BBVI for DCM

August 27, 2024

1 Black Box Variational Inference on Diagnostic Classification Models

Motivation behind ELBO cost function

The evidence lower bound is a lower bound of the log marginalized joint distribution $p(x, z)$. The value $\log p(x)$ is called the evidence.

$$\begin{aligned}
\log p(x) &= \log\left(\int p(x,z)dz\right) \\
&= \log\left(\int p(x,z)\frac{q(z)}{q(z)}dz\right) \\
&= \log\left(E_q\left[\frac{p(x,z)}{q(z)}\right]\right) \\
&\geq E_q[\log p(x,z)] - E_q[\log q(z)] \quad \text{(ELBO)}
\end{aligned}$$

The last term obtained via Jensen's inequality and linearity of expectation. It is referred to as the evidence lower bound. The ELBO shows up in variational inference as an alternative cost function to the KL divergence.

$$\begin{aligned}
KL(q||p) &= E_q\left[\log\left(\frac{q(z)}{p(z|x)}\right)\right] \\
&= E_q[\log q(z)] - E_q[\log p(z|x)] \\
&= E_q[\log q(z)] - E_q[\log p(x,z)] + E_q[\log p(x)] \\
&= -\text{ELBO} + \log p(x)
\end{aligned}$$

Since the goal of variational inference is to optimize the parameters of q, maximizing the ELBO with respect to q is equivalent to minimizing the KL divergence with respect to q.

Noisy Gradient of the ELBO

The minimizer of the ELBO does not always have an analytical form. Instead, gradient ascent is used to find the maximizer. An unbiased estimator of the gradient is obtained by sampling from variational distirbution q. The gradient of the ELBO is

$$\nabla_\lambda L = E_q[\nabla_\lambda \log q(z|\lambda)(\log p(x,z) - \log q(z|\lambda))]$$

Assume each latent variable z is governed by independent variational distributions such that $q(z|\lambda) = \prod_{i=1}^{n} q(z_i|\lambda_i)$. Equation 5 from (Ranganath et. al., 2013) gives the Rao-Blackwellized estimator of the gradient for individual λ_i.

$$\nabla_{\lambda_i} L = E_{q_{(i)}}[\nabla_{\lambda_i} \log q(z_i|\lambda_i)(\log p_i(x, z_{(i)}) - \log q(z_i|\lambda_i))]$$

$E_{q_{(i)}}$ is the expectation with respect to the markov blanket of z_i, or the variables that depend on z_i, and $p_i(x, z_{(i)})$ is the terms of the join distribution that depend on x and z_i.

Diagnostic Classification Model (DCM)

In this problem setting, there are I items on an assesment, and each item measures up to A attributes. Each latent class c has an attribute profile $Z_c = [z_{c1}, ..., z_{cA}] \in \{0, 1\}^A$ where $z_{ca} = 0$ indicates that the a-th attribute is not mastered and $z_{ca} = 1$ indicates that the a-th attribute is mastered in latent class c.

$Q \in \{0, 1\}^{I \times A}$ is called the Q matrix and q_{ia} indicates whether question i measures mastery of attribute a. 0 indicates that the attribute is not measured and 1 indicates that it is measured.

$\Delta_i = (\delta_{i1}, ..., \delta_{iL})^T$ is the feature matrix for item i, where each row is indexed by an attribute profile and the columns are the attributes and all the interactions of attributes measured by an item on the assesment. 1 indicates that the attributes in the interaction term are all mastered in the attributes profile. 0 indicates that at least 1 attribute in the interaction is not mastered in the attribute profile.

For example, consider that item i measures mastery of 2 attributes. There are $L = 4$ possible attribute profiles (00, 01, 10, 11) corresponding to no masteries, mastering attribute 1, mastering attribute 2, and mastering both attributes. There are 4 features in Δ_i (00, 01, 10, 11) corresponding to intercept, main effect of attribute 1 mastery, main effect of attribute 2 mastery, and interaction effect.

	00	**01**	**10**	**11**
00	1	0	0	0
01	1	1	0	0
10	1	0	1	0
11	1	1	1	1

The DCM models $\pi_{ic} = P(Y_{ic} = 1|Z_c)$, the probability of a respondent with attribute profile Z_c answering the i-th item correctly.

Let σ be the sigmoid function, $\sigma(x) = \frac{1}{1+e^{-x}}$

$$\pi_{ic} = \sigma(\beta_i^T \delta_{ic})$$

$$P(Y_{ic} = y_{ic}|Z_c) = (\pi_{ic})^{y_{ic}} \cdot (1 - \pi_{ic})^{1-y_{ic}}$$

$$= \sigma((2y_{ic} - 1)\beta_i^T \delta_{ic}) \quad \text{(Only true when } y_{ic} \in \{0, 1\})$$

1.1 Naive Black Box

In this section black box variational inference will be used to estimate the parameters of a DCM model. Data is generated according to the probability model shown below.

[ ]:
```
display("image/png", read("DCM probability model.png"))
```

- $\sigma^2 \sim IG(a^0, b^0)$. We have the following :

$$P(\sigma^2|a^0, b^0) = \frac{1}{\Gamma(a^0)} b^{0^{a^0}} (\sigma^2)^{-a^0-1} e^{-\frac{b^0}{\sigma^2}}. \tag{1}$$

- $\boldsymbol{\beta}_j \in \mathbb{R}^L$, $\boldsymbol{\beta}_j|\sigma^2 \overset{\text{i.i.d.}}{\sim} N(0, \sigma^2 \boldsymbol{I}_{L\times L})$. $P(\boldsymbol{\beta}_j|\sigma^2) = (\frac{1}{2\pi\sigma^2})^{\frac{L}{2}} e^{-\frac{1}{2\sigma^2}\boldsymbol{\beta}_j^T\boldsymbol{\beta}_j}$. We have the following:

$$P(\boldsymbol{\beta}|\sigma^2) = \prod_{j=1}^{J} P(\boldsymbol{\beta}_j|\sigma^2) = (\frac{1}{2\pi\sigma^2})^{\frac{JL}{2}} e^{-\frac{1}{2\sigma^2}\sum_{j=1}^{J}\boldsymbol{\beta}_j^T\boldsymbol{\beta}_j} \tag{2}$$

- $\boldsymbol{\pi} \in [0,1]^L$, $\boldsymbol{\pi}|\mathbf{d}^0 \sim \text{Dirichlet}(\mathbf{d}^0 = (d_1^0, ..., d_L^0))$. We have the following:

$$P(\boldsymbol{\pi}|\mathbf{d}^0) = \frac{1}{Beta(\mathbf{d}^0)} \prod_{l=1}^{L} \pi_l^{d_l^0-1} \tag{3}$$

- $\mathbf{Z}_i = [z_{i1}, ..., z_{iL}]^T \in \{0,1\}^L$, $\mathbf{Z}_i|\boldsymbol{\pi} \sim \text{Categorical}(\boldsymbol{\pi})$. $P(\mathbf{Z}_i|\boldsymbol{\pi}) = \prod_{l=1}^{L} \pi_l^{z_{il}}$. We have the following:

$$P(\mathbf{Z}|\boldsymbol{\pi}) = \prod_{i=1}^{N} P(\mathbf{Z}_i|\boldsymbol{\pi}) = \prod_{i=1}^{N}\prod_{l=1}^{L} \pi_l^{z_{il}} \tag{4}$$

- $P(Y_{ij} = y_{ij}|\mathbf{z}_i, \boldsymbol{\beta}_j) = \prod_{l=1}^{L} [\sigma((2y_{ij}-1)\boldsymbol{\beta}_j^T\boldsymbol{\delta}_{jl})]^{z_{il}}$, where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function. $\boldsymbol{\Delta}_j = [\boldsymbol{\delta}_{j1}, ..., \boldsymbol{\delta}_{jL}]^T$. We have the following:

$$P(\mathbf{Y}|\mathbf{Z}, \boldsymbol{\beta}, \Delta) = \prod_{i=1}^{N}\prod_{j=1}^{J}\prod_{l=1}^{L} [\sigma((2y_{ij}-1)\boldsymbol{\beta}_j^T\boldsymbol{\delta}_{jl})]^{z_{il}} \tag{5}$$

The joint distribution is

$$P(Y, Z, \beta, \pi, \sigma^2|\Delta, d^0, a^0, b^0) = P(Y|Z, \beta, \Delta)P(Z|\pi)P(\pi|d^0)P(\beta|\sigma^2)P(\sigma^2|a^0, b^0)$$

$$= \prod_{i=1}^{N}\prod_{j=1}^{J}\prod_{l=1}^{L} [\sigma((2y_{ij}-1)\beta_j^T\delta_{jl})]^{z_{il}} \cdot \prod_{i=1}^{N}\prod_{l=1}^{L} \pi_l^{z_{il}} \cdot \frac{1}{\text{Beta}(d^0)} \prod_{l=1}^{L} \pi_l^{d_l^0-1} \cdot \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{JL}{2}} e^{-\frac{1}{2\sigma^2}\sum_{j=1}^{J}\beta_j^T\beta_j} \cdot \frac{1}{\Gamma(a^0)} (b^0)^{a^0} (\sigma^2)^{-a^0-1} e^{-\frac{b^0}{\sigma^2}}$$

The variational distribution follows the mean field family

$$Q(Z, \beta, \pi, \sigma^2) = q_1(Z)q_2(\beta)q_3(\pi)q_4(\sigma^2)$$

$$= \left(\prod_{i=1}^{N} q_{1i}(Z_i)\right)\left(\prod_{j=1}^{J} q_{2j}(\beta_j)\right) q_3(\pi)q_4(\sigma^2)$$

$$Z_i \sim \text{Categorical}(\pi_i^*) \quad \beta_j \sim N(\mu_j^*, V_j^*) \quad \pi \sim \text{Dirichlet}(d^*) \quad \sigma^2 \sim IG(a^*, b^*)$$

Noisy Gradient of $q_{1i}(Z_i)$

Since q_{1i} is a categorical distribution, it can be parameterized by logits $\rho_i^* = \{\rho_{i1}^*, ..., \rho_{iL}^*\}$ such that $\pi_{il}^* = \frac{e^{\rho_{il}^*}}{\sum_{k=1}^{L} e^{\rho_{ik}^*}}$ are obtained by the softmax transformation.

$$
\begin{aligned}
\log(q_{1i}(Z_i)) &= \sum_{l=1}^{L} z_{il} \log(\pi^*_{il}) \\
\nabla_{\rho^*_i} \log(q_{1i}(Z_i)) &= Z_i - \pi^*_i \quad \text{(Equivalent to gradient of logit w.r.t cross entropy loss)} \\
\log(p(Y, Z_{(i)})) &= \sum_{j=1}^{J} \sum_{l=1}^{L} z_{il} \log(\sigma((2y_{ij} - 1)\beta_j^T \delta_{jl})) + \sum_{l=1}^{L} z_{il} \log(\pi_l)
\end{aligned}
$$

Note that π_l are latent variables in the probability model different from π^*_{il} which are the parameters of the variational distribution.

The gradient is approximated by sampling

$$
Z_{im} \sim q_{1i}(Z_i|\pi^*_i), \quad \beta_{jm} \sim q_{2j}(\beta_j|\mu^*_j, V^*_j), \quad \pi_m \sim q_4(\pi|d^*), \quad \text{for } m = 1, 2, ..., M
$$

and taking the following average

$$
\begin{aligned}
\nabla_{\rho^*_i} L &= E_{q_{1i}(Z_i)}[\nabla_{\rho^*_{il}} \log(q_{1i}(Z_i)) \cdot (\log(p(Y, Z_{(i)})) - \log(q_{1i}(Z_i)))] \\
&\approx \frac{1}{M} \sum_{m=1}^{M} \Bigg[(Z_{im} - \pi^*_{il}) \cdot \Bigg(\sum_{j=1}^{J} \sum_{l=1}^{L} z_{iml} \log(\sigma((2y_{ij} - 1)\beta^T_{jm} \delta_{jl})) \\
&\quad + \sum_{l=1}^{L} z_{iml} \log(\pi_{ml}) - \sum_{l=1}^{L} z_{iml} \log(\pi^*_{il}) \Bigg) \Bigg]
\end{aligned}
$$

Noisy Gradient of $q_{2j}(\beta_j)$

To ensure that all gradient updates remain within the positive semi definite cone, let $V^*_j = C^*_j C^{*T}_j$ where C^*_j is the lower triangular Cholesky factor of the Cholesky decomposition of V^*_j. We will update C^*_j via noisy gradient descent.

Let k_j be the length of β_j or the number of features.

$$
\begin{aligned}
\log(q_{2j}(\beta_j)) &= -\frac{k_j}{2} \log(2\pi) - \frac{1}{2} \log(|V^*_j|) - \frac{1}{2} (\beta_j - \mu^*_j)^T (V^*_j)^{-1} (\beta_j - \mu^*_j) \\
\nabla_{\mu^*_j} \log(q_{2j}(\beta_j)) &= (V^*_j)^{-1} (\beta_j - \mu^*_j) \\
\nabla_{V^*_j} \log(q_{2j}(\beta_j)) &= \frac{1}{2} (V^*_j)^{-1} - \frac{1}{2} (V^*_j)^{-1} (\beta_j - \mu^*_j)(\beta_j - \mu^*_j)^T (V^*_j)^{-1} \\
\frac{d \log(q_{2j}(\beta_j))}{d \text{ vech}(C^*_j)} &= \frac{d \log(q_{2j}(\beta_j))}{d \text{ vec}(V^*_j)} \cdot \frac{d \text{ vec}(V^*_j)}{d \text{ vech}(C^*_j)} \\
&= \text{vec} \left(\nabla_{V^*_j} \log(q_{2j}(\beta_j)) \right) \cdot \frac{d \text{ vec}(V^*_j)}{d \text{ vech}(C^*_j)} \\
&= \text{vec} \left(\nabla_{V^*_j} \log(q_{2j}(\beta_j)) \right) \cdot \left[C^*_j \otimes I + I \otimes C^*_j K_{k_j} \right] D_{k_j} \\
&= \text{vec} \left(\frac{1}{2} (V^*_j)^{-1} - \frac{1}{2} (V^*_j)^{-1} (\beta_j - \mu^*_j)(\beta_j - \mu^*_j)^T (V^*_j)^{-1} \right) \cdot \left[C^*_j \otimes I + I \otimes C^*_j K_{k_j} \right] D_{k_j} \\
\log(p(Y, \beta_{(j)})) &= \sum_{i=1}^{N} \sum_{l=1}^{L} z_{il} \log(\sigma((2y_{ij} - 1)\beta^T_j \delta_{jl})) - \frac{1}{2\sigma^2} \beta^T_j \beta_j
\end{aligned}
$$

vec and **vech** are the vectorization and half vectorization operations for matricies. $\otimes$ is the kronecker product. K_{k_j} is the commutation matrix for the vectorization of a k_j by k_j matrix.

i.e. If $A \in R^{k_j \times k_j}$ then $K_{k_j}\text{vec}(A) = \text{vec}(A^T)$

D_{k_j} is the duplication matrix for a lower triangular k_j by k_j matrix.

i.e. If A is a lower triangular k_j by k_j matrix then $D_{k_j}\text{vech}(A) = \text{vec}(A)$ (Note: This is not the same as the duplication matrix for symmetric A)

The gradient is approximated by sampling

$$Z_{im} \sim q_{1i}(Z_i|\pi_i^*), \quad \beta_{jm} \sim q_{2j}(\beta_j|\mu_j^*, V_j^*), \quad \sigma_m^2 \sim q_4(\sigma^2|a^*, b^*), \quad \text{for } m = 1, 2, ..., M$$

and taking the following averages

$$\begin{aligned}
\nabla_{\mu_j^*} L &= E_{q_{2j}(\beta_j)}[\nabla_{\mu_j^*} \log(q_{2j}(\beta_j)) \cdot (\log(p(Y, \beta_{(j)})) - \log(q_{2j}(\beta_j)))] \\
&\approx \frac{1}{M}\sum_{m=1}^{M}\left[(V_j^*)^{-1}(\beta_{jm} - \mu_j^*) \cdot \left(\sum_{i=1}^{N}\sum_{l=1}^{L} z_{iml}\log(\sigma((2y_{ij}-1)\beta_{jm}^T\delta_{jl})) - \frac{1}{2\sigma_m^2}\beta_{jm}^T\beta_{jm}\right.\right. \\
&\left.\left. + \frac{1}{2}\log(|V_j^*|) + \frac{1}{2}(\beta_{jm} - \mu_j^*)^T(V_j^*)^{-1}(\beta_{jm} - \mu_j^*)\right)\right] \\
\frac{dL}{d\ \text{vech}(C_j)} &= E_{q_{2j}(\beta_j)}\left[\frac{d\log(q_{2j}(\beta_j))}{d\ \text{vech}(C_j)} \cdot (\log(p(Y, \beta_{(j)})) - \log(q_{2j}(\beta_j)))\right] \\
&\approx \frac{1}{M}\sum_{m=1}^{M}\left[\text{vec}\left(\frac{1}{2}(V_j^*)^{-1} - \frac{1}{2}(V_j^*)^{-1}(\beta_j - \mu_j^*)(\beta_j - \mu_j^*)^T(V_j^*)^{-1}\right) \cdot \left[C_j \otimes I + I \otimes C_j K_{k_j}\right] D_{k_j}\right. \\
&\left. \cdot \left(\sum_{i=1}^{N}\sum_{l=1}^{L} z_{iml}\log(\sigma((2y_{ij}-1)\beta_{jm}^T\delta_{jl})) - \frac{1}{2\sigma_m^2}\beta_{jm}^T\beta_{jm} + \frac{1}{2}\log(|V_j^*|) + \frac{1}{2}(\beta_{jm} - \mu_j^*)^T(V_j^*)^{-1}(\beta_{jm} - \mu_j^*)\right)\right]
\end{aligned}$$

Noisy Gradient of $q_3(\pi)$

In order to satisfy the constraints $d_l^* > 0$, $q_3(\pi)$ can be parameterized by $\delta_l^* = \log(d_l^*)$.

$$\begin{aligned}
\log(q_3(\pi)) &= \log\Gamma\left(\sum_{l=1}^{L} d_l^*\right) - \sum_{l=1}^{L}\log\Gamma(d_l^*) + \sum_{l=1}^{L}(d_l^*)\log(\pi_l) \\
\nabla_{\delta_k^*}\log(q_3(\pi)) &= \nabla_{d_k^*}\log(q_3(\pi)) \cdot \nabla_{\delta_l^*} d_l^* \\
&= \left(\psi\left(\sum_{l=1}^{L} d_l^*\right) - \psi(d_k^*) + \log(\pi_k)\right) \cdot d_l^* \\
\log(p(Y, \pi)) &= \sum_{i=1}^{N}\sum_{l=1}^{L} z_{il}\log(\pi_l) + \sum_{l=1}^{L}(d_l^0 - 1)\log(\pi_l)
\end{aligned}$$

$\Gamma(\cdot)$ is the gamma function and $\psi(\cdot)$ is the digamma function.

The gradient is approximated by sampling

$$Z_{im} \sim q_{1i}(Z_i|\pi_i^*), \quad \pi_m \sim q_3(\pi_m|d^*), \quad \text{for } m = 1, 2, ..., M$$

and taking the following average

$$\begin{aligned}\nabla_{\delta_k^*} L &= E_{q_3(\pi)}[\nabla_{\delta_l^*} \log(q_3(\pi)) \cdot (\log(p(Y,\pi)) - \log(q_3(\pi)))] \\ &\approx \frac{1}{M}\sum_{m=1}^{M}\Bigg[\left(\psi\left(\sum_{l=1}^{L} d_l^*\right) - \psi(d_k^*) + \log(\pi_{mk})\right)\cdot d_l^* \\ &\cdot \left(\sum_{i=1}^{N}\sum_{l=1}^{L} z_{iml}\log(\pi_{ml}) + \sum_{l=1}^{L}(d_l^0 - 1)\log(\pi_{ml}) - \log\Gamma\left(\sum_{l=1}^{L} d_l^*\right) + \sum_{l=1}^{L}\log\Gamma(d_l^*) - \sum_{l=1}^{L}(d_l^*)\log(\pi_{ml})\right)\Bigg]\end{aligned}$$

Noisy Gradient of $q_4(\sigma^2)$

In order to satisfy the constraints $a^* > 0$ and $b^* > 0$, we will parameterize $q_4(\sigma^2)$ by $\alpha^* = \log(a^*)$ and $c^* = \log(b^*)$.

$$\begin{aligned}\log(q_4(\sigma^2)) &= a^*\log(b^*) - \log\Gamma(a^*) - (a^*+1)\log(\sigma^2) - \frac{b^*}{\sigma^2} \\ \nabla_{\alpha^*}\log(q_4(\sigma^2)) &= (\log(b^*) - \psi(a^*) - \log(\sigma^2))\cdot a^* \\ \nabla_{c^*}\log(q_4(\sigma^2)) &= \left(\frac{a^*}{b^*} - \frac{1}{\sigma^2}\right)\cdot b^* \\ \log(p(Y,\sigma^2)) &= -\frac{JL}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{j=1}^{J}\beta_j^T\beta_j - (a^0+1)\log(\sigma^2) - \frac{b^0}{\sigma^2}\end{aligned}$$

$\Gamma(\cdot)$ is the gamma function and $\psi(\cdot)$ is the digamma function.

The gradient is approximated by sampling

$$\beta_{jm} \sim q_{2j}(\beta_j|\mu_j^*, V_j^*), \quad \sigma_m^2 \sim q_4(\sigma^2|a^*, b^*), \quad \text{for } m = 1, 2, ..., M$$

and taking the following averages

$$\begin{aligned}\nabla_{\alpha^*} L &= E_{q_4(\sigma^2)}[\nabla_{\alpha^*}\log(q_4(\sigma^2))\cdot(\log(p(Y,\sigma^2)) - \log(q_4(\sigma^2)))] \\ &\approx \frac{1}{M}\sum_{m=1}^{M}\Bigg[(\log(b^*) - \psi(a^*) - \log(\sigma^2))\cdot a^*\cdot\Bigg(-\frac{JL}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{j=1}^{J}\beta_j^T\beta_j - (a^0+1)\log(\sigma^2) - \frac{b^0}{\sigma^2} \\ &-a^*\log(b^*) + \log\Gamma(a^*) + (a^*+1)\log(\sigma^2) + \frac{b^*}{\sigma^2}\Bigg)\Bigg] \\ \nabla_{c^*} L &= E_{q_4(\sigma^2)}[\nabla_{c^*}\log(q_4(\sigma^2))\cdot(\log(p(Y,\sigma^2)) - \log(q_4(\sigma^2)))] \\ &\approx \frac{1}{M}\sum_{m=1}^{M}\Bigg[\left(\frac{a^*}{b^*} - \frac{1}{\sigma^2}\right)\cdot b^*\cdot\Bigg(-\frac{JL}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{j=1}^{J}\beta_j^T\beta_j - (a^0+1)\log(\sigma^2) - \frac{b^0}{\sigma^2} \\ &-a^*\log(b^*) + \log\Gamma(a^*) + (a^*+1)\log(\sigma^2) + \frac{b^*}{\sigma^2}\Bigg)\Bigg]\end{aligned}$$

[ ]:
```
using LinearAlgebra
include("BBVI_modeling.jl")
```

[ ]:
```
using RCall
```

```
R"""
load("data.RData")
"""
data = @rget data
Y = data[:Y]
Q = convert(Matrix{Int64}, data[:Q])
obs = DCMObs(Y, Q);
```

Initialize at true beta values

[ ]:
```
a0 = 1e-2
b0 = 1e-4
d0 = ones(size(obs.D[1], 1))
M = 1000
model = DCModel(obs, d0, a0, b0, M)

# Set beta to true values
for j in 1:30
    model.mu_star[j] .= data[:beta][j]
end
```

[ ]:
```
K = 1
for k in 1:K
    update_pi_star(model, step = 1e-2, maxiter = 100, verbose = false)
    # update_mu_star_V_star(model, init_step = 0.5, use_iter = true, maxiter =
 ↪20, verbose = true)
    update_d_star(model, step = 1e-5, maxiter = 20, verbose = false)
    update_a_star_b_star(model, step = 1e-5, maxiter = 30, verbose = false)
end
```

[ ]:
```
skill_profiles = Dict(1=>"00",
                      3=>"01",
                      2=>"10",
                      4=>"11")

pred = []
for i in 1:1000
    push!(pred, skill_profiles[argmax(model.pi_star[i])])
end
```

[ ]:
```
mean(pred .== data[:skill])
```

0.979

Initialize at true pi values

[ ]:
```
model = DCModel(obs, d0, a0, b0, M)

skill_to_pi = Dict("00"=>[.997, .001, .001, .001],
```

```
                "01"=>[.001, .001, .997, .001],
                "10"=>[.001, .997, .001, .001],
                "11"=>[.001, .001, .001, .997])

for i in 1:1000
    model.pi_star[i] .= skill_to_pi[data[:skill][i]]
end
```

[ ]:
```
K = 50
for k in 1:K
    # update_pi_star(model, step = 1e-2, maxiter = 100, verbose = true)
    update_mu_star_V_star(model, init_step = 0.5, use_iter = true, maxiter =␣
 ↪20, verbose = false)
    update_d_star(model, step = 1e-5, maxiter = 20, verbose = false)
    update_a_star_b_star(model, step = 1e-5, maxiter = 30, verbose = false)
end
```

[ ]:
```
model.mu_star
```

```
30-element Vector{Vector{Float64}}:
 [-1.2392690588369464, 1.1493650114315601, 1.7512313356322096, 1.
 ↪1176784213692486]
 [-0.7990398823688503, 1.0865075970622178, 1.325898346229483, 0.7993398049553667]
 [-0.9126727171272198, 2.436129551316525]
 [-0.9996775897779946, 2.3100380616916203]
 [-1.3730853329242618]
 [-0.7123550619661712, 2.0713817059133057]
 [-0.41605365714776327, 0.7875215265636807, 1.0730773134513716, 0.
 ↪7946779115236593]
 [-0.45160994999733545, 0.8077240390280532, 0.8972487844682199, 0.
 ↪8246447613602793]
 [-0.9675184636662003]
 [-0.5038747940747368, 1.8136138520453056]

 [-0.10409842377696393, 1.0265670977509598]
 [-0.05767227900697194, 0.9963820521134111]
 [-0.8228129781973679]
 [-0.04322084990364942, 0.9309274085123161]
 [-0.06006169125223673, 0.9001484394542788]
 [-0.08116822348669141, 0.8780180560444568]
 [0.028327620869163662, 0.8364717996310813]
 [-0.8168393427161795]
 [0.006011971529259193, 0.45005224113588427, 0.4058263110240864, 0.
 ↪3908362761767494]
```

[ ]:
```
data[:beta]
```

```
30-element Vector{Any}:
```

```
  [-1.0908464698819444, 1.4969379037618638, 1.4836376605555415, 0.
↪5467837428674103]
  [-1.085415811627172, 1.4696301169227808, 1.4857180385151878, 0.
↪5013018082128838]
  [-1.0617003259481863, 2.9305335130542516]
  [-1.1269538402557373, 2.9325977156870064]
-1.0885653140954674
  [-1.0998363556107507, 2.9376237776130436]
  [-1.1092285147402434, 1.5082388046896085, 1.480892906500958, 0.
↪4980965640163049]
  [-1.056060520769097, 1.454604054824449, 1.5034861729713156, 0.
↪49426607822533697]
-1.059363453043625
  [-1.0618957618251443, 2.9326482232660056]

  [-1.1135568148689343, 3.0277511939406394]
  [-1.1467085672542452, 3.039238517684862]
-1.0684435825329275
  [-1.0992577146505937, 2.9225185251794756]
  [-1.0593676698859782, 2.982667790632695]
  [-1.1026244212174787, 3.042742836009711]
  [-1.1258128789486364, 2.9599281255155803]
-1.0795521178515628
  [-1.0728325051954015, 1.4513840340543538, 1.5051313953474164, 0.
↪5152461190475152]
```

Learn full model

[ ]:
```
model = DCModel(obs, d0, a0, b0, M);
```

[ ]:
```
# 200 iterations of coordinate ascent
K = 200
for k in 1:K
    update_pi_star(model, step = 1e-2, maxiter = 10, verbose = false)
    update_mu_star_V_star(model, init_step = 1.0, use_iter = true, maxiter =␣
 ↪20, verbose = false)
    update_d_star(model, step = 1e-5, maxiter = 20, verbose = false)
    update_a_star_b_star(model, step = 1e-5, maxiter = 30, verbose = false)
end
```

[ ]:
```
# Overall classification accuracy

skill_profiles = Dict(1=>"00",
                      3=>"01",
                      2=>"10",
                      4=>"11")

pred = []
```

```
for i in 1:1000
    push!(pred, skill_profiles[argmax(model.pi_star[i])])
end

mean(pred .== data[:skill])
```

0.967

[ ]:
```
# Intra-class classification accuracy

mean(pred[data[:skill] .== "00"] .== "00")
```

0.9959677419354839

[ ]:
```
mean(pred[data[:skill] .== "01"] .== "01")
```

0.9590163934426229

[ ]:
```
mean(pred[data[:skill] .== "10"] .== "10")
```

0.9384057971014492

[ ]:
```
mean(pred[data[:skill] .== "11"] .== "11")
```

0.978448275862069

[ ]:
```
# Misclassified pi vectors
# Correct classification always in top 2

model.pi_star[data[:skill] .== "00"][pred[data[:skill] .== "00"] .!= "00"]
```

```
1-element Vector{Vector{Float64}}:
 [0.11050727963043894, 0.8636977468400452, 0.014962901810850172, 0.
 ↪010832071718665707]
```

[ ]:
```
model.pi_star[data[:skill] .== "01"][pred[data[:skill] .== "01"] .!= "01"]
```

```
10-element Vector{Vector{Float64}}:
 [0.7004468385850918, 0.018764461584782913, 0.27242772277178917, 0.
 ↪008360977058336057]
 [0.011633141639797995, 0.015065188140715561, 0.13711150394878938, 0.
 ↪8361901662706971]
 [0.548057987672103, 0.0301620378939421, 0.40762109733362784, 0.
 ↪014158877100327115]
 [0.007845531201806066, 0.00802957365801706, 0.2808369442656743, 0.
 ↪7032879508745026]
 [0.5078916649162213, 0.01018309350766702, 0.4756103528790468, 0.
 ↪006314888697064903]
 [0.014813810156742368, 0.015830657024895726, 0.458978812425687, 0.
 ↪5103767203926749]
```

```
 [0.023469807622752237, 0.04428908642234675, 0.24169656954576832, 0.
 ↪6905445364091327]
 [0.011523723991044786, 0.01867850819041867, 0.10064325993324119, 0.
 ↪8691545078852954]
 [0.007385176762989079, 0.007722907417462996, 0.20639542783893944, 0.
 ↪7784964879806086]
 [0.9544264957616442, 0.013165813434936972, 0.028379371788991224, 0.
 ↪004028319014427698]
```

[ ]:
```
model.pi_star[data[:skill] .== "10"][pred[data[:skill] .== "10"] .!= "10"]
```

```
17-element Vector{Vector{Float64}}:
 [0.4918426400646139, 0.4846536190292401, 0.015687989449590396, 0.
 ↪00781575145655558]
 [0.9476237560321342, 0.03827416237911564, 0.009523526102202128, 0.
 ↪0045785554865481]
 [0.6122806714797372, 0.35136562132053606, 0.023381591746580728, 0.
 ↪012972115453146017]
 [0.5315996648709657, 0.38439146403117386, 0.06062905961984336, 0.
 ↪023379811478017175]
 [0.7995857899965878, 0.17732839142312307, 0.015549705040035795, 0.
 ↪007536113540253223]
 [0.8621392385109086, 0.12550739345052686, 0.00790094487995615, 0.
 ↪004452423158608379]
 [0.8351968346794238, 0.1391766290173688, 0.018117897798648964, 0.
 ↪007508638504558506]
 [0.7486044795202037, 0.21762250826572738, 0.02442090952684559, 0.
 ↪009352102687223163]
 [0.6730688579341072, 0.2769115277062606, 0.03329331791197709, 0.
 ↪016726296447654988]
 [0.9106621590518936, 0.07254111534930509, 0.011495182362208698, 0.
 ↪005301543236592735]
 [0.8073252809644657, 0.16840942469278056, 0.016074560685245834, 0.
 ↪008190733657507873]
 [0.5830787607700761, 0.39818115777163815, 0.011644352026317152, 0.
 ↪007095729431968566]
 [0.4793796829951817, 0.4723754438822575, 0.035263317653762095, 0.
 ↪01298155546879874]
 [0.9607950961818639, 0.02457455270065659, 0.01047951297884968, 0.
 ↪00415083813862979]
 [0.8176595990195008, 0.17151550840303817, 0.006569217423809092 4, 0.
 ↪004255675153651978]
 [0.6046365559928486, 0.33765802570624276, 0.041960906443975265, 0.
 ↪015744511856933402]
 [0.5065608389776936, 0.47230732485696875, 0.013670094615144521, 0.
 ↪007461741550193129]
```

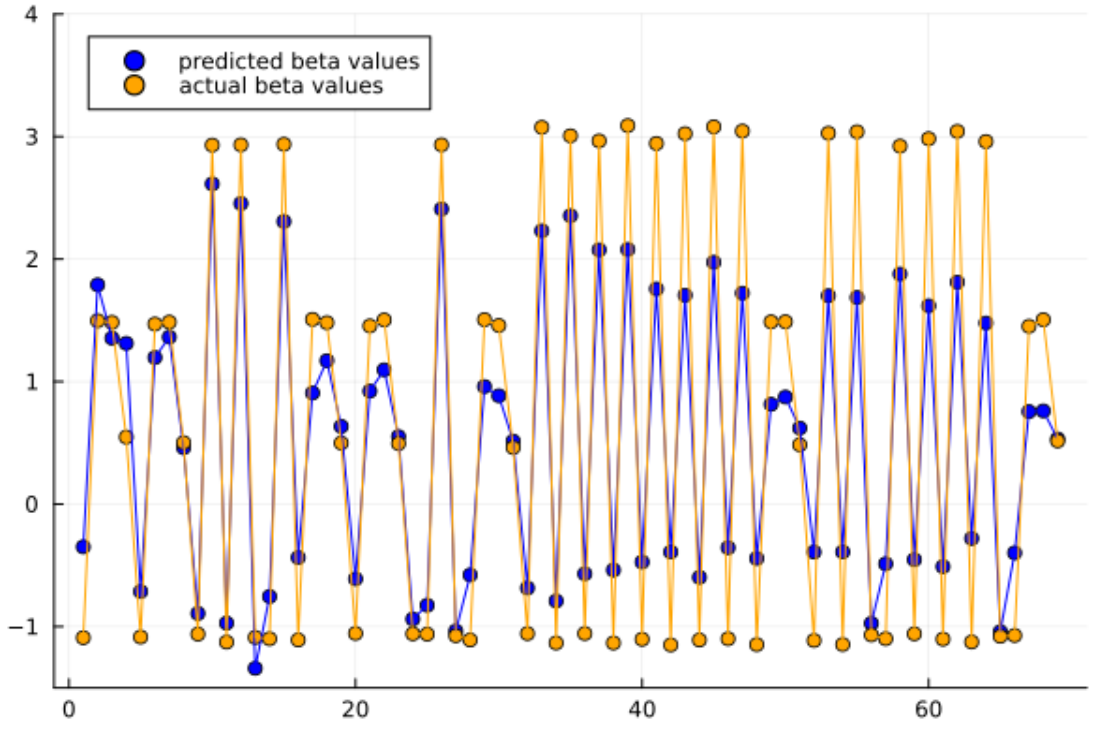
[ ]:
```
model.pi_star[data[:skill] .== "11"][pred[data[:skill] .== "11"] .!= "11"]
```

```
5-element Vector{Vector{Float64}}:
 [0.010253689988555106, 0.00742445959578943, 0.8531790774241982, 0.
 ↪12914277299145718]
 [0.10511585928640647, 0.3882644668814505, 0.22396531395236693, 0.
 ↪2826543598797761]
 [0.027652122875724532, 0.03174582005390812, 0.542599563099999, 0.
 ↪39800249397036846]
 [0.031008924757590428, 0.03752245387659398, 0.5352040326429194, 0.
 ↪3962645887228963]
 [0.06275850925949514, 0.5016351769924267, 0.11927637595812933, 0.
 ↪31632993778994883]
```

[ ]:
```
pred_beta_values = []
actual_beta_values = []
for j in 1:30
    for val in model.mu_star[j]
        push!(pred_beta_values, val)
    end
    for val in data[:beta][j]
        push!(actual_beta_values, val)
    end
end
```

[ ]:
```
using Plots

x = 1:69
plot(x, pred_beta_values, seriestype=:scatter, label = "predicted beta values",␣
 ↪mc=:blue)
plot!(x, actual_beta_values, seriestype=:scatter, label = "actual beta values",␣
 ↪mc=:orange)
plot!(x, pred_beta_values, label = "", lc=:blue)
plot!(x, actual_beta_values, label = "", lc=:orange)
ylims!(-1.5, 4)
```



[ ]:
```
using JLD2

save_object("BBVI_model_final.jld2", model)
save_object("BBVI_pi_star.jld2", model.pi_star)
save_object("BBVI_mu_star.jld2", model.mu_star)
```