Maximum Likelihood Estimation of the Random-Clumped Multinomial Model using High Performance Computing

Andrew M. Raim†, Matthias K. Gobbert†, Nagaraj K. Neerchal†, Jorge G. Morel*
† Department of Mathematics and Statistics, University of Maryland, Baltimore County
* Regulatory and Clinical Development, Procter & Gamble Company

UMBC
AN HONORS UNIVERSITY IN MARYLAND

Project Summary

- Parallel computing has become popular in statistics
- But tends to focus on "embarassingly parallel" problems
  - Split problems into smaller subproblems, which can be solved independently
  - For example: repeating a simulation many times
  - Easy to do, e.g. using the SNOW package for R by Tierney et al
- How could we use high performance computing (HPC) in a more sophisticated way?
- We consider maximum likelihood estimation (MLE) for a special multinomial model
- We show that HPC is very useful for improving computation time, for large problems

Background

- Consider the model

$$X_1, \ldots, X_n \overset{iid}{\sim} f(x \mid \boldsymbol{\theta}), \quad \boldsymbol{\theta} = (\theta_1, \ldots, \theta_k) \in \Theta, \quad x \in \mathcal{X}$$

- $f(x \mid \Theta)$ is the *density* of each X_i
- The *joint likelihood function* is

$$L(\boldsymbol{\theta} \mid \boldsymbol{x}) = \prod_{i=1}^{n} f(x_i \mid \boldsymbol{\theta}), \qquad \boldsymbol{x} = (x_1, \ldots, x_n)$$

- A standard inference problem — *parametric point estimation*:
  - f is a known function, but parameters $\boldsymbol{\theta}$ are unknown
  - We've got data $x_1, \ldots, x_n$ generated by the model
  - Use the data to estimate $\boldsymbol{\theta}$
- **Maximum likelihood estimation (MLE)** — choose the estimator

$$\widehat{\boldsymbol{\theta}}(\boldsymbol{x}) := \arg\max_{\boldsymbol{\theta}} L(\boldsymbol{\theta} \mid \boldsymbol{x}) \qquad \left[\equiv \arg\max_{\boldsymbol{\theta}} \log L(\boldsymbol{\theta} \mid \boldsymbol{x})\right]$$

Random-Clumped Multinomial Model

First described in [Morel and Nagaraj, Biometrika, 1993],

$$f(\boldsymbol{t} \mid \boldsymbol{\pi}, \rho) = \sum_{j=1}^{k} \pi_j \, g(\boldsymbol{t} \mid \boldsymbol{p}_j, m), \qquad \boldsymbol{t} = (t_1, \ldots, t_k)$$

is the density for a random variable $\boldsymbol{T} = (T_1, \ldots, T_k)$, where

- $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_k)$ forms a discrete probability distribution
- $g(\boldsymbol{t} \mid \boldsymbol{p}_j, m)$ is the density of a standard multinomial
- $\boldsymbol{e}_j$ is a vector with 1 in the jth position, zeros elsewhere
- $\boldsymbol{p}_j = (1-\rho)\boldsymbol{\pi} + \rho\, \boldsymbol{e}_j, \quad j = 1, 2, \ldots, k-1$
- $\boldsymbol{p}_k = (1-\rho)\boldsymbol{\pi}$

- There are $k+1$ parameters $\boldsymbol{\theta} = (\pi_1, \ldots, \pi_k, \rho)$ to estimate, which live in:

$$\Theta = \{\boldsymbol{\theta} \in \mathbb{R}^{k+1} : \pi_1, \ldots, \pi_k, \rho \in (0,1), \sum_{j=1}^{k} \pi_j = 1\}$$

Joint likelihood function for $\boldsymbol{X} = (\boldsymbol{T_1}, \ldots, \boldsymbol{T_n})$ (iid) is

$$L(\boldsymbol{\theta} \mid \boldsymbol{x}) = \prod_{i=1}^{n} f(\boldsymbol{t}_i \mid \boldsymbol{\pi}, \rho) = \prod_{i=1}^{n} \left\{ \sum_{j=1}^{k} \pi_j \left[\frac{m!}{t_{i1}! \cdots t_{ik}!} p_{j1}^{t_{i1}} \cdots p_{jk}^{t_{ik}} \right] \right\}$$

Why Consider This Model?

The standard multinomial model handles this scenario:

- A survey question is asked to m people
- k possible answers
- Responses are **independent**
- After the survey, $\boldsymbol{t} = (t_1, \ldots, t_k)$ contains counts for each answer

The Random Clumped model addresses a scenario where independence doesn't hold:

- Some of the respondents are influenced by a common leader
- Rest answer independently

MLE computation for this model makes a good test problem

- Solving by hand isn't tractable, numerical methods must be used
- Lots of theoretical results for efficient computation are available. E.g. [Liu, PhD Thesis 2005], [Neerchal and Morel, Computational Statistics & Data Analysis, 2005]
- We've used the model for a test problem, but not much of the theory
- Also, easy to generate samples from this distribution

Implementation Details

- MLE solution coded in C++ using PGI compiler & OpenMPI
- Used the Toolkit for Advanced Optimization (TAO) library from Argonne National Lab
  - contains parallel algorithms for constrained & unconstrained optimization
  - built on top of PETSc and MPI
  - See `www.mcs.anl.gov/tao`
- Used unconstrained optimization in $\mathbb{R}^{k+1}$, even though Θ is a constrained subset
  - Let the optimizer work in $\mathbb{R}^{k+1}$
  - Use logit transformation on each parameter

$$\text{logit}(x) = \frac{1}{1+e^{-x}}, \qquad \text{logit} : \mathbb{R} \to (0,1)$$

  - Then normalize π_j's so that $\sum_{i=j}^{k} \pi_i = 1$
- Use `lgamma_r` C function to compute $\log[\Gamma(x)]$ instead of naive $x!$ calculation

Optimization method: Limited-Memory, Variable-Metric (LMVM)

- Iteratively searches for a maximum
- Only **objective** and **gradient** functions $h(\boldsymbol{\theta})$ and $\nabla h(\boldsymbol{\theta})$ need to be specified
- $h(\boldsymbol{\theta})$ first transforms $\boldsymbol{\theta}$ (see above), then computes the likelihood L

Compute gradient vector numerically, using finite differences

$$\nabla h(\boldsymbol{\theta}) = \left(\frac{\partial h(\boldsymbol{\theta})}{\partial \theta_1}, \ldots, \frac{\partial h(\boldsymbol{\theta})}{\partial \theta_{k+1}} \right), \quad \frac{\partial h(\boldsymbol{\theta})}{\partial \theta_i} \approx \frac{h(\boldsymbol{\theta} + \delta \boldsymbol{e}_i) - h(\boldsymbol{\theta})}{\delta}, \quad \delta = 10^{-8}$$

Key point for parallel computing

- Evaluating $L(\boldsymbol{\theta} \mid \boldsymbol{x})$ is very expensive
- Must be evaluated many times in our finite difference scheme
- But $\left(\frac{\partial h(\boldsymbol{\theta})}{\partial \theta_1}, \ldots, \frac{\partial h(\boldsymbol{\theta})}{\partial \theta_{k+1}} \right)$ components can be computed independently
- We can split computation of the $k+1$ components among $p \le k+1$ parallel processes

Performance Study

Estimation experiment:

- Select true parameters $\boldsymbol{\theta} = (\pi_1, \ldots, \pi_k, \rho)$ as a function of k

$$v := (1, 2, 3, \ldots, 3, 2, 1) \text{ so that } v \in \mathbb{N}^k, \quad \text{let } \pi_i = v_i \Big/ \left(\sum_{j=1}^{k} v_j \right)$$

- Generate a sample $(\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n)$ from the random clumped distribution given $\boldsymbol{\theta}$
- Initial guess for algorithm: $\boldsymbol{\theta}^{(0)} = (\pi_1^{(0)} = \frac{1}{k}, \ldots, \pi_k^{(0)} = \frac{1}{k}, \rho = \frac{1}{2})$
- Optimize, record the estimate, elapsed time, memory usage, etc
- Parallel performance is measured by *speedup* and *efficiency*

Repeat the experiment many times, varying number of processes p along with

- number of categories k **(shown below)**, sample size n, and cluster size m

Results

Parallel performance varying k (# of π_i's)

Walltime, speedup, and efficiency varying k, for n = 128, m = 256, r = 1. Tests were performed with 4 processes per node, except for $p = 1$ which uses 1 process per node, and $p = 2$ which uses 2 processes per node.

(a) Wall clock time in seconds

k	$p=1$	$p=2$	$p=4$	$p=8$	$p=16$	$p=32$	$p=64$	$p=128$
1	0.004	0.005	—	—	—	—	—	—
3	0.058	0.036	0.025	—	—	—	—	—
7	0.471	0.255	0.148	0.151	—	—	—	—
15	4.913	2.507	1.375	0.824	0.599	—	—	—
31	41.724	21.414	11.010	5.912	3.394	5.165	—	—
63	390.403	197.000	99.962	51.808	27.544	14.721	9.063	—
127	2513.446	1259.716	635.585	306.806	167.395	92.008	49.710	30.367

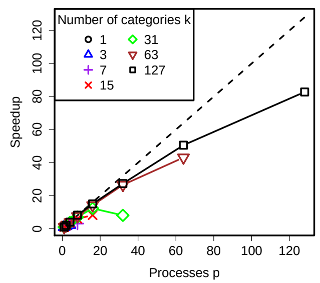
(b) Observed speedup S_p

k	$p=1$	$p=2$	$p=4$	$p=8$	$p=16$	$p=32$	$p=64$	$p=128$
1	1.00	0.87	—	—	—	—	—	—
3	1.00	1.59	2.28	—	—	—	—	—
7	1.00	1.85	3.18	3.11	—	—	—	—
15	1.00	1.96	3.57	5.96	8.20	—	—	—
31	1.00	1.95	3.79	7.06	12.29	8.08	—	—
63	1.00	1.98	3.91	7.54	14.17	26.52	43.08	—
127	1.00	2.00	3.95	8.19	15.02	27.32	50.56	82.77

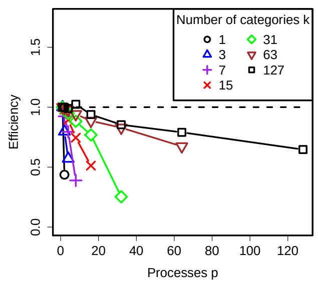
(c) Observed efficiency E_p

k	$p=1$	$p=2$	$p=4$	$p=8$	$p=16$	$p=32$	$p=64$	$p=128$
1	1.00	0.44	—	—	—	—	—	—
3	1.00	0.80	0.57	—	—	—	—	—
7	1.00	0.92	0.79	0.39	—	—	—	—
15	1.00	0.98	0.89	0.74	0.51	—	—	—
31	1.00	0.97	0.95	0.88	0.77	0.25	—	—
63	1.00	0.99	0.98	0.94	0.89	0.83	0.67	—
127	1.00	1.00	0.99	1.02	0.94	0.85	0.79	0.65

Speedup varying number of categories
for n = 128, m = 256, r = 1



Efficiency varying number of categories
for n = 128, m = 256, r = 1



- Experiments were run on the cluster `hpc` at the UMBC High Performance Computing Facility
- See [Raim, Gobbert, Tech Report HPCF-2009-8] at `http://www.umbc.edu/hpcf` for full results
- Good parallel performance for fixed k, if k large enough
- Future work: take advantage of the computational theory for the Random Clumped model
- Also: could this be done in a more statistician-friendly language like R?