{\alpha})$. This implies that the number expected of clusters is much smaller than the number of observations and the precision parameter α is in a positive relationship with it. However, under the condition of α known and fixed, Miller and Harrison (2013) showed that posterior mixture distribution induced by DP is not consistent for

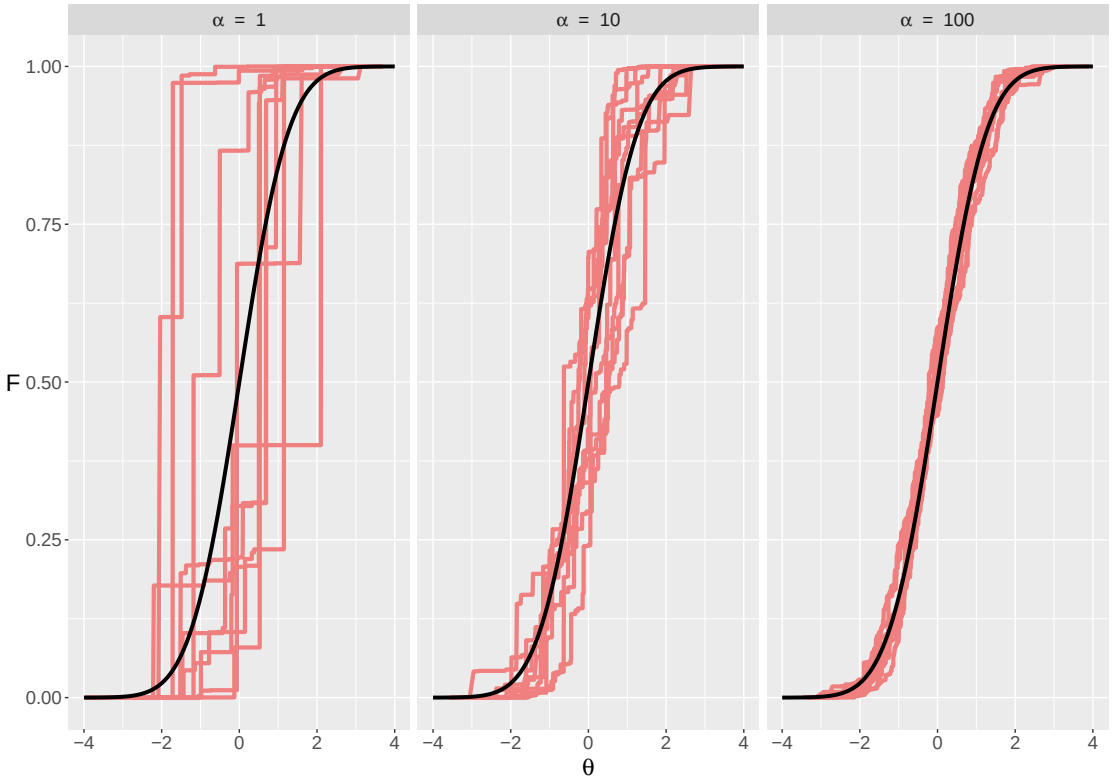

Figure 1.1: Representation of the empirical distribution function of 10 simulated samples from a DP(α, P_0) with P_0 the standard Gaussian distribution and $\alpha = (1, 10, 100)$. The black line is the distribution function P_0.

the number of components. Given the influence of α on the posterior distribution and in order to have more flexibility on the clustering of data, most implementations assume a weakly hyper-prior distribution and recently Ascolani et al. (2022) demonstrated that, under fairly general conditions on the hyper-prior distribution, the posterior distribution of the number of clusters is consistent.

1.2 DP posterior distribution

An important property of the DP is that of conjugacy. This is essential for easy computation of the posterior distribution. If $\theta_1, \ldots, \theta_n$ are i.i.d. with distribution $P \sim DP(P_0, \alpha)$, the resulting posterior distribution is still a DP

$$P|\theta_1, \ldots, X_n \sim DP(\alpha + n, \frac{\alpha P_0 + \sum_{i=1}^{n} \delta_{\theta_i}}{\alpha + n}).$$

The posterior expectation is hence

$$\mathbb{E}[P|\theta_1, \ldots, \theta_n] = \frac{\alpha}{\alpha + n} P_0 + \frac{n}{\alpha + n} \frac{\sum_{i=1}^{n} \delta_{\theta_i}}{n},$$

i.e., the posterior base distribution is a weighted mean of the prior base distribution and the empirical distribution of θ. The weight given to P_0 is proportional to the concentration parameters while for the empirical distribution the weight is proportional to the number of elements. In other words, as more information is available from the data, the posterior base distribution is more centred around the empirical distribution.

The same result occurs considering the stick-breaking representation. Given $\theta_1^*, \ldots, \theta_K^*$ unique elements of θ, the posterior distribution is:

$$\begin{aligned} P|\theta_1 \ldots, \theta_n &= \sum_{k=1}^{\infty} \pi_h \delta_{\theta_k^*}, \\ \pi_k &= V_k \prod_{l<k} (1 - V_l), \\ V_k &\sim Beta(1 + n_k, \alpha + m_k), \end{aligned}$$

where $n_k = \sum_{i=1}^{n} \mathbb{I}_{[\theta_i = \theta_k]}$ and $m_k = n - \sum_{l<=k} \sum_{i=1}^{n} \mathbb{I}_{[\theta_i = \theta_l]}$.

1.3 Posterior computation in DP mixtures

The posterior distribution of the infinite mixture model is not analytically tractable, therefore Markov Chain Monte Carlo (MCMC) methods are necessary for inference. Specifically, by appropriately choosing the prior distributions, it is possible to derive the full conditional distributions and consequently implement a Gibbs sampling algorithm. However, an additional challenge in Bayesian nonparametric methods is how to sample from a space where the number of parameters is infinite. To overcome this problem, two main classes of algorithms were developed: those called marginal and those called conditional.

The marginal methods rely on integrating out the underlying mixing probability P (Escobar and West, 1995; Müller et al., 1996). The conditional distribution of θ_i given all θ_js such that $j \neq i$, hereafter referred to as θ_{-i}, obtained integrating P is given by the Pólya urn scheme (Blackwell and MacQueen, 1973):

$$\theta_i|\theta_{-i} \sim \frac{1}{\alpha + n - i} \sum_{j \neq i} \delta_{\theta_j} + \frac{\alpha}{\alpha + n - i} P_0.$$

Given this formulation for the prior distribution, the computation of the posterior distribution $\text{pr}(\theta_i|\theta_{-i}, y_i) = \text{pr}(\theta_i|\theta_{-i}, y_i)\mathcal{K}(y; \theta_i)$ is straightforward and correspond to

$$\theta_i|\theta_{-i}, y_i \sim b \sum_{j \neq i} \delta_{\theta_j} \mathcal{K}(y_i; \theta_j) + b\alpha \text{pr}(y_i) \frac{\mathcal{K}(y_i; \theta_i) P_0(\theta_i)}{\text{pr}(y_i)},$$

where $\text{pr}(y_i) = \int \mathcal{K}(y_i; \theta) dP_0(\theta)$ and b is such that the posterior distribution is a probability distribution. By choosing the prior distribution P_0 and the likelihood $\mathcal{K}(\cdot; \theta)$ such that they are conjugate, sampling from the full conditional distribution is simple and it is equivalent to

- set $\theta_i = \theta_k^*$ with probability proportional to $n_{-i,k}\mathcal{K}(y_i; \theta_k^*)$ for $k = 1, \ldots, K_{-i}$,
- sample a new value from the posterior distribution $\text{pr}(y_i)^{-1}\mathcal{K}(y_i; \theta_i)P_0(\theta_i)$ with probability proportional to $\alpha \text{pr}(y_i)$.

where $n_{-i,k}$ is the number of elements in the cluster k without the subject i and K_{-i} is the number of clusters without subject i.

However, this algorithm is not efficient because the chain to move must pass through a low-probability intermediate state in which the observations in the group do not have all the same value and this occurs rarely. This results in slow convergence and poor mixing.

To overcome this problem, Neal (2000) introduced an auxiliary cluster indicator variable S_i which identifies the cluster-specific parameters $\theta_i = \theta_{S_i}^*$. As done before,

by integrating out the probability distribution P, the indicator variable S_i given S_{i-1}, the indicator variables without the subject i, has the following distribution

$$S_i|S_{-i} \sim \sum_{j \neq i} \frac{n_{S_j}}{\alpha + n - 1} \delta_{\theta^*_{S_j}} + \frac{\alpha}{\alpha + n - 1} \delta_{\theta^*_{S_{K_{-i}}}}.$$

Sampling the S_is from the posterior full conditional $\text{pr}(S_i|S_{-i}, \theta^*_1, \ldots, \theta^*_{K_{-i}}, y_i) = \text{pr}(S_i|S_{-i})\mathcal{K}(y_i; \theta^*_{S_i})$ can be done by

- drawing the cluster k among those already present with probability proportional to $n_{-i,k}\mathcal{K}(y_i; \theta^*_k)$,
- drawing a new cluster with probability proportional to $\alpha\mathcal{K}(y_i; \theta_{K+1})$ where θ_{K+1} is sampled from the prior P_0.

Finally, the posterior full condition for θ^*_ks is simply given by $\text{pr}(\theta^*_k|S_1, \ldots, S_n, y) = \prod_{i:S_i=k} \mathcal{K}(y_i; \theta^*_k) P_0(\theta^*_k)$.

On the other hand, the conditional approach derives a Gibbs sampling algorithm by conditioning on the probability P. This can be carried out by approximating the distribution P by truncating the stick-breaking representation in a finite sum of elements (Ishwaran and James, 2001). This approach is justified by the fact that the weights π_k decrease geometrically as the number of components K increases. Formally, (Ishwaran and James, 2001) showed that the error of approximation is of the order of $4ne^{-\frac{K-1}{\alpha}}$, and hence a close approximation is achievable by taking K sufficient large. Given the truncation, the problem is reduced to a finite mixture model and the derivation of a Gibbs sampling algorithm is straightforward.

Rather than implementing a truncation and an approximation of the distribution P, Walker (2007); Kalli et al. (2011) introduced a data augmentation step simulating n auxiliary latent variable u such that

$$f(u; \pi) = \sum_{k=1}^{\infty} \pi_k U(u; 0, \pi_k) = \sum_{k=1}^{\infty} \mathbb{I}(u < \pi_k),$$

with the joint distribution that is

$$f(y, u; \theta^*, \pi) = \sum_{k=1}^{\infty} \mathbb{I}(u < \pi_k) \mathcal{K}(y; \theta^*_k).$$

Given u, the number of components is finite and the conditional distribution is

$$f(y|u) = \frac{1}{N_u} \sum_{k \in A_u} \mathcal{K}(y; \theta^*_k),$$

where $A_u = \{j : \pi_j > u\}$ a finite set of indices and $N_u = \sum_{j \in A_u} \pi_k$. The algorithm is completed by introducing a further latent variable which indicates to which component the observation belongs.

Recently, Canale et al. (2022) proposed the Importance Conditional Sampling algorithm which reduces the infinite dimension problem by evaluating P in a finite-dimensional partition of the parameter space and sapling via the sampling importance resampling procedure from a distribution proportional to $\mathcal{K}(y;\theta)\mathrm{d}P$. Approximate realizations of the posterior distribution of f are then obtained by marginalizing θ out with a Gibbs sampler.

1.4 Clusters inference

A relevant problem of Bayesian nonparametric clustering is how to summarize the MCMC samples from the posterior distribution. The first difficulty is given by the label-switching phenomenon. The allocation of the labels may differ from iteration to iteration, but the resulting partition may be the same. In other words, due to the likelihood of being invariant to the labels used for the components, one can permute the labels on the different mixture components without having any impact on the likelihood. Hence, the interpretation of the parameters is not consistent with different samples of the MCMC algorithm. A second problematic aspect relies on the nature of the posterior sample provide by the MCMC algorithm. In MCMC methods, it is used to summarise the posterior distribution by computing the mean and some quantiles of interest of the MCMC draws, but doing this for a sample of partitions is not trivial. For this reason, appropriate summary tools are needed to extract information from the MCMC samples.

A simple solution is to estimate the clustering with the posterior mode, that is, the partition most frequently visited by the MCMC chain. However, there are

$$B_n = \sum_{k=1}^{n} \frac{1}{k!} \sum_{j=1}^{k} (-1)^j \binom{k}{j} (k-j)^n$$

partitions and, even for small n, this number is very large (e.g. $B_{50} \approx 10^{46}$). Hence, in most applications, the MCMC chain does not visit a partition more than once and therefore the posterior mode is meaningless. Some specific posterior mode search techniques have been proposed (Heller and Ghahramani, 2005), however, the mode is not a good estimator of the centre of the distribution, especially in high dimensional domains.

An alternative approach widely used is to compute the posterior similarity

matrix, which represents the probability that two observations belong to the same cluster and can be easily estimated from the Monte Carlo draws by counting how often two data points are clustered together. Given the similarity matrix, standard hierarchical clustering methods can be computed. See Medvedovic and Sivaganesan (2002) for an example.

Although in some applications this method provides an adequate estimate of the posterior distribution of partition, a more elegant approach based on decision theory has been proposed. The idea behind this approach is to define a suitable loss function $\mathcal{L}(\mathbf{c}, \hat{\mathbf{c}})$ which measures the loss to estimating true cluster allocation $\mathbf{c}$ with the estimate $\hat{\mathbf{c}}$. The true cluster allocation $\mathbf{c}$ is unknown, hence the goal becomes the minimization of the expected value of the loss function:

$$\mathbf{c}^* = \underset{\hat{\mathbf{c}}}{\operatorname{argmin}}\ \mathbb{E}[L(\mathbf{c}, \hat{\mathbf{c}})|\mathcal{Y}] = \underset{\hat{\mathbf{c}}}{\operatorname{argmin}}\ \sum_{c} L(\mathbf{c}, \hat{\mathbf{c}}) p(\mathbf{c}|\mathcal{Y}),$$

where $\mathcal{Y}$ represents the data. In other words, this approach looks for the clusters allocation closest to the posterior distribution of partitions according to a chosen loss function.

The definition of a suitable loss function is not trivial because it should satisfy basic principles such as invariance to permutations of the data point indices and invariance to permutations of the cluster labels for both the true and estimated clustering (Binder, 1978). This approach is first followed in Bayesian nonparametric clustering by Lau and Green (2007) and it was based on the Binder's loss function (Binder, 1978) which is defined as follows

$$\mathrm{B}(\mathbf{c}, \hat{\mathbf{c}}) = \sum_{i<j} l_1 \mathbb{I}[c_j = c_j] \mathbb{I}[\hat{c}_i \neq \hat{c}_j] + l_2 \mathbb{I}[c_i \neq c_j] \mathbb{I}[\hat{c}_i = \hat{c}_j],$$

that is, a quadratic function that, for all possible pairs of observations, penalizes the two errors of allocating two observations to different clusters when they should be in the same cluster or allocating them to the same cluster when they should be in different clusters. If the two types of error are penalized equally, $l_1 = l_2 = 1$, the n invariant version of the Binder loss is

$$\overline{\mathrm{B}}(\mathbf{c}, \hat{\mathbf{c}}) = \left(\sum_{i=1}^{k} \frac{n_{i+}^2}{n} + \sum_{j=1}^{\hat{k}} \frac{n_{+j}^2}{n} - 2 \sum_{i=1}^{k} \sum_{j=1}^{\hat{k}} \frac{n_{ij}^2}{n} \right),$$

where $n_{ij} = |C_i \cap \hat{C}_j|$ is the cardinality of the intersection between C_i, the set of data point indices in cluster i under $\mathbf{c}$, and $\hat{C}_i$, the set of data point indices in cluster j under $\hat{\mathbf{c}}$, k and $\hat{k}$ are the number of clusters respectively in $\mathbf{c}$ and in $\mathbf{c}'$,

$n_{i+} = \sum_j n_{ij}$ and $n_{+j} = \sum_i n_{ij}$.

Inspired by this approach, Wade and Ghahramani (2018) proposed the use of the VI loss (Meilă, 2007) which is defined as follows:

$$\begin{aligned}\mathrm{VI}(\mathbf{c}, \hat{\mathbf{c}}) &= \mathrm{H}(\mathbf{c}) + \mathrm{H}(\hat{\mathbf{c}}) - 2\mathrm{I}(\mathbf{c}, \hat{\mathbf{c}}) \\ &= -\sum_{i=1}^{k_n} \frac{n_{i+}}{n} \log\left(\frac{n_{i+}}{n}\right) - \sum_{j=1}^{\hat{k}_n} \frac{n_{+j}}{n} \log\left(\frac{n_{+j}}{n}\right) - 2\sum_{i=1}^{k_n}\sum_{j=1}^{\hat{k}_n} \frac{n_{ij}}{n} \log\left(\frac{n_{ij}n}{n_{i+}n_{+j}}\right) \\ &= \sum_{i=1}^{k_n} \frac{n_{i+}}{n} \log\left(\frac{n_{i+}}{n}\right) + \sum_{j=1}^{\hat{k}_n} \frac{n_{+j}}{n} \log\left(\frac{n_{+j}}{n}\right) - 2\sum_{i=1}^{k_n}\sum_{j=1}^{\hat{k}_n} \frac{n_{ij}}{n} \log\left(\frac{n_{ij}}{n}\right).\end{aligned}$$

The first two terms measure the uncertainty of the cluster allocation of an unknown randomly chosen data point given a particular clustering of the data points. The last term is the mutual information between the two clusterings and measures the reduction in the uncertainty of the cluster allocation of a data point in $\mathbf{c}$ when its cluster allocation in $\hat{\mathbf{c}}$ is known. The introduction of this alternative loss function is motivated by the fact that the Binder loss presents an asymmetric behaviour which prefers to split clusters than merge. This results in an overestimate of the number of clusters as shown by the simulation studies in Wade and Ghahramani (2018).

An important aspect of this approach is to develop an efficient algorithm to find the best partition $\hat{\mathbf{c}}$ that minimized the expected loss function which for the VI loss corresponds to

$$\mathbb{E}[\mathrm{VI}(\mathbf{c}, \hat{\mathbf{c}})|\mathcal{Y}] = \sum_{i=1}^{n} \log(\sum_{i'=1}^{n} \mathbb{I}(\hat{c}_{i'} = \hat{c}_i)) - 2\sum_{i=1}^{n} \mathbb{E}[\log(\sum_{i'=1}^{n} \mathbb{I}(c_{i^i} = c_i, \hat{c}_{i'} = \hat{c}_i))|\mathcal{Y}].$$

The research of the best partition is computationally challenging since is not feasible to explore the entire space of partitions. A simple but effective approach is to restrict the search space to only those partitions visited by the MCMC chain or those explored in a hierarchical clustering algorithm based on posterior similarity matrix (Fritsch, 2022).

Arguing that in many applications the clustering with the lower value of the loss function is not among the sampled partition, Wade and Ghahramani (2018) proposed a greedy algorithm based on the Hasse diagram that considers the whole partitions space. The Hasse diagram is built on the following definitions:

- for $\mathbf{c}, \hat{\mathbf{c}} \in \mathbf{C}$, $\mathbf{c} \leq \hat{\mathbf{c}}$ if for $i = 1, \ldots, k_n$, $C_i \subseteq \hat{C}_j$ for some $j = 1, \ldots, \hat{k}_n$;
- for any $\mathbf{c}, \hat{\mathbf{c}} \in \mathbf{C}$, $\mathbf{c}$ is covered by $\hat{\mathbf{c}}$ if $\mathbf{c} < \hat{\mathbf{c}}$ and there is no $\hat{\hat{\mathbf{c}}} \in \mathbf{C}$ such that

$$\mathbf{c} < \hat{\hat{\mathbf{c}}} < \hat{\mathbf{c}}.$$

Hence, at each step, the greedy algorithm moves locally to the best partition according to the expected loss function. In particular, given some partition $\hat{\mathbf{c}}$, it considers the l closest partitions that cover $\hat{\mathbf{c}}$ and the l closest partitions that $\hat{\mathbf{c}}$ covers, where the distance to determine the closest partitions corresponds to the loss function. Then, the posterior expected loss is computed for all the $2l$ considered partitions and the algorithm moves to the partition with the minimal value. The algorithm stops when the expected loss function does not improve or when it reaches the maximum number of iterations.

Moreover, Wade and Ghahramani (2018) suggest using the lower bound of the expected loss function obtained via Jensen's inequality swapping the log and the expectation in the second term:

$$-2\sum_{i=1}^{n}\log(\sum_{i'=1}^{n}P(c_{i'}=c_i|\mathcal{Y})\mathbb{I}(\hat{c}_{i'}=\hat{c}_i)).$$

In this way, the minimization depends only on the posterior similarity matrix which can be pre-computed based on the MCMC chain. In this way, the computation cost pass from $O(Mn^2)$ to $O(n^2)$, where M is the number of MCMC samples.

Wade and Ghahramani (2018) also proposed a method to characterized the uncertainty of the point estimate $\mathbf{c}^*$ by defining a credible ball of a given credible level $1-\alpha$, $\alpha \in (0,1)$, as

$$B_{\epsilon^*}(\mathbf{c}^*) = \{\mathbf{c} : d(\mathbf{c}^*, \mathbf{c}) \leq \epsilon^*\},$$

where $d(\cdot,\cdot)$ is a metric on the space of partitions (e.g. VI and Binder loss) and ϵ^* is the smallest $\epsilon > 0$ such that $\mathrm{P}(B_{\epsilon^*}|\mathcal{Y}) \geq 1-\alpha$. An estimate of ϵ^* and thus the credible ball can be computed from the MCMC chain. Given the MCMC samples $\{\mathbf{c}^m\}$ and the point estimate $\mathbf{c}^*$, for any $\epsilon > 0$,

$$\mathrm{P}(B_{\epsilon^*}|\mathcal{Y}) = \mathbb{E}[\mathbb{I}(d(\mathbf{c}^*,\mathbf{c}) \leq \epsilon^*)|\mathcal{Y}] \approx \frac{1}{M}\sum_{m=1}^{M}\mathbb{I}(d(\mathbf{c}^*,\mathbf{c}^m) \leq \epsilon^*),$$

and ϵ^* is the smallest $\epsilon > 0$ such that $\frac{1}{M}\sum_{m=1}^{M}\mathbb{I}(d(\mathbf{c}^*,\mathbf{c}^m) \leq \epsilon^*) \geq 1-\alpha$.

The credible ball can be a large set of partitions and is difficult to interpret. For this reason Wade and Ghahramani (2018) defined three types of bounds:

- the vertical upper bounds consist of the partitions in the credible ball with the smallest number of clusters that are most distant from $\mathbf{c}^*$;

- the vertical lower bounds consist of the partitions in the credible ball with the largest number of clusters that are most distant from $\mathbf{c}^*$;
- The horizontal bounds consist of the partitions in the credible ball that are most distant from $\mathbf{c}^*$.

For the formal definitions and some examples refer to Wade and Ghahramani (2018).

Chapter 2

Latent mixture for Bayesian clustering

2.1 Curse of dimensionality

In statistics, the curse of dimensionality (Bellman, 1966) is a well-known problem which occurs when the space dimension p of observed data is huge. As p increases, the points quickly scatter and get closer to the border of the sample space. To compensate for the increasing space between points, the sample size n has to grow exponentially with the number of parameters and this is not observable in any real problem. Formally, considering the Euclidean distance, the following result occurs

$$\lim_{p\to\infty} \mathbb{E}\left(\frac{\text{dist}_{\max}(p) - \text{dist}_{\min}(p)}{\text{dist}_{\min}(p)}\right) \to 0,$$

i.e., the difference between the minimum and the maximum distance between two data points becomes indiscernible compared to the minimum distance. Hence, in the clustering context, the distance-based methods fail because the distances between data points are indistinguishable from each other. In the same way, in Gaussian model-based clustering, as p increases, the model quickly becomes over-parameterized since the number of parameters grows as p^2. A serious computational issue also arises because it requires to deal with a huge $p \times p$ matrix. Moreover, in the worst case $p > n$, the sample covariance matrix is singular and its inversion is not even possible.

In Bayesian nonparametric clustering, Chandra et al. (2020) showed that when p is large, the posterior inference tends to assign either all subjects to the same cluster ($K = 1$) or all subjects to a different cluster ($K = n$). The different behaviour strongly depends on the choice of the kernel density $\mathcal{K}(\cdot;\cdot)$ and the base

measure P_0 for the θ_hs. Indeed, considering a Gaussian kernel, if we assume the conjugate multivariate normal inverse Wishart prior for the cluster-specific means and dispersion matrix, that is, we allow maximum flexibility to the model, as p increases the model soon becomes over-parameterized and as a result, it assigns all subjects to the same cluster thus limiting the number of parameters. On the other hand, we fall in the other extreme case $K = n$ if we assume an over-simplistic structure where all mixture components are spherical. From these results, it is clear that it is necessary to find a trade-off between parsimony and flexibility. Many methodologies have been proposed in the literature to overcome this problem and they can be split into two main families.

One approach focus on selecting only the features that bring the discriminant information. In standard approaches, this can be achieved by adding a penalty term to the log-likelihood function in the context of model-based clustering (Hastie et al., 2015) or some chosen criteria in model-free clustering (Bouveyron and Brunet-Saumard, 2014). Adding a penalty term leads to sparsity in parameter estimation, and hence an automatic features selection. In Bayesian nonparametric clustering, Kim et al. (2006) introduced a variable selection approach by incorporating into the model a latent variable that identifies Whether a feature is important for discriminating groups. As well as solving the high dimensions problem, the main advantage of these methods is interpretability which is of main interest in many applications. However, the computation of the posterior distribution is difficult.

The second approach is to reduce the dimension by projecting the data into a low-dimension space. In other words, this approach assumes that the information to split the data into groups is in a space of lower dimension and it looks for a transformation of data from the original space to the reduced space. A widely used transformation is the Principal Components Analysis and its sparse version (Zou et al., 2006). Once data are projected into the low-dimensional space, standard clustering techniques can be applied. A disadvantage of these methods is that the projection does not take into consideration the clustering goal, and therefore some discriminating information can be lost. In model-based clustering, dimension reduction can be achieved by the specification of a factorial model that reduces the number of parameters to estimate by a factorization of the cluster-specific covariance matrices. For an extensive review refer to Bouveyron and Brunet-Saumard (2014).

2.2 Latent factor mixture model

Following a dimensional reduction idea and the Bayesian nonparametric framework, Chandra et al. (2020) proposed a general class of latent factor mixture models (Lamb) defined as

$$y_i \sim N_p(\Lambda\eta_i, \Sigma), \quad \eta_i \sim \sum_{k=1}^{\infty} \pi_k N_H(\mu_k, \Delta_k), \tag{2.1}$$

where $\eta_i = (\eta_{i1}, \ldots, \eta_{iH})^T$ are H-dimensional latent variable with $H \ll p$, Λ is a $p \times H$ matrix of loadings, $\Sigma = \text{diag}(\sigma_1^2, \ldots, \sigma_p^2)$ is a $p \times p$ diagonal matrix and $\{\pi_k\}_{k=1}^{\infty}$ follow a stick-breaking representation. Under this model, the clustering occurs at the level of low-dimensional space of the latent variables η_i thus ensuring parsimony in the dimension of cluster-specific parameters. At the same time, high flexibility is favoured by leaving the mean through μ_k and the shape, size, and orientation through Δ_k of the distribution vary across clusters.

A distinction with the Mixture of factor analyzer (MFA) (Ghahramani et al., 1996) and with its improved Bayesian version (Murphy et al., 2020), which considers an infinite number of both mixture distributions and factors, is important to stress. In these models, clustering occurs at the level of the original sample space and a factorization is used to reduce the dimension of the cluster-specific covariance matrices. Although a substantial complexity reduction is performed that drops the number of parameters for the cluster specif covariance matrices from $\propto Kp^2$ to $\propto K(p \times H)$, the model still remains over-parameterized for p large. In order to overcome the dimensionality pitfall, Calo et al. (2006) introduced a model-based clustering based on Independent Factor Analysis (IFA) (Attias, 1999), where clustering occurs by a univariate Gaussian mixture of mutually independent latent factors. Clearly, it is an oversimplified representation and the range of the number of clusters that one can choose is limited by construction. In the same spirit as Lamb clustering, with the aim to sets a trade-off between the low flexibility of IFA and the over parameterization of MFA, Montanari and Viroli (2010) introduced the Heteroscedastic Factor Mixture Analysis (HFMA), which consists in a Gaussian mixture of latent factors with heteroscedastic components where observation are projected into the latent space by a unique loadings matrix. The main difference from Lamb is that it follows a frequentist approach and the estimation occurs by EM algorithm. On the other hand, a Bayesian approach allows some advantages as the number of clusters can be inferred along with posterior distribution, as for the dimension of the latent space. In addition, a penalization on the parameters of the loadings Λ to obtain a sparse structure and a more parsimony model can be

easily implemented by the definition of a shrinkage prior distribution.

2.3 Prior specification

A key task is to specify a suitable prior for the loadings matrix Λ. Given the high-dimension context $p \gg n$ and that Λ is a large $p \times H$ matrix, it is fundamental to reduce the number of parameters to estimate. In Bayesian statistics, a classical approach is to assume a shrinkage prior which induces sparsity (Bernardo et al., 2003; Carvalho et al., 2008) . These techniques were introduced following the same idea of Lasso regression (Tibshirani, 1996) which shrinks towards zero the coefficients in a regression model, improving both interpretability and prediction error. For some examples refer to Park and Casella (2008); Carvalho et al. (2009); Hans (2011); Armagan et al. (2013).

Following Bhattacharya et al. (2015), let $\lambda \in \mathbb{R}^H$ the vector of parameters, a general framework is to assume a global-local (GL) mixture of Gaussian,

$$\lambda_j \sim N(0, \psi_j \tau), \quad \psi_j \sim f, \quad \tau \sim g, \quad j = 1, \dots, d, \tag{2.2}$$

where τ controls global shrinkage toward the origin while the local scales ψ_js allow deviations in the degree of shrinkage. The distribution f and g are chosen so as to produce a resulting distribution that puts sufficient mass near zero but maintains heavy tails. In this way, the irrelevant parameters are estimated close to zero while, thanks to the fat tails, the large parameters are not excessively penalized.

Chandra et al. (2020) proposed to use the Dirichlet-Laplace (DL) prior (Bhattacharya et al., 2015) thus it is convenient both computationally and theoretically. From now with Lamb-DL we refer to the Lamb model based on DL prior. The DL prior assumes the following hierarchical form:

$$\begin{aligned} \lambda_{jh} | \psi, \phi, \tau \overset{\text{ind}}{\sim} N(0, \psi_{jh} \phi_{jh} \tau), \quad \psi_h \overset{\text{ind}}{\sim} \text{Exp}(1/2), \\ \phi \sim \text{Dir}(a, \dots, a), \qquad \tau \sim \text{Ga}(pa, 1/2), \end{aligned} \tag{2.3}$$

where λ_{jh} is the element in position (j, h) of the matrix Λ with $j = 1, \dots, p$ and $h = 1, \dots, H$, and ϕ is a vector of length pH.

One problematic aspect relating to the use of DL prior is the definition of the dimension H of the latent space. Chandra et al. (2020) used a pragmatic approach which consists in performing a PCA and setting H equal to the number of the first principal components that explain at least 95% of the variability of the data. By doing this, H assumes a large value and the prior shrinks to zero the extra

parameters of Λ ensuring parsimony. However, more elegant prior definitions for the loadings matrix have been proposed.

2.4 Infinite factorization model

Bhattacharya and Dunson (2011), Legramanti et al. (2020) and Schiavon et al. (2022) exploit an over-parameterized model that consider an infinite number of factors with loadings increasingly shrunk towards zero as the column index increases. In this way, flexibility is guaranteed by an infinite number of factors while overfitting is avoided due to the shrinkage prior.

Multiplicative gamma process

The Multiplicative gamma process (MGP) introduced by Bhattacharya and Dunson (2011) defines a shrinkage-type prior with the degree of shrinkage increasing along the column's index as follows,

$$\begin{aligned} \lambda_{jh}|\phi_{jh},\tau_h \sim N(0,\phi_{jh}^{-1}\tau_h^{-1}), \quad \phi_{jh} \sim Ga(\nu/2,\nu/2), \quad \tau_h = \prod_{l=1}^{h}\delta_l, \\ \delta_1 \sim Ga(a_1,1), \quad \delta_l \sim Ga(a_2,1),\ l \geq 2, \end{aligned} \tag{2.4}$$

for $j = 1,\dots,\infty$ where δ_l are independent, τ_h is a global shrinkage parameter for the h-th column and ϕ_{jh} are local shrinkage parameters for the elements in the h-th columns. Under the restriction $a_2 > 1$, the τ_hs are stochastically increasing which favours more shrinkage as the columns index increase. The introduction of the local shrinkage parameters ϕ_{jh} are deployed because the global shrinkage has the tendency to over-shrink nonzero loadings. However, Durante (2017) highlighted some critical aspect about hyper-parameters definitions. Beside Bhattacharya and Dunson (2011) claim that the parameters τ_h are stochastically increasing under the condition $a_2 > 1$, Durante (2017) showed, via simulations, that for $a_2 = 1.1$ an undesirable decreasing behaviour affected τ_h which leads to stochastic increasing distribution on $1/\tau_h$ and consequently to increasingly diffuse loadings λ_{jh}, under h increasing. Indeed, given $\mathbb{E}(\tau_h) = a_1 a_1^{h-1}$ and $\mathbb{E}(1/\tau_h) = 1/\{(a_1-1)(a_2-1)^{h-1}\}$, for $1 < a_1 < 2$ both τ_h and $1/\tau_h$ increase across the columns of Λ in expectation which is an undesirable behavior. In addition, the condition $a_1 > 1$ does not ensure stochastic order. However, it can be demonstrated that the mass assigned by the prior to small neighborhoods of zero increases with h, for all $a_1 > 0$ and $a_2 > 0$, facilitating growing shrinkage, and therefore Durante (2017) provides a set

of suitable values for a_1 and a_2 which ensure an increasing shrinkage.

Cumulative shrinkage process

Motivated by the practical disadvantages of MGP just described, Legramanti et al. (2020) formulated an alternative solution based on the idea of increasing shrinkage along with columns index, named Cumulative shrinkage process (CUSP). CUSP assumes that the variances follow a spike and slab distribution (Mitchell and Beauchamp, 1988) that assigns increasing mass to the spike for increasing column index. The main idea behind spike and slab prior distributions is to define a shrinkage prior with a mass of probability around 0, that is the spike, to shrink small values and fat tails to guarantee large values estimation.

Formally, let $(\lambda_{jh}|\theta_h) \sim N(0, \theta_h)$ for $(j = 1, \dots, p$ and $h = 1, 2, \dots,$ the Cumulative shrinkage process prior on θ_h is defined by assuming

$$(\theta_h|\pi_h) \sim (1 - \pi_h)G_0 + \pi_h \delta_{\theta_\infty}, \quad \pi_h = \sum_{l=1}^{h} \omega_l, \quad \omega_l = v_l \prod_{m=1}^{l-1}(1 - v_m), \tag{2.5}$$

where $v_1, v_2, \dots$ are independent $Beta(1, \alpha)$, G_0 is a diffuse continuous distribution representing the slab and δ_{θ_∞} defines a spike close to zero with θ_∞ a small value.

By assuming $G_0 \sim \text{InvGa}(a_\theta, b_\theta)$ and integrating out θ_h, each loading λ_{jh} has the marginal prior

$$\lambda_{jh} \sim (1 - \pi_h)t_{2a_\theta}(0, b_\theta/a_\theta) + \pi_h N(0, \theta_\infty), \tag{2.6}$$

where $t_{2a_\theta}(0, b_\theta/a_\theta)$ denotes the Student-t distribution with $2a_\theta$ degrees of freedom, location 0 and scale b_θ/a_θ. Hence, as showed in Figure 2.1. the prior distribution has a spike around 0 and fat tails deriving from the Student-t distribution. As in Section 1.1 for the infinite mixture model, the formulation 2.5 assumes an infinite number of parameters and the weights ω_ls follow a stick-breaking process. Hence, the probabilities π_hs assigned to the spike increase with the dimension of the latent space and $\lim_{h\to\infty} \pi_h = 1$ almost surely. In other words, the formulation 2.5 implies increasing shrinkage as the complexity grows in such a way as to facilitate the deletion of redundant terms for the benefit of the relevant ones which are represented from the slab. This behaviour is well shown in Figure 2.1.

The formulation 2.5 could alternative replaced with some pre-specified non-decreasing functions bounded between 0 and 1. However, Legramanti et al. (2020) showed that the probability function induced on $\pi = \{\pi_h \in (0, 1) : h = 1, 2, \dots\}$ by 2.5 has a large support on the whole space of non decreasing sequences taking values

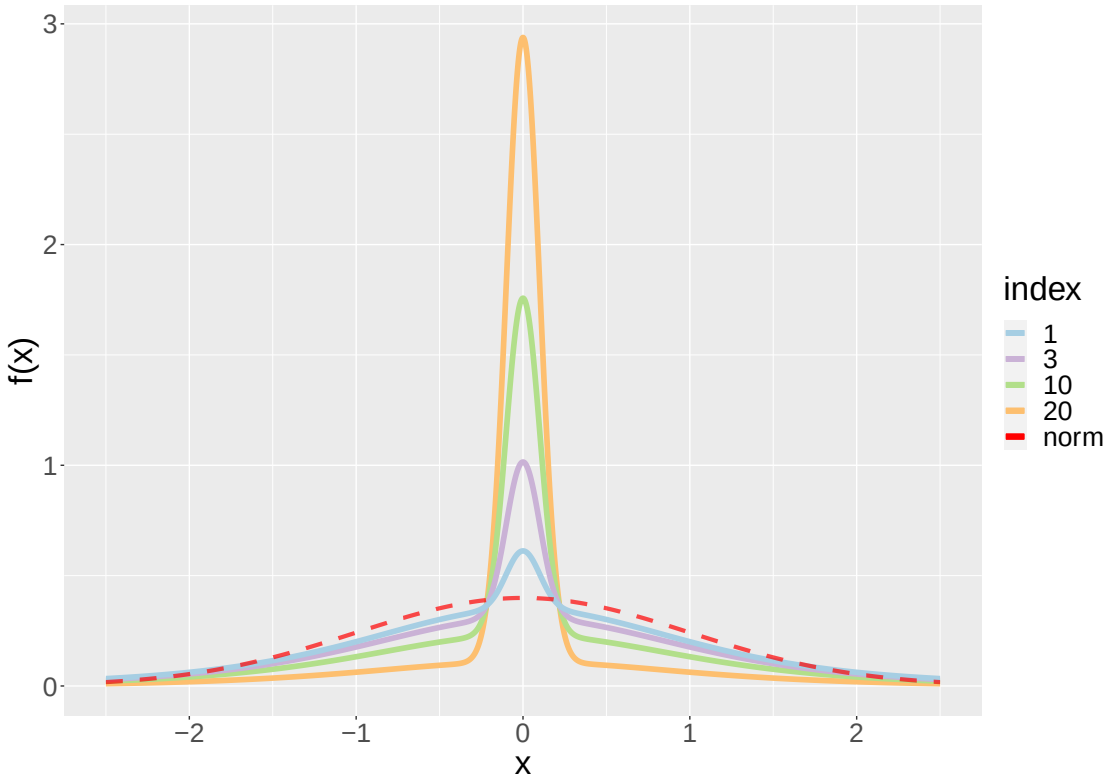

Figure 2.1: CUSP prior density function as the index columns increase, with $a_\theta = 2$, $a_\theta = 2$, $\theta_\infty = 0.01$, $\alpha = 20$ and $\{\pi_h\}_{h=1}^{\infty}$ randomly sampled. The red dashed line is the density function of standard Gaussian distribution.

in $(0, 1)$. Moreover, the shrinkage process derived is straightforward to understand and it allows a simple interpretation of the parameter α. Indeed, the expectation of θ_h is

$$\mathbb{E}(\theta_h) = \theta_\infty + \{\alpha(1+\alpha)^{-1}\}^h(\theta_0 - \theta_\infty),$$

where ϕ_0 is the expected value under the slab G_0. Hence, the prior expectation of θ_h converges exponentially towards the spike as h increases. Legramanti et al. (2020) demonstrated that, on the contrary to MGP, the following stronger notion of shrinkage in distribution holds

$$\text{pr}(|\theta_{h+1} - \theta_\infty| \leq \epsilon) > \text{pr}(|\theta_h - \theta_\infty| \leq \epsilon),$$

for any $\alpha > 0$, $h = 1, 2, \ldots$ and $\epsilon > 0$.

The hyper-parameter α plays an important role in the definition of the prior's properties. In particular, lower values of α induced faster concentration around θ_∞ and hence more rapid shrinkage of redundant terms, controlling the rate of increasing shrinkage without affecting the prior slab distribution of relevant terms. An insight as to what value fix α is derived by calculating the prior expectation of the number of active factors H^*, which correspond to the number of columns generated from the slab. The prior specification in 2.5 can be derived by marginalizing out the augmented indicator $c_h \sim Ber(1 - \pi_h)$ in $(\theta_h|c_h) \sim c_h P_0 + (1 - c_h)\delta_{\theta_\infty}$. Then, $H^* = \sum_{h=1}^{\infty} c_h$ and the expected value is

$$\mathbb{E}(H^*) = \sum_{h=1}^{\infty} \mathbb{E}(c_h) = \sum_{h=1}^{\infty} \mathbb{E}\{\mathbb{E}(c_h|\pi_h)\} = \sum_{h=1}^{\infty} \mathbb{E}(1-\pi_h) = \sum_{h=1}^{\infty} = \{\alpha(1+\alpha)^{-1}\}^h = \alpha,$$

hence, α should be set to the expected number of factors. Regarding the variance θ_∞ of the spike, it should be set close to 0 to allow shrinkage of redundant factors. However, the choice $\theta_\infty = 0$ should be avoided to guarantee a continuous prior function which improves MCMC chain mixing (Ishwaran and Rao, 2005).

Schiavon et al. (2022) proposed an improved version of CUSP prior by adding a structured shrinkage driven by meta covariates and controlling for the multiplicity problem within each column of Λ as p increases. Let $(\lambda_{jh}|\theta_h) \sim N(0, \theta_h)$, the structured increasing shrinkage process (SIS) follows a similar formulation as in 2.5

$$\begin{gathered} \theta_h = \vartheta_h \rho_h \psi_{jh}, \quad \vartheta_h^{-1} \sim Ga(a_\theta, b_\theta), \quad \rho_h \sim Ber(1 - \pi_h), \\ \psi_{jh} \sim Ber(\text{logit}^{-1}(x_j^T \beta_h) c_p), \quad \beta_h \sim N_H(0, \sigma_\beta I_H), \end{gathered} \tag{2.7}$$

where $\{\pi_h\}_{h=1}^{\infty}$ follow a stick-breaking construction, $a_\theta > 1$ and $c_p \in (0, 1)$ represent a positive offset. The introduction of meta-covariates in ψ_{jh} distribution allows to differently shrink the loading rows in function of some characteristics of the covariates. A positive offset such that $c_p \propto O(\log(p)/p)$ is introduced to guarantee asymptotically increasingly sparsity in p.

All methods avoid strict constraints on the number of factors or the order of the data. This is because the identifiability of a unique loadings matrix is not necessary if the purpose is the prediction or the inference of the covariance data. However, if one is interested in the loadings matrix inference, there is rich literature proposing post-processing algorithms that align posterior samples of Λ in such a way as to obtain interpretable posterior summaries (Schiavon et al., 2022).

For a computational purpose, the infinite loadings matrix is approximated by a finite matrix Λ_H having few columns H relative to the number of features p. The truncation to a conservative dimension is justified by the fact that, for CUSP and with similar motivation for MGP and SIS, Λ_H converges to Λ at a rate which is exponential in H. MGP truncates the columns of Λ for which all entries are within a distance $\epsilon > 0$ from zero, while for CUSP and SIS the inactive columns drooped correspond as those modelled by the spike and those for which $\rho_h = 0$ respectively. More details on how to perform truncation are provided in Section 2.5. Schiavon and Canale (2020) showed, via simulations, that, in MGP truncation described above, the estimation of the number of factors is several biased, and therefore provides an alternative method based on the trace of the covariance matrix. Schiavon and Canale (2020) also pointed out that the truncation of redundant columns of Λ strongly depends on the scale of the data. In particular, considering CUSP, let q be the full conditional probability of dropping a set of columns Λ_s and $r = q/(1-q)$, consider the case when the marginal distribution of the slab in 2.6 can be approximated by a Gaussian distribution, that is when a_θ is large, if the covariance matrix of the data is scaled by a factor c, then the new ratio r_c is given by $r_c = r^c$. For this reason, a standardization procedure of the data seems crucial to the inference of the number of latent factors.

Considering the practical advantages of CUSP on MGP and that in many applications meta-covariates are not available to perform the structure shrinkage of SIS, we decided to assume the CUSP prior for the loadings matrix Λ in 2.1, given also its computational simplicity.

2.5 Posterior computation of Lamb-CUSP model

The specification of the latent mixture model in 2.1 is completed by assuming a CUSP prior 2.5 for the matrix of loadings Λ and an independent Gaussian inverse-Wishart prior with locations μ_0, precision parameter κ_0, inverse scale matrix Δ_0 and degrees of freedom ν_0 for the cluster-specific parameters $\{\mu_k, \Delta_k\}_{k=1}^{\infty}$. From now, we refer to this model with Lamb-CUSP. Hence, by considering the DP prior, the η_is prior specification can be equivalently represented in the following hierarchical form

$$\eta_i|\mu_i, \Delta_i \overset{\text{ind}}{\sim} \mathrm{N}_H(\mu_i, \Delta_i), \quad \mu_i, \Delta_i|P \overset{\text{ind}}{\sim} P, \quad P \sim \mathrm{DP}(\alpha_{\mathrm{DP}}, P_0), \tag{2.8}$$

where $P_0 = \mathrm{NIW}(\mu_0, \Delta_0, \kappa_0, \nu_0)$. We set $\mu_0 = 0$ and $\Delta_0 = \xi^2 I_H$. Following West (1992), we assume the hyper-prior $\alpha_{\mathrm{DP}} \sim \mathrm{Ga}(a_\alpha, b_\alpha)$ to provide more flexibility to the model since the parameter α_{DP} plays an important role in the posterior distribution of the number of clusters as noted by Ascolani et al. (2022). Finally, we assume $\sigma_j^2 \sim \mathrm{InvGa}(\alpha_\sigma, \beta_\sigma)$ for $j-$element of the diagonal covariance matrix Σ.

For posterior computation we define a Gibbs sampling algorithm based on a marginalization approach as explained in section 1.3 with the introduction of the cluster indicators c_i for $i = 1, \ldots, n$. Its derivation is quite straightforward since we have assumed conjugate distributions. An alternative unconjugated model specification is, however, possible by incorporating into the Gibbs sampling one or more Metropolis-Hasting steps. The algorithm consists of the following steps:

Step 1 Update the cluster-specific dispersion matrices Δ_ks.

Following Chandra et al. (2020), μ_k is integrated out to improve mixing and the resulting distribution is analytically known due to conjugation. Hence, for $k = 1, \ldots, K$ sample Δ_ks from the inverse-Wishart distribution $\mathrm{IW}(\hat{\psi}_k, \hat{\nu}_k)$ where

$$n_k = \sum_{i=1}^{n} \mathbb{I}(c_i = k) \quad \bar{\eta}_k = \frac{1}{n_k} \sum_{i:c_i=k} \eta_i, \quad \hat{\nu}_k = \nu_0 + n_k,$$
$$\hat{\psi}_k = \xi^2 I_H + \sum_{i:c_i=k} (\eta_i - \bar{\eta}_k)(\eta_i - \bar{\eta}_k)^T + \frac{\kappa_0 n_k}{\kappa_0 + n_k} \bar{\eta}_k \bar{\eta}_k^T.$$

Step 2 Update of the latent factors η_is

As for Δ_ks, the full conditional distribution is derived by integrating out μ_k and it is still a Gaussian distribution, hence for $i = 1, \ldots, n$ sample

$$(\eta_i|-) \sim \mathrm{N}_H\{\Omega_k^{-1}\hat{\rho}_k, \Omega_k^{-1} + \Omega_k^{-1}(\hat{\kappa}_{k,-i}\Delta_k)^{-1}\Omega_k^{-1}\},$$

where $n_{k,-i} = \sum_{j \neq i} \mathbb{I}(c_j = k)$, $\hat{\kappa}_{k,-i} = \kappa_0 + n_{k,-i}$, $\bar{\eta}_{k,-i} = \frac{1}{n_{k,-i}} \sum_{j:c_i=k; j \neq i} \eta_i$, $\hat{\mu}_{k,-i} = \frac{n_{k,-i}\bar{\eta}_{k,-i}}{n_{k,-i}+\kappa_0}$, $\hat{\rho}_k = \Lambda^T \Sigma^{-1} Y_i + \Delta_k^{-1} \hat{\mu}_{k,-i}$ and $\Omega_k = \Lambda^T \Sigma^{-1} \Lambda + \Delta_k^{-1}$.

Step 3 Update of the loadings matrix Λ.

Letting λ_j^T denote the jth row of Λ, $\eta = (\eta_1, \ldots, \eta_n)^T$, $D = \mathrm{diag}(\theta_1, \ldots, \theta_H)$ and $y^{(j)} = (y_{1j}, \ldots, y_{nj})^T$ for $j = 1, \ldots, p$ sample

$$(\lambda_j | -) \sim \mathrm{N}_H\{(D^{-1} + \sigma_j^{-2}\eta^T\eta)^{-1}\eta^T\sigma_j^{-2}y^{(j)}, (D^{-1} + \sigma_j^{-2}\eta^T\eta)^{-1}\}.$$

Step 4 Update of the cluster indicators c_is.

For $i = 1 \ldots, n$ sample the indicator variables c_i with probabilities:

$$\Pi(c_i = k) \propto \begin{cases} n_{k,-i} \int \mathrm{N}_H(\eta_i; \mu_k, \Delta_k) \mathrm{d}p(\mu_k, \Delta_k | c_{-i}, \eta_{-i}) & \text{for } k \in c_{-i} \\ \alpha_{DP} \int \mathrm{N}_H(\eta_i; \mu_k, \Delta_k) \mathrm{d}p(\mu_k, \Delta_h) & \text{for } k = K+1, \end{cases}$$

where $\eta_{-i} = \{\eta_j : j \neq i\}$ and $c_{-i} = \{c_j : j \neq i\}$. The integrals above can be interpreted respectively as a posterior and prior predictive distribution, and due the conjugacy they are analytically available:

$$\Pi(c_i = k) \propto \begin{cases} n_{k,-i} t_{\hat{\nu}_{k,-i}-H+1}(\eta_i; \hat{\mu}_{k,-i}, \hat{\psi}_{k,-1}) & \text{for } k \in c_{-i} \\ \alpha_{DP} t_{\nu_0-H+1}(\eta_i; 0, \frac{\kappa_0+1}{\kappa_0(\nu_0-H+1)}\Delta_0) & \text{for } k = K+1, \end{cases}$$

where $\hat{\nu}_{k,-i} = \nu_0 + n_{k,-i}$ and

$$\hat{\psi}_{k,-i} = \frac{\hat{\kappa}_{k,-i}+1}{\hat{\kappa}_{k,-i}(\hat{\nu}_{k,-i}-H+1)} \times \left[\Delta_0 + \sum_{j:c_j=k, j\neq i} (\eta_j - \bar{\eta}_{k,-i})^T(\eta_j - \bar{\eta}_{k,-i}) + \frac{\kappa n_{k,-i}}{\kappa + n_{k,-i}} \bar{\eta}_{k,-i}\bar{\eta}_{k,-i}^T\right].$$

Remove the empty clusters and update the number of clusters K.

Step 5 Update of the concentration parameters α_{DP} of DP process of the mixture model.

Following West (1992), first generate $\varphi \sim \mathrm{Beta}(\alpha_{DP}+1, n)$, evaluate $\pi/(1-\pi) = (a_\alpha + K - 1)/(n(b_\alpha - \log\varphi))$ and generate

$$\alpha_{DP} | \varphi, K \sim \begin{cases} \mathrm{Ga}(\alpha_{DP} + K, b_\alpha - \log\varphi) & \text{with probability } \pi \\ \mathrm{Ga}(\alpha_{DP} + K - 1, b_\alpha - \log\varphi) & \text{with probability } 1-\pi. \end{cases}$$

Step 6 Update of the variance of each variables

For $j = 1, \dots, p$ sample independently σ_j^2 from:

$$(\sigma_j^2|-) \sim \text{InvGa}\left(a_\sigma + \frac{n}{2}, b_\sigma + \frac{1}{2}\sum_{i=1}^{n}(y_{ij} - \lambda_j^T \eta_i)^2\right).$$

Step 7 Update of the hyper-parameters of the CUSP prior.
We use the sampler proposed by Legramanti et al. (2020) which is based on a data augmentation step that introduces the latent indicator variable z_h such that $\text{pr}(z_h = l|\omega_l) = \omega_l$ and

$$(\theta_h|z_h) \sim (1 - \mathbb{I}(z_h \leq h))\text{InvGa}(a_\theta, b_\theta) + \mathbb{I}(z_h \leq h)\delta_{\theta_\infty}.$$

Integrating out z_h brings back to the original prior expression. In other words, z_h identifies active columns and, conditioning on it, it is possible to sample from the full conditional as follows:

(a) letting λ_h denote the hth columns of Λ, for $h = 1, \dots, H$ sample z_h from the categorical distribution obtained by marginalizing out ϕ_h

$$\text{pr}(z_h = l|-) \propto \begin{cases} \omega_l N_p(\lambda_h; 0, \theta_\infty I_p), & \text{for } l = 1, \dots, h, \\ \omega_l t_{2a_\theta}(\lambda_h; 0, (b_\theta/a_\theta) I_p) & \text{for } l = h+1, \dots, H; \end{cases}$$

(b) for $l = 1, \dots, H-1$ sample v_l from

$$(v_l|-) \sim \text{Beta}\left(1 + \sum_{h=1}^{H}\mathbb{I}(z_h = l), \alpha + \sum_{h=1}^{H}\mathbb{I}(z_h > l)\right),$$

set $v_H = 1$, and update ω_l accordingly 2.5;

(c) for $h = 1, \dots, H$, if $z_h > h$ sample θ_h from

$$(\theta_h|-) \sim \text{InvGa}(a_\theta + 0.5p, b_\theta + 0.5\sum_{j=1}^{p}\lambda_{jh}^2),$$

else set $\phi_h = \phi_\infty$.

An important characteristic of the framework of the infinite factors is the inference of the dimension of the latent factor space, that is the number of columns of the loadings matrix Λ. While Lamb-DL formulation of Chandra et al. (2020) sets a conservative value $H > H_0$ in such a way to ensure the identification of the factors, Bhattacharya and Dunson (2011) proposed an adaptive Gibbs sampling

that infers H as the sampler proceeds. In this way, the latent space dimension estimation takes into account the entire complexity of the model and provides greater flexibility. Under the CUSP prior, the identification of the inactive factors is straightforwards since they correspond to the columns of Λ modelled by the spike. Hence, let $H^{(t)}$ and $H^{*(t)} = \sum_{h=1}^{H^{(t)}} \mathbb{I}(z_h^{(t)} > h)$ be at the iteration t the dimension of the latent space and the number of the active factors respectively

- if $H^{*(t)} < H^{(t)} - 1$ then set $H^{(t)} = H^{*(t)} + 1$, drop the inactive columns of Λ together with the associate parameters and add a final component to Λ, η, θ, ω sampled from the prior;
- else set $H^{(t)} = H^{(t-1)} + 1$ and add a final component to Λ, η, θ, ω sampled from the prior.

To satisfy the diminishing adaptation condition in Roberts and Rosenthal (2007), the adaptation occurs at each iteration t with probability $\exp(\alpha_0+\alpha_1 t)$, where $\alpha_0 < 0$ and $\alpha_1 < 0$. In addition, Legramanti et al. (2020) suggest to start the adaptation after a fixed number $\bar{t}$ of iterations to let the chain stabilize. Legramanti et al. (2020) initialized $H = p+1$ to ensure that $H > H_0$, and hence the converge to the true value. However, in the context of high-dimension problems, this initialization is not feasible for computational reasons. Following Bhattacharya and Dunson (2011), we initialize $H = 5 \log p$.

Chapter 3

Numerical implementation

The proposed model sets the basis for a considerable computational challenge. It is widely known that the main drawback of Bayesian statistics is its computational cost. Expect for a simple model where the posterior distribution is analytically known, posterior inference is performed through MCMC algorithms, which require repeating a series of operations for a large number of iterations to ensure that the algorithm reaches the convergence to the posterior distribution. For this reason, much effort has been made to implement an efficient algorithm and particular attention was given to choosing an appropriate programming language and methodology.

3.1 R and Cpp

While `R` provides a natural framework to work with data analysis, it is also popular for being slower compared with other programming languages. One of its strengths is extreme dynamism (Wickham, 2019), which means that almost anything can be modified after it is created. Therefore, operations that in a static programming language are performed during compilation, in `R` are executed at run time. Thanks to this dynamism, code writing is more natural and intuitive, allowing an interactive approach between code writing and execution that enables get quick data analysis. However, it also represents a pitfall and involves inefficiency in both execution time and memory usage (Morandat et al., 2012). This is because `R`, every time it has to execute an instruction, it has to infer the type of the objects involved and apply accordingly the right method. Take as example the method `` `+` `` applied to two object, i.e. `` `x + y` ``, `R` has to infer if `` `x` `` and `` `y` `` are for example integers, floats, vectors or matrices and then has to find the right method accordingly. Regarding memory usage, consistent with object run time alteration,

R implicitly needs to (re-)allocate memory based on program individual operations. This does not happen in compiled languages where the object types and memory allocation are declared, and during execution, they are known and fixed. More in general, the easier it is to predict what is going to happen, the easier it is for an interpreter or compiler to make optimization.

These drawbacks affect especially loop execution, where type inference and memory allocation are performed for every instruction of every loop. For this reason, in R is used to avoid loops and work with vectors. Since vectors are required to have the same type for every element, R does not need to check each element and can apply highly specific optimized methods. However, not for all problems R vectorization is possible, especially for iterative algorithms in which the execution of an iteration depends on the values at the previous iteration. This is the case of MCMC algorithms and therefore a R implementation of the Gibbs sampling algorithm described in section 2.5 would be severely affected by these inefficiencies.

For this reason, the core implementation of Gibbs sampling is written in Cpp, a compiled programming language, and based on the high-quality linear algebra library Armadillo. Then the model estimation occurs by calling the Cpp compiler on R by Rcpp library. The data manipulation, data visualization, and post-inference analysis are performed in R accordingly to the use of appropriate packages.

To give an idea of the run time differences, we compared the implementation in R and Cpp of the Gibbs sampling algorithm for the factorization model described in Legramanti et al. (2020). The R implementation is the one provided by the authors, while the Cpp implementation is a personal written version. As shown by Table 3.1, the gains in run times are huge, and the computational cost for type inference is especially notable for small dimensions p and H where the run time for algebra operations is shorter. In this case, the run time of R implementation is about 17 times that of Cpp implementation.

3.2 Efficient sampling from multivariate Gaussian distribution

Referring to the Gibbs sampler in Section 2.5, Steps 2, 3 and 4 result particularly computationally onerous because they require sampling/calculation from/of multivariate distributions. The algorithm implemented in most software routines to sample $x \in R^H$ from a multivariate Gaussian distribution $N_H(\mu, \Sigma)$ is by decomposing the covariance matrix Σ via the Cholesky decomposition $\Sigma = LL^T$, with L a lower triangular matrix $H \times H$, and set $x = \mu + Lz$, where z is a random

vector of length H of independents univariate standard Gaussian variables. In this way, exploiting the properties of the Gaussian distribution, x is a draw form a multivariate Gaussian distribution since is a linear combination of independent normally distributed variables, with mean $\mathbb{E}(x) = \mathbb{E}(\mu + Lz) = \mu + L\mathbb{E}(z) = \mu$ and covariance matrix $\text{Var}(x) = \text{Var}(\mu + Lz) = L\text{Var}(z)L^T = LL^T = \Sigma$.

However, the sampling from the multivariate Gaussian distributions of the form of the full conditional distributions in Steps 2 and 3 requires the inversion of a $H \times H$ matrix to obtain the mean and covariance matrix from the precision matrix. This operation, considering the Gauss–Jordan elimination algorithm, has a complexity $O(H^3)$ and consequently brings a bottleneck in speed.

A way to speed up the computation is to perform the inversion directly on the triangular matrix L derived by the Cholesky decomposition of the precision matrix. Hence, exploiting the structure of triangular matrices, a draw from a multivariate Gaussian distribution with mean $W^{-1}\mu$ and covariance W^{-1} can be obtained more efficiently by setting $x = L^{-1T}(\mu + z)$, given the result $W^{-1} = (LL^T)^{-1} = L^{-1T}L^{-1}$.

However, the inversion of the triangular matrix L has still a complexity of $O(H^3)$. To overcome this problem, the solution adopted, inspired by Rue (2001), avoids the inversion calculation and instead solves a series of linear systems. Hence, according to the notation of the section 2.5 and omitting some indexes for clarity of writing, let $W = D^{-1} + \sigma^{-2}\eta^T\eta$ and $\mu = \eta^T\sigma^{-2}y$, the algorithm to sample λ_j from its full conditional in Step 3 is the following

1. compute the Cholesky decomposition of the matrix $W = LL^T$,
2. sample H independent draws from the univariate standard Gaussian distribution and collect them in the column vector z,
3. solve the linear equation $La = \mu$ for a using forward-substitution,
4. solve the linear equation $L^T x = x + z$ for y using back-substitution,
5. return x as a draw from the full conditional distribution of λ_j.

The proof is straightforward, indeed, rewriting the two linear systems in one equation, we have that $x = L^{-1T}L^{-1}\mu + L^{-1T}z$ and applying the property of Cholesky decomposition and of Gaussian distribution, x is a sample from a multivariate Gaussian distribution with mean $\mathbb{E}(x) = L^{-1T}L^{-1}\mu = W^{-1}\mu$ and covariance matrix $\text{Var}(x) = L^{-1T}L^{-1}\text{Var}(z) = L^{-1T}L^{-1} = W^{-1}$. In this way, we sample from the target distribution by solving two linear systems, and, given the triangular structure of L, they can be solved with back/forward substitution which required a complexity of only $O(H^2)$.

Following a similar procedure for Step 2, according to the notation of the section 2.5 and omitting some indexes for clarity of writing, the algorithm to sample η_i from its full conditional is as follows

1. compute the Cholesky decomposition of $\Delta = L_\Delta L_\Delta^T$ and of $\Omega = L_\Omega L_\Omega^T$,
2. sample H independent draws from the univariate standard Gaussian distribution and collect them in the column vector z_1,
3. solve the linear equation $\hat{\kappa}^{-\frac{1}{2}} L_\Delta^T a_1 = z_1$ for a_1 using back-substitution and set $u = \hat{\rho} + a_1$, where $\hat{\rho} = \Lambda\Sigma^{-1}Y_i + \Delta^{-1}\hat{\mu}$
4. solve the linear equation $L_\Omega a_2 = u$ for a_2 using forward-substitution,
5. sample H independent draws from the univariate standard Gaussian distribution and collect them in the column vector z_2,
6. solve the linear equation $L_\Omega^T x = z_2 + a_2$ for y using back-substitution,
7. return x as a draw from the full conditional distribution of η_i.

Again, the proof that η_i comes from the target distribution can be obtained by rewriting

$$\begin{aligned} x &= L_\Omega^{-1T} z_2 + L_\Omega^{-1T} a_2 \\ &= L_\Omega^{-1T} z_2 + L_\Omega^{-1T} L_\Omega^{-1} u \\ &= L_\Omega^{-1T} z_2 + \Omega^{-1}(\hat{\rho} + a_1) \\ &= L_\Omega^{-1T} z_2 + \Omega^{-1}(\hat{\rho} + \hat{\kappa}^{-\frac{1}{2}} L_\Delta^{-1T} z_1); \end{aligned}$$

therefore, x is a sample from a multivariate Gaussian distribution with

$$\begin{aligned} \mathbb{E}(x) &= \Omega^{-1}\hat{\rho} \\ \mathrm{Var}(x) &= L_\Omega^{-1T} L_\Omega^{-1} \mathrm{Var}(z_2) + \Omega^{-1}(\hat{\kappa}^{-1} L_\Delta^{-1T} L_\Delta^{-1} \mathrm{Var}(z_1))\Omega^{-1} \\ &= \Omega^{-1} + \Omega^{-1}(\hat{\kappa}\Delta)^{-1}\Omega^{-1}. \end{aligned}$$

Bhattacharya et al. (2016) proposed an alternative algorithm for sampling from distribution as in Step 3 which avoids the computation of Cholesky decomposition by a data augmentation step. In this way, the complexity of the algorithm is $O(n^2 H)$ which is linear with the dimension space. although the algorithm brings a large reduction in computation time for H large and n small, for our application it is not suitable since we are looking for a relatively small H with usually $n > H$.

To assess the real differences between the methods mentioned, we measured their computation time. Hence, we compared the *naive* method which inverts the precision matrix and uses a standard routine function for sampling from a multivariate Gaussian distribution, the method based on the inversion of the *triangular* matrix of Cholesky decomposition and the method based on *back-forward* substitution. Comparison are made on `Cpp` implementations. For the reasons described above, the sampling algorithm based on back-forward substitutions is the most efficient in both Steps 2 and 3, and the benefit becomes more important as the dimension H increases, as shown in Table 3.2 and Table 3.3. In particular, the differences in computation time are especially conspicuous for the sampling of η_is. Indeed, for all dimensions H considered, the back-forward algorithm presents a computation time that is about half that for the algorithm based on the inversion of the triangular matrix. On the other hand, the computation time for the naive algorithm tends to be explosive as the dimension increases, and in particular, it is more than 5 times that of the back-forward substitution algorithm. Besides one can think that in practice there is no real gain since microseconds is almost imperceptible, the Gibbs sampling algorithms require repeating each step many times, and, in addition, Steps 2 and 3 is performed p and n times in each iteration. Therefore, if we consider, as an example $H = 40$, the settings of the simulation study in Chapter 4 with $p = 2500$, $n = 2000$ and 5000 Gibbs sampling iterations, the mean total time of computation for sampling all λ_js and all η_is is 30 minutes, 17 minutes and 9 minutes respectively for the naive algorithm, the algorithm based on the inversion of a triangular matrix and the algorithm based on back-forward substitution.

A similar strategy is used to compute the multivariate t distribution density in Step 4 in order to sample the cluster membership. Let $x \sim t_\nu(\mu, \Sigma)$ which density function is characterize by the quantity

$$(x - \mu)^T \Sigma^{-1} (x - \mu), \tag{3.1}$$

to avoid the computation of the inversion of Σ, that in Step 4 correspond to $\hat{\psi}_k$, the quantity 3.1 can be computed as $a^T a$ where a is the solution of the linear system $La = x - \mu$ for a solved by back substitution, with $\Sigma = LL^T$.

Table 3.1: Comparison in computation time (seconds) of `R` and `Cpp` implementations of the Gibbs sampling algorithm for the factorization model described in Legramanti et al. (2020) with $n = 100$.

p	H	`R`	`Cpp`	`R`/`Cpp`
10	5	105	6	16.7
50	10	336	36	9.3
100	15	1001	217	4.6

Table 3.2: Comparison in computation time (microseconds) of different algorithms to sampling λ_j from the multivariate Gaussian distribution in Step 3 with different dimension H. The column *rate* refers to the rate with the back-forward algorithm.

	$H = 20$		$H = 40$		$H = 100$	
Method	time	rate	time	rate	time	rate
Naive	10	1.1	33	1.3	407	2.2
Triangular	10	1.1	30	1.2	278	1.5
Back-forward	9	1.0	25	1.0	181	1.0

Table 3.3: Comparison in computation time (microseconds) of different algorithms to sampling η_i from the multivariate Gaussian distribution in Step 2 with different dimension H. The column *rate* refers to the rate with the back-forward algorithm.

	$H = 20$		$H = 40$		$H = 100$	
Method	time	rate	time	rate	time	rate
Naive	20	2.2	116	4.5	1419	5.5
Triangular	15	1.9	58	2.2	555	2.2
Back-forward	9	1.0	26	1.0	257	1.0

Chapter 4

Simulation study

To apply the method just described to a real data set, we conducted a simulation study to investigate the suitability of the model proposed. In particular, Lamb-CUSP model is compared with the Lamb-DL model of Chandra et al. (2020), which assumes the latent space dimension fixed, and with a two-step approach (PCA-KM) that performs a K-means algorithm on the low dimension space of the first approximate principal components that explain at least the 95% of the variability and select the number of cluster K that maximize the average silhouette width (Rousseeuw, 1987). The DP mixture of Gaussian model with diagonal covariance matrix and the nonparametric mixture of infinite factor analyzers (Murphy et al., 2020) have been also considered but in the high dimensional simulation settings, they showed high instability, including lack of convergence, memory errors, extremely long running times and other issues. For this reason, they were excluded.

To compare the clustering algorithms performance, we compute the Adjusted Rand Index (ARI) (Hubert and Arabie, 1985). The standard formulation of Rand Index (Rand, 1971) is defined, given two partitions, as the number of agreements, that is if in both partitions a pair is the same cluster or split, in all possible pairs divided by the total number of possible pairs $\binom{n}{2}$. Hence, the Rand index takes values in $[0, 1]$: 0 when no pair is classified in the same way under both clusterings, while 1 when the two estimated clustering are identical. However, the value of the Rand index depends on both the number of clusters and the number of subjects. In addition, Fowlkes and Mallows (1983) proved that the Rand index for independent clusterings converges to 1 as the number of clusters increases, and this is undesirable for a similarity measure. Therefore, an adjusted formulation of the Rand index is introduced and, following the notation in section 1.4, is defined

as

$$\text{ARI} = \frac{\sum_{ij}\binom{n_{ij}}{2} - \left[\sum_i \binom{n_{i+}}{2}\sum_j \binom{n_{+j}}{2}\right]}{\frac{1}{2}\left[\sum_i \binom{n_{i+}}{2} + \sum_j \binom{n_{+j}}{2}\right] - \left[\sum_i \binom{n_{i+}}{2}\sum_j \binom{n_{+j}}{2}\right]/\binom{n}{2}},$$

for $i = 1, \ldots, K$ and $j = 1, \ldots, \hat{K}$. In other words, it corrects the Rand index for a baseline accounting for the expected similarity between clusterings produced by a random model and if the Rand index is less than the expected index, the ARI is penalized by a negative value.

4.1 Data generation

The models are compared in a high dimensional scenario with $p = 2500$ and $n = 2000$. Similarly to the scenario used in Chandra et al. (2020), the number of clusters K_0 is set to $\{10, 25\}$ with the first $\lfloor 2K_0/3 \rfloor$ main clusters having the same probability and the remaining ones having together the same probability of a single main cluster. For example, if $K_0 = 10$, each unit is assigned with probability $1/7$ to one of the 6 main clusters and with probability $1/28$ to one of the minor clusters. Especially for $K_0 = 25$, this clusters allocation is quite challenging because the size of the minor clusters is small (≈ 14) and any methods struggle and tend to incorporate them into larger groups unless there are a small number of close to equal weight clusters that are well separated.

The simulated data are generated from different models to assess the performance under different scenarios, which are the following:

Lamb: mixture of latent factor described in section 2.2 where the number of clusters is fixed to K_0, Λ is set to a sparse matrix with many entries equal to zero and we set $H = 20$;

MFA: mixture of sparse analyzers defined as

$$y_i \overset{\text{iid}}{\sim} \sum_{k=0}^{K_0} \pi_k N_p(\mu_k, \Lambda_k \Lambda_k^T + \Sigma_h),$$

where Λ_k is a $p \times H$ order sparse matrix with $H = 20$, Σ_k is a $p \times p$ diagonal matrix with positive entries and $\mu_h \in R^p$ for $k = 1, \ldots, K_0$;

SpCount: mixture of log-transformed zero-inflated sparse Poisson counts, sample

count data from

$$w_{ij}|c_i = k \overset{\text{iid}}{\sim} \begin{cases} Pois(\lambda_k) + N(0,1) & \text{for all } j \in S_k \\ 0 \quad \text{with probability 1 for all } j \notin S_k \end{cases}$$
$$\text{pr}(c_i = k) = \pi_k \quad \text{for all } k = 1, \dots, K_0,$$

and set $y_{ij} = \log(w_{ij}+1)$ for $i = 1, \dots, n$ and $j = 1, \dots, p$, where $\{\lambda_1, \dots, \lambda_{K_O}\}$ are fixed positive constants and $\{S_1, \dots, S_{K_0}\}$ are random partitions of $\{1, \dots, p\}$ of the same size $r = \lfloor p/K_0 \rfloor$.

The aim of the last scenario is to mimic the scRNAseq dataset structure analyzed in Chapter 5 which consists of sparse discrete data and the cluster-specific distribution is highly non-Gaussian. Therefore, simulations are used to evaluate the robustness and performance of our model and to investigate appropriate values for hyper-parameters.

4.2 Prior definition and initialization

The common hyper-parameters of the prior distribution for both Lamb implementations were chosen following a weakly informative approach as Chandra et al. (2020). Hence we set $\xi^2 = 20$, $\kappa_0 = 0.001$, $\nu_0 = H + 50$, $\alpha_\alpha = \beta_\alpha = 0.1$ for the DP mixture prior and $a_\sigma = 1$, $b_\sigma = 0.3$ for the prior distribution on the residual variances. Concerning the loadings matrix, we choose $a = 0.5$ for DL prior as in Chandra et al. (2020), while for the infinite factorization we followed Schiavon and Canale (2020) and we set $a_\theta = 15$, $b_\theta = 2$ for the slab distribution of loadings and $\alpha = 20$ for the DP prior of cumulative shrinkage given the high dimension framework. After some simulations, the choice of the variance parameter of spike prior of non-informative loadings turned out to be not trivial. Despite Schiavon and Canale (2020) proposed to scale the data and choose $\theta_\infty = 0.01$, in our application this value was too high and the Gibbs sampling proposed, due to the different structure of data, fails to identify the significant columns because the posterior distribution is too shrunk. Hence, after some simulations, we found the appropriate trade-off between selection and shrinkage by setting $\theta_\infty = 0.00001$ under Gaussian data in Lamb and MFA scenarios, and $\theta_\infty = 0.0005$ under SpCount scenario.

An important task is about the initialization of Λ and η, which includes how to set H. Chandra et al. (2020) suggested an initialization relying on a approximated SVD decomposition based on the augmented implicitly restarted Lanczos bidiagonalization algorithm (Baglama and Reichel, 2005) and to set H as the smallest

index for which the truncated decomposition explains at least the 95% of the variability in the data. In particular, they set $\eta = UD$ and $\Lambda = V$, where U and V are respectively the matrices of left and right approximated singular vectors and D the diagonal matrix of approximated singular value. Notice that this is equivalent to performing a truncated PCA. Besides this approach showed great performances both in simulations and in a real application, we assessed model performances under random initialization because a) the PCA does not always represent a good dimension space reduction for clustering; b) the approximated SVD decomposition required, before fitting, to set the maximum number of singular values to estimate, and the resulting size of the latent space is highly dependent from theta, especially if a high level of explained variance is considered. Evidence of this is shown in Chapter 5. Moreover, given the particular structure of Λ and η derived from PCA, Lamb-CUSP implementation struggle with this initialization and the setting of the parameter θ_∞ appears to be challenging because the factor selection step gets stuck in two extreme cases: for θ_∞ small, deletion of redundant factors is not performed due small shrinkage, while it falls in the appositive case of deletion of all factors for bigger θ_∞, without being able to find the right balance of parsimony and flexibility. Therefore, for Lamb-DL we set H based on PCA approach, while for Lamb-CUSP we set $H = 5\log(p)$, and for both we randomly initialized Λ and η. Notice that for Lamb-CUSP H is only an initial value and Gibbs sampler adapts the estimation according to the data.

As initial values of cluster membership, we used the K-means algorithm with 20 and 40 clusters under the cases $K_0 = 10$ and $K_0 = 25$ respectively. We initialized each element of η and Λ matrices sampling from a Gaussian distribution with mean 0 and variance 5 for Lamb and MFA scenarios and 1 under the SpCount scenario. We run the Gibbs sampling algorithms for 6,000 iterations discarding the first 1,000 as burn-in and taking one draw every five to reduce autocorrelation, and the posterior clustering is computed by minimizing the VI loss as described in section 1.4.

4.3 Clustering comparisons

Figure 4.3 reports the distribution of the 20 replicates of the adjusted Rand index and the estimated number of clusters for the three methods considered. As already shown in Chandra et al. (2020), Lamb methods outperform the PCA-KM approach which achieved an ARI always smaller. Moreover, under Lamb scenario, sometimes it completely failed clustering and achieved an ARI below 0.5. Besides

PCA-KM approach seems to detect with an acceptable degree of accuracy the true number of clusters K_0, it misclassifies units clusters membership by associating them to the same cluster when they belong to two different clusters or by splitting them when they belong to the same group. On the other hand, both Lamb-DL and Lamb-CUSP yield an ARI close to 1 in all scenarios, with some exceptions in MFA and SpCount scenarios. However, we strongly believe that this relative under-performances is due to unlucky converges of Gibbs sampling and a re-estimation achieved performance in agreement with the other experiments. Both Lambs implementations failed in most cases to allocate adequately the observations of smaller clusters which are joined together into one or two larger clusters or with one of the major clusters. However, this represents a really challenging task.

In Figure 4.3 are compared the estimated dimensions of the latent space between Lamb-DL based on approximate SVD decomposition and Lamb-CUSP based on infinite factorization. Under the Lamb scenario for which the true dimension of latent space is known, both methods under-estimated the true value, especially the Lamb-DL method. Considering MFA and SpCount scenarios, we can not assess the suitability of the methodologies since the dimension of the latent space is not known, and there may not even be a real reduced space of clustered latent factors. However, considering $K_0 = 25$, they yield to similar estimation. On the other hand, for $K_0 = 10$ the two estimations differ significantly, especially under the MFA scenario for which the median are respectively 30 and 14 for Lamb-DL and Lamb-CUSP. This has an important impact on computation time, indeed, the mean time for 5000 iterations is about 67 minutes for Lamb-DL and about 49 minutes for Lamb-CUSP. In general, the estimated values vary in function of the true number of clusters K_0. In particular, Lamb-CUSP H is usually higher when $K_0 = 25$ compared to $K = 10$, and this seems quite reasonable since a larger dimension space may help to represent a more complex clustered structure. Moreover, Lamb-CUSP estimations showed more variability between experiments under the same scenario, and therefore a better ability in adapting to data.

To understand the impact of hyperparameters on the posterior partitions distribution, we perturbed the key hyperparameters. The scale parameter ξ^2 and the degrees of freedom ν of the Normal inverse-Wishart base measure resulted in having a significant impact on posterior clustering, especially in some cases. Indeed, for a small value of the quantity ξ^2/ν, that is for ξ^2 small and ν large, the posterior distribution tends to produce more clusters. This occurs because the prior distribution tends to be informative and therefore the posterior distribution struggle to join two clusters if there is not much information on data. This particularly

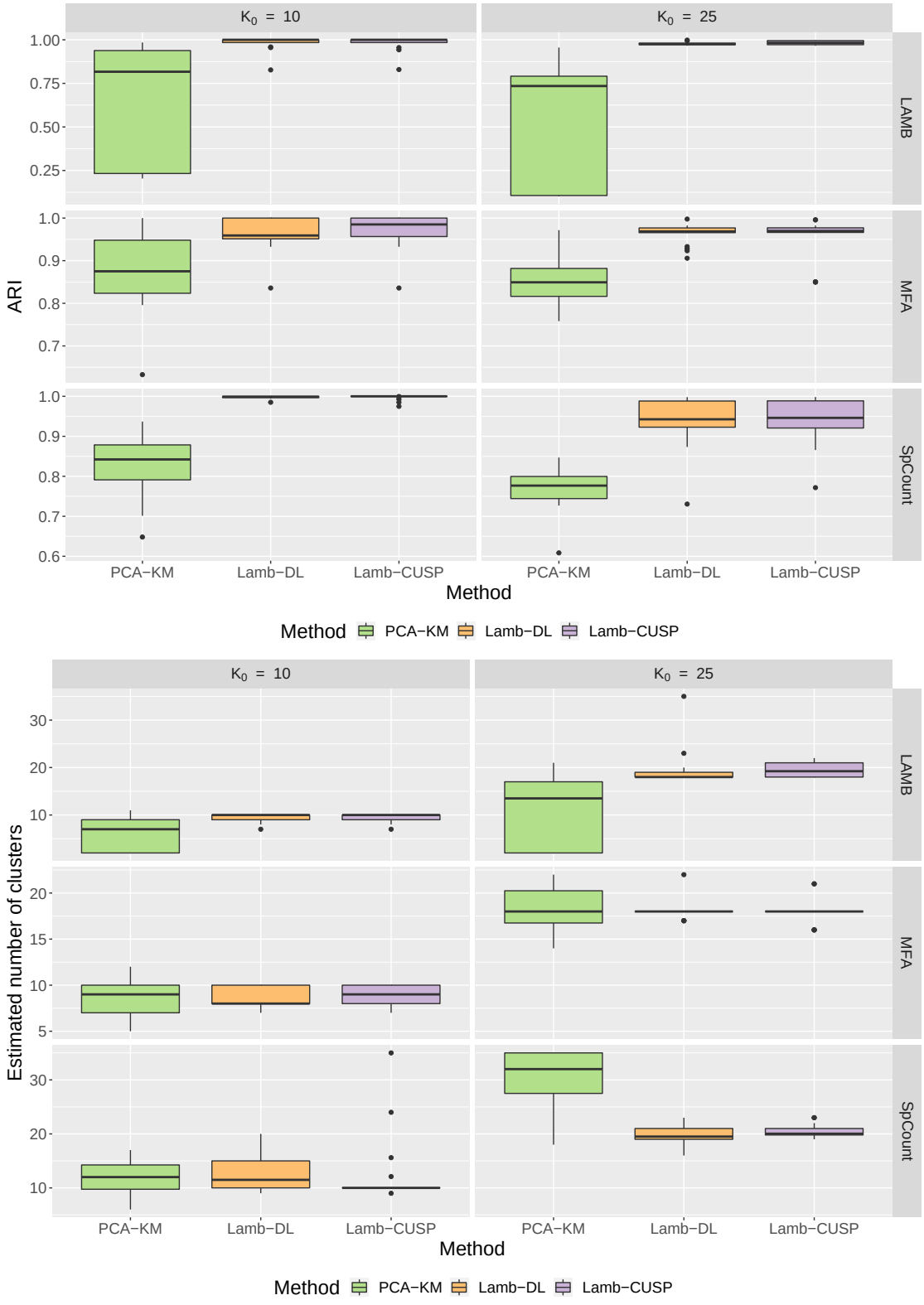

Figure 4.1: Comparison between our Lamb-CUSP, the Lamb-DL and the two-stage PCA-KM approach: distributions of the adjusted Rand indices (upper plot) and estimated number of clusters (lower plot) in 20 replicated experiments, for $p = 2500$, $n = 2000$ and $K_0 = (10, 25)$, and each row refers to the true data model generator between Lamb with $H = 20$, MFA and SpCount.

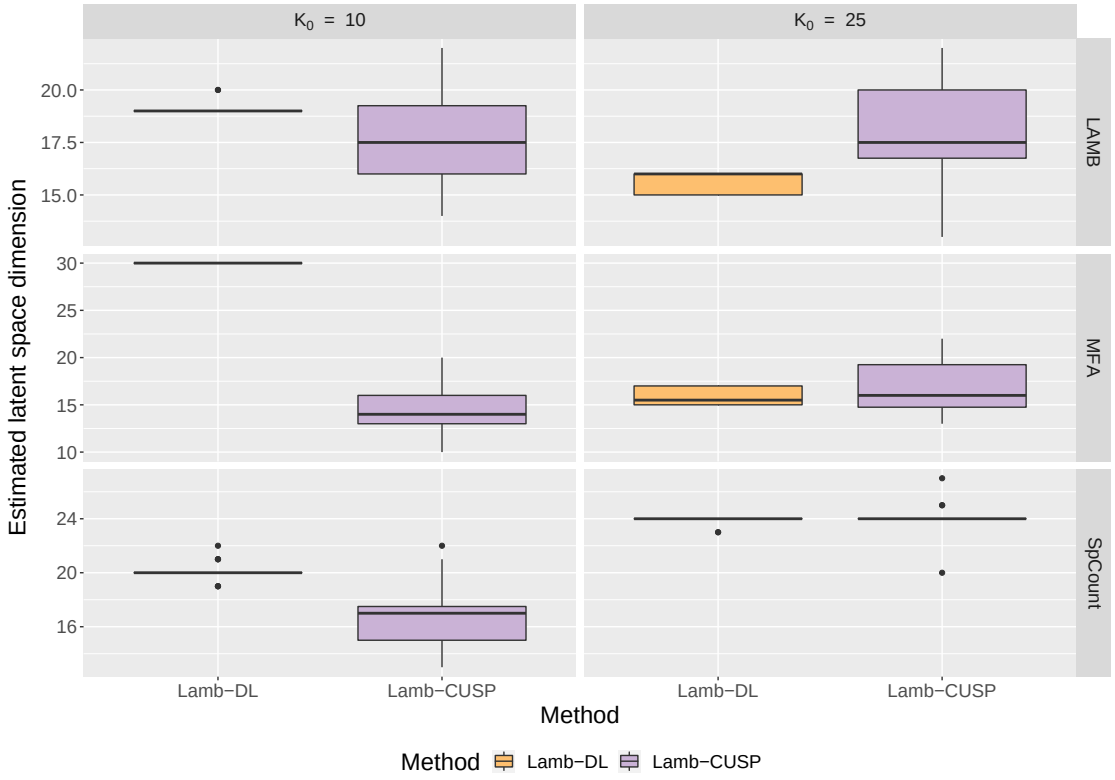

Figure 4.2: Comparison between our Lamb-CUSP and the Lamb-DL on the distributions of the estimation of dimension H of the latent space in 20 replicated experiments, for $p = 2500$, $n = 2000$ and $K_0 = (10, 25)$, and each row refers to the true data model generator between Lamb with $H = 20$, MFA and SpCount. H in Lamb-DL corresponds also to the number of principal components in the PCA-KM approach and for this reason, is not reported.

effects small clusters estimation. On the other hand, when ξ^2/ν is large, the prior distribution is weakly informative and the posterior distribution is more flexible to merge clusters, and therefore it yields a smaller number of clusters. Hence, ξ^2 and ν can be adjusted based on some prior information on the clustered data structure. If one expects to observe a few big clusters it may assume a weakly informative distribution. Otherwise, for smaller and more numerous clusters, a stronger prior may be more appropriate.

For what concerns the CUSP prior and the estimation of the latent space dimension, some luck of robustness emerged. While the parameter α, which controls the cumulative weights of spike and slab distribution, does not have any evident effect on the posterior dimension of the latent space, the setting of parameter θ_∞ appears more delicate. Indeed, even small perturbations can generate different estimations, in particular, it is easy to fall in one of the extreme cases for which, if θ_∞ is too small, selection of loadings columns of Λ does not occur, or if θ_∞ is too large, all columns are sampled from the spike. Moreover, this behaviour strongly depends on initialization, mainly from the initial values of the number of clusters and the matrices η and Λ. However, our recommended hyperparameters and initialization seem quite robust along different replications.

Simulations showed some significant mixing issues concerning both Lamb-DL and Lamb-CUSP samplers. In particular, if a unit is assigned to a medium-large cluster it is unlikely it will move to a different cluster. This is especially evident in the simulations with $K_0 = 25$ where the small clusters, once merged with other clusters, are no longer split apart. The mixing problem is evident also when the initial clusters allocation consists of many partitions. In this case, the sampler struggles to merge clusters and gets stuck in the initialization values. Low mixing issues are common in Bayesian nonparametric clustering, for this reason in Chandra et al. (2020) Lamb implementation proposed to adopt a split-merge MCMC (Jain and Neal, 2004) procedure which especially helps with major changes in clusters memberships, switching randomly between it and the Gibbs sampling. Although this procedure helps in some applications, we observed that in some cases the Lamb-DL Gibbs sampling implementation leads to poor performance since it forms incorrect clusters and then no longer separates them. A mixing criticism is also evident in the adaptation of the dimension of the latent space. As it was already clear from Legramanti et al. (2020), the relevant columns of Λ selected at the beginning of Gibbs sampler remain the same along all computation and the algorithm struggle to delete and generate new latent factors.

Chapter 5

Application to Single-Cell RNA-Seq data

5.1 Data description

In this section, Lamb-CUSP is tested in a real challenging application which is the clustering of Single-Cell RNA-Seq (scRNAseq) data. For researchers, having a data-driven tool to divide cells into groups is very important as it provides important guidance toward further analysis. In this way, they are able, for example, to identify cells that are expressed by unique genes or that exhibit interesting structures or nonconforming phenomena. One important application is, for example, tumour tissue analysis in which an accurate clustering technique makes it possible to identify and study new types of cells.

Single-Cell RNA-Seq data consist of the expression profile of single cells sampled from a specific tissue. In particular, for each cell, the expression of individual genes is represented by a count obtained by overlapping the cell RNA segments with the genome. As the number of genes is huge (> 20000) and the number of cells is usually small due to the elevated cost, we face a high dimensional scenario. Besides the most recent technologies allowing to have large sample size with $n \approx p$, the curse of dimensionality and computational issues still remain as described in section 2.1.

For our test, we analyzed the scRNAseq dataset provided by Consortium et al. (2018). In particular, it consists of a big collection of cell expression profiles from mice (7 male e 4 female) of 20 organs. The dataset is public and can be downloaded from Figshare. For our purpose, we focused only on the dataset regarding pancreas cells since it presents a challenging structure and an appropriate number of cells, number of expressed genes and number of cell types. Hence, the data

considered includes the expression profile for 23341 genes of 1564 cells divided into 9 clusters: pancreatic B cells (449), pancreatic A cells (390), pancreatic acinar cells (182), pancreatic ductal cells (161), pancreatic D cells (140), pancreatic PP cells (73), endothelial cells (66), leukocyte (54), pancreatic stellate cells (49). A graphical representation of the cells are shown in Figure 5.3 via UMAP projection McInnes et al. (2018). While the pancreatic B, pancreatic ductal and leukocyte cells clusters are well separated in UMAP projection, the same is not observed for the remaining groups. In particular, the figure shows a relatively big cluster not well separated composted by pancreatic A, pancreatic PP and pancreatic D cells and a partially overlapped group of clusters formed by endothelial, pancreatic acinar and pancreatic stellate cells.

Following standard procedure in scRNAseq data analysis, we applied data preprocessing using `Seurat` package. After checking cells with low count reads, we normalized gene expression by dividing by the total counts for that cell and multiplying by the scale factor of 10000. They are then transformed by the natural logarithm after adding 1. Genes with more than 50% of zero log counts are discarded. The final dataset consists in $n = 1564$ cells and $p = 2940$ gene expressions.

5.2 Results

Lamb-CUSP is compared with the PCA-KM approach described in Chapter 4, the Lamb-DL method and the graph-based method included in the popular Seurat pipeline (Butler et al., 2018) which represents the reference workflow in scRNAseq clustering analysis. Specifically, this algorithm identifies clusters of cells by a shared nearest neighbour (SNN) modularity optimization. For a full description of the algorithms, see Waltman and Van Eck (2013).

Lamb-DL is implemented with the prior and initialization as described in Chapter 4. However, in this application, the estimation of latent space dimension strongly depends on the number of approximated principal components estimated. For example, by computing 40 and 80 components we obtained an estimate for H of 37 and 73 respectively. The same behaviour is observed for higher values. Considering the computation cost, for Lamb-DL we set $H = 37$ after computing 40 approximated principal components.

For the Lamb-CUSP clustering, the prior hyper-parameters are set according to the simulation results under SpCount scenario, in particular we set $\theta_\infty = 0.0005$, and a random initialization is used for the matrices Λ and η, with $H = 5\log(p) = 39$.

For both Gibbs samplers, 5000 iterations of the MCMC chain are collected after a burn-in of 5000. For Lamb-CUSP, the adaptation of the latent space dimension starts after 500 iteration. The posterior clustering is computed by minimizing the VI loss as described in section 1.4.

In Figure 5.1 is displayed a draw from the posterior distribution of the matrix of latent factors η derived from Lamb-CUSP, after the burn in, reorder for cluster assignment. It clearly shows a clustered structure in which observations in the same cluster assume similar values in the latent space, thus proving that the Gibbs sampler, starting from a random initialization, can reconstruct the clustered structure of data in a reduced dimension space.

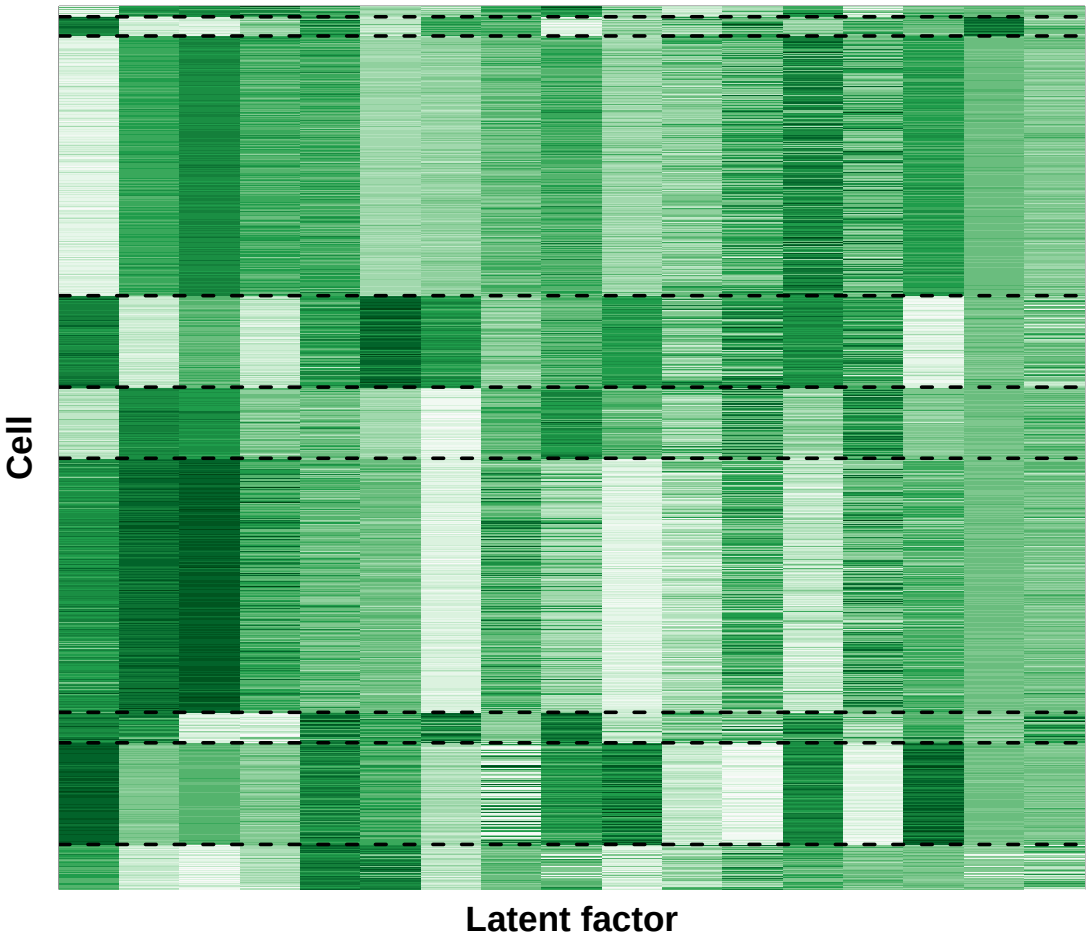

Figure 5.1: Draw from the posterior distribution of the latent factors $n \times H$ matrix η where columns represent latent factor and rows represent cells, reordered by cluster assignment.

The same insight can be concluded from Figure 5.2 where cell clusters can be partly identified from bivariate distributions of the H components of the latent space. More or less, all combinations of η components display a clustered structure.

This is particularly evident from the pairs plot of the 3rd and 7th component for which all clusters are well separated, except for stellate cells, which, however, are clearly distinguishable by looking at the 6th component.

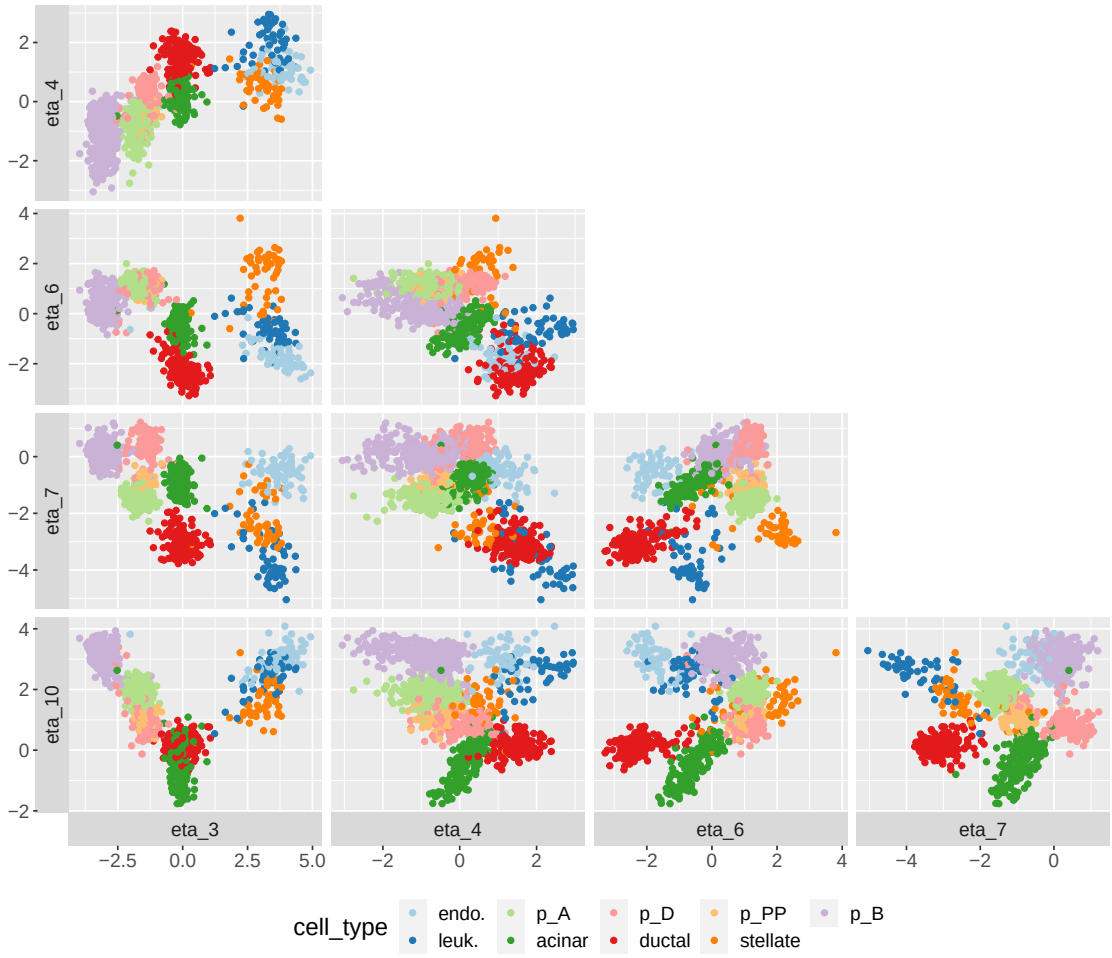

Figure 5.2: Pairs plot of components 3,4,6,7,10 of the latent factor matrix η drawn from the posterior distribution, and coloured by true cell types.

Graphical representations of the different clustering are shown in Figure 5.3 via UMAP projections. The PCA-KM approach partly failed clustering achieving an ARI of 0.726 since it yielded only 5 clusters and was unable to identify medium/small clusters even if in the UMAP projection they are well separated.

Lamb-CUSP reached a successful cells partition with an ARI of 0.879 and the dimension of latent space H converged to 17. On the contrary, Lamb-DL yielded to lower ARI of 0.766 since it merged type A, type D and type PP cells in a single cluster, and this is probably a result of the high dimension of the latent space that favours the joining of clusters. The Lamb-DL is also estimated considering a more conservative latent space dimension $H = 21$ which corresponds to the first approximated principal components that explain at least 80% of variability.

Under this setting, Lamb-DL reached a similar performance compared to Lamb-CUSP, thus highlighting how misspecification on the reduced space dimension leads to biased estimates and that the introduction of an adaptive estimation of H is important.

The Seurat approach achieved an ARI of 0.867. However, compared with Lamb-CUSP estimation, the two partitions differ significantly. Lamb-CUSP identified 9 clusters and, referring to UMAP projection, succeeded to split clusters that are not well separated and to allocate even those cells that are not properly close to their respective cluster. One example is about some leukocyte and ductal cells that are mixed in a confusing cluster of acinar, stellate and endotendothelial cells. However, the Lamb model is unable to identify pancreas PP cells, incorporating them into the larger cluster of type A cells, and split the pancreas stellate cells into two minor clusters. Seurat approach, in contrast, yields 6 clusters and identifies more precisely the clusters of pancreas D and pancreas B cells, however it merges ductal and stellar cells in a single cluster, as for leukocyte and acinar cells, even if in UMAP projection are well separated. It is worth mentioning that while the number of clusters is not be specified before fitting accordingly with the Seurat pipeline, in practice the reduction parameter of SNN algorithm has an important behaviour on the estimation of the number of clusters. In particular, for a value of 0.1, we obtained the partition described above, while for a value of 0.2, the model led to an unsatisfactory result that identifies 10 clusters and splits the largest groups of cells into two.

One advantage of Lamb clustering as model-based clustering is to perform post-clustering inference. As shown by Figure 5.4, most of the cells have a high posterior probability of being assigned to a specific cluster. However, for some cells, the uncertainty regarding the cluster assignment is relevant. An example is a pancreas leukocyte cell that in UMAP projection is among the acinar cells cluster which our model assigns a probability of 0.55 in the appropriate acinar cells cluster and a probability of 0.45 to be in the leukocyte cells cluster.

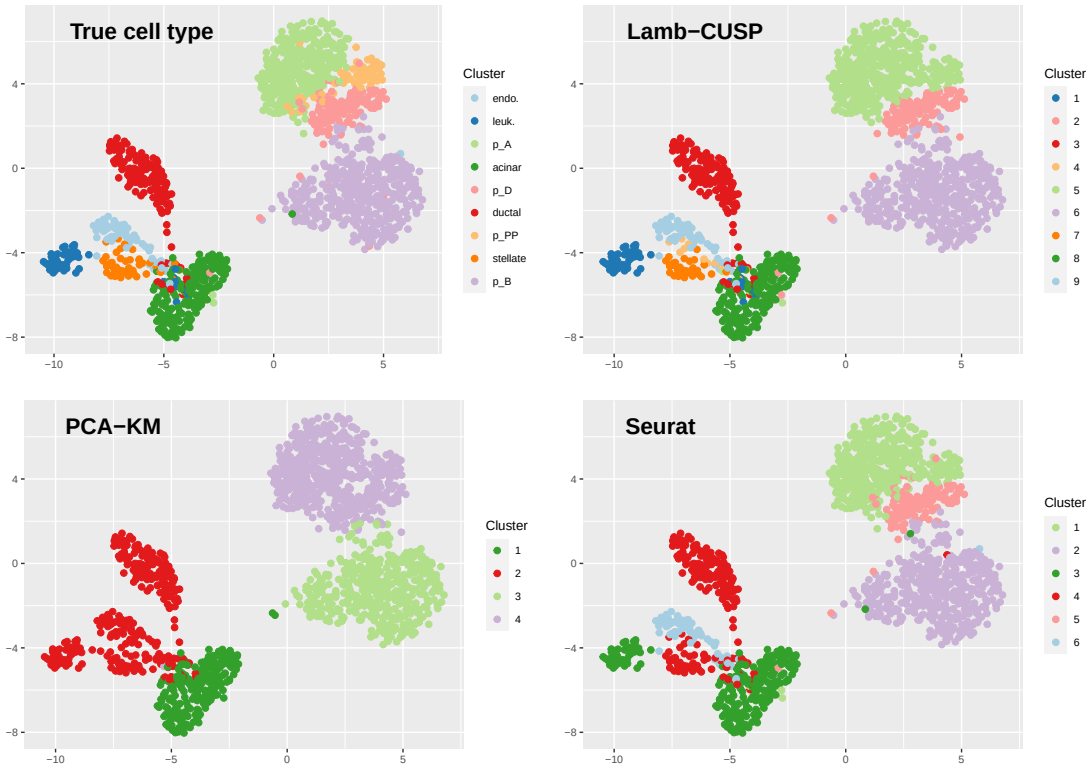

Figure 5.3: UMAP plots of the cell line dataset: Clusterings corresponding to the true cell types, Lamb estimate, PCA-KM estimate and Seurat estimate are plotted in a clockwise manner.

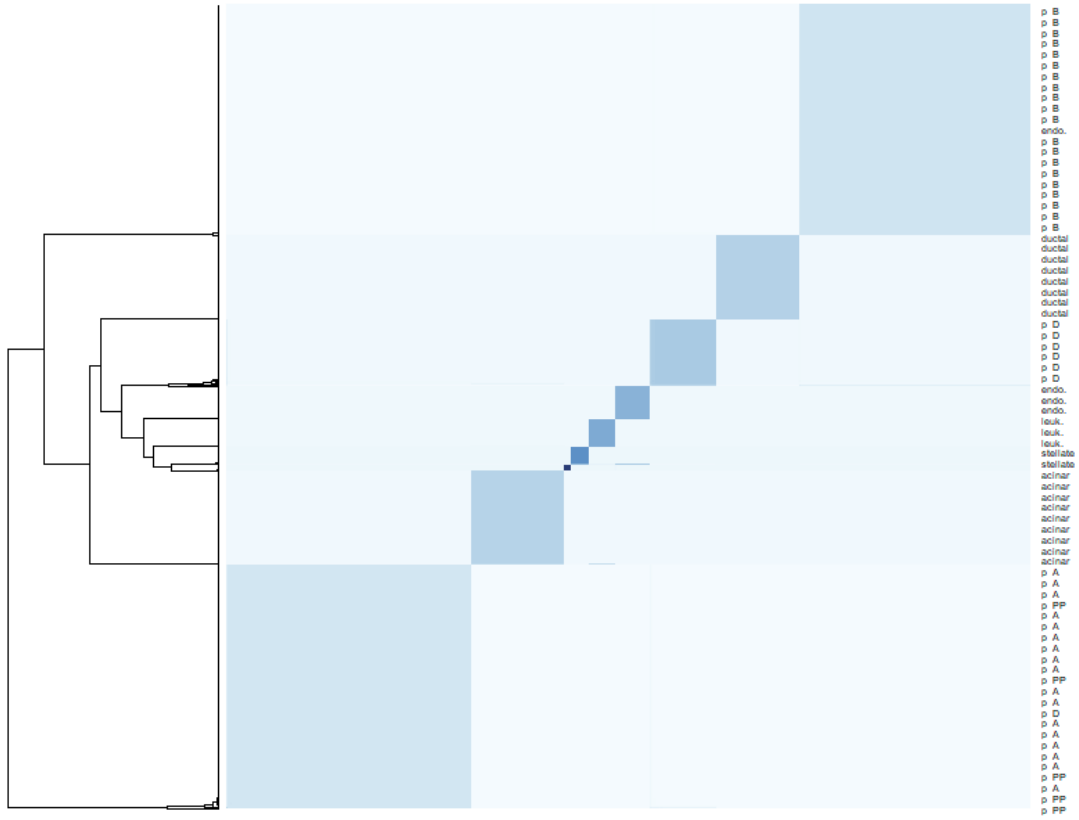

Figure 5.4: Posterior similarity matrix obtained from the MCMC samples of the Lamb-CUSP clustering

Conclusion

In this work, an alternative formulation of Lamb clustering based on an infinite factorization is proposed. Lamb clustering overcomes the curse of dimensionality in high-dimensional applications by defining a factorization of sample space and performing clustering in a reduced dimension space of latent factors. In contrast with the standard formulation of Lamb clustering (Lamb-DL) which estimates the dimension of the latent space empirically before posterior computation and assumes a general shrinkage prior distribution for the loading matrix, the model proposed (Lamb-CUSP) infers the dimension of the latent space along with the posterior computation by assuming a loading matrix specific shrinkage prior distribution, which, in the spirit of DP process, allows an infinite number of latent factors. In particular, we chose as prior distribution the Cumulative Shrinkage Process which shrinkage is based on a spike and slab framework that assigns more probability mass to the spike as the columns index of the loading matrix increases.

Simulations study showed comparable performance between the two Lamb implementations in terms of ARI and number of clusters estimation, and both tend to underestimate the true dimension of the latent space. However, the Lamb-CUSP seems to be more flexible between different replications. A notable gain of Lamb-CUSP on Lamb-DL is not evident probably because the estimation of the dimension of the latent space by PCA result is quite effective due to the well-defined structure present in the simulated data.

On the other hand, some evident differences appeared when the methods are applied to a real application. In the context of scRNAseq data considered, the estimated dimension of the latent space in Lamb-DL appeared less accurate and it led to an over-parameterized model which partly failed clustering. Lamb-CUSP, instead, thanks to the latent factor selection, yielded a satisfactory clustering that is comparable and in some terms superior to the scRNAseq data-specific clustering of the Seurat workflow.

However, some criticism appeared. These concern the setting of the hyper-parameter of the CUSP prior which strongly depends on initialization and data scale, and small perturbations may lead to undesirable posterior behavior. For this

reason, the implementation of an accurate initialization procedure seems crucial to guarantee convergence to the posterior distribution. Moreover, the estimation of the dimension of the latent space along with the Gibbs sampling algorithm is characterized by low mixing. One possibility to improve this and consequently the posterior distribution computation is to follow the approach described in Section 3.3 of Kowal and Canale (2021), which introduces redundant parameter expansion that multiples each loading factor for a quantity randomly centered in 1 or -1.

Other more general improvements or extensions of Lamb methods are proposed in Chandra et al. (2020), and they include exploiting parallel and distributed computing for bigger sample sizes, handling more complex data structures such as not real-valued data, and allowing for kernel misspecification.

Appendix A

Computation of full conditional distributions

The calculation to derive the full conditional distributions of the Gibbs sampling algorithm in Section 2.5 for Steps 2 and 3 are reported below. The remaining full conditional distributions are easily derived by applying the properties of conjugate distributions.

The standard technique to derive the full conditional distribution is to isolate from the posterior distribution only the terms that depend on the parameter of interest and recognize the kernel of known distribution.

Step 2. Let $k = c_i$, the full conditional distribution of the latent factors η_is can be computed as

$$
\begin{aligned}
p(\eta_i|-) &\propto \int_{-\infty}^{\infty} p(y_i|\Lambda\eta_i, \Sigma)p(\eta_i; \mu_k, \Delta_k) \prod_{j:c_j=k, j\neq i} p(\eta_j; \mu_k, \Delta_k)p(\mu_k)\mathrm{d}\mu_k \\
&\propto \int_{-\infty}^{\infty} p(\eta_i|y_i, -)p(\mu_k|\eta_{k,-i})\mathrm{d}\mu_k \\
&\propto \int_{-\infty}^{\infty} N_p(y_i; \Lambda\eta_i, \Sigma)N_H(\eta_i; \mu_k, \Delta_k) \prod_{j:c_j=k, j\neq i} N_H(\eta_j; \mu_k, \Delta_k)N_H(\mu_k; 0, \Delta_k/\kappa_0)\mathrm{d}\mu_k \\
&\propto \int_{-\infty}^{\infty} N_H(\eta_i; \Omega_k^{-1}\rho_k, \Omega_k^{-1})N_H(\mu_k; \hat{\mu}_{k,-i}, \Delta_{k,-i}/\hat{\kappa}_{k,-i})\mathrm{d}\mu_k \\
&\propto \int_{-\infty}^{\infty} N_H(\eta_i; \Omega_k^{-1}\rho_k, \Omega_k^{-1})N_H(\Omega_k^{-1}\rho_h; \Omega_k^{-1}\hat{\rho}_k, \Omega_k^{-1}(\Delta_{k,-i}\hat{\kappa}_{k,-i})^{-1}\Omega_k^{-1})\mathrm{d}\mu_k,
\end{aligned}
$$

where $\rho_k = \Lambda^T \Sigma^{-1} Y_i + \Delta_k^{-1} \mu_k$ and the last step derives by noting

$$
\begin{aligned}
N_H(\Omega_k^{-1}\rho_h; \Omega_k^{-1}\hat{\rho}_k, \Omega_k^{-1}(\Delta_{k,-i}\hat{\kappa}_{k,-i})^{-1}\Omega_k^{-1}) \propto \\
&\propto (\Omega_k^{-1}\rho_h - \Omega_k^{-1}\hat{\rho}_k)^T (\Omega_k^{-1}(\Delta_{k,-i}\hat{\kappa}_{k,-i})^{-1}\Omega_k^{-1})^{-1} (\Omega_k^{-1}\rho_h - \Omega_k^{-1}\hat{\rho}_k) = \\
&= (\mu_k - \hat{\mu}_k)^T (\Delta_{k,-i}^{-1}\Omega_k^{-1}\Omega_k(\Delta_{k,-i}\hat{\kappa}_{k,-i})\Omega_k\Omega_k^{-1}\Delta_{k,-i}^{-1})(\mu_k - \hat{\mu}_k)^T = \\
&= (\mu_k - \hat{\mu}_k)^T (\Delta_{k,-i}/\hat{\kappa}_{k,-i})^{-1} (\mu_k - \hat{\mu}_k) \propto \\
&\propto N_H(\mu_k; \hat{\mu}_{k,-i}, \Delta_{k,-i}/\hat{\kappa}_{k,-i}).
\end{aligned}
$$

Exploiting the conjugacy properties of multivariate Gaussian distribution, that is the last integral can be interpreted as a posterior predictive distribution, we obtain the full conditional distribution in Step 2.

Step 3. The full conditional distribution of the loadings matrix Λ is given by

$$
p(\lambda_j|-) \propto p(\lambda_j|\theta) p(y^{(j)}|\Lambda, \eta, \Sigma) = N_H(\lambda_j; 0, D) \prod_{i=1}^{n} N(y_{ij}; \lambda_j \eta_i, \sigma_j^2).
$$

Passing to the log-probability we have that

$$
\begin{aligned}
\log p(\lambda_j|-) = \\
&= -\frac{1}{2}\lambda_j^T D^{-1} \lambda_j - \frac{1}{2}\sigma_j^{-2}(y^{(j)} - \eta\lambda_j)^T (y^{(j)} - \eta\lambda_j) + const = \\
&= -\frac{1}{2}\lambda_j^T D^{-1} \lambda_j - \frac{1}{2}\sigma_j^{-2}\lambda_j^T \eta^T \eta \lambda_j + \sigma_j^{-2}\lambda_j^T \eta^T y^{(j)} + const \\
&= -\frac{1}{2}\lambda_j^T (D^{-1} + \sigma_j^{-2}\eta^T\eta)\lambda_j + \sigma_j^{-2}\lambda_j^T \eta^T y^{(j)} + const = \\
&= -\frac{1}{2}(\lambda_j - W^{-1}\sigma_j^{-2}\eta^T y^{(j)})^T W (\lambda_j - W^{-1}\sigma_j^{-2}\eta^T y^{(j)}) + const,
\end{aligned}
$$

which is the kernel of multivariate Gaussian distribution with mean $W^{-1}\sigma_j^{-2}\eta^T y^{(j)}$ end variance W^{-1}, with $W = (D^{-1} - \sigma_j^{-2}\eta^T\eta)$.

Appendix B

Plots of simulated dataset

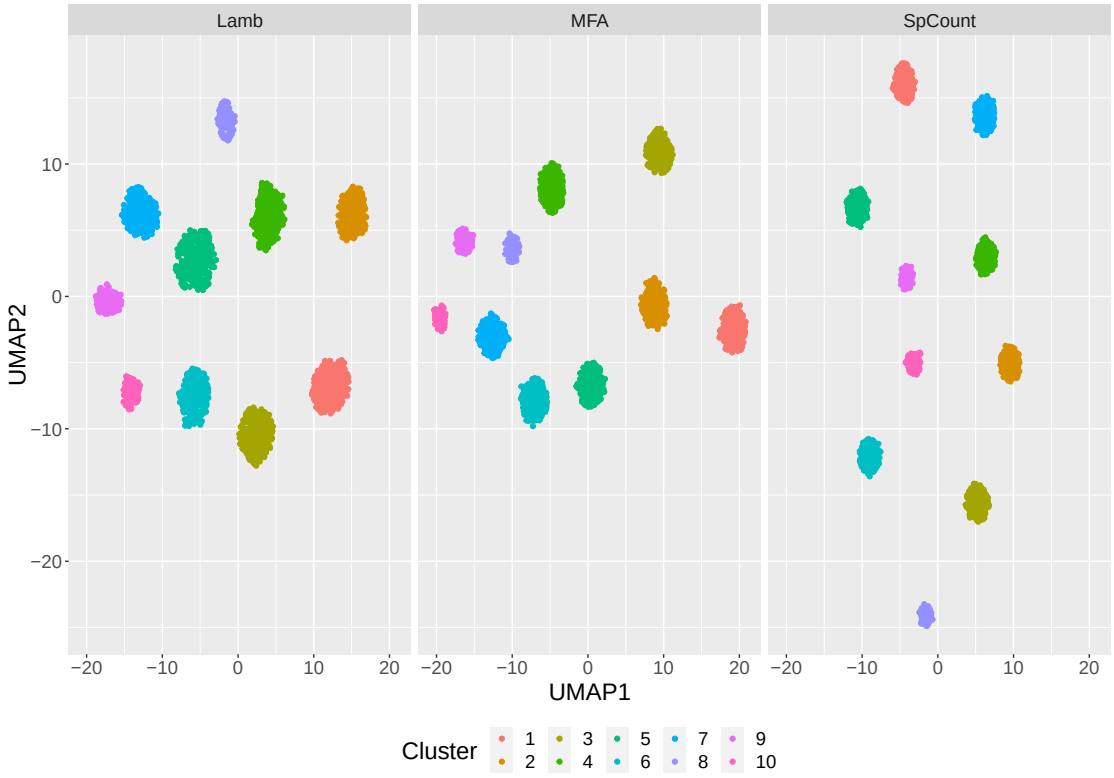

Figure B.1: UMAP plots of simulated dataset under the three different model considered (Lamb, MFA, SpCount) with number of clusters $K_0 = 10$, and $p = 2500$, $n = 2000$, $H = 20$.

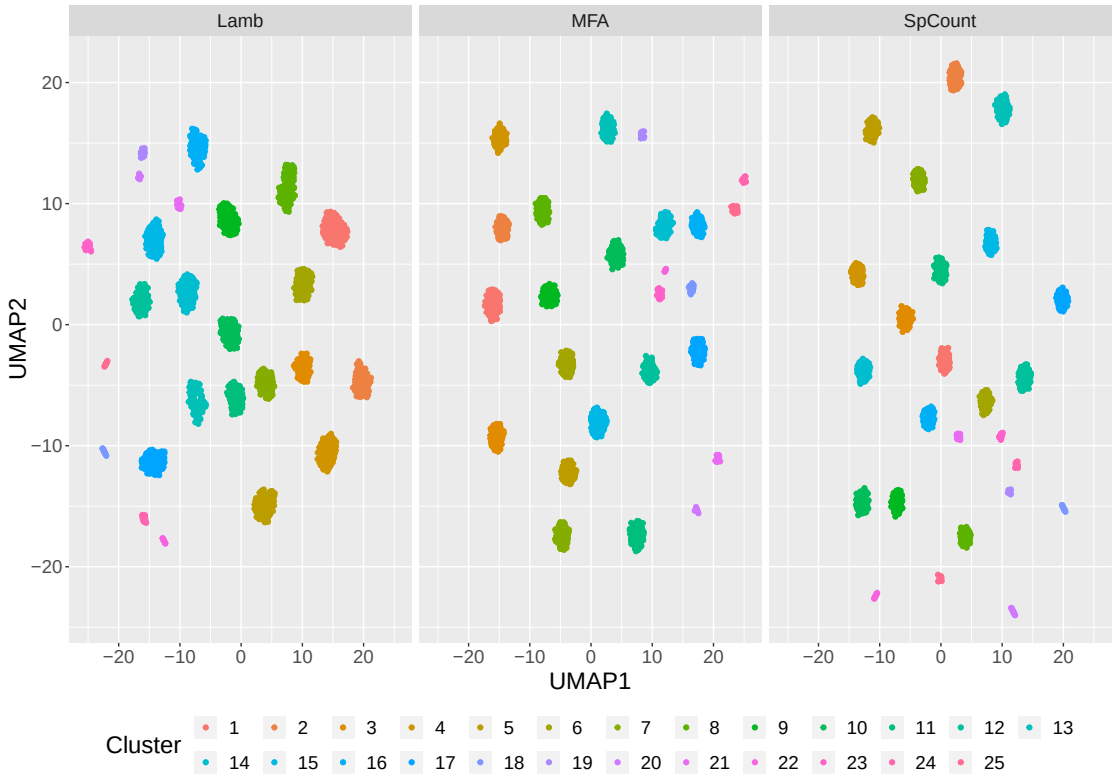

Figure B.2: UMAP plots of simulated dataset under the three different model considered (Lamb, MFA, SpCount) with number of clusters $K_0 = 25$, and $p = 2500$, $n = 2000$, $H = 20$.

Appendix C

Cpp code

The `Cpp` code of the implementation of Gibbs sampling algorithm proposed in Section 2.5 is reported below.

```
#include <RcppArmadillo.h>
#include <RcppGSL.h>
#include <mvt.h>
#include <sstream>
#include <iostream>
#include <fstream>
#include <iomanip>
#include <omp.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_sf_gamma.h>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>
#include <progress.hpp>
#include <progress_bar.hpp>

// [[Rcpp::depends(RcppArmadillo, RcppDist)]]
// [[Rcpp::depends(RcppArmadillo)]]
// [[Rcpp::depends(RcppGSL)]]
// [[Rcpp::plugins(cpp11)]]
// [[Rcpp::plugins(openmp)]]
// [[Rcpp::depends(RcppProgress)]]

using namespace arma;
using namespace std;

static double const log2pi = std::log(2.0 * M_PI);

//utility function ============================================================

// multivariate t log density
inline double log_t_density(const double nu,
                            const vec &x,
                            const vec &mu,
                            const mat &lower_chol){
  int k=x.n_elem;
  double det_sig_half= sum(log(lower_chol.diag() ) );
  vec resid = solve(trimatl ( lower_chol ) , x - mu );
```

```
  double quad_form=dot(resid,resid)/nu;

  double density =gsl_sf_lnpoch(nu/2 , ((double)k)/2)-
    (k* log( (datum::pi)*nu) + (nu+k)*log1p(quad_form) )/2 -det_sig_half;

  return density;
}

// multivariate t log density  with independent components
inline double log_t_density_diag(const double nu,
                                 const double sigma,
                                 const vec &x){
  int k=x.n_elem;
  double density = k * (lgamma((nu+1)/2 ) - lgamma(nu/2)) -
    k * log((datum::pi)*nu)/2 - k * log(sigma)/2-
    (nu+1)/2 * sum(log(1 + ( x % x ) / (nu+sigma)));
  return density;
}

// rescale log probability to avoid overflow issues
double log_sum_exp(const arma::vec &x) {
  double maxVal= x.max();
  double sum_exp=sum(exp(x-maxVal));
  return log(sum_exp)+maxVal ;
}

// delete void cluster e reorder labels
void clean_clusters(uvec &c, uword &c_old, cube &Delta, uvec &n_k,
                    mat &m_eta_temp, cube &V_eta_temp,
                    cube &psi_eta_k, cube &chol_psi_eta_k){

  uword K = Delta.n_slices;
  uword H = Delta.n_cols;
  int cut = 0;
  for (uword i=0; i<K; i++){

    if (accu(c == i) == 0){

      for (uword j=(K-1); j>i; j--){

        if (accu(c == j) > 0){

          c(find(c == j)).fill(i);

          if (c_old == j){
            c_old = i;
          }

          Delta.slice(i) = Delta.slice(j);
          n_k(i) = n_k(j);
          m_eta_temp.col(i) = m_eta_temp.col(j);
          V_eta_temp.slice(i) = V_eta_temp.slice(j);
          psi_eta_k.slice(i) = psi_eta_k.slice(j);
          chol_psi_eta_k.slice(i) = chol_psi_eta_k.slice(j);

          break;
```

```
        }
      }
    }
  }

  while( sum(c == cut) > 0 ){
    cut++;
  }
  n_k.resize(cut);
  Delta.resize(H, H, cut);
  m_eta_temp.resize(H, cut);
  V_eta_temp.resize(H, H, cut);
  psi_eta_k.resize(H, H, cut);
  chol_psi_eta_k.resize(H, H, cut);
}

// update alpha of DP
double update_alpha(double alpha,
                    double a,
                    double b,
                    unsigned n,
                    unsigned k){
  double phi=R::rbeta(alpha+1,(double) n);
  double gam1=a+k, gam2=b- log(phi);
  double pi=(gam1-1)/(n* gam2 );
  return (log(randu()) < log(pi)-log1p(pi)  ) ?
  (randg(  distr_param(gam1,1/gam2) )) : (randg(  distr_param(gam1-1,1/gam2) ));
}

// ============================================================================
//' @export
//' @name Lamb_CUSP
//' @title C++ function to estimate the Lamb-CUSP
//'  clustering method via Gibbs sampling
//' @keywords internal
//'
//' @param y a matrix of observations
//' @param n_iter nuumber of iteration
//' @param n_burn number of burn-in iterations
//' @param start_adapt iteration to start adapting latent space dimension
//' @param prior_par list of prior parameters:
//'   * xi variance of Delta
//'   * nu_0 df of Delta
//'   * kappa_0 precision of mu
//'   * a_sigma shape of sigma
//'   * b_sigma rate of sigma
//'   * a_dir shape of alpha DP for clustering
//'   * b_dir rate of alpha DP for clustering
//    * a_theta shape of theta CUSP
//'   * b_theta rate of theta CUSP
//'   * alpha_f precision od DP for CUSP
//'   * theta_lim variance of spike CUSP
//' @param start list of initial values of parameters:
//'   * H dimension of latent space
//'   * K number of clusters
//'   * Lambda matrix of loadings
//'   * eta matrix of latent factors
```

```
//'   * sigma vector of diagonal variances of data
//'   * c vector of cluster menbership [0, K-1]
//'
//' @return list:
//'   * H latent space dimension
//'   * n_cluster number of clusters
//'   * c clusters allocation
//

// [[Rcpp::export]]
Rcpp::List Lamb_CUSP(mat y, uword n_iter, uword n_burn, uword start_adapt,
                     Rcpp::List prior_par, Rcpp::List start){

  Progress progress(n_iter, true);
  // prior parameters
  double a_sigma   = prior_par["a_sigma"];
  double b_sigma   = prior_par["b_sigma"];
  double a_theta   = prior_par["a_theta"];
  double b_theta   = prior_par["b_theta"];
  double xi        = prior_par["xi"];
  double kappa_0   = prior_par["kappa_0"];
  uword  nu_0      = prior_par["nu_0"];
  double a_dir     = prior_par["a_dir"];
  double b_dir     = prior_par["b_dir"];
  double alpha_f   = prior_par["alpha_f"];
  double theta_lim = prior_par["theta_lim"];

  // values for the adaptation of latent space dimension
  double alpha_0 = -1;
  double alpha_1 = -0.0005;

  uword p = y.n_cols; // number of features
  uword n = y.n_rows; // number of observation
  uword H = start["H"]; // latent space dimension
  uword K = start["K"]; // number of cluster

  // initialization ============================================================
  mat Lambda = start["Lambda"];// p x H
  mat eta = start["eta"];      // n x H
  vec sigma = start["sigma"];  // p x 1
  vec theta = start["theta"];  // H x 1
  uvec c = start["c"];
  cube Delta(H, H, K);
  uvec z(H);
  vec v(H);
  vec w(H, fill::value(1));
  w = w/H;
  double alpha_cl = 1;
  uword H_star=H;
  uvec active(H);

  // temp variables=============================================================

  //// temp for Lambda
  mat V_Lambda_temp(H, H);
  mat eta2(H, H);
```

```
//// temp for eta
mat L_sigma_L_prod(H, H);
mat L_sigma_prod(p, H);
cube chol_Omega_temp(H, H, K);
cube chol_Delta(H, H, K);
vec rho_k(H);
mat m_eta_temp(H, K);
cube V_eta_temp(H,H, K);
cube psi_eta_k(H,H, K);
cube chol_psi_eta_k(H,H, K);

//// temp for c
uvec n_k(K);
uword n_k_wo_i;
uvec idx_k(1), idx_k_wo_i(1);
vec m_eta_wo_i(H);
mat V_eta_wo_i(H, H);
mat psi_eta_wo_i(H, H);

//// temp for sigma
double a_sigma_temp;
double b_sigma_temp;
double m1_temp;
double m2_temp;

//// temp for CUSP
double n1_v;
double n2_v;
double Lambda_sum_squared=0;

//// other temp variables
mat Lambda_temp(p, H);
mat eta_temp(n, H);
vec theta_temp(H);
vec w_temp(H);
vec v_temp(H);
cube Delta_temp(H, H, K);
bool req_clean = false;
double flag = 0;

// output ========================================================================
Rcpp::List out;
vec H_out(n_iter-n_burn);
umat c_out(n, n_iter-n_burn);
uvec n_cluster( n_iter-n_burn);

// initial comoutation of temp variable ==========================================
for (uword k=0; k<K; k++){
  n_k(k) = accu(c == k);
  idx_k.set_size(n_k(k));
  idx_k = find(c == k);

  m_eta_temp.col(k) = sum(eta.rows(idx_k),0).as_col();
  V_eta_temp.slice(k) = trans(eta.rows(idx_k)) * eta.rows(idx_k);

  psi_eta_k.slice(k) = symmatu(V_eta_temp.slice(k) -
    m_eta_temp.col(k) * trans(m_eta_temp.col(k)) / (kappa_0 + n_k(k)));
```

```
  psi_eta_k.slice(k).diag() += xi;

}

// START GIBBS ================================================================
for(uword it=0; it<n_iter; it++){

  // update Delta=============================================================

  for (uword k=0; k<K; k++){

    Delta.slice(k) = iwishrnd(psi_eta_k.slice(k), nu_0 + n_k(k));

    chol( chol_Delta.slice(k), Delta.slice(k) );
  }

  if (Delta.has_nan()){
    Rcpp::stop("Delta has nan");
  }

  // update Lambda ===========================================================

  eta2 = eta.t() * eta;
  for (uword j=0; j<p; j++){
    V_Lambda_temp = chol(diagmat(1/theta) + 1/sigma(j)*eta2);
    // back forward substitution
    Lambda.row(j) = (solve(trimatu(V_Lambda_temp ),
                     randn<vec>(H) +solve(trimatl((V_Lambda_temp ).t()),
                                1/sigma(j) * (eta.t() * y.col(j)) ) ) ).t();
  }

  if (Lambda.has_nan()){
    Rcpp::stop("Lambda has nan");
  }

  // update eta ==============================================================
  L_sigma_prod = Lambda.each_col() % (1/sigma);
  L_sigma_L_prod =  symmatu(L_sigma_prod.t() * Lambda);

  //// compute Omega
  for (uword k=0; k<K; k++){
    chol_Omega_temp.slice(k) = (chol(L_sigma_L_prod +
      inv_sympd(Delta.slice(k))));
  }

  for (uword i=0; i<n; i++){

    rho_k = L_sigma_prod.t() * y.row(i).t();
    m_eta_temp.col(c(i)) = m_eta_temp.col(c(i)) - eta.row(i).as_col();

    rho_k += inv_sympd(Delta.slice(c(i))) *
      m_eta_temp.col(c(i)) / (n_k(c(i))-1+kappa_0);
    rho_k += solve(trimatu(chol_Delta.slice(c(i))),
                   randn<vec>(H)) / sqrt(n_k(c(i))-1+kappa_0);

    eta.row(i) = solve(trimatu(chol_Omega_temp.slice(c(i))),
            randn<vec>(H) +
```

```
            solve(trimatl((chol_Omega_temp.slice(c(i))).t()), rho_k)).t();
  m_eta_temp.col(c(i)) = m_eta_temp.col(c(i)) + eta.row(i).as_col();
}

if (eta.has_nan()){
  Rcpp::stop("eta has nan");
}

// update of temp variables ================================================
for (uword k=0; k<K; k++){

  idx_k.set_size(n_k(k));
  idx_k = find(c == k);

  V_eta_temp.slice(k) = trans(eta.rows(idx_k)) * eta.rows(idx_k);

  psi_eta_k.slice(k) = symmatu(V_eta_temp.slice(k) -
    m_eta_temp.col(k) * trans(m_eta_temp.col(k)) / (kappa_0 + n_k(k)));
  psi_eta_k.slice(k).diag() += xi;
  chol_psi_eta_k.slice(k) = chol(psi_eta_k.slice(k), "lower");
}

// update clutsers ==========================================================
for (uword i=0; i<n; i++){

  uword c_old = c(i);
  req_clean = false;

  // if i is a singleton, I have to remove its cluster
  if (accu(c == c(i)) == 1){
    req_clean = true;
    c_old = n;
  }

  // I assign i to a non-existing cluster for convenience of code writing
  c(i) = n;

  if (req_clean == true){
    clean_clusters(c, c_old, Delta, n_k, m_eta_temp, V_eta_temp,
                   psi_eta_k, chol_psi_eta_k);
    K = Delta.n_slices;
  }

  Rcpp::NumericVector log_prob_cluster(K+1);
  Rcpp::NumericVector prob_cluster(K+1);

  for (uword k=0; k<K; k++){

    if (c_old == k){ // for i old cluster
      n_k_wo_i = n_k(k)-1;
      m_eta_wo_i = m_eta_temp.col(k) - eta.row(i).as_col();
      V_eta_wo_i = V_eta_temp.slice(k) - trans(eta.row(i)) * eta.row(i);

      psi_eta_wo_i = symmatu(V_eta_wo_i - m_eta_wo_i * m_eta_wo_i.t() /
        (kappa_0 + n_k_wo_i));
      psi_eta_wo_i.diag() += xi;
```

```
        log_prob_cluster(k) = log(n_k_wo_i) +
          log_t_density(nu_0 + n_k_wo_i - H + 1 ,
                        (eta.row(i)).t(),
                         m_eta_wo_i / (kappa_0 + n_k_wo_i),
                         chol((kappa_0 + n_k_wo_i + 1) /
                           ((kappa_0 + n_k_wo_i) * (nu_0 + n_k_wo_i - H + 1))
                                *psi_eta_wo_i, "lower"));
        prob_cluster(k) = exp(log_prob_cluster(k));
      } else{ // for all other clusters

        log_prob_cluster(k) = log(n_k(k)) +
          log_t_density(nu_0 + n_k(k) - H + 1 ,
                        (eta.row(i)).t(),
                         m_eta_temp.col(k) / (kappa_0 + n_k(k)),
                         sqrt((kappa_0 + n_k(k) + 1) /
                           ((kappa_0 + n_k(k)) * (nu_0 + n_k(k) - H + 1))) *
                           chol_psi_eta_k.slice(k));
        prob_cluster(k) = exp(log_prob_cluster(k));
      }
    }

    // prob for a new cluster
    log_prob_cluster(K) = log(alpha_cl) +
      log_t_density_diag((double) (nu_0 - H + 1),
                         (kappa_0+1)/(kappa_0*(nu_0-H+1))*xi,
                          (eta.row(i)).t());
    prob_cluster(K) = exp(log_prob_cluster(K));

    Rcpp::IntegerVector c_idx = Rcpp::seq(0, K);
    c(i) = Rcpp::sample(c_idx, 1, false, prob_cluster)(0);

    // I need to update the utility variables
    // if i is assigned to a different cluster
    if (c(i) != c_old){
      // if i was not a sigleton
      if (c_old != n){
        // I need to remove its contribution from the old cluster
        n_k(c_old)--;
        m_eta_temp.col(c_old) = m_eta_temp.col(c_old) - eta.row(i).as_col();
        V_eta_temp.slice(c_old) = V_eta_temp.slice(c_old) -
          trans(eta.row(i)) * eta.row(i);

        psi_eta_k.slice(c_old) = symmatu(V_eta_temp.slice(c_old) -
          m_eta_temp.col(c_old) * m_eta_temp.col(c_old).t() /
          (kappa_0 + n_k(c_old)));
        psi_eta_k.slice(c_old).diag() += xi;
        chol_psi_eta_k.slice(c_old) = chol(psi_eta_k.slice(c_old), "lower");
      }

      if (c(i) == K){
        // i is assigned to a completely new cluster
        K = K+1;
        Delta.resize(H, H, K);
        Delta.slice(K-1) = iwishrnd(xi * eye(H,H), nu_0);
        n_k.resize(K);
        n_k(K-1) = 1;
        m_eta_temp.resize(H, K);
```

```
        m_eta_temp.col(K-1) = eta.row(i).as_col();
        V_eta_temp.resize(H, H, K);
        V_eta_temp.slice(K-1) = trans(eta.row(i)) * eta.row(i);
        psi_eta_k.resize(H, H, K);
        psi_eta_k.slice(K-1) = symmatu(V_eta_temp.slice(K-1) -
          m_eta_temp.col(K-1) * m_eta_temp.col(K-1).t() /
          (kappa_0 + n_k(K-1)));
        psi_eta_k.slice(K-1).diag() += xi;
        chol_psi_eta_k.resize(H, H, K);
        chol_psi_eta_k.slice(K-1) = chol(psi_eta_k.slice(K-1), "lower");

      } else {
        // i is assigned to a new cluster among those that already exist
        n_k(c(i))++;

        m_eta_temp.col(c(i)) = m_eta_temp.col(c(i)) + eta.row(i).as_col();
        V_eta_temp.slice(c(i)) = V_eta_temp.slice(c(i)) +
          trans(eta.row(i)) * eta.row(i);

        psi_eta_k.slice(c(i)) = symmatu(V_eta_temp.slice(c(i)) -
          m_eta_temp.col(c(i)) * m_eta_temp.col(c(i)).t() /
          (kappa_0 + n_k(c(i))));
        psi_eta_k.slice(c(i)).diag() += xi;
        chol_psi_eta_k.slice(c(i)) = chol(psi_eta_k.slice(c(i)), "lower");
      }
    }
  }

  chol_Omega_temp.set_size(H, H, K);
  chol_Delta.set_size(H, H, K);

  // update z ====================================================================
  for (uword h=0; h<H; h++){
    Rcpp::NumericVector log_prob_z(H);
    Rcpp::NumericVector prob_z(H);
    for (uword l=0; l<H; l++){
      if (l<=h){
        log_prob_z(l) = log(w(l)) +
          as_scalar(sum(log_normpdf(Lambda.col(h), 0, sqrt(theta_lim))));
      } else{
        log_prob_z(l) = log(w(l)) +
          log_t_density_diag( 2*a_theta, b_theta/a_theta, Lambda.col(h));
      }
    }

    // to avoid overflow issues
    prob_z = exp(log_prob_z - log_sum_exp(log_prob_z));

    if (sum(prob_z) == 0){
      cout << "WARNING: all probabilities for z equal to 0..." << endl;
      prob_z.fill(0);
      prob_z(H-1) = 1;
    }

    Rcpp::IntegerVector h_index = Rcpp::seq(0, H-1);
    z(h) = Rcpp::sample(h_index, 1, false, prob_z)(0);
  }
```

```
// update v ============================================================
for (uword l=0; l<(H-1); l++){
  n1_v = sum(z == l);
  n2_v = sum(z > l);
  v(l) = Rcpp::as<double>(Rcpp::wrap(R::rbeta(1 + n1_v, alpha_f + n2_v)));
}
v(H-1) = 1;

// update w ============================================================
w(0) = v(0);
for (uword l=1; l<H; l++){
  w(l) = v(l) * (1-v(l-1)) * (w(l-1)) / (v(l-1));
}

// update theta ========================================================
for (uword h=0; h<H; h++){
  if (z(h) <= h){
    theta(h) = theta_lim;
  } else{

    Lambda_sum_squared = as_scalar(trans(Lambda.col(h)) * Lambda.col(h));
    theta(h) = 1 / randg(distr_param(a_theta + 0.5*p,
                         1/( b_theta + 0.5*Lambda_sum_squared)));
  }
}

// update alpha DP =====================================================
alpha_cl = update_alpha(alpha_cl, a_dir, b_dir, n, K);

// update sigma ========================================================
for (uword j=0; j<p; j++){
  m1_temp = 0;
  m2_temp = 0;
  a_sigma_temp = a_sigma + 0.5*n;
  for (uword i=0; i<n; i++){
    m1_temp = as_scalar(Lambda.row(j) * eta.row(i).as_col());
    m2_temp = m2_temp + pow(y(i,j) - m1_temp, 2);
  }
  b_sigma_temp = b_sigma + 0.5*m2_temp;

  sigma(j) = 1 / randg(distr_param(a_sigma_temp, 1/b_sigma_temp));
}

// update latent space dimension=======================================
if ((it > start_adapt) && (randu() < exp(alpha_0+alpha_1*it))){

  Lambda_temp = Lambda;
  eta_temp = eta;
  Delta_temp = Delta;
  theta_temp = theta;
  w_temp = w;
  v_temp = v;

  uvec H_index = linspace<uvec>(0, H-1, H);
  H_star = sum(z > H_index);
  active.resize(H_star);
```

```
    active = find(z > H_index);

    if (H_star < (H-1)){
      flag = 1;
      H = H_star + 1;
      Lambda.cols(0, H-2) = Lambda_temp.cols(active);
      eta.set_size(n, H);
      eta.cols(0, H-2) = eta_temp.cols(active);
      theta.set_size(H);
      theta.subvec(0, H-2) = theta_temp(active);
      v.set_size(H);
      v.subvec(0, H-2) = v_temp(active);
      w.set_size(H);
      w.subvec(0, H-2) = w_temp(active);
      Delta.set_size(H ,H, K);
      eta.tail_cols(1) = randn(n) *
        sqrt(1 / randg(distr_param(nu_0/2, 1/(xi/2))));
      theta.tail(1) = theta_lim;
      w.tail(1) = 0;
      w.tail(1) = 1 - sum(w);
      Lambda.tail_cols(1) = randn(p) * sqrt(theta_lim);
    } else {
      flag = 1;
      H = H + 1;
      Lambda.resize(p,H);
      eta.resize(n, H);
      theta.resize(H);
      v.resize(H);
      w.resize(H);
      Delta.set_size(H ,H, K);
      eta.tail_cols(1) = randn(n) *
        sqrt(1 / randg(distr_param(nu_0/2, 1/(xi/2))));
      theta.tail(1) = theta_lim;
      v(H-2) = Rcpp::as<double>(Rcpp::wrap(R::rbeta(1 , alpha_f)));
      v(H-1) = 1;
      w(H-2) = v(H-2) * (1-v(H-3)) * (w(H-3)) / (v(H-3));
      w.tail(1) = v.tail(1) * (1-v(H-2)) * (w(H-2)) / (v(H-2));
      Lambda.tail_cols(1) =  randn(p) * sqrt(theta_lim);
    }

    if (flag==1){ // resize the temp variable accordly to the new H
      flag = 0;
      Lambda_temp.set_size(p, H);
      eta_temp.set_size(n, H);
      theta_temp.set_size(H);
      w_temp.set_size(H);
      v_temp.set_size(H);
      V_Lambda_temp.set_size(H, H);
      eta2.set_size(H,H);
      z.set_size(H);
      Delta_temp.resize(H, H, K)

      m_eta_temp.resize(H, K);
      V_eta_temp.resize(H,H,K);
      psi_eta_k.resize(H,H,K);
      chol_psi_eta_k.resize(H,H,K);
```

```
        for (uword k=0; k<K; k++){
          idx_k.set_size(n_k(k));
          idx_k = find(c == k);

          m_eta_temp.col(k) = sum(eta.rows(idx_k),0).as_col();
          V_eta_temp.slice(k) = trans(eta.rows(idx_k)) * eta.rows(idx_k);

          psi_eta_k.slice(k) = symmatu(V_eta_temp.slice(k) - m_eta_temp.col(k) *
            trans(m_eta_temp.col(k)) / (kappa_0 + n_k(k)));
          psi_eta_k.slice(k).diag() += xi;

        }

        L_sigma_prod.set_size(p,H);
        L_sigma_L_prod.set_size(H,H);

        chol_Omega_temp.set_size(H, H, K);
        chol_Delta.set_size(H, H, K);
        m_eta_wo_i.set_size(H);
        V_eta_wo_i.set_size(H, H);

        Rho.set_size(n, H);
        rho_k.set_size(H);
        psi_eta_wo_i.set_size(H, H);
      }
    }

    // save output ============================================================
    if ((it >= n_burn)){
      H_out(it-n_burn) = H_star;
      // Omega_out.slice(it-n_burn) = Lambda * Lambda.t() + diagmat(sigma);
      c_out.col(it-n_burn) = c;
      n_cluster(it-n_burn) = Delta.n_slices;
    }

    progress.increment();
  }
  // END GIBBS ================================================================

  out["H"] = H_out;
  out["clust"] = c_out;
  out["n_cluster"] = n_cluster;

  return out;
}
```

Bibliography

Antoniak, C. E. (1974). Mixtures of Dirichlet Processes with applications to Bayesian nonparametric problems. *The Annals of Statistics*, 2(6):1152–1174.

Armagan, A., Dunson, D. B., and Lee, J. (2013). Generalized double Pareto shrinkage. *Statistica Sinica*, 23(1):119.

Ascolani, F., Lijoi, A., Rebaudo, G., and Zanella, G. (2022). Clustering consistency with Dirichlet Process mixtures. *arXiv preprint arXiv:2205.12924*.

Attias, H. (1999). Independent factor analysis. *Neural Computation*, 11(4):803–851.

Baglama, J. and Reichel, L. (2005). Augmented implicitly restarted lanczos bidiagonalization methods. *SIAM Journal on Scientific Computing*, 27(1):19–42.

Bellman, R. (1966). Dynamic programming. *Science*, 153(3731):34–37.

Bernardo, J., Bayarri, M., Berger, J., Dawid, A., Heckerman, D., Smith, A., and West, M. (2003). Bayesian factor regression models in the "large p, small n" paradigm. *Bayesian Statistics*, 7:733–742.

Bhattacharya, A., Chakraborty, A., and Mallick, B. K. (2016). Fast sampling with Gaussian scale-mixture priors in high-dimensional regression. *Biometrika*, 103(4):985.

Bhattacharya, A. and Dunson, D. (2011). Sparse Bayesian infinite factor models. *Biometrika*, 98(2):291–306.

Bhattacharya, A., Pati, D., Pillai, N. S., and Dunson, D. B. (2015). Dirichlet–Laplace priors for optimal shrinkage. *Journal of the American Statistical Association*, 110(512):1479–1490.

Binder, D. A. (1978). Bayesian cluster analysis. *Biometrika*, 65(1):31–38.

Blackwell, D. and MacQueen, J. B. (1973). Ferguson distributions via Pólya urn schemes. *The Annals of Statistics*, 1(2):353–355.

Bouveyron, C. and Brunet-Saumard, C. (2014). Model-based clustering of high-dimensional data: A review. *Computational Statistics & Data Analysis*, 71:52–78.

Butler, A., Hoffman, P., Smibert, P., Papalexi, E., and Satija, R. (2018). Integrating single-cell transcriptomic data across different conditions, technologies, and species. *Nature Biotechnology*, 36:411–420.

Calo, D. G., Montanari, A., and Viroli, C. (2006). Model-based density estimation by independent factor analysis. In *From Data and Information Analysis to Knowledge Engineering*, pages 166–173. Springer.

Canale, A., Corradin, R., and Nipoti, B. (2022). Importance conditional sampling for Pitman–Yor mixtures. *Statistics and Computing*, 32(3):1–18.

Carvalho, C. M., Chang, J., Lucas, J. E., Nevins, J. R., Wang, Q., and West, M. (2008). High-dimensional sparse factor modeling: applications in gene expression genomics. *Journal of the American Statistical Association*, 103(484):1438–1456.

Carvalho, C. M., Polson, N. G., and Scott, J. G. (2009). Handling sparsity via the horseshoe. In *Artificial Intelligence and Statistics*, pages 73–80. PMLR.

Chandra, N. K., Canale, A., and Dunson, D. B. (2020). Escaping the curse of dimensionality in Bayesian model based clustering. *arXiv preprint arXiv:2006.02700*.

Consortium, T. M. et al. (2018). Single-cell transcriptomics of 20 mouse organs creates a tabula muris. *Nature*, 562(7727):367–372.

Durante, D. (2017). A note on the Multiplicative Gamma Process. *Statistics & Probability Letters*, 122:198–204.

Escobar, M. D. and West, M. (1995). Bayesian density estimation and inference using mixtures. *Journal of the American Statistical Association*, 90(430):577–588.

Ferguson, T. S. (1973). A Bayesian analysis of some nonparametric problems. *The Annals of Statistics*, pages 209–230.

Fowlkes, E. B. and Mallows, C. L. (1983). A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78(383):553–569.

Fritsch, A. (2022). *mcclust: Process an MCMC Sample of Clusterings*. R package version 1.0.1.

Ghahramani, Z., Hinton, G. E., et al. (1996). The EM algorithm for mixtures of factor analyzers. Technical report, CRG-TR-96-1, University of Toronto.

Hans, C. (2011). Elastic net regression modeling with the orthant normal prior. *Journal of the American Statistical Association*, 106(496):1383–1393.

Hastie, T., Tibshirani, R., and Wainwright, M. (2015). *Statistical Learning with Sparsity: The Lasso and Generalizations*. CRC Press.

Heller, K. A. and Ghahramani, Z. (2005). Bayesian hierarchical clustering. In *Proceedings of the 22nd international conference on Machine learning*, pages 297–304.

Hjort, N. L., Holmes, C., Müller, P., and Walker, S. G. (2010). *Bayesian nonparametrics*, volume 28. Cambridge University Press.

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

Ishwaran, H. and James, L. F. (2001). Gibbs sampling methods for stick-breaking priors. *Journal of the American Statistical Association*, 96(453):161–173.

Ishwaran, H. and Rao, J. S. (2005). Spike and slab variable selection: frequentist and Bayesian strategies. *The Annals of Statistics*, 33(2):730–773.

Jain, S. and Neal, R. M. (2004). A split-merge Markov chain Monte Carlo procedure for the Dirichlet Process mixture model. *Journal of Computational and Graphical Statistics*, 13(1):158–182.

Kalli, M., Griffin, J. E., and Walker, S. G. (2011). Slice sampling mixture models. *Statistics and Computing*, 21(1):93–105.

Kim, S., Tadesse, M. G., and Vannucci, M. (2006). Variable selection in clustering via Dirichlet Process mixture models. *Biometrika*, 93(4):877–893.

Kowal, D. R. and Canale, A. (2021). Semiparametric functional factor models with Bayesian rank selection. *arXiv preprint arXiv:2108.02151*.

Lau, J. W. and Green, P. J. (2007). Bayesian model-based clustering procedures. *Journal of Computational and Graphical Statistics*, 16(3):526–558.

Legramanti, S., Durante, D., and Dunson, D. B. (2020). Bayesian cumulative shrinkage for infinite factorizations. *Biometrika*, 107(3):745–752.

McInnes, L., Healy, J., and Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

Medvedovic, M. and Sivaganesan, S. (2002). Bayesian infinite mixture model based clustering of gene expression profiles. *Bioinformatics*, 18(9):1194–1206.

Meilă, M. (2007). Comparing clusterings—an information based distance. *Journal of Multivariate Analysis*, 98(5):873–895.

Miller, J. W. and Harrison, M. T. (2013). A simple example of Dirichlet Process mixture inconsistency for the number of components. *Advances in Neural Information Processing Systems*, 26.

Mitchell, T. J. and Beauchamp, J. J. (1988). Bayesian variable selection in linear regression. *Journal of the American Statistical Association*, 83(404):1023–1032.

Montanari, A. and Viroli, C. (2010). Heteroscedastic factor mixture analysis. *Statistical Modelling*, 10(4):441–460.

Morandat, F., Hill, B., Osvald, L., and Vitek, J. (2012). Evaluating the design of the R language. In *European Conference on Object-Oriented Programming*, pages 104–131. Springer.

Müller, P., Erkanli, A., and West, M. (1996). Bayesian curve fitting using multivariate normal mixtures. *Biometrika*, 83(1):67–79.

Murphy, K., Viroli, C., and Gormley, I. C. (2020). Infinite mixtures of infinite factor analysers. *Bayesian Analysis*, 15(3):937–963.

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

Park, T. and Casella, G. (2008). The Bayesian lasso. *Journal of the American Statistical Association*, 103(482):681–686.

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

Roberts, G. O. and Rosenthal, J. S. (2007). Coupling and ergodicity of adaptive Markov chain Monte Carlo algorithms. *Journal of Applied Probability*, 44(2):458–475.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65.

Rue, H. (2001). Fast sampling of Gaussian Markov random fields. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):325–338.

Schiavon, L. and Canale, A. (2020). On the truncation criteria in infinite factor models. *Stat*, 9(1):e298.

Schiavon, L., Canale, A., and Dunson, D. B. (2022). Generalized infinite factorization models. *Biometrika*, 109(3):817–835.

Sethuraman, J. (1994). A constructive definition of Dirichlet priors. *Statistica Sinica*, pages 639–650.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288.

Wade, S. and Ghahramani, Z. (2018). Bayesian cluster analysis: Point estimation and credible balls (with discussion). *Bayesian Analysis*, 13(2):559–626.

Walker, S. G. (2007). Sampling the Dirichlet mixture model with slices. *Communications in Statistics—Simulation and Computation®*, 36(1):45–54.

Waltman, L. and Van Eck, N. J. (2013). A smart local moving algorithm for large-scale modularity-based community detection. *The European Physical Journal B*, 86(11):1–14.

West, M. (1992). *Hyperparameter estimation in Dirichlet Process mixture models*. Duke University ISDS Discussion Paper# 92-A03.

Wickham, H. (2019). *Advanced R*. CRC press.

Zou, H., Hastie, T., and Tibshirani, R. (2006). Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, 15(2):265–286.