Laboratoire de Probabilité, Statistiques et Modélisation

May 2023

Internship report notes

Master 2 - Data Science

Alexandre CHAUSSARD

Contents

1 Introduction

Given observations of random variables (X, Y), we suppose that there exist another set of random variables Z that we do not observe, yet that characterize (X, Y) conditionally to Z. Z is then called a latent variable, or a hidden variable.

For instance, if we observe the weights of a given population through X, and that we aim at inferring their height Y, knowing the sex of each individual through Z could improve our predictions on Y. Hence, assuming that there exist a latent variable to a given model adds structure to the model while improving the explainability, as Z characterizes the behavior of our dataset. Typically, clustering methods like KMeans or Gaussian Mixtures Models (GMM) provide a discrete Z given the observations, which are interesting in the sense that they provide a categorical representation of our data.

However, finding such Z given the observations is not straight forward, and not all dataset respond to a latent process, and may be not even a discrete one.

During this research internship, we aim at exploring latent models with discrete latent space in order to analyze the microbiota structure. Our first focus will be on various methods to conceive latent models like Expectation-Maximization and variational models.

2 Expectation-Maximization

2.1 Overview

The Expectation-Maximization (EM) algorithm, first introduced in [1], is vast class of latent models. It is based on the following smart decomposition of the data:

$$\log p_\theta(X) = \underbrace{\mathbb{E}_{p_{\widehat{\theta}}(Z|X)}[\log p_\theta(X,Z)|X]}_{Q(\widehat{\theta},\theta)} - \mathbb{E}_{p_{\widehat{\theta}}(Z|X)}[\log p_\theta(Z|X)|X]$$

The idea behind this decomposition is that $\log p_\theta(X)$ is generally not tractable since it's an integral, while the complete likelihood $p_\theta(X,Z)$ is generally manageable. Note that since $\log p_\theta(X)$ can not be computed, $\log p_\theta(Z|X)$ can't either by extension. Hence, we introduce $\log p_{\widehat{\theta}}(Z|X)$ where $\widehat{\theta}$ is the maximum likelihood estimator of θ:

$$\widehat{\theta} = arg\max_\theta \log p_\theta(X)$$

The main trick of the EM algorithm relies in the idea that $Q(\widehat{\theta},\theta)$ is sufficient to compute a maximum likelihood estimator of θ. Indeed, consider the following algorithm:

Algorithm 1 Expectation-Maximization

Require: $\widehat{\theta}$
  Repeat until convergence
    Expectation: compute $p_{\widehat{\theta}}(Z|X)$ to compute $Q(\widehat{\theta},.)$
    Maximization: $\widehat{\theta} = arg\max_\theta Q(\widehat{\theta},\theta)$
  return $\widehat{\theta}$ =0

If we denote by $\widehat{\theta}^h$ the iterates of this algorithm, one can show using Jensen's inequality that:

$$\log p_{\widehat{\theta}^{h+1}}(X) \geq \log p_{\widehat{\theta}^h}(X)$$

As a result, the EM algorithm maximizes the likelihood, producing an estimator $\widehat{\theta}$ that is an MLE of θ. Note that we don't have a convergence certainty towards the best maximizer of the likelihood, only to a local maxima. Hence, the EM algorithm is heavily sensitive to the initialization we pick.

All is required now is to choose a latent model so we can perform the EM algorithm, meaning that we have to define the distributions of the followings:

- $Z \sim \mathcal{B}(K,\pi)$, conveniently set to a binomial of parameter π so that it's discrete and simple to manage.
- $X|Z=k \sim p_{\gamma(k)}(X|Z=k)$, which is where we have the most choice to make.

In such models, $\theta = (\pi, \gamma(0), ..., \gamma(K))$. In the next section, we will study a specific fork architecture of the gaussian mixture case.

2.2 Gaussian Mixture Linear Classifier

2.2.1 Framework and computations

Consider the case for which we observe $(X_i, Y_i)_{1 \leq i \leq n}$ i.i.d samples, where X_i denotes a feature vector and Y_i a label in a classification framework. We aim at introducing a linear classifier that exploits a latent structure over (X, Y), so that we have the following latent model:

- $Z_i \sim \mathcal{B}(K, \pi)$, we note $\pi_k = \mathbb{P}(Z_i = k)$
- $X_i | Z_i = k \sim \mathcal{N}(\mu_k, \sigma_k I)$, we denote by $f_k(X_i)$ its density.
- $\mathbb{P}(Y_i = 1 | X_i, Z_i = k) = \sigma(W_{e,k}^T e_k + W_{x,k}^T X_i) = p_k(X_i)$, where e_k denotes a vector from the canonical basis of $\mathbb{R}^K$.

As we want to perform the EM algorithm find an MLE of

$$\theta = (\pi, \mu_1, ..., \mu_K, \sigma_1, ..., \sigma_K, W_{e,1}, ..., W_{e,K}, W_{x,1}, ..., W_{x,K})$$

we start by computing the **expectation** step by assessing $p_{\widehat{\theta}}(Z_i = k | X_i, Y_i)$, using Bayes rules ($\hat{\pi}, \hat{p}, \hat{f}$ signify that we evaluate these quantities using the current estimate $\widehat{\theta}$):

$$\begin{aligned} p_{\widehat{\theta}}(Z_i = k | X_i, Y_i) &= \frac{\hat{\pi}_k \hat{f}_k(X_i) \left(Y_i \hat{p}_k(X_i) + (1 - Y_i)(1 - \hat{p}_k(X_i))\right)}{\sum_{j=1}^K \hat{\pi}_j \hat{f}_j(X_i) \left(Y_i \hat{p}_j(X_i) + (1 - Y_i)(1 - \hat{p}_j(X_i))\right)} \\ &= \tau_{ik} \end{aligned}$$

Now, we can safely evaluate $Q(\widehat{\theta}, \theta)$ for any θ:

$$\begin{aligned} Q(\widehat{\theta}, \theta) &= \mathbb{E}_{p_{\widehat{\theta}}(Z|X)}[\log p_\theta(X, Y, Z) | X] \\ &= \sum_{i=1}^n \sum_{k=0}^K \log p_\theta(X_i, Y_i, Z_i = k) \tau_{ik} \\ &= \sum_{i=1}^n \sum_{k=0}^K (\log \pi_k + \log f_k(X_i) + Y_i \log p_k(X_i) + (1 - Y_i) \log(1 - p_k(X_i))) \tau_{ik} \end{aligned}$$

The **maximization** step now consists in deriving $Q(\widehat{\theta}, \theta)$ regarding each parameters in θ so that we obtain an either explicit value or iterative procedure to compute the next iterate of $\widehat{\theta}$.

- Maximization regarding π_k under constraint that $\sum_{k=0}^K \pi_k = 1$ can be solved explicitly using Lagrange duality:

$$\pi_k^* = \frac{1}{n} \sum_{i=1}^n \tau_{ik}$$

- Maximization regarding $(\mu_k, sigma_k)$ is given by the maximum of likelihood estimator on $\sum_{i=1}^n \tau_{ik} \log f_k(X_i)$:

$$\begin{aligned} \mu_k^* &= \frac{1}{\sum_{i=1}^n \tau_{ik}} \sum_{i=1}^n \tau_{ik} X_i \\ \sigma_k^* &= \frac{1}{\sum_{i=1}^n \tau_{ik}} \sum_{i=1}^n \tau_{ik} (X_i - \mu_k^*)(X_i - \mu_k^*)^\top \end{aligned}$$

- Maximization regarding $(W_{e,k}, W_{x,k})$ is not explicit, and requires a fixed point algorithm like gradient descent to determine an estimate of the optimal parameters. The iterations using full batch gradient descent are given below, with learning rate α:

$$W_{e,k}^{l+1} \leftarrow W_{e,k}^{l} - \alpha \sum_{i=1}^{n} (p_k(X_i) - Y_i)\tau_{ik} e_k$$
$$W_{x,k}^{l+1} \leftarrow W_{x,k}^{l} - \alpha \sum_{i=1}^{n} (p_k(X_i) - Y_i)\tau_{ik} X_i$$

  Each iteration should be confronted to the maximization criterion, so that each iterate improves $Q(\widehat{\theta}, \theta)$:

$$Q(\widehat{\theta}^{l+1}, \theta) \geq Q(\widehat{\theta}^{l}, \theta)$$

  In the end, only the best improvement iterate is kept for $W_{e,k}^*$ and $W_{x,k}^*$. Note that other methods could be used like SGD or CMAES as implemented during the internship.

After the maximization, we can update $\widehat{\theta}$ with the previously computed parameters, and redo the (E) and (M) steps up to convergence. The convergence can be measured relatively to $Q(\widehat{\theta}, \theta)$, so that for a threshold ϵ, we can use the following stopping criterion:

$$\frac{|Q(\widehat{\theta}^{h+1}, \widehat{\theta}^{h}) - Q(\widehat{\theta}^{h}, \widehat{\theta}^{h})|}{|Q(\widehat{\theta}^{h}, \widehat{\theta}^{h})|} \leq \epsilon$$

We now have a ready-to-go Gaussian Mixture Linear Classifier that we can benchmark on an suited dataset against other common methods.

2.2.2 Dataset generation

Before we benchmark the Gaussian Mixture Linear Classifier, we need to generate a dataset that is well suited to its usage. Hence, we set random parameters for θ and generate a new dataset following the latent framework we have set previously:

- For all $k \leq K$, generate n/K points following a gaussian parameterized by $(\mu_k, \sigma_k I)$. X is given by each point coordinate, Z by the gaussian from which the point was generated.
- For each sample, characterized by (X_i, Z_i), draw the label of the sample as $Y_i \sim \mathcal{B}(\sigma(W_{e,Z_i}^T e_{Z_i} + W_{x,Z_i}^T X_i))$.

The following figure illustrates a generation with $K = 2$:

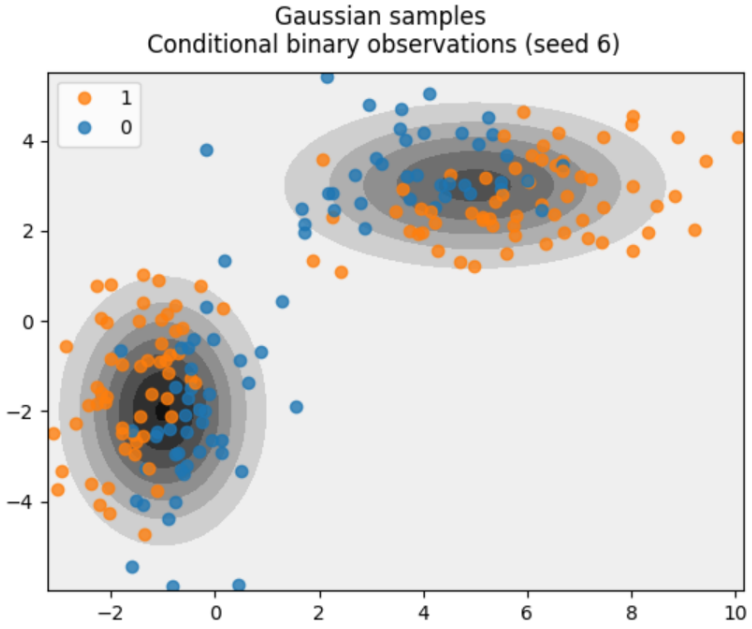

Figure 1: Generated samples out of $K = 2$ gaussians, labelized using a logistic model (label 1 or 0). The density of the hidden gaussians is represented in the background using shades of grey (black intense, white almost 0)

Note on the previous figure 1 that the limit between the labels in each subset gaussian is not sharp, as it is sorted out of a probabilistic modelisation (logistic). Consequently, an interesting observation can be made as we force the gaussians to have small values of mean and variance. Indeed, as shown on the next figure, if we take the same parameters as previously and divide them by a factor 10, the logistic model is ill conditioned.

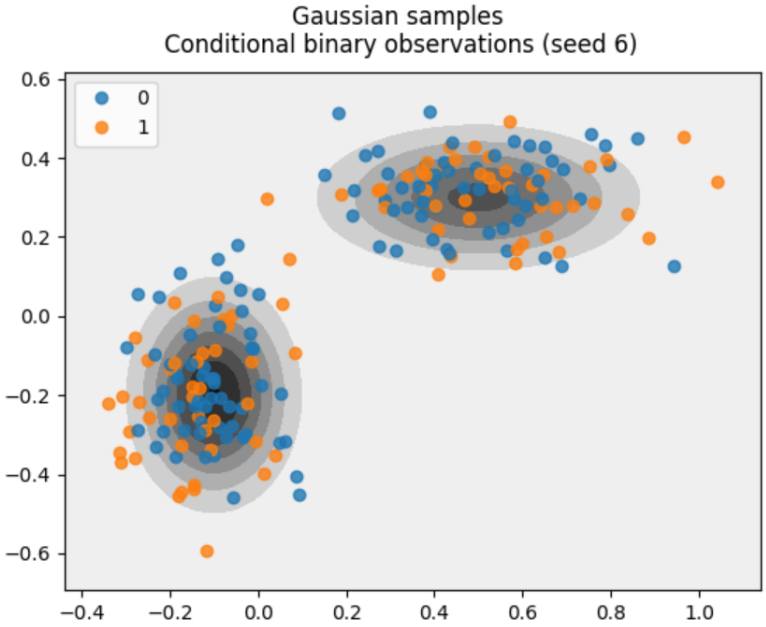

Figure 2: Generated samples out of $K = 2$ gaussians, using the same parameters as for figure 1 divided by a factor 10.

All points are close to the frontier in terms of norm on figure 2. Since we do not exploit any normalization term in the sigmoid modelization, this leads to a blurry area in which the linear separation model is not useful at all. This situation is heavily problematic as it prevents us from performing a general benchmark on that dataset. Indeed, the further larger the variance is, with sufficient samples, the better the linear approximation will be and therefore the better the modelization gets. On the contrary, with smaller variance the linear model isn't descriptive of the generated samples as they are all close to the frontier, therefore with heavily noisy labelization.

2.3 Dirichlet Mixture Linear Classifier

2.3.1 Framework and objectives

In the previous section, we have derived a very classical model (Gaussian Mixture Model) into a linear classifier using the EM algorithm. However, gaussian latent modelization is far from general, and may not be suited to our specific usage on the microbiota. Indeed, after performing the previous method onto our microbiota dataset, it turned out to be performing just as bad as the classical logistic regression, no matter the chosen latent space dimension. Therefore, we deduce that the gaussian latent modelization is not adapted to our practical settings.

As we analyze the data, we observe that each X_i belongs to the simplex. A natural distribution supported on the simplex is the Dirichlet distribution, parameterized by $\alpha = (\alpha_1, \dots, \alpha_p)$ where p is the dimension of X_i. We denote the dirichlet distribution by $\mathcal{D}(\alpha)$, for which the density is given by the following:

$$f(x|\alpha) = \frac{\Gamma\left(\sum_{j=1}^{p} \alpha_j\right)}{\Pi_{j=1}^{p}\Gamma(\alpha_j)} \Pi_{j=1}^{p} x_j^{\alpha_j - 1}$$

where Γ denotes the gamma function. For notation simplicity, we also introduce the digamma function that will play a key role in our model:

$$\psi(\alpha) = \frac{d}{d\alpha} \log \Gamma(\alpha) = \frac{\Gamma'(\alpha)}{\Gamma(\alpha)}$$

Since we are doing a Dirichlet mixture model, we have K Dirichlet distributions to handle. We introduce the notation $\alpha^{(k)}$ to parameterize the k-th Dirichlet distribution.

As previously, we first perform the **expectation** step obtain the same result with a different conditional a priori distribution on $X|Z$:

$$\begin{aligned} p_{\widehat{\theta}}(Z_i = k|X_i, Y_i) &= \frac{\hat{\pi}_k f(X_i|\hat{\alpha}^{(k)})(Y_i \hat{p}_k(X_i) + (1 - Y_i)(1 - \hat{p}_k(X_i)))}{\sum_{j=1}^{K} \hat{\pi}_j f(X_i|\hat{\alpha}^{(j)})(Y_i \hat{p}_j(X_i) + (1 - Y_i)(1 - \hat{p}_j(X_i)))} \\ &= \tau_{ik} \end{aligned}$$

We can now evaluate $Q(\widehat{\theta}, \theta)$ for any θ, which enables us to perform the **maximization** step:

- The maximization over pi_k under the simplex constraint $\sum_{k=1}^{k} \pi_k = 1$ is again given by Lagrange duality as:

$$\pi_k^* = \frac{1}{n} \sum_{i=1}^{n} \tau_{ik}$$

- The maximization over $\alpha^{(k)_j}$ is not straightforward on the other hand, and requires a fixed point algorithm. Indeed, deriving over $\alpha^{(k)_j}$ we obtain:

$$\begin{aligned}
\partial_{\alpha_j^{(k)}} Q(\widehat{\theta}, \theta) &= \sum_{i=1}^{n} \left(\psi \left(\sum_{l=0}^{K} \alpha_l^{(k)} \right) - \psi(\alpha_j^{(k)}) + \log x_{ij} \right) \tau_{ik} \\
&= \left(\psi \left(\sum_{l=0}^{K} \alpha_l^{(k)} \right) - \psi(\alpha_j^{(k)}) \right) \sum_{i=1}^{n} \tau_{ik} + \sum_{i=1}^{n} \tau_{ik} \log x_{ij}
\end{aligned}$$

  Hence, as we look for $\partial_{\alpha_j^{(k)}} Q(\widehat{\theta}, \theta) = 0$, we obtain:

$$\psi(\alpha_j^{(k)}) - \psi \left(\sum_{l=0}^{K} \alpha_l^{(k)} \right) = \frac{\sum_{i=1}^{n} \tau_{ik} \log x_{ij}}{\sum_{i=1}^{n} \tau_{ik}}$$

  Thankfully, [4] provides a few tricks to solve iteratively such equation, so that we can iterate as follows (5 steps are sufficient to obtain high-accuracy solution according to [4]):

$$\begin{aligned}
\alpha_j^{(k)} &\leftarrow \psi^{-1} \left(\frac{\sum_{i=1}^{n} \tau_{ik} \log x_{ij}}{\sum_{i=1}^{n} \tau_{ik}} + \psi \left(\sum_{l=0}^{K} \hat{\alpha}_l^{(k)} \right) \right) \\
\hat{\alpha}_j^{(k)} &\leftarrow \alpha_j^{(k)}
\end{aligned}$$

  However, this solution is a lower bound to the true objective, which makes our EM a generalized version of it.

- The maximization over $(W_{e,k}, W_{x,k})$ is also given by a fixed point algorithm, which ends up being the same computation as previously for the Gaussian case:

$$\begin{aligned}
W_{e,k}^{l+1} &\leftarrow W_{e,k}^{l} - \alpha \sum_{i=1}^{n} (p_k(X_i) - Y_i) \tau_{ik} e_k \\
W_{x,k}^{l+1} &\leftarrow W_{x,k}^{l} - \alpha \sum_{i=1}^{n} (p_k(X_i) - Y_i) \tau_{ik} X_i
\end{aligned}$$

2.3.2 Dataset generation

Now that we have defined the EM algorithm in the previous section, we aim at generating a dataset to benchmark the dirichlet mixture classifier. Following a similar procedure as for the gaussian case, we are able to generate a dataset that matches a dirichlet mixture and is labeled following the sigmoid modelisation. The next figure illustrates a given generation:

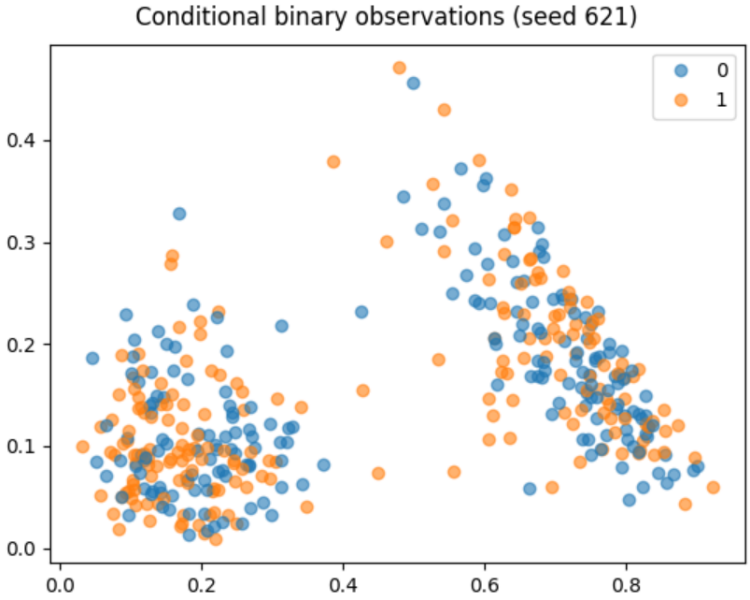

Figure 3: Generated samples out of $K = 2$ dirichlet distributions, labelized using the sigmoid modelisation on $\mathbb{P}(Y_i = 1)$.

As previously, since the data lives in the simplex, they are too close to the border of the next label set, which ends up creating a blurry dataset for which the linear model is not relevant anymore.

3 Variational methods

In this section, we aim at resourcing some variational methods that we are going to use in this study.

3.1 Variational Auto-Encoders

3.1.1 Framework and optimization objective

We are interested in another kind of latent models, this time based on variational inference results to achieve a new kind of deep latent structure: the Variational Auto-Encoder (VAE). These latent models were introduced in 2013 by Kingma, better described in a more in depth paper in 2019: see [3]

Once again, we assume the observations X to be modelizable by a given distribution parameterized by θ:

$$X \sim p_\theta(x)$$

Determining θ holds to find one θ^* that would optimize a given objective, generally chosen as the maximum of likelihood. Indeed, if $\theta^* \in arg\max_\theta p_\theta(x)$, then such θ^* maximizes the density around the dense areas of the observations, which makes them highly likely to happen under such distribution p_{θ^*}. Hence, the maximum likelihood is a natural criterion:

$$\theta^* \in arg\max_\theta p_\theta(x)$$

However, such modelization does not include a latent structure. As a result, we try to enforce it by rewriting the objective as follows:

$$p_\theta(x) = \int_{\mathcal{Z}} p(x,z)dz$$

Using Bayes decomposition, we obtain the following objective:

$$p_\theta(x,z) = p_\theta(z)p_\theta(x|z)$$

Recall that the prior $p_\theta(z)$ and the a priori $p_\theta(x|z)$ are defined by the framework (ex: Bernoulli prior and Gaussian posterior gives the Gaussian mixture framework). However, the computation of the evidence $p_\theta(x)$ is generally intractable in practice, which also leads to a non-tractable posterior distribution: $p_\theta(z|x)$. As a result, not being able to compute the evidence leads to not being able to provide a gradient regarding θ, so we can not perform the backpropagation in a deep learning approach.

Note that there exist approximate inference techniques to compute the evidence and the posterior, but these are quite expensive and often yield poor convergence results.

To overcome this issue, we introduce a smart rewriting of the objective using variational

inference. Indeed, let $q_\Phi(z|x) \approx p_\theta(z|x)$ to be learnt over Φ, one can write:

$$\begin{aligned}
\log p_\theta(x) &= \mathbb{E}_{q_\Phi(z|x)}[\log p_\theta(x)] \\
&= \mathbb{E}_{q_\Phi(z|x)}\left[\log \frac{p_\theta(x)}{q_\Phi(z|x)} \frac{q_\Phi(z|x)}{p_\theta(z|x)}\right] \\
&= \underbrace{\mathbb{E}_{q_\Phi(z|x)}\left[\log \frac{p_\theta(x)}{q_\Phi(z|x)}\right]}_{ELBO(q_\phi(z|x), p_\theta(x,z))} + D_{KL}\left[q_\Phi(z|x) \| p_\theta(z|x)\right]
\end{aligned}$$

The first term of that decomposition is generally called the Evidence Lower BOund (ELBO), as it marks a lower bound to the evidence $\log p_\theta(x)$ since the KL divergence is a positive quantity:

$$\log p_\theta(x) \geq ELBO(q_\phi(z|x), p_\theta(x, z))$$

$q_\Phi(z|x)$ is an approximation of the true posterior $p_\theta(z|x)$ that we aim at learning in a family of distributions. For instance,

$$q_\Phi(.|x) \sim \mathcal{N}(\mu(x), \Sigma(x))$$

would be an approximation of the true posterior by a Gaussian distribution. Notice that the true posterior may very not likely be Gaussian, which creates a first complexity error in our model.

Despite being a lower bound on the true maximum likelihood objective, the ELBO is actually tractable. Indeed, as we continue the computation:

$$\begin{aligned}
ELBO(q_\phi(z|x), p_\theta(x, z)) &= \mathbb{E}_{q_\Phi(z|x)}[\log p_\theta(x, z)] - \mathbb{E}_{q_\Phi(z|x)}[\log q_\Phi(z|x)] \\
&= \mathbb{E}_{q_\Phi(z|x)}[\log p_\theta(x|z)] - D_{KL}[\log q_\Phi(z|x) \| p_\theta(z)]
\end{aligned}$$

Another remarkable fact, is that when maximizing the ELBO, we are actually minimizing the KL divergence between the estimated and the true posterior. Hence, one can define the ELBO as a suboptimal objective to our problem that we get to maximize to obtain (Φ^*, θ^*), the parameters of our model.

3.1.2 Reparameterization trick

Even though the gradient of the ELBO is well defined for θ, it is not possible to compute the differential relatively to Φ yet, as it requires samples from the approximation to the posterior $q_\Phi(z|x)$ to compute $\mathbb{E}_{q_\Phi(z|x)}[\log p_\theta(x|z)]$.

Since, sampling is not a differentiable operation, we make use of the change of variable formula, so that for a bijective transformation $z = \phi_x(\epsilon)$, we get:

$$p(z) = p(\epsilon) \det \left| \frac{\partial \epsilon}{\partial z} \right|$$

Hence, if we take ϵ a random variable of density $p(\epsilon)$ that does not depend on θ, Φ nor x, so that $z = \phi_x(\epsilon)$, for any L_1 function f,

$$\mathbb{E}_{q_\Phi(z|x)}[f(z)] = \mathbb{E}_{p(\epsilon)}[f(z)]$$

As a result, the samples are not obtained through q_Φ anymore but through $p(\epsilon)$, so that can safely perform derivation of the ELBO relatively to Φ and backpropagate our gradient through the network.

3.1.3 Architecture

The vanilla architecture of the VAE is described by the following illustration:

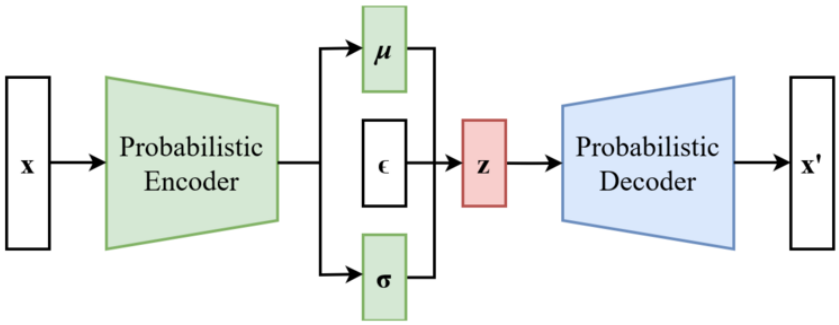

Figure 4: Illustration of a VAE with Gaussian prior (wikipedia)

The first part is generally called the encoder, as it turns a sample x into its latent representation z by modelizing the posterior $q_\Phi(z|x)$. The second part is then called the decoder, as it throws a latent representation in the sample space. The latest can even serve as a generative architecture, as one can sample from the latent space through $q_\Phi(z|x)$, and decode it to obtain a new sample.

As we can see more clearly in that illustration, we can see that Φ and θ are trained jointly through the ELBO, both serving for one part of the VAE at a time.

The training procedure is straightforward: the entry is a sample x and the output objective is the same sample x. We aim at train the VAE for learning the data space and its latent representation by learning how to reconstruct the samples through it.

3.1.4 Limits for our problem

As we have seen through the reparameterization trick, training a VAE architecture requires to be able to backpropagate the gradient of the ELBO at each step. We namely had to perform the reparameterization trick to circumvent the randomness operation which is not differentiable. As a result, learning a discrete posterior is not possible with such architecture, since we would have to perform a projection of the output of the encoder on a discrete space, which is not a differentiable operation.

Yet, learning discrete representation of our data seems much more natural than continuous latent ones. As we tend to categorize things as much as we can, describing behaviors with words for instance. Furthermore, a discrete representation facilitates the interpretation of the latent space by ordering data distribution in simple bins.

The next architecture, called the VQ-VAE, stands as a first fork option to the VAE with discrete posterior.

3.2 VQ-VAE

3.2.1 Quick overview

Introduced in [7], VQ-VAE architecture provides a framework to compute discrete posterior distributions $q_\Phi(z|x)$. To compare that model with the VAE, we start by introducing the architecture of the model for which an illustration is given below:

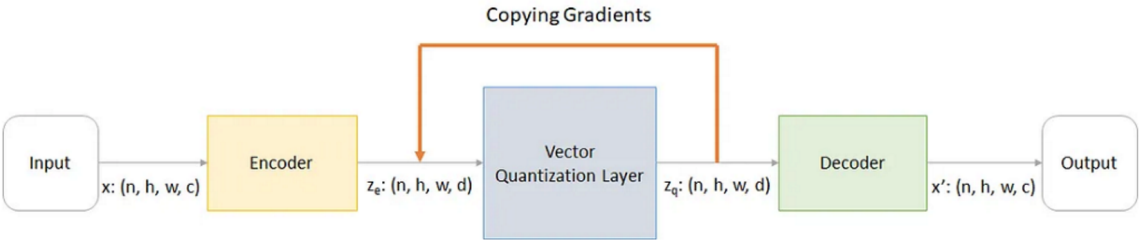

Figure 5: VQ-VAE architecture (source: Medium)

As we can see on figure 5, the major difference with the vanilla VAE architecture lies in the vector quantization step which enables to project the output of the encoder denoted by $z_e(x)$ onto a discrete embedding dictionary $(e_1, \ldots, e_K)$ by a simple distance argument:

$$k = arg\min_j \|z_q(x) - e_j\|_2, \quad z_q(x) = e_k$$

The projection of $z_e(x)$ on that discrete dictionary is denoted by $z_q(x)$, and serves as the input of the decoder. For further visual representation of the vector quantization layer, an illustration is given below.

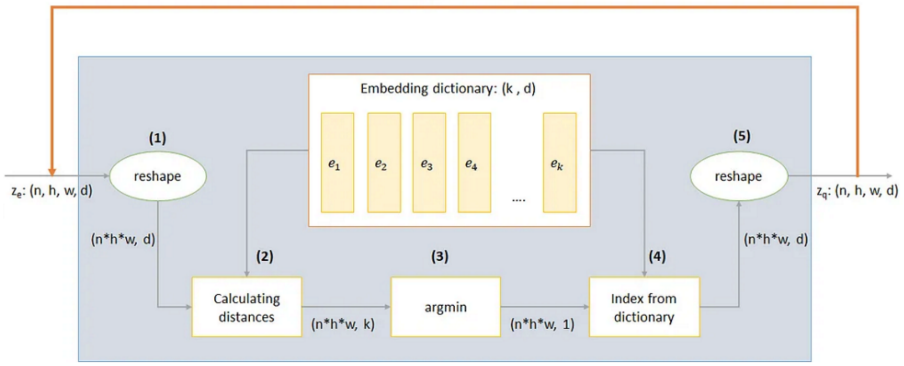

Figure 6: Architecture of the VQ-VAE: quantization layer (source: Medium)

Looking at the previous figures, we can grasp the challenge of backpropagation in such model with discrete prior and posterior. In the next section, we enter in the mathematical definition of the objective and how to train this architecture.

3.2.2 Framework and optimization objective

Contrary to the vanilla VAE, we have the following categorical distributions assumption:

- The prior $p_\theta(z)$ is categorical. In the original paper, it is taken as uniform supported in $\{1, \dots, K\}$ during the training. When the training is over, it is fit to an autoregressive distribution through a PixelCNN (see [6]). It is left as an exploration research field to be able to learn the prior while training the model.

- The posterior $q_\Phi(z|x)$ is categorical and set to the following:

$$q_\Phi(k|x) = \mathbb{1}_{\{k=arg\min_j \|z_q(x)-e_j\|_2\}}$$

This modelization of the posterior enables to obtain a discrete latent space, but it does not allow to differentiate regarding Φ. As a result, the authors suggest two possible strategies:

- *Straight-through*: propagate the gradient through the discrete part (vector quantization layer) without changing it. The intuition is that the gradient propagated from the encoder contains sufficient information to update the encoder accordingly, but this is just intuition.

- *Subgradient*: compute the subgradient of the quantization layer (unexplored yet)

The optimization objective of the VQ-VAE is based on the ELBO, that one can compute as follow:

$$\begin{aligned} ELBO(q_\Phi(z|x), p_\theta(z|x)) &= \mathbb{E}_{q_\Phi(z|x)}[\log p_\theta(x,z)] - \mathbb{E}_{q_\Phi(z|x)}[\log q_\Phi(z|x)] \\ &= \mathbb{E}_{q_\Phi(z|x)}[\log p_\theta(x|z)] - D_{KL}[q_\Phi(z|X)\|p_\theta(z)] \end{aligned}$$

Notice then that:

- Since $q_\Phi(k|x) = \mathbb{1}_{\{k=arg\min_j \|z_q(x)-e_j\|_2\}}$, we have:

$$\mathbb{E}_{q_\Phi(z|x)}[\log p_\theta(x|z)] = \log p_\theta(x|z_q(x))$$

- Since $Z \sim \mathbb{U}(\{1, ..., K\})$, $\mathbb{P}(Z = k) = \frac{1}{K}$. Also, notice that $q_\Phi(z_q(x)|x) = 1$ by definition of $q_\Phi(z|x)$ and $z_q(x)$. Combining those results, we obtain:

$$\begin{aligned} D_{KL}[q_\Phi(z|x)\|p_\theta(z)] &= \mathbb{E}_{q_\Phi(z|x)}\left[\log \frac{q_\Phi(z|x)}{p_\theta(z)}\right] \\ &= \log \frac{q_\Phi(z_q(x)|x)}{p_\theta(z_q(x))} \\ &= \log K \end{aligned}$$

  Therefore, this KL divergence does not impact the optimization objective as it does not depend on (θ, Φ).

Computing the previous quantities, we would obtain the following suboptimal objective for the VQ-VAE:

$$ELBO(\Phi, \theta) = log p_\theta(x|z_q(x))$$

However, such objective does not enable to learn the dictionary since we use a straight-through approach over the quantization layer. To update the dictionary, the authors suggest to add the following term to the loss:

$$\|sg[z_e(x)] - e\|_2^2$$

Where sg denotes the stop-gradient operator, meaning we do not consider any gradient after the given operation.

Finally, to prevent embedding space over expansion, the authors suggest the addition of a commitment loss parameterized by $\beta > 0$:

$$\beta \|z_e(x) - sg[e]\|_2^2$$

Indeed, the embedding space was unconstrained so far, so its dimension could grow arbitrarily. Intuitively, adding that term forces the model to commit to a given embedding.

Overall, we obtain this final objective to maximize:

$$\mathcal{L}(\theta, \Phi) = \log p_\theta(x|z_q(x)) + \|sg[z_e(x)] - e\|_2^2 + \beta \|z_e(x) - sg[e]\|_2^2$$

3.2.3 Discussion over some limiting aspects

Even though the previous objective seems natural, we only rigorously justified the first term thanks to the ELBO. Indeed, the dictionary update loss as well as the commitment loss keep dropping out from nowhere, while they seem necessary to ensure the training of our model.

Furthermore, we did not really deal with the non differentiability of the vector quantization loss, and the straight-through estimator can definitely be criticized about what information it actually provides to the encoder to update appropriately.

Finally, the usage of the PixelCNN after the training seems very unnatural, and could significantly boost the model as the PixelCNN did show amazing performances so far.

3.3 PixelCNN

3.3.1 Overview

Introducing the VQ-VAE, we have seen that an under-table tool that was being used is the Pixel CNN, first introduced in [6]. In the context of the VQ-VAE, it plays a major role after training by learning the prior $p(z)$, that was set to a discrete uniform previously. This makes a drastic difference with the VAE, as we are not setting the prior ourselves as it's being learnt and modeled by the PixelCNN in this process.

Indeed, the PixelRNN/PixelCNN architectures are sequential deep neural networks that aims at modeling the distribution of a data space in an autoregressive fashion. Their main characteristic is that they take advantage of the structure of an image to learn the data space distribution.

- *PixelRNN*: Bi-directional recurrent networks with variable smart directions are used to model the spatial dependencies between pixels. (3 architectures are presented in the original paper, but our focus will be on the PixelCNN here).
- *PixelCNN*: the dependencies between the pixels is modeled through stacking of masked convolution layers (it's faster since the receptive field is bounded by the size of the convolution), no pooling layer is used. The masking ensures that we keep an autoregressive estimation of a new pixel, without seeing the future pixels.

The following figure illustrates the masked convolution technique.

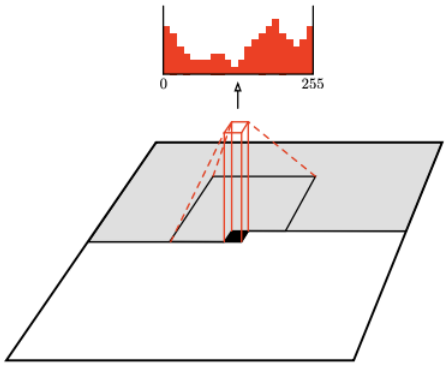

Figure 7: Masked convolution on an image (largest square), the big square represents the receptive field of the black pixel, the white area is masked

Hence, given x an image represented by its pixels $x = (x_1, \ldots, x_n)$, as usual for these distribution modeling problems, we aim at finding $\theta^* = arg\max_\theta p_\theta(x)$ in a family of distributions parameterized by θ: the maximum of likelihood. Contrary to the usual independent framework, we consider a local dependency between pixels given by the receptive field of our convolution. This local dependency is limited to the already seen pixels only as well, thanks to the masked convolution.

Furthermore, rather than using continuous outputs, these architecture use a softmax layer to determine the pixel of a given generation, leading to a discrete prior rather than a continuous

one, which is required for the VQ-VAE for instance. As a result, the distribution of a pixel conditionally to the ones in its receptive field is given by a multinomial in $\{0, \dots, 256\}$.

3.3.2 Usage in the VQ-VAE

Once we have trained the VQ-VAE, the original paper states that we can replace the uniform prior on $p(z)$ by a PixelCNN to model the prior. To perform the training of the PixelCNN, we turn the samples x in their latent representations z, and train the PixelCNN over the latent representations z. This way, we have created dependency between the z in the latent feature mapping, and we obtain a prior over their distribution modeled by the PixelCNN.

4 Microbiota analysis

This section aims at using the previous elements to design microbiota adapted latent methods.

4.1 Microbiota dataset

As we look into the microbiota data, we notice a major phylogenetic architecture to describe various levels of precision in the microbiota composition. Indeed, such phylogenetic structure can be represented as a tree as on the following figure:

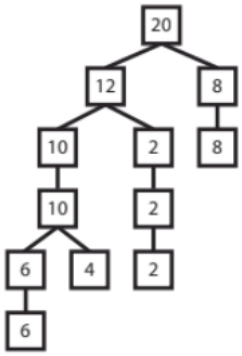

Figure 8: Phylogenetic tree example with abundance data (in the nodes) at each layer of the tree. Each node represents a bacterium specie at a given precision layer in the tree. From [2].

Such structure can not be used directly in a machine learning system since it's not a vectorizable representation. Hence, we first suggest to transform the tree in a matrix to image structure as in the following example:

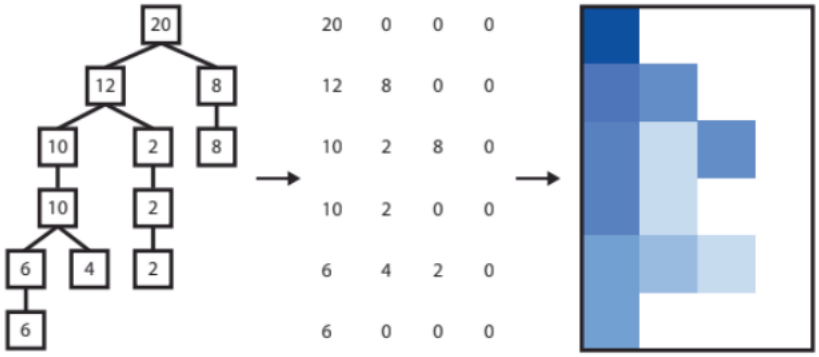

Figure 9: Phylogenetic tree to image representation: opacity of the pixel relates to the abundance of the species at the given level of precision (normalized between 0 and 1). From [2].

To give some notations, we introduce the following framework:

- The maximum depth of a phylogenetic tree is D, the maximum number of nodes N_T, the maximum amount of unique species at any level is U.
- X_i is the abundance matrix of the individual i (image of size $D \times U$) , which is observed.

- T_i is the adjacent matrix of the phylogenetic tree of individual i, of size $N_T \times N_T$, which is observed. We could use any other encoding of a tree (contour function, depth function, ...).
- For a given tree, we will denote by U_l the number of species at a given precision layer l of the tree.
- We assume that we have n individuals observed through $(X_i, T_i)_{1 \leq i \leq n}$ i.i.d samples.

4.2 Simple generative model: no latent variable

4.2.1 Context and objective

In first approximation, we would like to define a generative model that does not exploit any latent structure. Such model, parameterized by θ, aims at finding an optimal distribution in the sense of the maximum of likelihood, within a family of distributions yet to be defined. The maximum likelihood objective is given below as:

$$\theta^* = arg\max_{\theta} p_\theta(X, T)$$

One can rewrite the joint distribution as follow:

$$\begin{aligned} p_\theta(X, T) &= \prod_{i=1}^{n} p_\theta(X_i, T_i) \\ &= \prod_{i=1}^{n} p_\theta(T_i) p_\theta(X_i | T_i) \end{aligned}$$

As a result, to compute this objective, we need to define a prior $p_\theta(T_i)$ that generates trees, and a posterior distribution $p_\theta(X_i|T_i)$ that generates abundance data from a sampled tree.

Note that in this section we assume that the trees do not have missing entries, meaning all the branches of reach the maximum precision layer of the tree, so that the abundance data sum to 1 at each level.

4.2.2 Design of the prior

We aim at defining a parameterized distribution $p_\theta(T)$ from which one can sample trees. We would also like this distribution to model the trees of the microbiota dataset, meaning that the generated trees should look like the ones from the dataset as well, and respect the phylogenetic constraints.

Consequently, we introduce a first simple generative process to characterize $p_\theta(T)$ that we would call the markovian parenting tree generator. Before describing the generation method, let us introduce the framework of it:

- Describe T as a succession of L layers: $T = (T^{(1)}, \ldots, T^{(L)})$. We assume that we have no missing data, so to say, all the leaves the tree are reaching the precision layer L.

- Describe a given layer l as a discrete vector in $\{0,1\}^U$. To each possible node at layer l we can associate an index k so that we denote the nodes by $u_k^{(l)} \in \{0,1\}$. A node $u_k^{(l)}$ is activated if it is valued as 1 in $T^{(l)}$, otherwise it is not.
- We introduce the function $\mathcal{P}$ that takes a node $u_k^{(l)}$ as input, and output the parent of the node if it is well defined in T, otherwise it outputs 0 as if the parent was not activated:

$$\mathcal{P}(u_k^{(l)}) = \begin{cases} 1 & \text{if parent of } u_k^{(l)} \text{ exists and is activated} \\ 0 & \text{otherwise} \end{cases}$$

Now that the framework is clear and defined, we describe the generative process:

- The root node of the tree is deterministic, since all trees begin to the same root ancestor. Hence, we have:

$$p(T^{(1)}) = \mathbb{1}_{T^{(1)}=e_1}$$

- For all $l \geq 2, k \in \{1,\dots,U\}$, we assume that the activation of the node is distributed as Bernoulli conditionally to its parent activation, parameterized by $\pi_k^{(l)}$:

$$u_k^{(l)} | \left\{ \mathcal{P}(u_k^{(l)}) = 1 \right\} \sim \mathcal{B}(\pi_k^{(l)})$$

To respect the tree architecture, we assume that if $\mathcal{P}(u_k^{(l)}) = 0$ then the probability for the children to be activated is deterministic and set to 0, as a child can not exist without his parent.

- For now, we make the major assumption that all nodes are independent within a given layer conditionally to their parents, and they only depend on their respective parent, so that:

$$p\left(u_1^{(l)},\dots,u_K^{(l)} | \mathcal{P}(u_1^{(l)}),\dots,\mathcal{P}(u_K^{(l)})\right) = \prod_{k=1}^{K} p\left(u_k^{(l)} | \mathcal{P}(u_k^{(l)})\right)$$

- The dependency between the layers of the tree is markovian:

$$p_\theta(T^{(l+1)} | T^{(1:l)}) = p_\theta(T^{(l+1)} | T^{(l)})$$

Noting these framework properties, we can describe the distribution of such prior model on the trees:

$$\begin{aligned}
p_\theta(T) &= p_\theta\left(T^{(1)},\dots,T^{(L)}\right) \\
&= p\left(T^{(1)}\right) \prod_{l=1}^{L-1} p_\theta\left(T^{(l+1)} | T^{(l)}\right) \\
&= \mathbb{1}_{T^{(1)}=e_1} \prod_{l=1}^{L-1} p\left(u_1^{(l+1)},\dots,u_K^{(l+1)} | \mathcal{P}(u_1^{(l+1)}),\dots,\mathcal{P}(u_K^{(l+1)})\right) \\
&= \mathbb{1}_{T^{(1)}=e_1} \prod_{l=1}^{L-1} \prod_{k=1}^{K} p\left(u_k^{(l+1)} | \mathcal{P}(u_k^{(l+1)})\right) \\
&= \mathbb{1}_{T^{(1)}=e_1} \prod_{l=1}^{L-1} \prod_{k=1}^{K} \underbrace{\mathcal{P}(u_k^{(l+1)})}_{\text{1 if node has parent}} \left(\underbrace{u_k^{(l+1)}}_{\text{1 if activated}} \pi_k^{(l+1)} + (1-u_k^{(l+1)})(1-\pi_k^{(l+1)}) \right)
\end{aligned}$$

Looking at the previous formula, we obtain that such prior is parameterized by the activation probabilities $\pi_k^{(l)}$ of each node $u_k^{(l)}$. Furthermore, due to the indicator function expressed through $\mathcal{P}(u_k^{(l+1)})$, the update of a given activation probability will only be impacted by the trees which have the node $u_k^{(l)}$ in any branch.

Now that the prior of the trees is well defined, we would like to compute an optimal value of $\pi^{(l)} = \left(\pi_1^{(l)}, \dots, \pi_U^{(l)}\right)$ in the sense of the maximum of likelihood.
The maximum log-likelihood objective for $\pi_k^{(l)}$ over our microbiota dataset can then be written as:

$$\begin{aligned}\left(\pi_j^{(m)}\right)^* = arg\max_{\pi_j^{(m)}} \quad & \sum_{i=1}^{n} \mathbb{1}_{T_i^{(1)}=e_1} \sum_{l=1}^{L-1}\sum_{k=1}^{K} \mathcal{P}(u_{k,i}^{(l+1)}) \left[u_{k,i}^{(l+1)} \log \pi_k^{(l+1)} + (1-u_{k,i}^{(l+1)}) \log(1-\pi_k^{(l+1)})\right] \\ \text{s.t.} \quad & \forall k, \pi_k^{(L)} \in [0,1]\end{aligned} \tag{1}$$

Then, simply by deriving the log-likelihood we obtain:

$$\begin{aligned}\partial_{\pi_j^{(m)}} \log p(T_1, \dots, T_n) &= \sum_{i=1}^{n} \mathbb{1}_{T_i^{(1)}=e_1} \mathcal{P}(u_{j,i}^{(m)}) \left[u_{j,i}^{(m)} \frac{1}{\pi_j^{(m)}} + (1-u_{j,i}^{(m)}) \frac{-1}{1-\pi_j^{(m)}}\right] \\ &= \frac{1}{\pi_j^{(m)}} \sum_{i=1}^{n} \mathbb{1}_{T_i^{(1)}=e_1} \mathcal{P}(u_{j,i}^{(m)}) u_{j,i}^{(m)} - \frac{1}{1-\pi_j^{(m)}} \sum_{i=1}^{n} \mathbb{1}_{T_i^{(1)}=e_1} \mathcal{P}(u_{j,i}^{(m)})(1-u_{j,i}^{(m)})\end{aligned}$$

Looking for 0 valued gradient, we end up with:

$$\begin{aligned}(1-\pi_j^{(m)}) \sum_{i=1}^{n} \mathbb{1}_{T_i^{(1)}=e_1} \mathcal{P}(u_{j,i}^{(m)}) u_{j,i}^{(m)} - \pi_j^{(m)} \sum_{i=1}^{n} \mathbb{1}_{T_i^{(1)}=e_1} \mathcal{P}(u_{j,i}^{(m)})(1-u_{j,i}^{(m)}) &= 0 \\ \pi_j^{(m)} \sum_{i=1}^{n} \mathbb{1}_{T_i^{(1)}=e_1} \mathcal{P}(u_{j,i}^{(m)}) &= \sum_{i=1}^{n} \mathbb{1}_{T_i^{(1)}=e_1} \mathcal{P}(u_{j,i}^{(m)}) u_{j,i}^{(m)} \\ \pi_j^{(m)} &= \frac{\sum_{i=1}^{n} \mathbb{1}_{T_i^{(1)}=e_1} \mathcal{P}(u_{j,i}^{(m)}) u_{j,i}^{(m)}}{\sum_{i=1}^{n} \mathbb{1}_{T_i^{(1)}=e_1} \mathcal{P}(u_{j,i}^{(m)})}\end{aligned}$$

Since the previous quantity respects the constraint to be in $[0,1]$, we obtain the optimal activation probability for every bacteria as:

$$\boxed{\left(\pi_k^{(l)}\right)^* = \frac{\sum_{i=1}^{n} \mathbb{1}_{T_i^{(1)}=e_1} \mathcal{P}(u_{k,i}^{(l)}) u_{k,i}^{(l)}}{\sum_{i=1}^{n} \mathbb{1}_{T_i^{(1)}=e_1} \mathcal{P}(u_{k,i}^{(l)})}}$$

This estimator is actually the common MLE for a Bernoulli parameter estimation, except that it limits the computation of the estimation to all trees that respect the root constraint, and that could possess the node $u_k^{(l)}$ since they must have the parent node $\mathcal{P}(u_k^{(l)})$.

4.2.3 Design of the posterior and maximum likelihood estimator

Now that we have a way to generate trees, we need to define an explicit stochastic relationship between the abundance data and the structure of the tree. Such inevitably exists, as when observing T, the abundance of an entity that isn't present in T is necessarily 0. Similarly, it is highly likely that when observing the presence of certain entities in T that induces a high abundance of another neighbour entity (interaction between bacteria).

For context, we recall that X is a matrix of shape (D, U). We denote by $X^{(l)}$ the l-th line of the abundance matrix, for which up to U_l elements should be non-zero.

Since we would like an explicit model here, we design a posterior $p_\theta(X|T)$ which is markovian relatively to the layers of the tree:

$$p_\theta(X^{(l)}|X^{(1:l-1)}, T) = p_\theta(X^{(l)}|X^{(l-1)}, T)$$

We assume the following framework:

- We denote $X^{(l)} = (x_l^{(1)}, \dots, x_l^{(U)})$ the abundance vector at layer l.
- $X^{(1)} = [1, 0, \dots, 0]$, since it's the root of the tree, only one entity gets the whole weight.
- We assume that for all $l \geq 2, X^{(l)}|T \sim \mathcal{D}(\alpha_l)$.
- Each value in $X^{(l)}$ is restricted by the following set of constraints:
  - If node k at layer $l-1$ has one child, then its abundance value is the same for the child node.
  - If node k at layer $l-1$ has at least two children, the children abundance sums to the parent's abundance value.

  We denote by $\mathcal{C}_T$ the set of vectors that verify the previous constraints for the tree T. Hence, we obtain the following conditional distribution:

$$\begin{aligned} p_{\gamma_l}(X^{(l+1)}|X^{(l)}, T) &= \frac{p_{\gamma_l'}(X^{(l+1)}, X^{(l)}, T)}{p_{\alpha_l}(X^{(l)}, T)} \\ &= \frac{p_{\alpha_{l+1}}(X^{(l+1)}|T)}{p_{\alpha_l}(X^{(l)}|T)} p(X^{(l)}|X^{(l+1)}, T) \\ &= \frac{p_{\alpha_{l+1}}(X^{(l+1)}|T)}{p_{\alpha_l}(X^{(l)}|T)} \mathbb{1}_{\mathcal{C}_T}\left(X^{(l)}, X^{(l+1)}\right) \end{aligned}$$

The whole posterior is then given by:

$$\begin{aligned}
p_\theta(X|T) &= \prod_{i=1}^{n} p_\theta(X_i|T_i) \\
&= \prod_{i=1}^{n} p_\theta(X_i^{(1)}, \dots, X_i^{(D)}|T_i) \\
&= \prod_{i=1}^{n} p(X_i^{(1)}|T_i) \prod_{l=1}^{D-1} p_{\gamma_j}(X_i^{(l+1)}|X_i^{(l)}, T_i) \\
&= \prod_{i=1}^{n} \mathbb{1}_{X_i^{(1)}=e_1} \prod_{l=1}^{D-1} \frac{p_{\alpha_{l+1}}(X_i^{(l+1)}|T_i)}{p_{\alpha_l}(X_i^{(l)}|T_i)} \mathbb{1}_{\mathcal{C}_{T_i}} \left(X_i^{(l)}, X_i^{(l+1)}\right) \\
&= \prod_{i=1}^{n} p_{\alpha_D}(X_i^{(D)}|T_i) \mathbb{1}_{X_i^{(1)}=e_1} \prod_{l=1}^{D-1} \mathbb{1}_{C_{T_i}}(X_i^{(l)} X_i^{(l+1)})
\end{aligned}$$

Hence, to generate an abundance matrix X out of a tree T, we just need to sample the highest precision layer of the tree and roll it up to the top thanks to the tree structure that we observe. As a result, this posterior is only characterized by one parameter α_D that is the dirichlet parameter of the ultimate layer of the tree. Sampling from the last layer D has to be made explicit though. We assume that the abundance at the last layer only depends on the last layer of the tree, so that:

$$p_{\alpha_D}(X_i^{(D)}|T_i) = p_{\alpha_D}(X_i^{(D)}|T_i^{(D)})$$

Knowing the last layer structure $T_i^{(D)}$ enables us to tell for sure where some 0 abundances are going to be. Recall $\mathcal{L}_{T_i}^{(D)}$, the set of indexes so that the node associated to that index belongs to any branch at the level set $T_i^{(D)}$ in T_i:

$$\mathcal{L}_{T_i}^{(D)} = \left\{ k \in \{1, \dots, U\} | u_k^{(D)} \in T_i^{(D)} \right\}$$

We can now describe the last layer's abundance conditionally to the tree:

$$
\begin{aligned}
p(X_i^{(D)}|T_i) &= p(X_i^{(D)}|T_i^{(D)}) \\
&= p\left(\left[X_i^{(D)}\right]_1, \dots, \left[X_i^{(D)}\right]_U | \bigcap_{k' \notin \mathcal{L}_{T_i}^{(D)}} \left[X_i^{(D)}\right]_{k'} = 0\right) \\
&= \frac{p_{\alpha_D}\left(\bigcap_{k \in \mathcal{L}_{T_i}^{(D)}} \left[X_i^{(D)}\right]_k, \bigcap_{k' \notin \mathcal{L}_{T_i}^{(D)}} \left[X_i^{(D)}\right]_{k'} = 0\right)}{\int p_{\alpha_D}\left(\bigcap_{k \in \mathcal{L}_{T_i}^{(D)}} \left[\tilde{X}_i^{(D)}\right]_k, \bigcap_{k' \notin \mathcal{L}_{T_i}^{(D)}} \left[X_i^{(D)}\right]_{k'} = 0\right) d\tilde{X}} \\
&= \frac{\frac{1}{B(\alpha_D)} \prod_{k \in \mathcal{L}_{T_i}^{(D)}} \left[X_i^{(D)}\right]_k^{\alpha_D^{(k)}-1} \prod_{k' \notin \mathcal{L}_{T_i}^{(D)}} \left[X_i^{(D)}\right]_{k'}^{\alpha_D^{(k')}-1}}{\frac{1}{B(\alpha_D)} \prod_{k' \notin \mathcal{L}_{T_i}^{(D)}} \left[X_i^{(D)}\right]_{k'}^{\alpha_D^{(k')}-1} \prod_{k \in \mathcal{L}_{T_i}^{(D)}} \int_0^1 \left[\tilde{X}_i^{(D)}\right]_k^{\alpha_D^{(k)}-1} d\left[\tilde{X}_i^{(D)}\right]_k} \\
&= \prod_{k \in \mathcal{L}_{T_i}^{(D)}} \alpha_D^{(k)} \left[X_i^{(D)}\right]_k^{\alpha_D^{(k)}-1}
\end{aligned}
$$

Plugin that result into the whole distribution, we obtain that final formulation of the posterior distribution:

$$
p_\theta(X|T) = \prod_{i=1}^{n} \prod_{k \in \mathcal{L}_{T_i}^{(D)}} \alpha_D^{(k)} \left[X_i^{(D)}\right]_k^{\alpha_D^{(k)}-1} \mathbb{1}_{X_i^{(1)}=e_1} \prod_{l=1}^{D-1} \mathbb{1}_{C_{T_i}}(X_i^{(l)}, X_i^{(l+1)})
$$

Now that we have described the posterior distribution, we would like to compute an optimal parameter for such distributions under a maximum likelihood objective, that is given by $\alpha_D^* = (\alpha_D^{(1)}, \dots, \alpha_D^{(U)})$. The optimization objective for each component is the following:

$$
(\alpha_D^{(j)})^* = arg\max_{\alpha_D^{(k)}} \sum_{i=1}^{n} \log p_{\alpha_D}(X_i^{(D)}|T_i) + \log \mathbb{1}_{X_i^{(1)}=e_1} + \sum_{k=1}^{D-1} \log \mathbb{1}_{C_{T_i}}(X_i^{(l)}, X_i^{(l+1)})
$$

Since the indicator function do not depend on the parameter, we only have to solve the following problem:

$$
(\alpha_D^{(j)})^* = arg\max_{\alpha_D^{(k)}} \sum_{i=1}^{n} \sum_{k \in \mathcal{L}_{T_i}^{(D)}} \log \alpha_D^{(k)} + \left(\alpha_D^{(k)} - 1\right) \log \left[X_i^{(D)}\right]_k
$$

Deriving with respect to $\alpha_D^{(j)}$, if any tree has the node j, we get:

$$
\boxed{(\alpha_D^{(j)})^* = \frac{\sum_{i=1}^{n} \mathbb{1}_{j \in \mathcal{L}_{T_i}^{(D)}}}{\sum_{i=1}^{n} \mathbb{1}_{j \in \mathcal{L}_{T_i}^{(D)}} \log \frac{1}{\left[X_i^{(D)}\right]_j}}}
$$

4.2.4 Optimization of the objective

Recall the optimization objective, written under the maximum of the log-likelihood:

$$\begin{aligned}\theta^* &= arg\max_\theta \log p_\theta(X, T) \\ &= arg\max_\theta \sum_{i=1}^{n} \log p_\theta(X_i, T_i) \\ &= arg\max_\theta \sum_{i=1}^{n} \log p_\theta(T_i) + \log p_\theta(X_i|T_i)\end{aligned}$$

Notice that the first term of the loss is made only of the prior, while the second one is made only of the posterior. Since these two distributions do not share common parameters in θ, we can proceed to optimize the parameters of each distribution without taking care of the other term.

Hence, the optimal θ^* is given by the concatenation of the MLE from the prior on the trees and the posterior of the abundance knowning the trees, which we both have computed in the previous sections.

4.2.5 Experiments

We have implemented the previous modelization and tested it on a large microbiota dataset.

4.2.6 Conclusions

This first model is interesting as it provides a benchmark baseline for our upcoming tree-structured models. However, it clearly lacks of complexity:

- The nodes are modeled as independent, which prevents any modelisation of correlation between entities. As for many ecosystems, we would expect some bacteria to have symbiotic relationship, or domination roles, especially when we come in critical systems like disease detection in which some bacteria may proliferate over others. One idea could be to model an interaction graph (see [5]) and use it as a correlation restriction between our bacteria.

- The abundance data generation takes the tree constraints into account, yet it is poorly flexible as we sample from a unique Dirichlet parameterized by α_D and masked according to $T^{(D)}$, which completely omits the tree's global structure. It would be highly interesting to try to group the trees through a latent variable for instance, and adapt the parameter to that specific group to enhance the modelization. We will explore that situation in the next section.

- So far we have only considered trees without any missing entries at precision level L, which is not what we observe for microbiota datasets. Inference for missing data implementation could be interesting in the future.

5 Bibliography

References

[1] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum Likelihood from Incomplete Data via the EM Algorithm". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 39.1 (1977), pp. 1–38. ISSN: 00359246. URL: http://www.jstor.org/stable/2984875 (visited on 05/06/2023).

[2] Ricardo Hernández Medina et al. "Machine learning and deep learning applications in microbiome research". In: *ISME Communications* 2.1 (2022), p. 98.

[3] Diederik P Kingma, Max Welling, et al. "An introduction to variational autoencoders". In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392.

[4] Thomas P. Minka. "Estimating a Dirichlet distribution". In: 2012.

[5] Raphaëlle Momal, Stéphane Robin, and Christophe Ambroise. "Tree-based Inference of Species Interaction Network from Abundance Data". In: *arXiv preprint arXiv:1905.02452* (2019).

[6] Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. "Pixel Recurrent Neural Networks". In: *CoRR* abs/1601.06759 (2016). arXiv: 1601.06759. URL: http://arxiv.org/abs/1601.06759.

[7] Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. "Neural Discrete Representation Learning". In: *CoRR* abs/1711.00937 (2017). arXiv: 1711.00937. URL: http://arxiv.org/abs/1711.00937.