Homework 5: Poisson matrix factorization

STAT 348, UChicago, Spring 2025

Your name here: Haolin Yang

Hours spent: 10

(Please let us know how many hours in total you spent on this assignment so we can calibrate for future assignments. Your feedback is always welcome!)

Open in Colab

Instructions

This homework focuses on themes in lectures 10-12 on coordinate ascent variational inference (CAVI), admixture models, and Poisson matrix factorization.

For reference, this homework is a close adaption of HW5 for Scott Linderman's STATS 305C.

Assignment is due **Wednesday May 14, 11:59pm** on GradeScope.

Background

Poisson matrix factorization (PMF) is a mixed membership model like LDA, and it has close ties to non-negative factorization of count matrices. Let $\mathbf{X} \in \mathbb{N}^{N \times M}$ denote a count matrix with entries $x_{n,m}$. We model each entry as a Poisson random variable,

$$x_{n,m} \sim \mathrm{Po}\Big(\boldsymbol{\theta}_n^\top \boldsymbol{\phi}_m\Big) = \mathrm{Po}\Big(\sum_{k=1}^{K} \theta_{n,k}\phi_{m,k}\Big),$$

where $\boldsymbol{\theta}_n \in \mathbb{R}_+^K$ and $\boldsymbol{\phi}_m \in \mathbb{R}_+^K$ are *non-negative* feature vectors for row n and column m, respectively.

PMF has been used for recommender systems, aka collaborative filtering. In a recommender system, the rows correspond to users, the columns to items, and the entries $x_{n,m}$ to how much user n liked item m (on a scale of $0, 1, 2, \ldots$ stars, for example). The K feature dimensions capture different aspects of items that users may weight in their ratings.

Note that the Poisson rate must be non-negative. It is sufficient to ensure $\boldsymbol{\theta}_n$ and $\boldsymbol{\phi}_m$ are non-negative. To that end, PMF uses gamma priors,

$$\begin{aligned}\theta_{n,k} &\sim \mathrm{Ga}(\alpha_\theta, \beta_\theta) \\ \phi_{m,k} &\sim \mathrm{Ga}(\alpha_\phi, \beta_\phi),\end{aligned}$$

where $\alpha_\star$ and $\beta_\star$ are hyperparameters. When $\alpha_\star < 1$, the gamma distribution has a sharp peak at zero and the prior induces sparsity in the feature vectors.

Latent variable formulation

PMF can be rewritten in terms of a latent variable model. Note that,

$$\begin{aligned}x_{n,m} \sim \mathrm{Po}\Big(\sum_{k=1}^K \theta_{n,k}\phi_{m,k}\Big) \iff x_{n,m} &= \sum_{k=1}^K z_{n,m,k} \\ z_{n,m,k} &\sim \mathrm{Po}(\theta_{nk}\phi_{mk}) \quad \text{independently.}\end{aligned}$$

From this perspective, a user's rating of an item is a sum of ratings along each feature dimension, and each feature rating is an independent Poisson random variable.

The joint distribution is,

$$\begin{aligned}p(\mathbf{X}, \mathbf{Z}, \mathbf{\Theta}, \mathbf{\Phi}) = &\left[\prod_{n=1}^N \prod_{m=1}^M \mathbb{I}\Big[x_{n,m} = \sum_{k=1}^K z_{n,m,k}\Big] \prod_{k=1}^K \mathrm{Po}(z_{n,m,k} \mid \theta_{n,k}\phi_{m,k})\right] \\ &\times \left[\prod_{n=1}^N \prod_{k=1}^K \mathrm{Ga}(\theta_{n,k} \mid \alpha_\theta, \beta_\theta)\right] \times \left[\prod_{m=1}^M \prod_{k=1}^K \mathrm{Ga}(\phi_{m,k} \mid \alpha_\phi, \beta_\phi)\right]\end{aligned}$$

where $\mathbf{Z} \in \mathbb{N}^{N \times M \times K}$ denotes the *tensor* of feature ratings, $\mathbf{\Theta} \in \mathbb{R}_+^{N \times K}$ is a matrix with rows $\boldsymbol{\theta}_n$, and $\mathbf{\Phi} \in \mathbb{R}_+^{M \times K}$ is a matrix with rows $\boldsymbol{\phi}_m$.

Setup

In [2]:
```
import torch
from torch.distributions import Distribution, Gamma, Poisson, Multinomial
from torch.distributions.kl import kl_divergence

from tqdm.auto import trange

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_context("notebook")
```

Problem 1: Conditional distributions [math]

Since this model is constructed from conjugate exponential family distributions, the conditionals are available in closed form. We will let $\mathbf{z}_{n,m} = (z_{n,m,1}, \ldots, z_{n,m,K})$.

Problem 1a: Derive the conditional for $\mathbf{z}_{n,m}$

Find the conditional density $p(\mathbf{z}_{n,m} \mid x_{n,m}, \boldsymbol{\theta}_n, \boldsymbol{\phi}_m)$.

Your answer here.

$p(\mathbf{z}_{n,m} \mid x_{n,m}, \boldsymbol{\theta}_n, \boldsymbol{\phi}_m) = Multinomial(\mathbf{z}_{n,m}; \sum_{k=1}^K z_{n,m,k}, \frac{\theta_{n,1}\phi_{m,1}}{\lambda_{n,m}}, \ldots, \frac{\theta_{n,K}\phi_{m,K}}{\lambda_{n,m}})$

where $\lambda_{n,m} = \sum_{k=1}^K \theta_{n,k}\phi_{m,k}$

Problem 1b: Derive the conditional for $\theta_{n,k}$

Find the conditional density $p(\theta_{n,k} \mid \mathbf{Z}, \mathbf{\Phi})$.

Your answer here.

$p(\theta_{n,k} \mid \mathbf{Z}, \mathbf{\Phi}) \propto Gamma(\theta_{n,k}|\alpha_\theta, \beta_\theta) \prod_{m=1}^M Po(Z_{n,m,k}|\theta_{n,k}\phi_{m,k}) \propto \theta_{n,k}^{\alpha_\theta + \sum_{m=1}^M Z_{n,m,k}} e^{(-\theta_{n,k})(\beta_\theta +}$
So the posterior distribution is $Gamma(\alpha_\theta + \sum_{m=1}^M Z_{n,m,k}, \beta_\theta + \sum_{m=1}^M \phi_{m,k})$

Problem 1c: Derive the conditional for $\phi_{m,k}$

Find the conditional density $p(\phi_{m,k} \mid \mathbf{Z}, \mathbf{\Theta})$.

Your answer here.

$p(\phi_{m,k} \mid \mathbf{Z}, \mathbf{\Phi}) \propto Gamma(\phi_{m,k}|\alpha_\phi, \beta_\phi) \prod_{n=1}^N Po(Z_{n,m,k}|\theta_{n,k}\phi_{m,k}) \propto \phi_{m,k}^{\alpha_\phi + \sum_{n=1}^N Z_{n,m,k}} e^{(-\phi_{m,k})(}$
So the posterior distribution is $Gamma(\alpha_\phi + \sum_{n=1}^N Z_{n,m,k}, \beta_\phi + \sum_{n=1}^N \theta_{n,k})$

Problem 2: Coordinate ascent variational inference [math]

We will perform inference in this model using a mean-field variational posterior which factorizes according to:

$$
\begin{aligned}
q(\mathbf{Z}, \mathbf{\Phi}, \mathbf{\Theta}) &= q(\mathbf{Z})q(\mathbf{\Phi})q(\mathbf{\Theta}) \\
&= \left[\prod_{n=1}^N \prod_{m=1}^M q(\mathbf{z}_{n,m})\right] \left[\prod_{n=1}^N \prod_{k=1}^K q(\theta_{n,k})\right] \left[\prod_{m=1}^M \prod_{k=1}^K q(\phi_{m,k})\right]
\end{aligned}
$$

The optimal mean field factors will have the same forms as the conditional distributions above.

Problem 2a: Derive the CAVI update for $q(\mathbf{z}_{n,m})$

Show that, fixing $q(\mathbf{\Phi})$ and $q(\mathbf{\Theta})$, the optimal $q(\mathbf{z}_{n,m})$ is given by:

$$\begin{aligned} q(\mathbf{z}_{n,m}; \boldsymbol{\lambda}^{(z)}_{n,m}) &= \mathrm{Mult}(\mathbf{z}_{n,m}; x_{n,m}, \boldsymbol{\lambda}^{(z)}_{n,m}) \\ \log \lambda^{(z)}_{n,m,k} &= \mathbb{E}_q[\log \theta_{n,k} + \log \phi_{m,k}] + c \end{aligned}$$

Your answer here.

since

$p(\mathbf{z}_{n,m} | x_{n,m}, |) = Multinomial(\mathbf{z}_{n,m}; \sum_{k=1}^K z_{n,m,k}, \frac{\theta_{n,1}\phi_{m,1}}{\lambda_{n,m}}, \ldots, \frac{\theta_{n,K}\phi_{m,K}}{\lambda_{n,m}}) = \frac{x_{n,m}!}{\prod_{k=1}^K z_{n,m,k}!} e^{\sum_{k=1}^K z_{r}$

This is an expoenential family, hence the optimal CAVI update to each coordinate of the natural parameter is $\mathbb{E}_q[\log \theta_{n,k} + \log \phi_{m,k}] + c$, where $c = -\mathbb{E}_q \log \lambda_{n,m}$. This is the log of the probabilities in the posterior multinomial distribution.

Problem 2b: Derive the CAVI update for $q(\theta_{n,k})$

Show that, fixing $q(\mathbf{Z})$ and $q(\mathbf{\Phi})$, the optimal $q(\theta_{n,k})$ is given by:

$$\begin{aligned} q(\theta_{n,k}; \lambda^{(\theta)}_{n,k,1}, \lambda^{(\theta)}_{n,k,2}) &= \mathrm{Ga}(\theta_{n,k}; \lambda^{(\theta)}_{n,k,1}, \lambda^{(\theta)}_{n,k,2}) \\ \lambda^{(\theta)}_{n,k,1} &= \alpha_\theta + \sum_{m=1}^M \mathbb{E}_q[z_{n,m,k}] \\ \lambda^{(\theta)}_{n,k,2} &= \beta_\theta + \sum_{m=1}^M \mathbb{E}_q[\phi_{m,k}] \end{aligned}$$

Your answer here.

Gamma is one of the exponential family, and both the prior and the posterior of $\theta_{n,k}$ is the Gamma distribution, the optimal CAVI update invovles taking expectation of the posterior parameters. Since the true posterior is $Gamma(\alpha_\theta + \sum_{m=1}^{M} Z_{n,m,k}, \beta_\theta + \sum_{m=1}^{M} \phi_{m,k})$. The optimal CAVI update is $Gamma(\alpha_\theta + \sum_{m=1}^{M} \mathbb{E}_q[Z_{n,m,k}], \beta_\theta + \sum_{m=1}^{M} \mathbb{E}_q[\phi_{m,k}])$

Problem 2c: Derive the CAVI update for $q(\phi_{m,k})$

Show that, fixing $q(\mathbf{Z})$ and $q(\mathbf{\Theta})$, the optimal $q(\phi_{m,k})$ is given by:

$$\begin{aligned} q(\phi_{m,k}; \lambda_{m,k,1}^{(\phi)}, \lambda_{m,k,2}^{(\phi)}) &= \mathrm{Ga}(\phi_{m,k}; \lambda_{m,k,1}^{(\phi)}, \lambda_{m,k,2}^{(\phi)}) \\ \lambda_{m,k,1}^{(\phi)} &= \alpha_\phi + \sum_{n=1}^{N} \mathbb{E}_q[z_{n,m,k}] \\ \lambda_{m,k,2}^{(\phi)} &= \beta_\phi + \sum_{n=1}^{N} \mathbb{E}_q[\theta_{n,k}] \end{aligned}$$

Your answer here.

Gamma is one of the exponential family, and both the prior and the posterior of $\phi_{m,k}$ is the Gamma distribution, the optimal CAVI update invovles taking expectation of the posterior parameters. Since the true posterior is $Gamma(\alpha_\phi + \sum_{n=1}^{N} Z_{n,m,k}, \beta_\phi + \sum_{n=1}^{N} \theta_{n,k})$. The optimal CAVI update is $Gamma(\alpha_\phi + \sum_{n=1}^{N} \mathbb{E}_q[Z_{n,m,k}], \beta_\phi + \sum_{n=1}^{N} \mathbb{E}_q[\theta_{n,k}])$

Problem 2d: Find the expected sufficient statistics

To update the variational factors, we need the expectations $\mathbb{E}_q[z_{n,m,k}]$, $\mathbb{E}_q[\log \theta_{n,k} + \log \phi_{m,k}]$, $\mathbb{E}_q[\theta_{n,k}]$, and $\mathbb{E}_q[\phi_{m,k}]$. Assume that each factor follows the forms derived above. That is, assume $q(\mathbf{z}_{n,m})$ is multinomial with parameters $\lambda_{n,m}^{(z)}$ while $q(\theta_{n,k})$ and $q(\phi_{mk})$ are gamma with parameters $\left(\lambda_{n,k,1}^{(\theta)}, \lambda_{n,k,2}^{(\theta)}\right)$ and $\left(\lambda_{m,k,1}^{(\phi)}, \lambda_{m,k,2}^{(\phi)}\right)$, respectively. Derive what each of these expectations are in closed form.

Your answer here.

$\mathbb{E}_q[z_{n,m,k}] = x_{n,m}\lambda_{n,m,k}^{(z)}$

$\mathbb{E}_q[\theta_{n,k}] = \frac{\lambda_{n,k,1}^{(\theta)}}{\lambda_{n,k,2}^{(\theta)}}$

$$\mathbb{E}_q[\phi_{m,k}] = \frac{\lambda_{m,k,1}^{(\phi)}}{\lambda_{m,k,2}^{(\phi)}}$$

Problem 3: Implement Coordinate Ascent Variational Inference [code]

First we'll give some helper functions and objects. Because PyTorch doesn't offer support for batched multinomial distributions in which the total counts differ (e.g. each $\mathbf{z}_{n,m}$ follows a multinomial distribution in which the total count is $x_{n,m}$), we have defined a `BatchedMultinomial` distribution for your convenience. This distribution doesn't support sampling, but will return the mean of each Multinomial variable in its batch. This is exactly what is needed for the CAVI updates.

In [3]:
```
def gamma_expected_log(gamma_distbn):
    """Helper function to compute the expectation of log(X) where X follows a
    gamma distribution.
    """
    return torch.digamma(gamma_distbn.concentration) - torch.log(gamma_distbn.rate)

class BatchedMultinomial(Multinomial):
    """
    Creates a Multinomial distribution parameterized by `total_count` and
    either `probs` or `logits` (but not both). The innermost dimension of
    `probs` indexes over categories. All other dimensions index over batches.

    The `probs` argument must be non-negative, finite and have a non-zero sum,
    and it will be normalized to sum to 1 along the last dimension. `probs` will
    return this normalized value. The `logits` argument will be interpreted as
    unnormalized log probabilities and can therefore be any real number. It will
    likewise be normalized so that the resulting probabilities sum to 1 along
    the last dimension. `logits` will return this normalized value.

    Args:
        total_count (Tensor): number of trials
        probs (Tensor): event probabilities
            Has shape total_count.shape + (num_categories,)
        logits (Tensor): event log probabilities (unnormalized)
            Has shape total_count.shape + (num_categories,)

    Note: this text is mostly from the PyTorch documentation for the
        Multinomial distribution
    """
    def __init__(self, total_count, probs=None, logits=None, validate_args=None):
        super().__init__(probs=probs, logits=logits, validate_args=validate_args)
        self.total_count = total_count

    @property
    def mean(self):
        return self.total_count[..., None] * self.probs
```

Problem 3a: Implement a CAVI update step

Using the update equations derived in Problem 2, complete the `cavi_step` function below.

Hint: Given a `Distribution` named `d`, `d.mean` returns the mean of that distribution.

In [28]:
```python
import torch.nn.functional as F

def cavi_step(X, q_z, q_theta, q_phi, alpha_theta, beta_theta, alpha_phi, beta_phi):
    """One step of CAVI.

    Args:
        X: torch.tensor of shape (N, M)
        q_z: variational posterior over z, BatchedMultinomial distribution
        q_theta: variational posterior over theta, Gamma distribution
        q_phi: variational posterior over eta, Gamma distribution

    Returns:
        (q_z, q_theta, q_phi): Updated distributions after performing CAVI updates
    """
    ###
    # Your code here
    N, M = X.shape
    K = q_theta.concentration.shape[1]  # latent dimension

    log_theta = gamma_expected_log(q_theta)
    log_phi = gamma_expected_log(q_phi)

    log_theta_expand = log_theta[:, None, :]
    log_phi_expand = log_phi[None, :, :]
    log_lambda = log_theta_expand + log_phi_expand

    probs = F.softmax(log_lambda, dim=-1)
    q_z = BatchedMultinomial(total_count=X, probs=probs)


    expected_z = q_z.mean
    sum_z_over_m = expected_z.sum(dim=1)
    expected_phi = q_phi.mean
    sum_phi_over_m = expected_phi.sum(dim=0).unsqueeze(0).expand(N, K)

    q_theta = Gamma(
        concentration=alpha_theta + sum_z_over_m,
        rate=beta_theta + sum_phi_over_m
    )


    sum_z_over_n = expected_z.sum(dim=0)
    expected_theta = q_theta.mean
    sum_theta_over_n = expected_theta.sum(dim=0).unsqueeze(0).expand(M, K)

    q_phi = Gamma(
        concentration=alpha_phi + sum_z_over_n,
        rate=beta_phi + sum_theta_over_n
    )
```

```
    return q_z, q_theta, q_phi
```

Problem 3b: ELBO Calculation [math]

Recall that the evidence lower bound is defined as:

$$\mathcal{L}(q) = \mathbb{E}_q \left[\log p(\mathbf{X}, \mathbf{Z}, \mathbf{\Phi}, \mathbf{\Theta}) - \log q(\mathbf{Z}, \mathbf{\Phi}, \mathbf{\Theta}) \right]$$

Assume that $q(\mathbf{Z})$ has support contained in $\{\mathbf{Z} : \sum_{k=1}^K z_{n,m,k} = x_{n,m} \text{ for all } n, m\}$. Show that we can rewrite $\mathcal{L}(q)$ as:

$$\mathcal{L}(q) = \mathbb{E}_q[\log p(\mathbf{Z} \mid \mathbf{\Theta}, \mathbf{\Phi}) - \log q(\mathbf{Z})] - \mathrm{KL}(q(\mathbf{\Theta}) || p(\mathbf{\Theta})) - \mathrm{KL}(q(\mathbf{\Phi}) || p(\mathbf{\Phi}))$$

Next, use that $q(\mathbf{z}_{n,m}; \boldsymbol{\lambda}_{n,m}^{(z)}) = \mathrm{Mult}(\mathbf{z}_{n,m}; x_{n,m}, \boldsymbol{\lambda}_{n,m}^{(z)})$ and by plug in the densities of the Poisson and Multinomial distributions to show that we have:

$$\mathbb{E}_q[\log p(\mathbf{Z} \mid \mathbf{\Theta}, \mathbf{\Phi}) - \log q(\mathbf{Z})] = \\ \sum_{n=1}^N \sum_{m=1}^M \mathbb{E}_q \left[\sum_{k=1}^K -\theta_{n,k}\phi_{m,k} + z_{n,m,k} \log(\theta_{n,k}\phi_{m,k}) - z_{n,m,k} \log(\lambda_{n,m,k}^{(z)}) \right] - \log(x_{n,m}!)$$

Explain why we have:

$$\mathbb{E}_q \left[-\theta_{n,k}\phi_{m,k} + z_{n,m,k} \log(\theta_{n,k}\phi_{m,k}) - z_{n,m,k} \log(\lambda_{n,m,k}^{(z)}) \right] = \\ - \mathbb{E}_q \left[\theta_{n,k} \right] \mathbb{E}_q \left[\phi_{m,k} \right] + \mathbb{E}_q \left[z_{n,m,k} \right] \left(\mathbb{E}_q \left[\log(\theta_{n,k}) \right] + \mathbb{E}_q \left[\log(\phi_{m,k}) \right] - \log(\lambda_{n,m,k}^{(z)}) \right)$$

Your answer here.

$\mathcal{L}(q) = \mathbb{E}_q \left[\log p(\mathbf{X}, \mathbf{Z}, \mathbf{\Phi}, \mathbf{\Theta}) - \log q(\mathbf{Z}, \mathbf{\Phi}, \mathbf{\Theta}) \right] = \mathbb{E}_q[\log p(\mathbf{X}, \mathbf{Z} | \mathbf{\Phi}, \mathbf{\Theta}) + \log p(\mathbf{\Phi}, \mathbf{\Theta}) - \mathrm{lo}$
by the mean-field assumption.

Since $\log p(\mathbf{X}, \mathbf{Z} | \mathbf{\Phi}, \mathbf{\Theta})$ = $\log p(\mathbf{Z} | \mathbf{\Phi}, \mathbf{\Theta}) \mathbf{1}_{\sum_{k=1}^K z_{n,m,k} = x_{n,m} \forall n,m}$, we have
$\mathcal{L}(q) = \mathbb{E}_q \left[\log p(\mathbf{X}, \mathbf{Z}, \mathbf{\Phi}, \mathbf{\Theta}) - \log q(\mathbf{Z}, \mathbf{\Phi}, \mathbf{\Theta}) \right] = \mathbb{E}_q[\log p(\mathbf{Z} | \mathbf{\Phi}, \mathbf{\Theta}) + \log p(\mathbf{\Phi}) + \log p(\mathbf{\Theta}$
$- \mathrm{KL}(q(\mathbf{\Theta}) || p(\mathbf{\Theta})) - \mathrm{KL}(q(\mathbf{\Phi}) || p(\mathbf{\Phi}))$

$\mathbb{E}_q[\log p(\mathbf{Z} \mid \mathbf{\Theta}, \mathbf{\Phi}) - \log q(\mathbf{Z})] = \sum_{n=1}^N \sum_{m=1}^M \mathbb{E}_q[\sum_{k=1}^K (\log p(Z_{n,m,k} \mid \mathbf{\Theta}, \mathbf{\Phi})) - \log q(\mathbf{Z}_{\mathbf{n,m}}$
$\sum_{m=1}^M \mathbb{E}_q[\sum_{k=1}^K (Z_{n,m,k} \log(\theta_{n,k}\phi_{m,k} - \theta_{n,k}\phi_{m,k} - \log Z_{n,m,k}!) - \log(X_{n,m}! + \sum_{k=1}^K \log Z_{n,m}$
$\sum_{m=1}^M \mathbb{E}_q \left[\sum_{k=1}^K -\theta_{n,k}\phi_{m,k} + z_{n,m,k} \log(\theta_{n,k}\phi_{m,k}) - z_{n,m,k} \log(\lambda_{n,m,k}^{(z)}) \right] - \log(x_{n,m}!)$

By the mean-field assumption $\theta_{n,k}, \phi_{m,k}, z_{n,m,k}$ are independent, thus we have

$$\mathbb{E}_q \left[-\theta_{n,k}\phi_{m,k} + z_{n,m,k} \log(\theta_{n,k}\phi_{m,k}) - z_{n,m,k} \log(\lambda_{n,m,k}^{(z)}) \right] = \\ - \mathbb{E}_q \left[\theta_{n,k} \right] \mathbb{E}_q \left[\phi_{m,k} \right] + \mathbb{E}_q \left[z_{n,m,k} \right] \left(\mathbb{E}_q \left[\log(\theta_{n,k}) \right] + \mathbb{E}_q \left[\log(\phi_{m,k}) \right] - \log(\lambda_{n,m,k}^{(z)}) \right)$$

Problem 3c: Implement the ELBO [code]

Using our expression above, write a function which evaluates the evidence lower bound.

Hints:

- Use the `kl_divergence` function imported above to compute the KL divergence between two `Distributions` in the same family.
- Recall that for integers n, $\Gamma(n+1) = n!$ where Γ is the Gamma function. $\log \Gamma$ is implemented in PyTorch as `torch.lgamma`.

In [8]:
```
def elbo(X, q_z, q_theta, q_phi, p_theta, p_phi):
    """Compute the evidence lower bound.

    Args:
        X: torch.tensor of shape (N, M)
        q_z: variational posterior over z, BatchedMultinomial distribution
        q_theta: variational posterior over theta, Gamma distribution
        q_phi: variational posterior over eta, Gamma distribution
        p_theta: prior over theta, Gamma distribution
        p_phi: prior over eta, Gamma distribution

    Returns:
        elbo: torch.tensor of shape []
    """
    N, M = X.shape
    K = q_theta.concentration.shape[1]
    E_log_theta = gamma_expected_log(q_theta)  # (N, K)
    E_log_phi   = gamma_expected_log(q_phi)    # (M, K)
    E_theta     = q_theta.mean                 # (N, K)
    E_phi       = q_phi.mean                  # (M, K)
    E_z         = q_z.mean                    # (N, M, K)
    log_lambda  = torch.log(q_z.probs + 1e-12) # (N, M, K)

    E_log_theta_phi = E_log_theta[:, None, :] + E_log_phi[None, :, :]  # (N, M, K)
    E_theta_phi     = torch.einsum("nk,mk->nmk", E_theta, E_phi)       # (N, M, K)

    term1 = torch.sum(E_z * E_log_theta_phi) - torch.sum(E_theta_phi)
    term2 = -torch.sum(E_z * log_lambda)
    term3 = -torch.sum(torch.lgamma(X + 1))

    kl_theta = kl_divergence(q_theta, p_theta).sum()
    kl_phi   = kl_divergence(q_phi, p_phi).sum()

    elbo = term1 + term2 + term3 - kl_theta - kl_phi
    return elbo / torch.sum(X)
```

Implement CAVI loop [given]

Using your functions defined above, complete the function `cavi` below. `cavi` loops for some number of iterations, updating each of the variational factors in sequence and evaluating the ELBO at each step.

In [29]:
```
from torch.distributions import Uniform

def cavi(data,
         num_factors=10,
         num_iters=100,
         tol=1e-5,
         alpha_theta=0.1,
         beta_theta=1.0,
         alpha_phi=0.1,
         beta_phi=1.0,
         seed=0
        ):
    """Run coordinate ascent VI for Poisson matrix factorization.

    Args:

    Returns:
        elbos, (q_z, q_theta, q_phi):
    """
    data = data.float()
    N, M = data.shape
    K = num_factors      # short hand

    # Initialize the variational posteriors.
    q_phi = Gamma(Uniform(0.5 * alpha_phi, 1.5 * alpha_phi).sample((M, K)),
                  Uniform(0.5 * beta_phi, 1.5 * beta_phi).sample((M, K)))
    q_theta = Gamma(Uniform(0.5 * alpha_theta, 1.5 * alpha_theta).sample((N, K)),
                    Uniform(0.5 * beta_theta, 1.5 * beta_theta).sample((N, K)))
    q_z = BatchedMultinomial(data, logits=torch.zeros((N, M, K)))

    p_theta = Gamma(alpha_theta, beta_theta)
    p_phi = Gamma(alpha_phi, beta_phi)

    # Run CAVI
    elbos = [elbo(data, q_z, q_theta, q_phi, p_theta, p_phi)]
    for itr in trange(num_iters):
        q_z, q_theta, q_phi = cavi_step(data, q_z, q_theta, q_phi,
                                        alpha_theta, beta_theta,
                                        alpha_phi, beta_phi)

        elbos.append(elbo(data, q_z, q_theta, q_phi, p_theta, p_phi))
    return torch.tensor(elbos), (q_z, q_theta, q_phi)
```

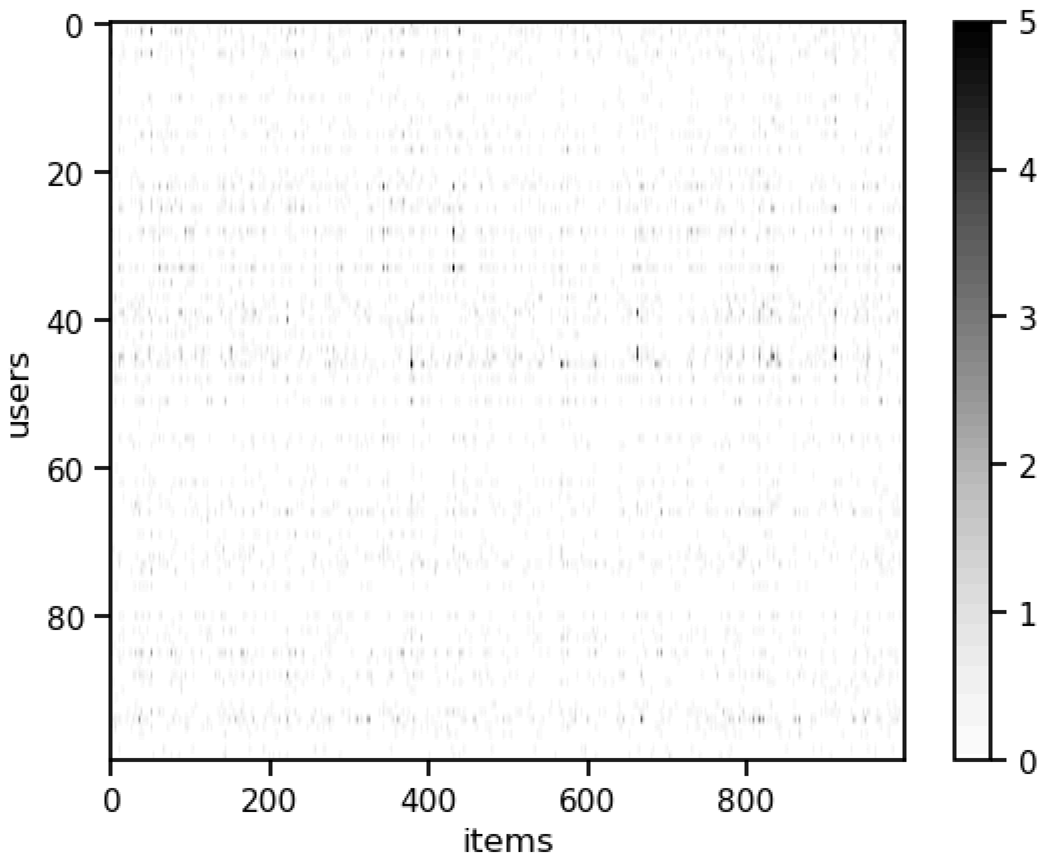
Test your implementation on a toy dataset

To check your implementation is working properly, we will fit a mean-field variational posterior using data sampled from the true model.

In [30]:
```
# Constants
N = 100    # num "users"
M = 1000 # num "items"
```

```python
K = 5      # number of latent factors

# Hyperparameters
alpha = 0.1  # sparse gamma prior with mean alpha/beta
beta = 1.0

# Sample data from the model
torch.manual_seed(305)
theta = Gamma(alpha, beta).sample(sample_shape=(N, K))
phi = Gamma(alpha, beta).sample(sample_shape=(M, K))
data = Poisson(theta @ phi.T).sample()

print(data.shape)
# Plot the data matrix
plt.imshow(data, aspect="auto", vmax=5, cmap="Greys")
plt.xlabel("items")
plt.ylabel("users")
plt.colorbar()

print("Max data:  ", data.max())
print("num zeros: ", torch.sum(data == 0))
```

```
torch.Size([100, 1000])
Max data:   tensor(14.)
num zeros:  tensor(95568)
```

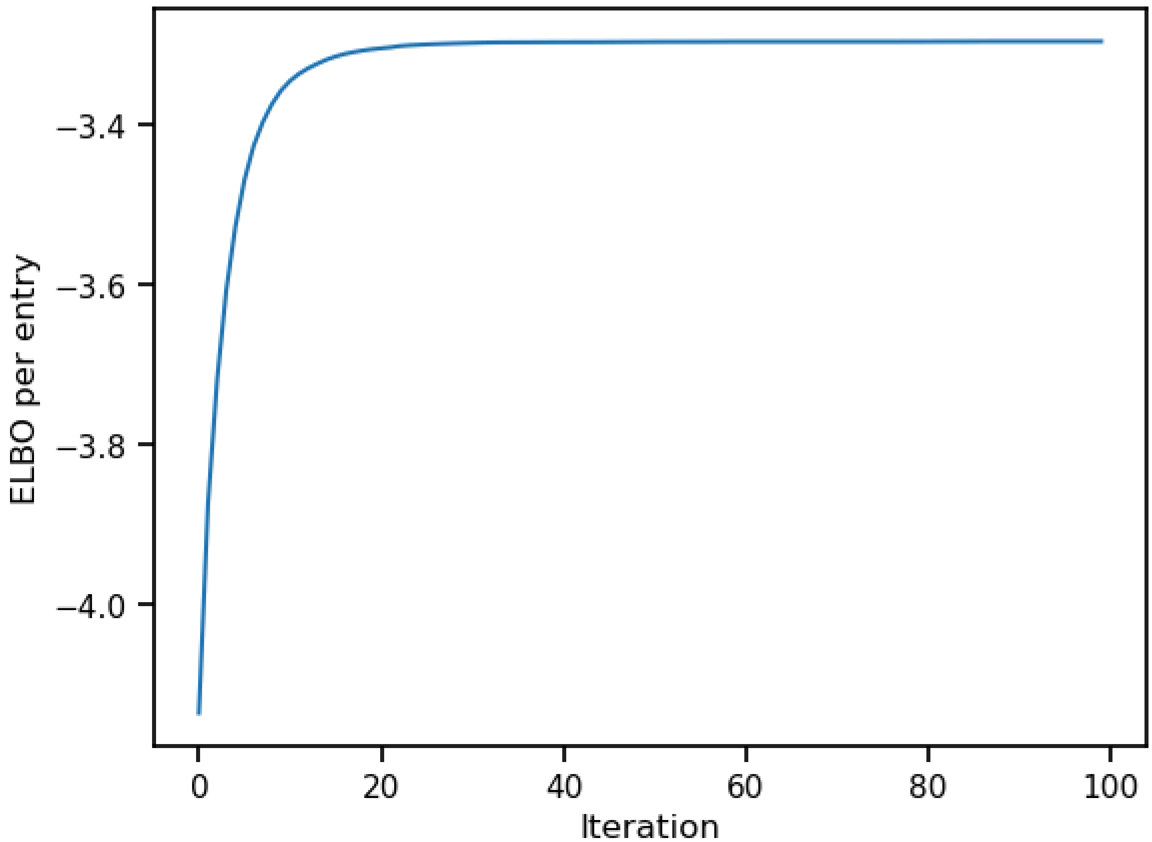


In [31]:
```python
elbos, (q_z, q_theta, q_phi) = cavi(data)
```

```
0%|          | 0/100 [00:00<?, ?it/s]
```

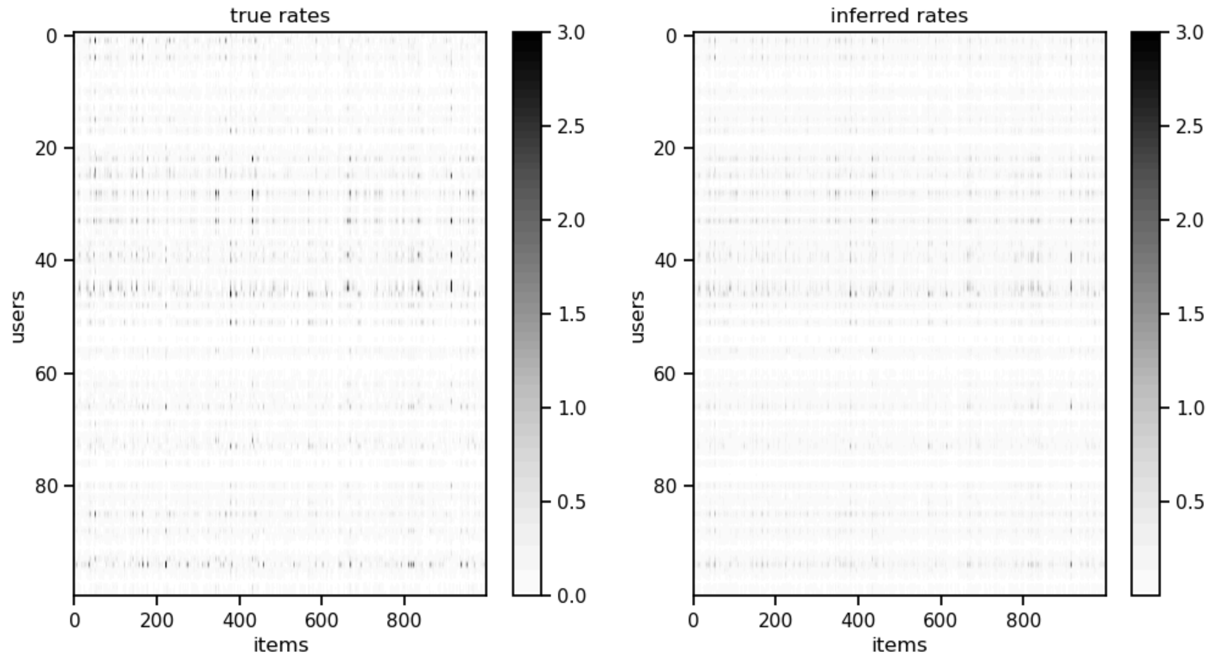
In [32]:
```
plt.plot(elbos[1:])
plt.xlabel("Iteration")
plt.ylabel("ELBO per entry")
```

Out[32]: `Text(0, 0.5, 'ELBO per entry')`



In [33]:
```
true_rates = theta @ phi.T
inf_rates = q_theta.mean @ q_phi.mean.T

# Plot the data matrix
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.imshow(true_rates, aspect="auto", vmax=3, cmap="Greys")
plt.xlabel("items")
plt.ylabel("users")
plt.title("true rates")
plt.colorbar()

plt.subplot(1, 2, 2)
plt.imshow(inf_rates, aspect="auto", vmax=3, cmap="Greys")
plt.xlabel("items")
plt.ylabel("users")
plt.title("inferred rates")
plt.colorbar()
```

Out[33]: `<matplotlib.colorbar.Colorbar at 0x20d0601b5f0>`



Problem 4: Run your code on a downsampled LastFM dataset

Next, we will use data gathered from Last.FM users to fit a PMF model. We use a downsampled version of the Last.FM-360K users dataset. This dataset records how many times each user played an artist's songs. We downsample the data to include only the 2000 most popular artists, as measured by how many users listened to the artist at least once, and the 1000 most prolific users, as measured by how many artists they have listened to.

In the code below , we use `lfm` to represent the data matrix X in the model. That is, `lfm[n, d]` denotes how many times the `n` -th user played a song by the `d` -th artist.

In [34]:
```
import pandas as pd

lfm_df = pd.read_csv('subsampled_last_fm.csv')
lfm = lfm_df.pivot_table(index='UserID', columns='ItemID', values='Count', aggfunc=
    .fillna(0).astype(int).to_numpy()
lfm = torch.tensor(lfm, dtype=torch.int)
print(lfm.shape)
```

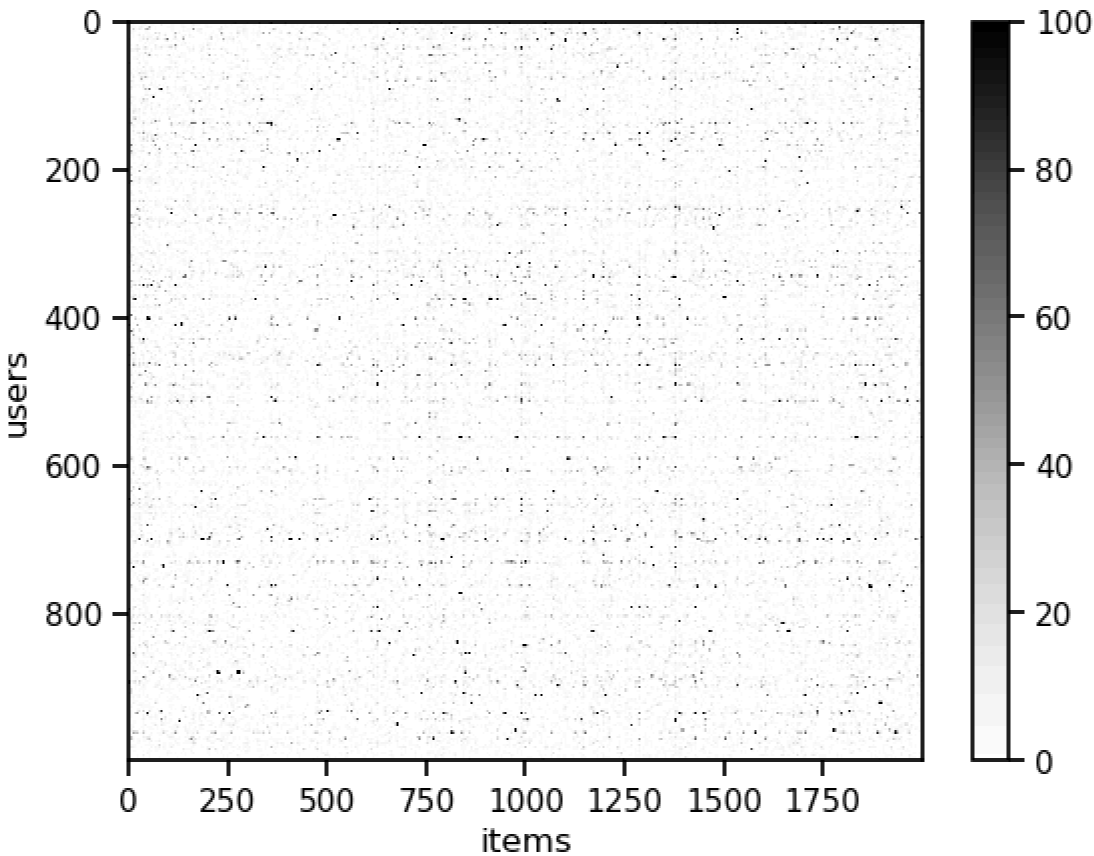
```
torch.Size([999, 2000])
```

```
C:\Users\Haoli\AppData\Local\Temp\ipykernel_176068\1531818262.py:4: FutureWarning: T
he provided callable <built-in function sum> is currently using DataFrameGroupBy.su
m. In a future version of pandas, the provided callable will be used directly. To ke
ep current behavior pass the string "sum" instead.
  lfm = lfm_df.pivot_table(index='UserID', columns='ItemID', values='Count', aggfunc
=sum)\
```

In [35]:
```
plt.imshow(lfm, aspect="auto", vmax=100, cmap="Greys")
plt.xlabel("items")
```

```
plt.ylabel("users")
plt.colorbar()
```

Out[35]: <matplotlib.colorbar.Colorbar at 0x20d0422f0b0>



Using the code below, run coordinate ascent variational inference on this dataset. Our implementation takes around 10-15 minutes to finish, and achieves a rescaled ELBO of around -2.

In [36]:
```
elbos, (q_z, q_theta, q_phi) = cavi(lfm,
     num_factors=40,
     num_iters=200,
     alpha_theta=1.,
     beta_theta=0.5,
     alpha_phi=1.,
     beta_phi=0.5)
```

```
0%|          | 0/200 [00:00<?, ?it/s]
```

Investigate "genres"

The columns of $\mathbf{H}$ correspond to weights on artists. Intuitively, each of the K columns should put weight on subsets of artists that are often played together. We might think of these columns as reflecting different "genres" of music. The code below the top 10 artists for a few of these columns.

In [38]:

```
# Find the 10 most used genres
genre_loading = q_theta.mean.sum(0)
genre_order = torch.argsort(genre_loading, descending=True)

# Print the top 10 artists for each of the top 10 genres
for genre in genre_order[:10]:
    print("genre ", genre)
    artist_idx = torch.argsort(q_phi.mean[:, genre],
                               descending=True)[:10].numpy()
    subset = lfm_df[lfm_df['ItemID'].isin(artist_idx)]
    print(subset[['ItemID', 'Artist']].drop_duplicates())
    print("")
```

genre tensor(1)

	ItemID	Artist
1	1647	tegan and sara
10	15	sufjan stevens
24	38	cat power
44	28	elliott smith
154	1306	modest mouse
155	1578	ryan adams
157	910	bright eyes
469	1959	fleet foxes
1019	815	sam cooke
1289	1933	jay-z

genre tensor(7)

	ItemID	Artist
49	1286	radiohead
204	627	david bowie
206	795	the cure
320	1658	devo
394	476	the smiths
475	12	morrissey
687	1877	new order
711	692	sonic youth
739	813	guided by voices
757	469	the magnetic fields

genre tensor(25)

	ItemID	Artist
60	990	pink floyd
61	758	metallica
105	773	led zeppelin
142	1591	iron maiden
165	806	judas priest
211	763	ac/dc
1365	614	black sabbath
1369	1301	megadeth
1375	651	motörhead
1677	1768	kiss
20257	763	acdc

genre tensor(14)

	ItemID	Artist
55	44	jimi hendrix
56	1377	the beatles
60	990	pink floyd
69	857	bob dylan
73	1210	the doors
105	773	led zeppelin
204	627	david bowie
551	988	pearl jam
563	1100	the clash
2920	1855	guns n' roses
2934	1855	guns n roses
18201	1377	beatles

genre tensor(29)

```
       ItemID                        Artist
336       442                     ghostface
817      1150          a tribe called quest
1419     1500                        common
1431      964                     the roots
2297     1639                           nas
2312     1213                       mos def
2693     1598                       j dilla
3613     1834                        madlib
3835       99                   wu-tang clan
4726      442               ghostface killah
10000      99                   wu tang clan
15054    1319                   blessthefall
24609     964  the roots featuring d'angelo

genre  tensor(2)
       ItemID               Artist
13        676                stars
38       1605             coldplay
49       1286            radiohead
83       1196                 muse
109       981          jason mraz
167         2  death cab for cutie
235       822                  m83
282      1066           bloc party
1080     1785              incubus
1852     1279          snow patrol
29203     676            the stars

genre  tensor(32)
       ItemID              Artist
56       1377         the beatles
67         26               queen
106       730        eric clapton
116        63     frédéric chopin
183       911             madonna
280      1705                abba
324      1370  the rolling stones
330        13       elvis presley
565      1254                  u2
876       832        mushroomhead
18201    1377             beatles

genre  tensor(28)
      ItemID           Artist
170       37        daft punk
218     1007     depeche mode
222      574          blondie
865     1494        the knife
951     1930        fever ray
973     1379  crystal castles
992      657  yeah yeah yeahs
1303    1400         ladytron
1443    1313  franz ferdinand
1447    1947          justice
```

```
genre  tensor(5)
       ItemID                     Artist
59        645                miles davis
64       1672               johnny cash
69        857                 bob dylan
277      1043           various artists
293      1533                 tom waits
297       163  nick cave and the bad seeds
324      1370         the rolling stones
330        13              elvis presley
343       755             leonard cohen
443       163    nick cave & the bad seeds
913       848          bruce springsteen
18256    1043                       v.a.

genre  tensor(0)
       ItemID          Artist
328      1819      bad brains
452      1791      have heart
454       820   comeback kid
788      1692         ramones
821       277          rancid
860      1735            nofx
885      1281     against me!
894       142   bad religion
1364     1170      black flag
11243    1769  sick of it all
```

Problem 4a

Inspect the data either using the csv file or the pandas dataframe and choose a user who has listened to artists you recognize. If you are not familiar with any of the artists, use the listener with UserID 349, who mostly listens to hip-hop artists. For the particular user n you choose, find the 10 artists who are predicted to have the most plays by sorting the vector of mean song counts predicted by the model, i.e. the n^{th} row of $\mathbb{E}_q[\boldsymbol{\Theta}\boldsymbol{\Phi}^\top]$. Are these artists you would expect the user would enjoy? Are there any artists that the user has not listened to?

Hint: Use `torch.argsort(..., descending=True)` *to return the indices of the largest elements of a vector in descending order.*

In [50]:

```
###
artist_idx=torch.argsort((q_theta.mean@q_phi.mean.T)[736], descending=True)[:10].nu
subset = lfm_df[lfm_df['ItemID'].isin(artist_idx)]
print(subset[['ItemID', 'Artist']].drop_duplicates())
print("")
##
```

```
     ItemID                Artist
49     1286            radiohead
204     627          david bowie
206     795             the cure
320    1658                 devo
394     476           the smiths
475      12            morrissey
687    1877            new order
711     692          sonic youth
739     813      guided by voices
757     469  the magnetic fields
```

UserID 736 listens to many Japanese musicians as can be seen from the original .csv file. The predicted list of top artists that he would listen to does not seem plausible.

Problem 5: Reflections

Problem 5a

Discuss one advantage and one disadvantage of fitting a posterior using variational inference vs. sampling from the posterior using MCMC.

Your answer here.

VI is typically much faster and more scalable than sampling-based methods like MCMC, especially on large datasets, because VI converts inference into an optimization problem that can be solved efficiently using gradient-based procedures without generating long sample chains.

VI often underestimates the complexity of the true posterior by searching only in a family of fully factorized variational distributions. Also it may not converge to the true posterior because non-convex optimization is involved.

Problem 5b

First, explain why the assumption that $\mathbf{Z}$, $\mathbf{\Phi}$ and $\mathbf{\Theta}$ are independent in the posterior will never hold.

Next, recall that maximizing the ELBO is equivalent to minimizing the KL divergence between the approximate posterior and the true posterior. In general, how will the approximate posterior differ from the true posterior, given that the variational family does not include the true posterior?

Your answer here.

First, the assumption that $\mathbf{Z}$, $\mathbf{\Phi}$, and $\mathbf{\Theta}$ are independent in the variational posterior will never hold in general because these variables are inherently dependent in the true posterior. For example, the latent variable $\mathbf{Z}$ depends on both user preferences $\mathbf{\Theta}$ and item features $\mathbf{\Phi}$ when generating the observed data. If a user's preference for a topic increases, it can reduce the need for higher weights in item features to explain the same observation. Besides, $\mathbf{Z}$ also depends on both $\mathbf{\Phi}$ and and $\mathbf{\Theta}$

If the variational family does not contain the true posterior, then the approximate posterior will likely be the one in the variational familly with the smallest KL divergence w.r.t. the posterior. Moreover, since VI minimizes the KL divergence,the approximate posteior will very small probabilities where the true posterior has low density. It tends to favor underestimating the posterior variance and capturing only a single mode.

Problem 5c

Suppose we are using this model to recommend new items to users. Describe one improvement that could be made to the model which you think would lead to better recommendations.

Your answer here.

One improvement would be to incorporate side information about users or items—such as user demographics or item metadata (e.g., artist genres, song tags)—into the model. This could be done by embedding the side information as features, and then let $\mathbf{Theta}$ and $\mathbf{\Phi}$ depend on these features to make the variational distributions more informative, or directly let the features to influence $\mathbf{X}$. Incorporating this auxiliary information helps the model make better predictions for new users or sparse users who have interacted with few items.

Submission Instructions

Formatting: check that your code does not exceed 80 characters in line width. If you're working in Colab, you can set *Tools → Settings → Editor → Vertical ruler column* to 80 to see when you've exceeded the limit.

Download your notebook in .ipynb format and use the following commands to convert it to PDF:

```
jupyter nbconvert --to pdf hw5_yourname.ipynb
```

Dependencies:

- `nbconvert` : If you're using Anaconda for package management,

```
conda install -c anaconda nbconvert
```

Upload your .pdf files to Gradescope.