Ordinal tensor data analysis

Chanwoo Lee

1 Ordinal tensor data

1.1 Modeling

For the ordinary tensor $Y = [y_{i_1,\cdots,i_N}] \in \{1, 2, \cdots, K\}^{d_1 \times \cdots \times d_N}$ we will use cumulative logit model. we assume it's entries are realization of independent random variables, such that, for all $(i_1, \cdots, i_N) \in [d_1] \times \cdots \times [d_N]$,

$$P(y_{i_1,\cdots,i_N} \leq j | \theta_{i_1,\cdots,i_N}) = \pi_1(\theta_{i_1,\cdots,i_N}) + \cdots + \pi_j(\theta_{i_1,\cdots,i_N})$$

To elaborate this more,

$$\begin{aligned} logit(P(y_{i_1,\cdots,i_N} \leq j | \theta_{i_1,\cdots,i_N})) &= \log \frac{P(y_{i_1,\cdots,i_N} < j | \theta_{i_1,\cdots,i_N})}{1 - P(y_{i_1,\cdots,i_N} \leq j | \theta_{i_1,\cdots,i_N})} = \log \frac{\pi_1(\theta_{i_1,\cdots,i_N}) + \cdots + \pi_j(\theta_{i_1,\cdots,i_N})}{\pi_{j+1} + \cdots + \pi_K(\theta_{i_1,\cdots,i_N})} \\ &= \alpha_j + \theta_{i_1,\cdots,i_N} \end{aligned}$$

where α_j is non-decreasing with regards to $j \in \{1, \cdots, K\}$ and we assume Θ has Tucker decomposition structure such that $\Theta = \mathcal{C} \times_1 A_1 \times_2 A_2 \cdots \times_N A_N$ where $\mathcal{C} \in \mathcal{R}^{r_1 \times \cdots \times r_N}$ and $A_i \in \mathcal{R}^{d_i \times r_i}$ where rank $r_i < d_i$ We also assume that the entries of $\mathcal{Y}$ are mutually independent conditional on Θ.

There is much better way to put this model adding one dimension to a given ordinal tensor. Let

$$Z_1 = I(Y = 1), \quad Z_2 = I(Y \leq 2), \quad \cdots \quad , Z_{K-1} = I(Y \leq K - 1)$$

and let's stack $Z_1, \cdots, Z_{K-1}$ together and yield an order $d_1 \times \cdots \times d_n \times (K-1)$ tensor Z such that $Z_{:,\cdots,:,i} = Z_i$ then our model is equivalent to following model.

$$f^{-1}(E(Z)) = 1_{d_1} \circ 1_{d_2} \circ \cdots \circ 1_{d_N} \circ \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{K-1} \end{pmatrix} + \Theta \circ 1_{K-1}$$

where $1_j = \underbrace{(1, \cdots, 1)^T}_{j}$ and f^{-1} is logit link.

Based on above formula, we can get another equivalent model using noise tensor. I would

call this model threshold model.

Consider an order-N+1 tensor such that $\tilde{\Theta} = 1_{d_1} \circ 1_{d_2} \circ \cdots \circ 1_{d_N} \circ \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{K-1} \end{pmatrix} + \Theta \circ 1_{K-1}$ and suppose that we cannot directly observe $\tilde{\Theta}$. Instead, we observe the quantized version $Z = [z_{i_1,\cdots,i_N,j}] \in \{0,1\}^{d_1 \times \cdots \times d_N \times K-1}$ following the realization of the scheme

$$z_{i_1,\cdots,i_N,j} = \begin{cases} 1 & \tilde{\theta}_{i_1,\cdots,i_N,j} + \epsilon_{i_1,\cdots,i_N,j} \geq 0 \\ 0 & \tilde{\theta}_{i_1,\cdots,i_N,j} + \epsilon_{i_1,\cdots,i_N,j} < 0 \end{cases}$$

where $\mathcal{E} = [\epsilon_{i_1,\cdots,i_N,j}]$ is a noise tensor. If we use an indicator function we can make above function more succinct $Z = \mathbb{1}(\tilde{\Theta} + \mathcal{E})$

Now let's show why above two become same.
First, let a noise tensor $\mathcal{E}$ is drawn from cumulative distribution of f defined by $P(\epsilon < \theta) = 1 - f(-\theta)$. Then,

$$\begin{aligned} P(z_{i_1,\cdots,i_N,j} = 1) &= P(\tilde{\theta}_{i_1,\cdots,i_N,j} + \epsilon_{i_1,\cdots,i_N,j} \geq 0) = P(\epsilon_{i_1,\cdots,i_N,j} \geq -\tilde{\theta}_{i_1,\cdots,i_N,j}) \\ &= 1 - (1 - f(\tilde{\theta}_{i_1,\cdots,i_N,j})) \\ &= f(\tilde{\theta}_{i_1,\cdots,i_N,j}) \end{aligned}$$

Secondly, define link function $f(\theta) = P(\epsilon \geq -\theta)$. Then,

$$\begin{aligned} P(z_{i_1,\cdots,i_N,j} = 1) &= P(\tilde{\theta}_{i_1,\cdots,i_N,j} + \epsilon_{i_1,\cdots,i_N,j} \geq 0) = P(\epsilon_{i_1,\cdots,i_N,j} \geq -\tilde{\theta}_{i_1,\cdots,i_N,j}) \\ &= f(\tilde{\theta}_{i_1,\cdots,i_N,j}) \end{aligned}$$

Therefore, we can use a noise related to cumulative distribution and can estimate our parameters using Threshold model.

There are 2 ways to estimate parameters Θ, and α based on which types of data we are going to use.

1.2 Estimation: For binary tensor Z

1.2.1 Loglikelihood based estimation

When we use Z which is made by stacking tensors $Z_i = I(Y \leq i)$, whole estimation problem becomes optimization problem for binary tensor:
we estimate the unknown parameter tensor $\tilde{\theta}$ using a likelihood loss function. The log-

likelihood function is

$$\begin{aligned}\mathcal{L}_Z(\Theta,\alpha) &= \sum_{i_1,\cdots,i_N,j}\Big[\mathbb{1}(z_{i_1,\cdots,i_N,j}=1)\log f(\tilde{\theta}_{i_1,\cdots,i_N,j}) + \mathbb{1}(z_{i_1,\cdots,i_N,j}=0)\log f(-\tilde{\theta}_{i_1,\cdots,i_N,j})\Big] \\ &= \sum_{i_1,\cdots,i_N,j}\log f((2z_{i_1,\cdots,i_N,j}-1)\tilde{\theta}_{i_1,\cdots,i_N,j})\end{aligned}$$

So our goal is to find parameters optimizing log-likelihood function constrained in our target parameter space $\mathcal{D}$:

$$\max_{(\Theta,\alpha)\in\mathcal{D}}\mathcal{L}_Z(\Theta,\alpha) \quad \text{where } \mathcal{D}=\{(\Theta,\alpha):\alpha_1\le\cdots\le\alpha_{K-1},\Theta=\mathcal{C}\times_1 A_1\times_2 A_2\cdots\times_N A_N,\|\Theta\|_\infty\le\alpha\}$$

for $\mathcal{C}\in\mathcal{R}^{r_1\times\cdots\times r_N}, A_i\in\mathcal{R}^{d_i\times r_i}$ and $r_i<d_i$

1.2.2 Optimization

In this section, we describe the algorithm which can be used to solve above optimization problem. We utilized a formulation of tucker decomposition, and turn the optimization into a block wise convex problem. It has really similar approach in the paper *Learning from Binary Multiway Data: Probabilistic Tensor Decomposition and its Statistical Optimality* where CP decomposition is used to make optimization problem into block-wise convex problem. First, suppose we already know $\alpha_1,\cdots,\alpha_{K-1}$ and decompose $\Theta=\mathcal{C}\times_1 A_1\cdots\times_N A_N$ to have N+1 blocks. Then our optimization problem of maximizing $\mathcal{L}_Z(\Theta,\alpha)$ becomes $\mathcal{L}_Z(\mathcal{C},A_1,\cdots,A_N)$ If any N out of the N + 1 blocks of variables are known, then the optimization with respect to the last block of variables reduced to a simple GLM. For example, suppose we know all blocks except A_j. Then, putting $X_{(j)}=unfold(\mathcal{C}\times_1 A_1\cdots\times_{j-1}A_{j-1}\times_{j+1}A_{j+1}\cdots\times_N A_N)$

$$Vec(\mathcal{Y})\sim Bernoulli(Vec((f(A_jX_{(j)}))))$$

Therefore, we can calculate optimal A_j given other variables.
Also, we can optimize $L_Z(\Theta,\alpha)$ with regards to α, given Θ. Therefore, our algorithm is as follows.

Algorithm 1 Binary tensor optimization using block-coordinate method

Input: Stacked tensor Z based on observed entries of $y_{i_1,\cdots,i_N}$ and initialization of an increasing sequence $\alpha_1^0,\cdots,\alpha_{K-1}^0\in\mathbf{R}$ and $\mathcal{C}^0\in\mathbf{R}^{r_1\times\cdots\times r_N}, A_1^0\in\mathbf{R}^{d_1\times r_1},\cdots,A_N^0\in\mathbf{R}^{d_N\times r_N}$

Output: Optimizor of $\mathcal{L}_Z(\alpha,\Theta)$

for t = 1,2,$\cdots$, **do** until convergence,
  $\mathcal{C}^{t+1}=\arg\max\mathcal{L}_Z(\alpha^t,A_1^t,\cdots,A_N^t)$
  for j = 1,2,$\cdots$,N **do**
    $A_j^{t+1}=\arg\max\mathcal{L}_Z(\alpha^t,\mathcal{C}^{t+1},A_1^{t+1},\cdots,A_{j-1}^{t+1},A_{j+1}^t,\cdots,A_N^t)$
  for i = 1,2,$\cdots$,K-1 **do**
    $\alpha_i^{t+1}=\arg\max\mathcal{L}_Z(\alpha_1^{t+1},\cdots,\alpha_{i-1}^{t+1},\alpha_{i+1}^t\cdots,\alpha_{K-1}^t,\Theta^{t+1})$
return α,Θ

Last thing I want to comment is that there is no convergence guarantees in general.

1.3 Estimation: For ordinal tensor Y

If we stick to use ordinal tensor Y then it becomes

$$\mathcal{L}_{\mathcal{Y}}(\Theta, \alpha) = -\sum_{i_1,\cdots,i_N} \Big[\sum_{l=1}^{K} \mathbb{1}(y_{i_1,\cdots,i_N} = l) \log(\pi_l(\theta, \alpha)\Big]$$

Where $\pi_l = P(Y_{i_1,\cdots,i_N} = l|\theta_{i_1,\cdots,i_N}, \alpha) = logit(\alpha_l + \theta_{i_1,\cdots,i_N}) - logit(\alpha_{l-1} + \theta_{i_1,\cdots,i_N})$
So our goal is to find parameters optimizing log-likelihood function constrained in our target parameter space $\mathcal{D}$:

$$\max_{(\Theta,\alpha)\in\mathcal{D}} \mathcal{L}_{\mathcal{Y}}(\Theta, \alpha) \quad \text{where } \mathcal{D} = \{(\Theta, \alpha) : \alpha_1 \leq \cdots \leq \alpha_{K-1}, \Theta = \mathcal{C}\times_1 A_1 \times_2 A_2 \cdots \times_N A_N, \|\Theta\|_\infty \leq \alpha\}$$

for $\mathcal{C} \in \mathcal{R}^{r_1\times\cdots\times r_N}, A_i \in \mathcal{R}^{d_i\times r_i}$ and $r_i < d_i$

Algorithmic approach for this setting is not that different from section 1.2.2. Only difference between this setting and setting in section 1.2 is loss function we use.

2 Probabilistic algorithm for Tensor approximation.

2.1 Accuracy

We are going to see how accurate our svd tensor approximation is by checking angle between a real vector and estimated vector. Let data A be $d_1 \times d_2$ matrix such that $A = \lambda a \circ b$ where λ is scalar and $\|a\|_F = \|b\|_F = 1$. Our given data is $D = A + E$ where elements of E is independently drawn from $N(0, \sigma^2)$.
Let Ω be $d_2 \times 1$ random vector drawn from $N(0, 1)$ and $\hat{a} = D\Omega = A\Omega + E\Omega$ then we can get a following theorem.

Theorem 1. *If $\lambda > \sigma\sqrt{2 \times d_2 \times max(d_1, d_2)}$ then, $\cos\theta(a, \hat{a})$ converges to 1 in probability*

I couldn't prove above Theorem 1 well. Instead, I proved a similar proposition denoted by Theorem 2. I needed following 3 lemmas to prove Theorem 2.

Lemma 1. *$\|A\Omega\| = \lambda\|\Omega\||Z_1|$ where $Z_1 \sim N(0.1)$*

Proof. Notice $A\Omega = \lambda(b_1\Omega_1 + \cdots + b_{d_2}\Omega_{d_2})a = \lambda\|\Omega\|N(0, 1)a$ □

Lemma 2. *$\|E\Omega\| = \|\Omega\|\sqrt{\sigma^2 V}$ where $V \sim \chi^2(d_1)$*

Proof.

$$\begin{aligned}\|E\Omega\| &= \sqrt{\sum_{i=1}^{d_1}(\sum_{j=1}^{d_2}\Omega_j E_{ij})^2} \\ &= \sqrt{\sum_{i=1}^{d_1}(\sum_{j=1}^{d_2}N_{ij}(0,\sigma^2\Omega_j))^2} = \sqrt{\sum_{i=1}^{d_1}(\sum_{j=1}^{d_2}\Omega_j^2)N_{i.}(0,\sigma^2)^2} = \sqrt{\sum_{j=1}^{d_2}\Omega_j^2\sum_{i=1}^{d_1}\sigma^2\chi_i^2(1)} \quad \text{for given } \Omega \\ &= \sqrt{\sum_{j=1}^{d_2}\Omega_j^2\sigma^2\chi^2(d_1)}\end{aligned}$$

Since this equality holds for all given Ω, we can say it holds for random variables □

Lemma 3. $\langle A\Omega, E\Omega\rangle = \|A\Omega\|\|\Omega\|\sigma Z_2 \quad$ *where* $Z_2 \sim N(0,1)$

Proof.

$$\begin{aligned}\langle A\Omega, E\Omega\rangle &= \|A\Omega\|\langle a, E\Omega\rangle = \|A\Omega\|\sum_{j=1}^{d_2}\Omega_j\langle E_{.,j}, a\rangle \\ &= \|A\Omega\|\sum_{j=1}^{d_2}\Omega_j\sum_{i=1}^{d_1}E_{ij}a_i = \|A\Omega\|\sum_{j=1}^{d_2}\Omega_j N_j(0,\sigma^2) \\ &= \|A\Omega\|\sum_{j=1}^{d_2}N_j(0,\sigma^2\Omega_j^2) = \|A\Omega\|N(0,\sigma^2)\sqrt{\sum_{j=1}^{d_2}\Omega_j^2}\end{aligned}$$

□

Now my Theorem 2 is like this.

Theorem 2. *If* $\sigma = o(\lambda)$ *then,* $\cos\Theta(\hat{a}, a) \to 1$

Proof. Notice,

$$\begin{aligned}\cos\Theta(\hat{a}, a) &= \frac{\langle A\Omega, A\Omega + E\Omega\rangle}{\|A\Omega\|\|A\Omega + E\Omega\|} = \frac{\|A\Omega\|^2 + \langle A\Omega, E\Omega\rangle}{\|A\Omega\|\|A\Omega + E\Omega\|} \\ &= \frac{\|A\Omega\|^2 + \langle A\Omega, E\Omega\rangle}{\|A\Omega\|\sqrt{\|A\Omega\|^2 + \|E\Omega\|^2 + 2\langle A\Omega, E\Omega\rangle}}\end{aligned}$$

Then, by plugging above lemmas into above fraction, we can get

$$\begin{aligned}&= \frac{\lambda|Z_1| + \sigma Z_2}{\sqrt{\lambda^2 Z_1^2 + \sigma^2 V + 2\lambda\sigma|Z_1|Z_2}} = \frac{|Z_1| + (\frac{\sigma}{\lambda})Z_2}{\sqrt{Z_1^2 + (\frac{\sigma}{\lambda})^2 V + 2(\frac{\sigma}{\lambda})|Z_1|Z_2}} \\ &\to 1\end{aligned}$$

2.2 Comparison between 2 algorithms with regards to computational costs

Let's review following 2 algorithms

Algorithm 2 Approx tensor SVD 1

1: **procedure** SVD($\mathcal{A}$)
2: **Step A: Approximate SVD of $\mathcal{A}$**
3: **for** $n \leftarrow 1 : N$ **do**
4: Unfold $\mathcal{A}$ as $A_{(n)}$
5: Generate an $I_1 \cdots I_{n-1} I_{n+1} \cdots I_N \times (k_i + p)$ Gaussian test matrix $\Omega^{(n)}$
6: For $\mathbf{Y^{(n)}} = \mathbf{A_{(n)}} \Omega^{(n)}$
7: Construct a matrix $\mathbf{Q}^{(n)}$ whose columns form an orthonormal basis for the range of $\mathbf{Y^{(n)}}$
8: Form $\mathbf{P_{Y^{(n)}}} = \mathbf{Q^{(n)}} \mathbf{Q^{(n)*}}$
9: get $\hat{\mathcal{A}} = \mathcal{A} \times_1 P_{Y^{(1)}} \times_2 P_{Y^{(2)}} \cdots \times_N P_{Y^{(N)}}$
10: **Step B: Get approximated SVD**
11: $\mathcal{S} = \mathcal{A} \times_1 Q^{(1)*} \times_2 Q^{(2)*} \cdots \times_N Q^{(N)*}$
12: **for** $i \leftarrow 1 : N$ **do**
13: $U^{(i)} = Q^{(i)}$
14: **return** $(\mathcal{S}, U^{(1)} \cdots U^{(n)})$

Algorithm 3 Approx tensor SVD 2

1: **procedure** SVD($\mathcal{A}$)
2: **Step A: Approximate SVD of $\mathcal{A}$**
3: **for** $i \leftarrow 1 : N$ **do**
4: Get Gaussian test matrix Ω_n whose size is $I_i \times (k_i + p)$
5: Form $A^{(i)} = \text{Unfold}_i(\mathcal{A} \times_1 \Omega_1^* \times \cdots \times_{i-1} \Omega_{i-1}^* \times_{i+1} \Omega_{i+1}^* \times \cdots \times_N \Omega_N^*)$
6: Find a matrix $Q^{(i)}$ whose size is $\prod_{j \neq i}(k_j + p) \times (k_i + p)$ using $\mathbf{Y^{(i)}} = \mathbf{A_{(i)}} \Omega^{(i)}$
7: $U^{(i)} = Q^{(i)}$
8: get $\mathcal{S} = \mathcal{A} \times_1 Q^{(1)*} \times_2 Q^{(2)*} \cdots \times_N Q^{(N)*}$
9: **return** $(\mathcal{S}, U^{(1)} \cdots U^{(n)})$

□

For simplicity, let $\mathcal{A} \in \mathbf{R}^{d \times \cdots \times d}$ with N-mode and our approximating rank of $\mathcal{A}$ be k for each mode. Also only difference between 2 algorithms is in Step A so it's enough to compute complexity in Step A.

1. 1st method.
   - $d^N k$ flops for matrix product (Algorithm 2: line 6)
   - $2dk^2 - \frac{2}{3}k^3$ flops for each QR decomposition (Algorithm 2: line 7)

- $Nd^Nk + N(2dk^2 - \frac{2}{3}k^3)$ flops for total

2. 2nd method.
   - $d^Nk + d^{N-1}k^2 + \cdots + d^2k^{N-1} = \frac{d^Nk(1-(\frac{k}{d})^{N-1})}{1-\frac{k}{d}}$ flops for tensor mode product (Algorithm 3: line 5)
   - dk^N flops for matrix product (Algorithm 3: line 6)
   - $2dk^2 - \frac{2}{3}k^3$ flops for each QR decomposition (Algorithm 3: line 6)
   - $N\frac{d^Nk(1-(\frac{k}{d})^{N-1})}{1-\frac{k}{d}} + Ndk^N + N(2dk^2 - \frac{2}{3}k^3)$ flops for total

Note that two methods pay exactly same amount to perform QR decomposition. Therefore, we can only focus on product parts of each algorithm. 2nd method has an advantage when implementing matrix product after unfolding process. However, it turn out that tensor mode product cost too much to cover benefit from small dimension matrix multiplication.

However, those flops are contradictory from my previous simulation results: the second method took less time than the first method when I implemented algorithms. I could explain this contradiction using extra simulations.
In this simulation, I implemented algorithm on a $100 \times 100 \times 100$ tensor and our target rank is 1. Each algorithm are run 1000 times and I averaged times spent to do SVD.
Simulation results are as follow. Algorithm 2 took 0.07517 seconds on average and Algorithm 3 took 0.0648 on average. But my conclusion is not contradict above flops argument. This is because we didn't take into account the time spent to draw random matrix for each case. I measured time for making random matrix in algorithm 2 (3 matrices whose sizes are 10000×1): 0.0149667.
Time for making random matrices in algorithm 3 are(6 matrices, 3 are size of 100×1 and 3 are size of 1×1) : 10^{-5}
If we subtract the time generating random matrices from total spending time, Algorithm 2 becomes 0.06020 and Algorithm 3 becomes 0.06479, which makes sense to theoretical flops.

To sum up, we need to consider both how many flops would be taken for each operations and how long does it take to generate random matrices together when we choose one algorithm. Therefore, computation cost for each method would be

$$T_{\text{Matrix multiplication}} + T_{\text{QR decomposition}} + T_{\text{Generating Random matrices}}$$

Finally, total cost for computation would be

1. 1st method: $\underbrace{Nd^Nk + N(2dk^2 - \frac{2}{3}k^3)}_{\text{Matrix multiplication + QR}} + \underbrace{Nd^{N-1}k \times T}_{\text{Generating random matrices}}$
2. 2nd method: $\underbrace{N\frac{d^Nk(1-(\frac{k}{d})^{N-1})}{1-\frac{k}{d}} + Ndk^N + N(2dk^2 - \frac{2}{3}k^3)}_{\text{Matrix multiplication + QR}} + \underbrace{N((N-1)dk + k^N) \times T}_{\text{Generating random matrices}}$

Where T is a cost for generating one random normal element.
In our simulation, the Algorithm 3 has less total computation cost than the Algorithm 2 because $d >> k$.

```
library(tictoc)
a <- as.matrix(rnorm(100))
b <- as.matrix(rnorm(100))
c <- as.matrix(rnorm(100))
tnsr <- as.tensor(array(1,dim = c(1,1,1)))
X <- ttm(ttm(ttm(tnsr,a,1),b,2),c,3)
eps <- as.tensor(array(rnorm(1000000,mean =0,sd = 0.75),dim = c
    (100,100,100)))
eps2 <- as.tensor(array(rnorm(1000000,mean =0,sd = 0.8),dim = c
    (100,100,100)))
tensor_svd1(X+eps,1,1,1,0)
tensor_svd2(X+eps,1,1,1,0)
for(i in 1:1000){
  tic()
  tensor_svd(X+eps,1,1,1,0)
  toc(log = T, quiet = T)
}
log.lst <- tic.log(format = F)
timings1 <- unlist(lapply(log.lst, function(x) x$toc - x$tic))
mean(timings1)
tic.clearlog()
for(i in 1:1000){
  tic()
  tensor_svd2(X+eps,1,1,1,0)
  toc(log = T,quiet = T)
}
log.lst <- tic.log(format = F)
timings2 <- unlist(lapply(log.lst, function(x) x$toc - x$tic))
mean(timings2)
tic.clearlog()


for(i in 1:1000){
  tic()
  rnorm(10000);rnorm(10000);rnorm(10000)
  toc(log = T, quiet = T)
}

log.lst <- tic.log(format = F)
timings1 <- unlist(lapply(log.lst, function(x) x$toc - x$tic))
mean(timings1)
tic.clearlog()

for(i in 1:1000){
  tic()
  rnorm(100);rnorm(100);rnorm(100);rnorm(1);rnorm(1);rnorm(1)
  toc(log = T, quiet = T)
}

log.lst <- tic.log(format = F)
```

```
timings1 <- unlist(lapply(log.lst, function(x) x$toc - x$tic))
mean(timings1)
tic.clearlog()
```

3 To do list

1. I am working on finding some statistical properties of MLE in this model.
2. I want to study more about algorithmic convergence and get some paper related to this work.
3. I am reading a proof part of your paper about binary tensor. I am expecting to find some properties that can be applied to this ordinal tensor model(Actually, I think your property can be applied to stacked tensor Z model directly)