# Assigment 3. Advanced machine learning.

Author:
Rodrigo Castellano Ontiveros
roco@kth.se

## 3.1 Success probability in the Johnson-Lindenstrauss lemma

In the proof of Johnson-Lindenstrauss lemma we first bounded the probability that a single projection maintains all pairwise distances with distortion that is between $(1\epsilon)$ and $(1+\epsilon)$. In particular, we showed that the probability of achieving such a distortion for all pairs of points is at least 1/n.

Assume now, that we want to boost the probability of success to be at least 95%.

An independent trial here refers to generating a new projection of the data points with a newlygenerated projection matrix.

**Question 3.1.1: Show that O(n) independent trials are sufficient for the probability of success to be at least 95%.**

According to the statement, we are trying to boost the probability of success to be at least 95%, taking into account that the probability of achieving the mentioned distortion is at least $\frac{1}{n}$.

If the probability of success is $\frac{1}{n}$, the probability not to success is $(1-\frac{1}{n})$. For $n$ trials, we get that the probability to not success is

$$(1-1/n)^n$$

.

To bound the probability we can use the inequality [1]

$$(1-1/k)^k \leq 1/e$$

and for $k = 1/n$, the expression below can be seen as the probability not to success for $n$ trials, bounded by $1/e$

$$\begin{aligned} &\left(1-\frac{1}{n}\right)^n \leq \frac{1}{e} \\ \Leftrightarrow &\left(1-\frac{1}{n}\right)^{n\ln 20} \leq \frac{1}{e}^{\ln 20} \\ \Leftrightarrow &\left(1-\frac{1}{n}\right)^{n\ln 20} \leq 0.05 \end{aligned} \tag{1}$$

Therefore it is shown that for $n \ln 20 \sim \mathcal{O}(n)$ trials, the probability not to success is smaller than 5%, and therefore the probability to success is at least 95%.

## 3.2 Node similarity for representation learning

Let G = (V,E) be an undirected and connected graph and let A be the adjacency matrix of G, that is, $A_{ij} = 1$ if $(i, j) \in E$ and $A_{ij} = 0$ otherwise. Let $D$ be a diagonal matrix with $D_{ii} = \sum_{k=1}^{\infty} A_{ij}$, and let $P = D^{-1}A$.

In graph representation learning, our goal is to learn vector representations (embeddings) for the nodes of the graph. The main idea is to define an appropriate similarity measure between the graph nodes, and then learn vector representations for the graph nodes, so that the similarity between pairs of learned vectors approximates the similarity between the corresponding graph nodes. Assume now that for a similarity measure between graph nodes, we define

$$S_{ij} = \sum_{k=1}^{\infty} \alpha^k [P^k]_{ij}$$

for each pair of nodes $i, j \in V$ , and for some real $0 < \alpha < 1$. Here, by $[P^k]_{ij}$ we refer to the $(i, j)$ entry of the matrix $P^k$.

**Question 3.2.2: Explain the intuition for the definition of the similarity measure S.**

The similarity matrix, also called Katz index, is a global overlap statistic. There are usually local and global approaches, and the problem with the local ones is that they consider only local nodes. Although two nodes could have no local overlap in their neighbours, they could still be members of the same community, and this is taken into account in global overlap statistics.

The main idea of the Katz index is to describe the similarity between nodes $u, v$ by counting the number of paths of all lengths between the pair of nodes. We can decide whether to give more or less weight to the long paths with $\beta$ [2].

**Question 3.2.3: Show that S can be computed efficiently using matrix addition, and a single matrix inversion operation, while avoiding computing an infinite series.**

In the book [2], the following theorem is shown

**Theorem 1**. Let $X$ be a real-valued square matrix and $\lambda_1$ its largest eigenvalue. Then

$$(I - X)^{-1} = \sum_{i=0}^{\infty} X^i$$

if and only if $(I - X)$ is non-singular and $\lambda_1 < 1$.

**Proof**. Let $s_n = \sum_{i=0}^{n} X^i$ Then,

$$\begin{aligned} X s_n &= X \sum_{i=0}^{n} X^i \\ &= \sum_{i=1}^{n+1} X^i \end{aligned}$$

and

$$\begin{aligned} s_n - Xs_n &= \sum_{i=0}^{n} X^i - \sum_{i=1}^{n+1} X^i \\ s_n(I - X) &= I - X^{n+1} \\ s_n &= \left(I - X^{n+1}\right) (\{I - X)^{-1} \end{aligned}$$

If $\lambda_1 < 1$ we have that $\lim_{n\to\infty} X^n = 0$, thus

$$\begin{aligned} \lim_{n\to\infty} s_n &= \lim_{n\to\infty} \left(I - X^{n+1}\right) (I - X)^{-1} \\ &= I(I - X)^{-1} \\ &= (I - X)^{-1} \end{aligned}$$

Taking into account the solution of **Theorem 1** and using $X = \alpha P_{ij}$

$$(I - \alpha P_{ij})^{-1} = \sum_{k=0}^{\infty} \alpha^k [P^k]_{ij} = \sum_{k=1}^{\infty} \alpha^k [P^k]_{ij} + I = S_{ij} + I \tag{2}$$

Therefore, we obtain

$$S_{ij} = (I - \alpha P_{ij})^{-1} - I \tag{3}$$

The computation, as mentioned in the exercise, is done by using matrix addition and a single matrix inversion operation.

## 3.3 Complicated likelihood for leaky units on a tree

Consider the following model. A binary tree T has random variables associated with its vertices. A vertex u has an observable variable $X_u$ and a latent class variable $Z_u$. Each class $c \in [C]$ has a Normal distribution $N(\mu_c, \sigma^2)$. If the three neighbors of u are v1, v2, and v3, then

$$p\left(X_u \mid Z_u = c, Z_{v_1} = c_1, Z_{\sigma_2} = c_2, Z_{v_3} = c_3\right) \sim \mathcal{N}\left(X_u \mid (1-\alpha)\mu_c + \sum_{i=1}^{3} \frac{\alpha}{3} \mu_{c_i}, \sigma^2\right) \tag{4}$$

The class variables are i.i.d., each follows the categorical distribution $\pi$.

Note: For root vertex you can assign the weights as $(1 - \alpha)$ and $(\alpha/2)$, and for a leaf vertex $(1 - \alpha)$ and $\alpha$.

**Question 3.3.4: Provide a linear time algorithm that computes $p(X|T, M, \sigma, \alpha, \pi)$ when given a tree T (with vertices $V(T)$), observable variables for its vertices $X = X_v : v \in V(T)$, and parameters $M = \{\mu_c : c \in [C]\}, \sigma, \alpha$.**

In contrast to the previous assignment, there are hidden variables and we have to marginalize over them. Then, the main point is to split the observations below a node, and to condition on the hidden variables so that we can have the probabilities of independent variables multiplied. Now, the observed variables are independent of their neighbours' observable variables given their hidden variables.

$$P(X \mid T, M, \sigma, \alpha, \pi) = \sum_Z P(X, Z \mid T, M, \sigma, \alpha, \pi) \tag{5}$$

Take into account that now the observations might not be in the leafs, so we denote $X_{\downarrow u}$ as the observable variables below a node u (instead of $X_{o \wedge \downarrow u}$), and the same applies to the hidden variables $Z_{\downarrow u}$. Having the observed values in a set of nodes instead of the leafs makes it a bit more complicated, but it can still be calculated.

For a root node r with children v,w, it is shown how to calculate the probability of the observations.

Therefore,

$$P(X) = \sum_Z P(X, Z) = \sum_Z P(X_r, X_v, X_w, X_{\downarrow v}, X_{\downarrow w}, Z_r, Z_v, Z_w, Z_{\downarrow v}, Z_{\downarrow w}) \tag{6}$$

In the expression above, X and Z have been decomposed. The next step is to marginalize over $Z_r, Z_v, Z_w$ due to (4). After that, it is needed group terms based on different nodes (independent variables) and apply conditional independence.

$$\begin{aligned}
&\sum_Z P(X_r, X_v, X_w, X_{\downarrow v}, X_{\downarrow w}, Z_r, Z_v, Z_w, Z_{\downarrow v}, Z_{\downarrow w}) \\
&= \sum_Z P(X_r, X_v, X_w, X_{\downarrow v}, X_{\downarrow w}, Z_{\downarrow v}, Z_{\downarrow w} \mid Z_r, Z_v, Z_w) P(Z_r, Z_v, Z_w) \\
&= \sum_Z P(X_r \mid Z_r, Z_v, Z_w) P(X_v, X_{\downarrow v}, Z_{\downarrow v} \mid Z_r, Z_v, Z_w) P(X_w, X_{\downarrow w}, Z_{\downarrow w} \mid Z_r, Z_v, Z_w) P(Z_r, Z_v, Z_w) \\
&= \sum_Z P(X_r \mid Z_r, Z_v, Z_w) P(X_v, X_{\downarrow v}, Z_{\downarrow v} \mid Z_r, Z_v) P(X_w, X_{\downarrow w}, Z_{\downarrow w} \mid Z_r, Z_w) P(Z_r) P(Z_v) P(Z_w) \\
&= \sum_Z \mathcal{N}\left(X_r \mid (1-\alpha)\mu_{c_{(Z_r)}} + \frac{\alpha}{2}(\mu_{c_{1(Z_v)}} + \mu_{c_{2(Z_w)}}), \sigma^2\right) P(Z_r) P(Z_v) P(Z_w) s(w) s(v)
\end{aligned} \tag{7}$$

It's written $\sum_Z$ to simplify notation, but Z should be splitted into all the variables it has been decomposed previously. Furthermore, we have defined, for a node u,

$$s(u) = \sum_Z P(X_u, X_{\downarrow u}, Z_{\downarrow u} \mid Z_{pa(u)}, Z_u) \tag{8}$$

By calculating $s(u)$, a linear time DP algorithm is found, by going from the root to the leafs nodes recursively (with s(node)), breaking the problem into smaller pieces.

For a node u with parent $pa(u)$ and children v,w, the calculations are very similar. As before, the observed and hidden variables are decomposed, then we condition on $Z_v, Z_w$. Finally, split probabilities based on independence, and remove conditional independence.

$$
\begin{aligned}
s(u) &= \sum_Z P(X_u, X_{\downarrow u}, Z_{\downarrow u} \mid Z_{pa(u)}, Z_u) = \sum_Z P(X_u, X_v, X_w, X_{\downarrow v}, X_{\downarrow w}, Z_v, Z_w, Z_{\downarrow v}, Z_{\downarrow w} \mid Z_{pa(u)}, Z_u) \\
&= \sum_Z P(X_u, X_v, X_w, X_{\downarrow v}, X_{\downarrow w}, Z_{\downarrow v}, Z_{\downarrow w} \mid Z_v, Z_w, Z_{pa(u)}, Z_u) P(Z_v, Z_w) \\
&= \sum_Z P(X_u \mid Z_v, Z_w, Z_{pa(u)}, Z_u) P(X_v, X_{\downarrow v}, Z_{\downarrow v} \mid Z_v, Z_w, Z_{pa(u)}, Z_u) P(X_w, X_{\downarrow w}, Z_{\downarrow w} \mid Z_v, Z_w, Z_{pa(u)}, Z_u) \\
& P(Z_v, Z_w) \\
&= \sum_Z P(X_u \mid Z_v, Z_w, Z_{pa(u)}, Z_u) P(X_v, X_{\downarrow v}, Z_{\downarrow v} \mid Z_v, Z_u) P(X_w, X_{\downarrow w}, Z_{\downarrow w} \mid Z_w, Z_u) P(Z_v) P(Z_w) \\
&= \sum_Z \mathcal{N}\left(X_r \mid (1-\alpha)\mu_{c_{(Z_u)}} + \frac{\alpha}{2}(\mu_{c_{1(Z_{pa(u)})}} + \mu_{c_{2(Z_v)}} + \mu_{c_{3(Z_w)}}), \sigma^2\right) P(Z_v) P(Z_w) s(w) s(v)
\end{aligned} \tag{9}
$$

Thanks to splitting the problem in this way, the algorithm becomes linear in the number of vertices.

When the node is a leaf, the recursion finishes and for a leaf node l, $s(l)$ is calculated as

$$
\begin{aligned}
s(l) &= \sum_Z P(X_l, X_{\downarrow l}, Z_{\downarrow l} \mid Z_{pa(l)}, Z_l) = \sum_Z P(X_l \mid Z_{pa(l)}, Z_l) \\
&= \sum_Z \mathcal{N}\left(X_l \mid (1-\alpha)\mu_{c_{(Z_l)}} + \alpha(\mu_{c_{1(Z_{pa(l)})}})\right)
\end{aligned} \tag{10}
$$

[3]. It has been shown, by doing DP, that it is possible to compute the likelihood in linear time, by being the algorithm linear in the number of vertices $N = |V(T)|$.

## 3.4 Super Epicentra - Variational Inference

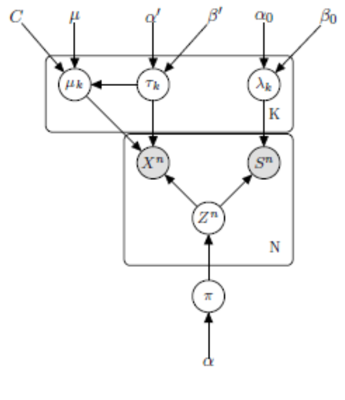


Figure 1: Figure 1: The K super epicentra model with priors.

As in Task 2.4 from Assignment 2, we have seismographic from an area with frequent earthquakes emanating from K super epicentra. In fact, the core of the present model is the model described in Task 2.4 from Assignment 2, but now the parameters also have conjugate prior distributions. As shown in Figure 1, the present model has the following prior distributions.

1. $\pi$ has a $\mathrm{Dir}(\alpha)$ prior.
2. $\tau_{k,i}$ has a $\mathrm{Ga}\left(\alpha', \beta'\right)$ prior.
3. $\mu_{k,i}$ has a $\mathcal{N}\left(\mu, \left(C\tau_{k,i}\right)^{-1}\right)$ prior.
4. $\lambda_k$ has a $\mathrm{Ga}\left(\alpha_0, \beta_0\right)$ prior.

**Question 3.4.5: Derive a VI algorithm that estimates the posterior distribution for this model.**

As shown in [3, 4], the VI algorithm is used to find the posterior distribution for the unknown model parameters as well as the hidden variables. The approximation is

$$p(Z, \theta \mid X) \approx q(Z, \theta) \tag{11}$$

We try to minimize the KL divergence between p and q, which measures their closeness. This is equivalent to maximizing the $ELBO(q) = E_\Psi[\log p(X, \Psi)] - E_\Psi[\log q(\Psi)]$. By using mean field theory, we assume factorization, i.e.,

$$q\left(Z_1, \ldots, Z_n, \theta_1, \ldots, \theta_K\right) = \prod_i q\left(Z_i\right) \prod_k q\left(\theta_k\right) \tag{12}$$

Then, each factor is updated in every iteration by performing the expectation as

$$\ln q_j^\star\left(Z_j\right) = \mathbb{E}_{i \neq j}[\ln p(X, S, Z, \theta)] + \text{ const.} \tag{13}$$

For the exercise, the distributions are the following:

$$\begin{aligned}
p(Z \mid \pi) &= \prod_{n=1}^N \prod_{k=1}^K \pi_k^{Z_{nk}} \\
p(X \mid Z, \mu, \tau) &= \prod_{n=1}^N \prod_{k=1}^K \mathcal{N}(X_n \mid \mu_k, (\tau_k)^{-1}))^{Z_{nk}} = \prod_{n=1}^N \prod_{k=1}^K \left(\frac{\tau_k}{2\pi}\right)^{\frac{1}{2}} e^{-\frac{\tau_k}{2}(x_n-\mu_k)^2} \\
p(S \mid Z, \lambda) &= \prod_{n=1}^N \prod_{k=1}^K P(S_n \mid \lambda_k)^{Z_{nk}} = \prod_{n=1}^N \prod_{k=1}^K \frac{\lambda_k^{S_n} e^{-\lambda_k}}{S_n!}
\end{aligned} \tag{14}$$

The priors are defined as

$$\begin{aligned}
p(\pi) &\sim Dir(\pi \mid \alpha) = f(\alpha) \prod_{k=1}^K \pi_k^{\alpha-1} \\
p(\mu_k \mid \tau_k) &\sim \mathcal{N}\left(\mu_k \mid \mu, (C\tau_k)^{-1}\right) = \left(\frac{C\tau_k}{2\pi}\right)^{\frac{1}{2}} e^{-\frac{C\tau_k}{2}(\mu_k-\mu)^2} \\
p(\tau_k) &\sim Ga(\alpha', \beta') = \frac{(\beta')^{\alpha'}}{\Gamma(\alpha')} \tau_k^{\alpha'-1} e^{-\beta'\tau_k} \\
p(\lambda_k) &\sim Ga(\alpha_0, \beta_0) = \frac{(\beta_0)^{\alpha_0}}{\Gamma(\alpha_0)} \tau_k^{\alpha_0-1} e^{-\beta_0\tau_k}
\end{aligned} \tag{15}$$

In order to compute (13), the joint distribution has to be calculated. Looking at the graphical model,

$$p(X, S, Z, \pi, \mu, \tau, \lambda) = p(X \mid Z, \mu, \tau) p(S \mid Z, \lambda) p(Z \mid \pi) p(\pi) p(\lambda) p(\mu \mid \tau) p(\tau) \tag{16}$$

The assumption for the factors is $q(Z,\pi,\mu,\tau,\lambda) = q(Z)q(\pi,\mu,\tau,\lambda)$. We need to find $\ln q^*(Z)$ and $\ln q^*(\pi,\mu,\tau,\lambda)$.

**-Factor** $\ln q^*(Z)$.

For $\ln q^*(Z)$, the terms that don't depend on Z are added as a constant,

$$\begin{aligned}
\ln q^\star(Z) &= \mathbb{E}_{\pi,\mu,\tau,\lambda}[\ln p(X,S,Z,\pi,\mu,\tau,\lambda)] + \text{ const.}\\
&= \mathbb{E}_{\pi,\mu,\tau,\lambda}[\ln p(X \mid Z,\mu,\tau)p(S \mid Z,\lambda)p(Z \mid \pi)p(\pi)p(\lambda)p(\mu \mid \tau)p(\tau)] + \text{ const.}\\
&= \mathbb{E}_{\pi,\mu,\tau,\lambda}[\ln p(X \mid Z,\mu,\tau)p(S \mid Z,\lambda)p(Z \mid \pi)] + \text{ const.}\\
&= \mathbb{E}_{\mu,\tau,\pi}[\ln p(X,Z \mid \mu,\tau,\pi)] + \mathbb{E}_\lambda[\ln p(S \mid Z,\lambda) + \text{ const.}\\
&= \mathbb{E}_{\mu,\tau,\pi}\Big[\ln\prod_{n=1}^{N}\prod_{k=1}^{K}\left(\pi_k\mathcal{N}(X_n \mid \mu_k,(\tau_k)^{-1})\right)^{Z_{nk}}\Big] + \mathbb{E}_\lambda\Big[\ln\prod_{n=1}^{N}\prod_{k=1}^{K}\frac{\lambda_k^{S_n}e^{-\lambda_k}}{S_n!}\Big]
\end{aligned} \tag{17}$$

Where

$$\begin{aligned}
\ln\prod_{n=1}^{N}\prod_{k=1}^{K}\left(\pi_k\mathcal{N}(X_n \mid \mu_k,(\tau_k)^{-1})\right)^{Z_{nk}} &= \sum_{n=1}^{N}\sum_{k=1}^{K}Z_{nk}\ln\left(\pi_k\mathcal{N}(X_n \mid \mu_k,(\tau_k)^{-1})\right)\\
&= \sum_{n=1}^{N}\sum_{k=1}^{K}Z_{nk}\left[\ln(\pi_k) + \frac{1}{2}\ln\tau_k - \frac{1}{2}\ln(2\pi) - \frac{1}{2}\left[(X_n-\mu_k)\,\tau_k\,(X_n-\mu_k)\right]\right]
\end{aligned} \tag{18}$$

$$\ln\prod_{n=1}^{N}\prod_{k=1}^{K}\frac{\lambda_k^{S_n}e^{-\lambda_k}}{S_n!} = \sum_{n=1}^{N}\sum_{k=1}^{K}\left(S^n\ln\lambda_k - \lambda_k - \ln(S^n!)\right) \tag{19}$$

Therefore, the result is

$$\ln q^\star(Z) \quad = \sum_{n=1}^{N}\sum_{k=1}^{K}Z_{nk}\ln\rho_{nk} + \text{ const.} \tag{20}$$

where

$$\begin{aligned}
\ln\rho_{nk} =&\mathbb{E}_{\pi_k}[\ln\pi_k] + \frac{1}{2}\mathbb{E}_{\tau_k}[\ln\tau_k]\\
&- \frac{1}{2}\mathbb{E}_{\mu_k,\tau_k}[(X_n-\mu_k)\,\tau_k\,(X_n-\mu_k)] + S^n\mathbb{E}[\ln\lambda_k] - \mathbb{E}[\lambda_k] - \ln(S^n!)\\
=&\mathbb{E}_{\pi_k}[\ln\pi_k] + \frac{1}{2}\mathbb{E}_{\tau_k}[\ln\tau_k] - \frac{1}{2}\mathbb{E}_{\tau_k}[\tau_k]\,\mathbb{E}_{\mu_k}\left[(X_n-\mu_k)^2\right] + S^n\mathbb{E}[\ln\lambda_k] - \mathbb{E}[\lambda_k] - \ln(S^n!)
\end{aligned} \tag{21}$$

The dimensionality of the observable variable X was assumed to be 1. By normalizing,

$$\ln q^\star(Z) = \sum_{n=1}^{N}\sum_{k=1}^{K}Z_{nk}\ln r_{nk} + \text{const.},\qquad r_{nk} = \frac{\rho_{nk}}{\sum_{j=1}^{K}\rho_{nj}} \tag{22}$$

**-Factor** $q^*(\pi,\mu,\Lambda)$. Besides, we express $\mathbb{E}[Z_{nk}] = r_{nk}$.

$$\begin{aligned}
\ln q^\star(\pi,\mu,\Lambda) &= \mathbb{E}_Z[\ln p(X,S,Z,\pi,\mu,\tau,\lambda)] + \text{ const.}\\
&= \ln p(\pi) + \sum_{k=1}^{N}\ln p(\mu_k,\tau_k) + \sum_{k=1}^{N}\ln p(\lambda_k)\\
&+\mathbb{E}_Z[\ln p(Z \mid \pi)] + \mathbb{E}_Z[\ln p(X \mid Z,\mu,\tau)] + \mathbb{E}_Z[\ln p(S \mid Z,\lambda)]
\end{aligned} \tag{23}$$

We can oberve that it decomposes further into terms involving only $\pi$ and other terms involving $\mu, \tau, \lambda$. The terms $\mu, \tau, \lambda$ involves a sum over $K$ of $\mu_k, \tau_k, \lambda_k$ which means

$$q^\star(\pi, \mu, \tau, \lambda) = q(\pi) \prod_{k=1}^{K} q(\mu_k, \tau_k) q(\lambda_k) \tag{24}$$

**-Factor** $q(\pi)$.

$$\ln q^\star(\pi) = (\alpha - 1) \sum_{k=1}^{K} \ln \pi_k + \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \ln \pi_k \sim Dir(\pi \mid \alpha^{new}) \tag{25}$$

with $\alpha^{new} = \alpha + N_k$. We define $N_k = \sum_{n=1}^{N} r_{nk}$

**-Factor** $q(\mu_k, \tau_k)$.

$$\begin{aligned} &\ln q\left(\mu_k, \tau_k\right) = \ln p(\tau_k) + \ln p(\mu_k \mid \tau_k) + \ln p(X, Z \mid \mu, \tau, \pi) \\ &= \Big( (\alpha' - 1) \ln \tau_k - \beta' \tau_k \Big) + \Big( \frac{1}{2} \ln C\tau_k - \frac{C\tau_k}{2} (\mu_k - \mu)^2 \Big) + \sum_{n=1}^{N} r_{nk} \ln( \frac{1}{2} \ln \tau_k - \frac{\tau_k}{2} (X_n - \mu_k)^2 ) \end{aligned} \tag{26}$$

From this expression it is possible to identify the distributions

$$\begin{aligned} q(\tau_k) &\sim Ga(\alpha'^{\text{ new}}, \beta'^{\text{ new}}) \\ q\left(\mu_k, \tau_k\right) &\sim \mathcal{N}(\mu_k \mid \mu^{\text{new}}, (\tau_k^{\text{new}})^{-1}) \end{aligned} \tag{27}$$

**-Factor** $q(\lambda_k)$. Taking only factors invoving $\lambda_k$,

$$\begin{aligned} \ln q(\lambda_k) &= \ln \frac{(\beta_0)^{\alpha_0}}{\Gamma(\alpha_0)} \lambda_k^{\alpha_0 - 1} e^{-\beta_0 \lambda_k} + \sum_{n=1}^{N} r_{nk} \ln \frac{\lambda_k^{S_n} e^{-\lambda_k}}{S_n!} \\ &= (\alpha_0 - 1)\lambda_k - \beta_0 \lambda_k + \sum_{n=1}^{N} r_{nk} \Big( S^n \ln \lambda_k - \lambda_k - \ln\left(S^n!\right) \Big) \sim Ga(\lambda_k \mid \alpha_0^{\text{new}}, \beta_0^{\text{new}}) \end{aligned} \tag{28}$$

where

$$\begin{aligned} \alpha_0^{\text{new}} &= \alpha_0 + \sum_{n=1}^{N} r_{nk} S^n \\ \beta_0^{\text{new}} &= \beta_0 + \sum_{n=1}^{N} r_{nk} = \beta_0 + N_k \end{aligned} \tag{29}$$

The way the algorithm works is equivalent to the EM steps. We need to initialize all the parameters. In the E step, all the moments are updated, and in the M step, the variational distributions are computed again.

## 3.5 The Casino Model and Sampling Tables Given Dice Sums

Consider the following generative model. There are 2K tables in a casino, $t_1, ..., t_k, t_1, ..., t_K$ K of which each is equipped with a single dice (which may be biased, i.e., any categorical distribution on {1, ..., 6}) and N players $P_1, ..., P_N$ of which each is equipped with a single dice (which also may be biased, i.e., any categorical distribution on {1, ..., 6}). Each player $P_i$ visits K tables. In the k:th step, if the previous table visited was $t_{k1}$, the player visits $t_k$ with probability 1/4 and $t_k$ with probability 3/4, and if the previous table visited was $t_{k1}$, the player visits $t_k$ with probability 1/4 and $t_k$ with probability 3/4. So, in each step the probability of staying among the primed or unprimed tables is 1/4. At table k player $i$ throws her own dice as well as the table's dice. We then observe the sum $S^i_k$ of the two dice, while the outcome of the table's dice $X_k$ and the player's dice $Z_k$ are hidden variables. So for player $i$, we observe $S^i = S^i_1, ..., S^i_K$ , and the overall observation for N players is $S_1, ..., S_N$.

**Question 3.5.6: Provide a drawing of the Casino model as a graphical model. It should have a variable indicating the table visited in the k:th step, variables for all the dice outcomes, variables for the sums, and plate notation should be used to clarify that N players are involved.**

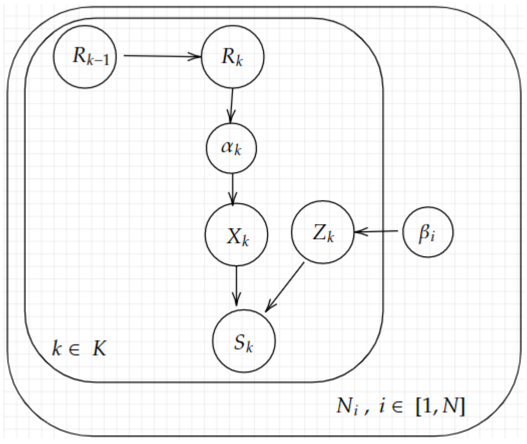


Figure 2: DAG model.

**Question 3.5.7: Implement the casino model.**

The casino model can be found in appendix A.

**Question 3.5.7: Provide data generated using at least three different sets of categorical dice distributions – what does it look like for all unbiased dice, i.e., uniform distributions, for example, or if some are biased in the same way, or if some are unbiased and there are two different groups of biased dice.**

In figure (3) is shown the ocurrencies for every number, when all dices are unbiased. It makes sense as result of the sum of the categorical distributions of the tables and the players. What is more, this is supported in figure (4), when the dices from both tables are biased to give always the number one. The result is coherent, given that it is similar to the distribution of a unbiased dice, except for the fact that it is shift by one.

Finally, the last test is done for all the dices biased. The dices of the tables are biased to one and the dices of the players are biased to 6. Therefore, as shown in figure (5), the result is always 7.

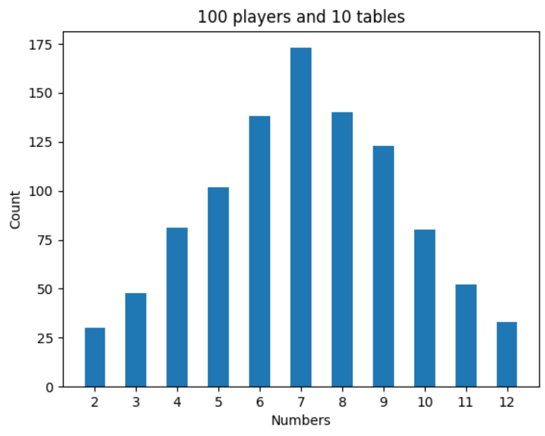


Figure 3: All the dices unbiased.

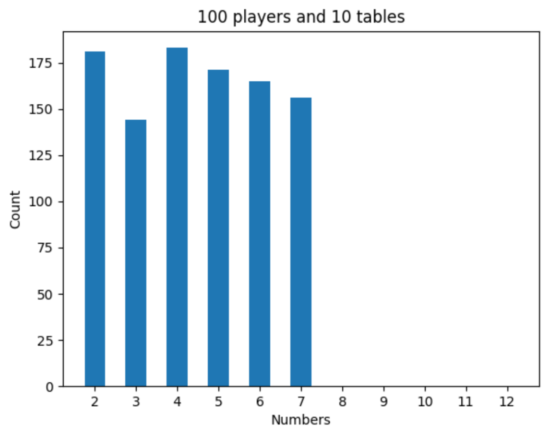


Figure 4: Only the dices from both tables biased to 1.

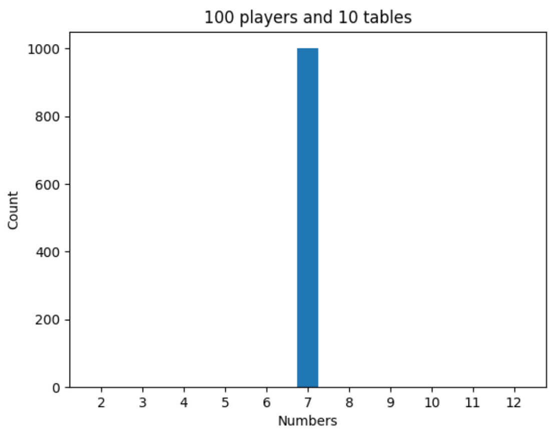


Figure 5: The dices from the tables biased to 1 and the player's dice biased to 6.

**Question 3.5.9: Describe an algorithm that, given (1) the parameters $\Theta$ of the full**

**casino model of Task 2.2 (so, $\Theta$ is all the categorical distributions corresponding to all the dice), (2) a sequence of tables $r_1 ..., r_K$ (that is, $r_i$ is $t_i$ or $t_i$) , and (3) an observation of dice sums $s_1, ..., s_K$, outputs $p(r_1, ..., r_K \mid s_1, ..., s_K, \Theta)$.**

We want to find

$$p(r_{1:K} \mid s_{1:k}, \theta) \tag{30}$$

The data we have are the matrices of the model.

The matrix pi for the initial states distribution is 1/2 for each entry.

The matrix A is defined by

| | $p(r_{t+1} = 0)$ | $p(r_{t+1} = 1)$ |
|---|---|---|
| $p(r_t = 0)$ | 3/4 | 1/4 |
| $p(r_t = 1)$ | 1/4 | 3/4 |

The matrix B is given by the variable $S$, which is the sum of the dice of the player and the dice of the table (can be primed or not). These follow different categorical distributions.

We can also define the elements of the model's matrices,

$$\begin{aligned} b_i(s_k = j) &= p(s_k = j \mid r_k = i) \\ a_{ij,k} &= p(r_k = j \mid r_{k-1} = i) \end{aligned} \tag{31}$$

with $b_j(s_k)$ being the probability to observe $s_k$ in state $r_j$.

Moreover, the forward algorithm is the following,

$$\begin{array}{lll} \text{Introduce:} & \alpha_k(i) = p(s_{1:k}, r_k = i \mid \theta) & \text{for every } k = 1, .., K \\ \text{Initialize as:} & \alpha_1(i) = \pi_i b_i(s_1) & \\ \text{For } 2 \leq k \leq K: & \alpha_k(i) = \left[ \sum_{j=1}^{N} \alpha_{k-1}(j) a_{ji,k} \right] b_i(s_k) & \\ \text{Which gives us:} & p\left(s_{1:K} \mid \theta\right) = \sum_{i=1}^{N} p\left(s_{1:K}, r_k = i \mid \theta\right) = \sum_{i=1}^{N} \alpha_K(i) & \end{array} \tag{32}$$

where $N$ is the number of hidden states.

By using Bayes, we can either work with the hidden variables conditioned on the observed variables or the observed variables given the hidden variables. Since the observed variables are independent given the hidden variables, we apply Bayes

$$p(r_{1:K} \mid s_{1:K}, \theta) = \frac{p(s_{1:K} \mid r_{1:K}, \theta) p(r_{1:K} \mid \theta)}{p(s_{1:K} \mid \theta)} \tag{33}$$

- It's easy to calculate $p(s_{1:K} \mid \theta)$ by following the forward algorithm

$$p(s_{1:K} \mid \theta) = \sum_{i=1}^{N} \alpha_K(i) \tag{34}$$

- For the term $p(r_{1:K} \mid \theta)$, we have that each $r_k$ is independent given $r_{k-1}$, and then

$$p(r_{1:K} \mid \theta) = p(r_1 \mid \theta) \prod_{k=1}^{K-1} p(r_{k+1} = i \mid r_k = j, \theta) = \pi_1 \prod_{k=1}^{K-1} a_{ji}(k) \tag{35}$$

- The last term $p(s_{1:K} \mid r_{1:K}, \theta)$ can be found by using the prod-rule and taking into account that the observations are independent given the hidden states

$$p(s_{1:K} \mid r_{1:K}, \theta) = \prod_{k=1}^{K} p(s_k \mid r_{1:K}, \theta) = \prod_{k=1}^{K} p(s_k = i \mid r_k = j, \theta) = \prod_{k=1}^{K} b_j(s_k) \tag{36}$$

By using the sum-rule, we find $b_{ij}(s_k)$ in the following way,

$$b_j(s_k) = p(s_k = i \mid r_k = j, \theta) = \sum_{l=1,m=1}^{6} p(s_k = l \mid r_k = j, \theta) p(z_k = m \mid r_k = j, \theta) \delta(l + m - s_i) = \tag{37}$$

$$= \sum_{l=1,m=1}^{6} Cat_j(l) Cat_{player}(m) \delta(l + m - s_i) \tag{38}$$

where $\delta$ is the Kronecker delta.

Therefore, now we can compute the probability of the sequence with the obtained expressions.

**Question 3.5.10: You should also show how to sample $r_1, ..., r_K$ from $p(R_1, ..., R_K \mid s_1, ..., s_K, \Theta)$ as well as implement and show test runs of this algorithm. In order to design this algorithm show first how to sample $r_K$ from $p(R_K | s_1, ..., s_K, \Theta) = p(R_K, s_1, ..., s_K \mid \Theta)/p(s_1, ..., s_K \mid \Theta)$ and then $r_{K1}$ from $p(R_{K1} \mid r_K, s_1, ..., s_K, \Theta) = p(R_{K1}, r_K, s_1, ..., s_K \mid \Theta)/p(r_K, s_1, ..., s_K \mid \Theta)$.**

We start by doing

$$\begin{aligned} p(R_{1:K} \mid s_{1:K}, \theta) &= p(R_k, R_{1:K-1} \mid s_{1:K}, \theta) \\ &= p(R_{1:K-1} \mid r_k, s_{1:K}, \theta) p(R_k \mid s_{1:K}, \theta) \end{aligned} \tag{39}$$

Then,

$$p(R_{1:K-1} \mid r_K, s_{1:K}, \theta) = p(R_{1:K-2} \mid r_{K-1}, s_{1:K-1}, \theta) p(R_{K-1} \mid r_K, s_{1:K-1}, \theta) \tag{40}$$

The calculations continue until $K = 1$. This leads to a algorithm in which the calculation is done backwards, from $k = K - 1$ to $k = 1$, and we need to calculate

$$p(R_K \mid s_{1:K}, \theta) = \frac{p(R_K = i, s_{1:K} \mid \theta)}{p(s_{1:K} \mid \theta)} = \frac{\alpha_K(R_K = i)}{\sum_{i=1}^{N} \alpha_K(i)}$$

and more generally, (41)

$$p(R_{k-1} \mid R_k, s_{1:k}, \theta) = \frac{p(R_k = j \mid R_{k-1} = i, s_{1:k}, \theta) p(R_{k-1} = i \mid s_{1:k}, \theta)}{p(R_k \mid s_{1:k}, \theta)} = \frac{a_{ij,k} \alpha_{k-1}(i)}{\sum_i a_{ij,k} \alpha_k(i)}$$

## References

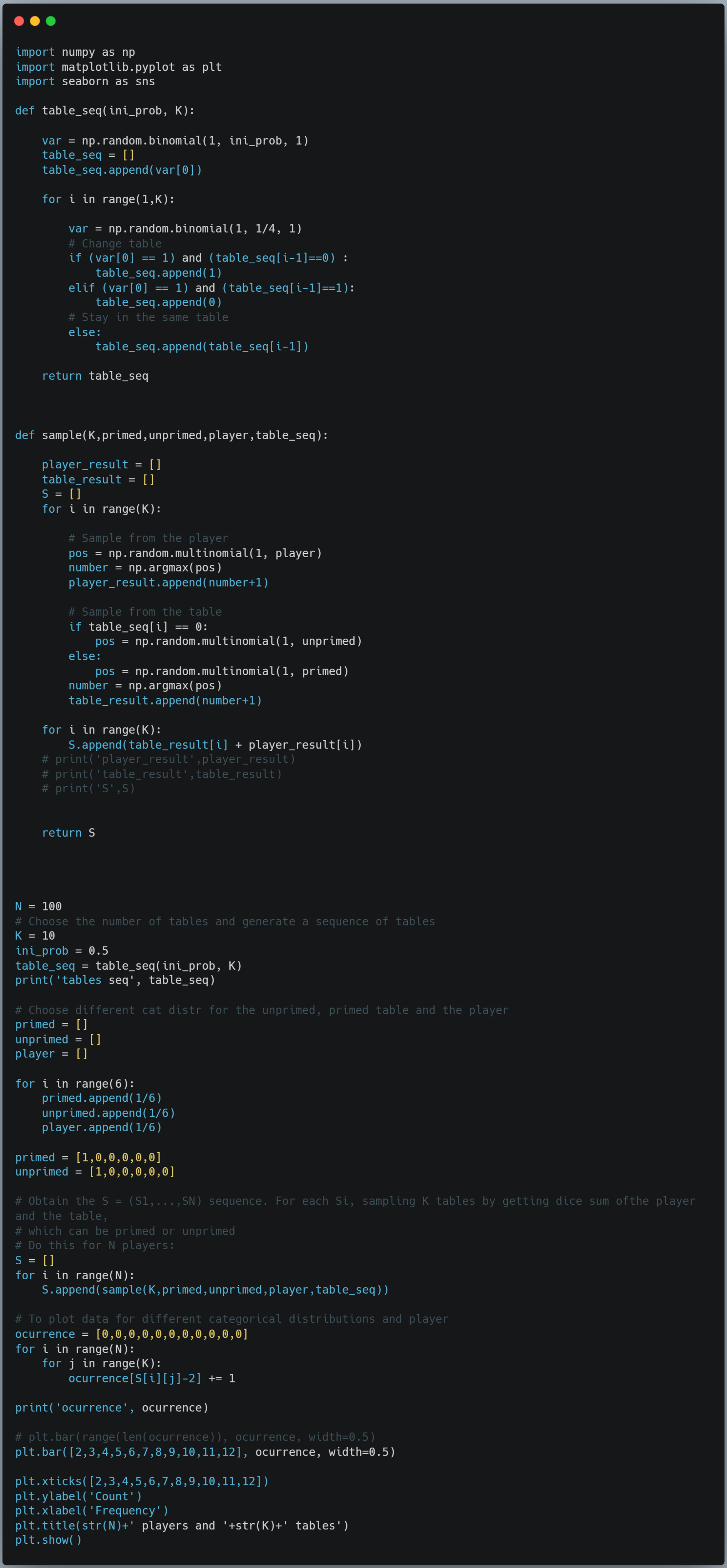
[1] Boaz Barak. *Introduction to Theoretical Computer Science.* Last version available in https://github.com/boazbk/tcs., 2021.

[2] William L. Hamilton. *Graph Representation Learning*. McGill University, 2020.

[3] Advanced machine learning, course material.

[4] Christopher M. Bishop. *Pattern recognition and Machine Learning.* Springer., 2006.

## A Apendix A: Code



```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

def table_seq(ini_prob, K):

    var = np.random.binomial(1, ini_prob, 1)
    table_seq = []
    table_seq.append(var[0])

    for i in range(1,K):

        var = np.random.binomial(1, 1/4, 1)
        # Change table
        if (var[0] == 1) and (table_seq[i-1]==0) :
            table_seq.append(1)
        elif (var[0] == 1) and (table_seq[i-1]==1):
            table_seq.append(0)
        # Stay in the same table
        else:
            table_seq.append(table_seq[i-1])

    return table_seq



def sample(K,primed,unprimed,player,table_seq):

    player_result = []
    table_result = []
    S = []
    for i in range(K):

        # Sample from the player
        pos = np.random.multinomial(1, player)
        number = np.argmax(pos)
        player_result.append(number+1)

        # Sample from the table
        if table_seq[i] == 0:
            pos = np.random.multinomial(1, unprimed)
        else:
            pos = np.random.multinomial(1, primed)
        number = np.argmax(pos)
        table_result.append(number+1)

    for i in range(K):
        S.append(table_result[i] + player_result[i])
    # print('player_result',player_result)
    # print('table_result',table_result)
    # print('S',S)


    return S




N = 100
# Choose the number of tables and generate a sequence of tables
K = 10
ini_prob = 0.5
table_seq = table_seq(ini_prob, K)
print('tables seq', table_seq)

# Choose different cat distr for the unprimed, primed table and the player
primed = []
unprimed = []
player = []

for i in range(6):
    primed.append(1/6)
    unprimed.append(1/6)
    player.append(1/6)

primed = [1,0,0,0,0,0]
unprimed = [1,0,0,0,0,0]

# Obtain the S = (S1,...,SN) sequence. For each Si, sampling K tables by getting dice sum ofthe player
and the table,
# which can be primed or unprimed
# Do this for N players:
S = []
for i in range(N):
    S.append(sample(K,primed,unprimed,player,table_seq))

# To plot data for different categorical distributions and player
ocurrence = [0,0,0,0,0,0,0,0,0,0,0]
for i in range(N):
    for j in range(K):
        ocurrence[S[i][j]-2] += 1

print('ocurrence', ocurrence)

# plt.bar(range(len(ocurrence)), ocurrence, width=0.5)
plt.bar([2,3,4,5,6,7,8,9,10,11,12], ocurrence, width=0.5)

plt.xticks([2,3,4,5,6,7,8,9,10,11,12])
plt.ylabel('Count')
plt.xlabel('Frequency')
plt.title(str(N)+' players and '+str(K)+' tables')
plt.show()
```