`PMD`: An R package for Computing the Probability Mass Function for the Poisson-Multinomial Distribution

Zhengzhi Lin Virginia Tech
Yueyao Wang Virginia Tech
Yili Hong Virginia Tech

July 25, 2021

Abstract

The Poisson Multinomial Distribution (PMD) is the sum of n independent indicators, each indicator is an m element vector, that follows a multinomial distribution. The PMD is useful in many areas such as, machine learning, uncertainty quantification, and voting theory. The distribution function (e.g., the probability mass function) has been studied by many authors for a long time, but there is no general computing method available for its distribution functions. In this paper, we develop algorithms to compute the probability mass function for the PMD, and we develop an R package that can calculate the probability mass function efficiently. We also study the accuracy of different methods and give several examples to illustrate different computing algorithms and the use of the R package.

Key Words: Binomial distribution; Classification; Poisson Binomial Distribution; Machine Learning; Uncertainty Quantification; Voting Theory

1 Introduction

The Poisson Multinomial Distribution (PMD) is defined as sum of different independent multinomial distributions. It has applications in game theory (Cheng, Diakonikolas, & Stewart, 2017), digital imaging (Akter, Moon, & Kwon, 2019),machine learning (Kamath, 2014) . One of the popular areas that involved PMD is voting theory. A very simple example is, suppose we have n independent voters to vote for a president from m candidates, each voter has its own chance to vote on a certain candidate. It is not our concern how voters vote, we care only the voting result, that is how many votes each candidate gets. An (n, m) PMD is a perfect model for this example. In machine learning, especially in classification context, when we classify n samples one by one into m categories, then the total number of samples assigned to each category follows a PMD. When (n, m) is small it is possible to calculate the distribution function by using enumeration method. However when n, m get large, we need better method, or algorithm to calculate it. Some former studies have uncover PMD's structure and properties, (Diakonikolas, Kane, & Stewart, 2016) shows the Fourier transformation of PMD is sparse and provide a theorem that there exist algorithms to calculate PMD's density. (Daskalakis, Kamath, & Tzamos, 2015) also prove us PMD is ϵ-cover and Central Limit Theory is valid for PMD. Other papers such as (Akter et al., 2019) shows us some interesting application of PMD and its sparsity property. Due to the huge practical value of PMD, computing its pmf is of great importance. However, there is no available algorithm for computing pmf for PMD. In this case, we are motivated to develop a thorough method to calculate its pmf. We design three methods to calculate it,

Method 1 MD-DFT(base on multi-dimensional discrete Fourier transform(MD-DFT)) method: we use Multidimensional Fast Discrete Fourier Transformation algorithm to calculate PMD's exact probability mass function.

Method 2 Simulation based method: we use multinomial distributions to simulate the process of voting and use the frequency as our result to approximate true result.

Method 3 Normal approximation based method: when n is large, we apply CLT and use a approximated normal distribution as our result.

We also compare the time efficiency and accuracy of each method, and come up a guide of the best situation for each method. A R package is developed to implement these three methods, users can call certain function to calculate user given (n, m) PMD pmf's.

2 Poisson Multinomial Distribution

Let $(I_{i1}, \ldots, I_{i,m})'$, $i = 1, \ldots, n$ be a sequence of independent indicator vectors where exactly one of (m) categories successes. That is there is one and only one of those I_{ij}, $j = 1, \ldots, m$ can take value one for each i. Take election as an example, suppose there are several candidates, denoted as $j = 1, \ldots, m$, and some voters, voters are independent to each other, denoted as $i = 1, \ldots, n$, every voter can only vote for one candidate, if the ith voter votes for jth candidate, then $I_{ij} = 1$, else $I_{ij} = 0$. Denote $p_{ij} = \Pr(I_{ij} = 1)$, as the success probability of jth candidate gets a vote from ith voter. The Poisson multinomial random variable $(X_1, \ldots, X_m)'$ is defined as the sum of n independent and non-identical distributed indicator vectors $(I_{i1}, \ldots, I_{i,m})'$, which is the result of election, the total votes each candidate gets. Here $X_j = \sum_{i=1}^{n} I_{ij}$. Assume all probabilities p_{ij}'s are known, we care about the distribution of our election result, which is the probability of a certain result. Because of the sum constraints that, $\sum_{j=1}^{m} I_{ij} = 1$, $\sum_{j=1}^{m} p_{ij} = 1$, and $\sum_{j=1}^{m} X_j = n$, one can drop the last random variable when the focus is on the distribution of $(X_1, \ldots, X_m)'$. That is we will focus on the random vectors $I_i = (I_{i1}, \ldots, I_{im})'$ and $X = (X_1, \ldots, X_m)'$. The distribution of the X is called the Poisson multinomial distribution (PMD), which is denoted by

$$X = \sum_{i=1}^{n} I_i \sim \text{PMD}(n, m, P_{n\times m}),$$

where the success probability matrix (SPM) is

$$P_{n\times m} = \begin{pmatrix} p_{11} & \cdots & p_{1m} \\ \vdots & \ddots & \vdots \\ p_{n1} & \cdots & p_{nm} \end{pmatrix}.$$

The probability mass function (pmf) of PMD,

$$p(x_1, \ldots, x_{m-1}, x_m) = \Pr\left(X_1 = x_1, \ldots, X_m = x_{m-1}, X_m = n - \sum_{i=1}^{m} x_i\right)$$

is of interest. Moreover, in the next section, we only concern $p(x_1, \ldots, x_{m-1})$ because $x_m = n - \sum_{i=1}^{m-1} x_i$.

Let's see a simple example, we have three candidates and four voters, and we have

$$P_{4\times 2} = \begin{pmatrix} 0.1 & 0.2 \\ 0.5 & 0.2 \\ 0.4 & 0.5 \\ 0.8 & 0.1 \end{pmatrix}.$$

We want to know the probability of a result that candidate 1 gets 4 votes and others gets 0 vote.

$$P\{X = (4, 0, 0)\} = 0.1 \times 0.5 \times 0.4 \times 0.8 = 0.016$$

The probability of $X = (1, 3, 0)$ is

$$\begin{aligned} P\{X = (1, 3, 0)\} &= 0.1 \times 0.2 \times 0.5 \times 0.1 \\ &+ 0.5 \times 0.2 \times 0.5 \times 0.1 + 0.4 \times 0.2 \times 0.2 \times 0.1 + 0.8 \times 0.2 \times 0.2 \times 0.5 = 0.0236 \end{aligned}$$

Keep doing this we can calculate all possible outcomes.

Note that when the SPM is identical across all rows, that is, I_i, $i = 1, \ldots n$ are identically distributed, the distribution of X can be simplified as multinomial distribution. Hence, the PMD is a generalization of the multinomial distribution. When $m = 1$, the PMD is reduced to the Poisson binomial distribution as in (Hong, 2013a).

In related literature, (Hong, 2013a) consider the exact and approximate methods for computing the pmf of the Poisson binomial distribution. (Zhang, Hong, & Balakrishnan, 2018) introduce the general Poisson binomial distribution and develop an algorithm to compute its distribution functions.

3 Computation of The Probability Mass Function

We introduce three methods for computing the pmf, which are the method based on multidimensional discrete Fourier transform (MD-DFT), the normal approximation (NA) method, and simulation based method.

3.1 The MD-DFT Method Multidimensional Fourier Transform

In this section we provide an exact formula to compute the pmf of the PMD. The formula is based on the characteristic function (CF) of the PMD and the MD-DFT. The CF of $X = (X_1, \ldots, X_{m-1})'$ is

$$\phi(t_1, \ldots, t_{m-1}) = \mathbf{E}\left[\exp\left(\boldsymbol{i}\sum_{j=1}^{m-1} t_j X_j\right)\right] = \mathbf{E}\left[\exp\left(\boldsymbol{i}\sum_{i=1}^{n}\sum_{j=1}^{m-1} t_j I_{ij}\right)\right].$$

Here $\boldsymbol{i} = \sqrt{-1}$. We notice

$$\mathbf{E}\left[\exp\left(\boldsymbol{i}\sum_{j=1}^{m-1} t_j X_j\right)\right] = \sum_{x_1=0}^{n}\cdots\sum_{x_{m-1}=0}^{n} p(x_1,\ldots,x_{m-1})\exp\left(\boldsymbol{i}\sum_{j=1}^{m-1} t_j x_j\right).$$

$$\mathbf{E}\left[\exp\left(\boldsymbol{i}\sum_{i=1}^{n}\sum_{j=1}^{m-1} t_j I_{ij}\right)\right] = \mathbf{E}\left[\exp\left(\boldsymbol{i}\sum_{j=1}^{m-1} t_j I_{1j} + \cdots + \boldsymbol{i}\sum_{j=1}^{m-1} t_j I_{nj}\right)\right].$$

$$= \prod_{i=1}^{n}\mathbf{E}\left[\exp\left(\boldsymbol{i}\sum_{j=1}^{m-1} t_j I_{ij}\right)\right] = \prod_{i=1}^{n}\left[(1-\sum_{j=1}^{m-1} p_{ij}) + \sum_{j=1}^{m-1} p_{ij}\exp(\boldsymbol{i}t_j)\right].$$

Therefore we get

$$\sum_{x_1=0}^{n}\cdots\sum_{x_{m-1}=0}^{n} p(x_1,\ldots,x_{m-1})\exp\left(\boldsymbol{i}\sum_{j=1}^{m-1} t_j x_j\right) = \prod_{i=1}^{n}\left[(1-\sum_{j=1}^{m-1} p_{ij}) + \sum_{j=1}^{m-1} p_{ij}\exp(\boldsymbol{i}t_j)\right].$$

Let $t_j = \omega l_j$, $l_j = 0,\ldots,n$, $\omega = 2\pi/(n+1)$. Then the equation becomes

$$\frac{1}{(n+1)^{m-1}}\sum_{x_1=0}^{n}\cdots\sum_{x_{m-1}=0}^{n} p(x_1,\ldots,x_{m-1})\exp\left(\boldsymbol{i}\omega\sum_{j=1}^{m-1} l_j x_j\right) = \frac{1}{(n+1)^{m-1}} q(l_1,\ldots,l_{m-1}), \tag{1}$$

where

$$q(l_1,\ldots,l_{m-1}) = \prod_{i=1}^{n}\left[(1-\sum_{j=1}^{m-1} p_{ij}) + \sum_{j=1}^{m-1} p_{ij}\exp(\boldsymbol{i}\omega l_j)\right].$$

Note that $q(l_1,\ldots,l_{m-1})$ can be computed directly. The left side of equation (1) is the inverse multi-dimensional discrete Fourier transform of the sequence $p(x_1,\ldots,x_{m-1}), x_i = 0,\ldots,n$. Therefore we can apply MD-DFT on both sides to recover the sequence, we obtain the pmf as

$$p(x_1,\ldots,x_{m-1}) = \frac{1}{(n+1)^{m-1}}\sum_{l_1=0}^{n}\cdots\sum_{l_{m-1}=0}^{n} q(l_1,\ldots,l_{m-1})\exp\left(-\boldsymbol{i}\omega\sum_{j=1}^{m-1} l_j x_j\right) \tag{2}$$

Let $\ell = (l_1,\ldots,l_{m-1})$, then we will have $(n+1)^{m-1}$ different ℓ as l_i values from 0 to n. For example, if we have $n = 4$,$m = 4$, then ℓ can be $(0,0,0),(0,0,1),\ldots,(4,4,4)$, 125 different vectors in total. Now we have $q(l_1,\ldots,l_{m-1}) = q(\ell)$. We design to use these vectors to generate respective $p(x_1,\ldots,x_{m-1})$. For each ℓ, we get a $p(x_1,\ldots,x_{m-1})$.

To get all $p(x_1,\ldots,x_{m-1})$ with respect to given n, $m-1$ and $P_{n\times(m-1)}$, we apply the Fast Fourier Transformation algorithm(FFT) from GSL Scientific Library to make calculation efficient. This FFT algorithm is C language based, we implemented it and wrote a new MD-DFT

algorithm. Our MD-DFT algorithm can calculate all values of distribution function as long as we input our $P_{n\times(m-1)}$ matrix, and it can be called from R.

3.2 Normal-Approximation Based Method

Let $\boldsymbol{p}_i$ be the ith row of the SPM P. Let $\boldsymbol{p} = \sum_{i=1}^{n} \boldsymbol{p}_i/n$ be the average of the rows of P. We first show that CLT can be applied to

$$\frac{X}{n} - \boldsymbol{p} = \frac{(X_1, \ldots, X_{m-1})}{n} - \boldsymbol{p} = \frac{(X_1 - \sum_{i=1}^{n} p_{i1}, \ldots, X_{m-1} - \sum_{i=1}^{n} p_{i,m-1})}{n}$$

We just need to show that for any $j = 1, \ldots, m-1$, $X_j - \sum_{i=1}^{n} p_{ij}$ satisfies conditions of CLT. Where $X_j = \sum_i I_{ij}$
Notice that for any $\delta > 0$

$$1 \geq p_{ij}(1 - p_{ij}) = \mathrm{Var}(I_{ij}) \geq E(|I_{ij}|^{2+\delta})$$

Therefore, let $s_n^2 = \sum_{i=1}^{n} \mathrm{Var}(I_{ij})$, we have

$$\frac{1}{s_n^{2+\delta}} \sum_{i=1}^{n} E|I_{ij}|^{2+\delta} \leq \frac{1}{s_n^{2+\delta}} \sum_{i=1}^{n} \mathrm{Var}(I_{ij}) = \frac{1}{s_n^{\delta}}$$

As $n \to \infty$, $s_n \to \infty$, so the above equation converges to 0. Therefore X_j satisfies Lyapunov condition, thus CLT can be applied to X_j.
And we have to figure out the covariance matrix for X. Observe for any fix i, I_{ij} and I_{ik} has covariance $-p_{ij}p_{ik}$.

Therefore, it is trivial to get the covariance matrix

$$\Sigma = \frac{1}{n} \sum_{i=1}^{n} [\mathrm{diag}(\boldsymbol{p}_i) - \boldsymbol{p}_i \boldsymbol{p}_i']$$

By central limit theorem (CLT),

$$\left(\frac{X}{n} - \boldsymbol{p}\right) \dot{\sim} \mathrm{N}\left(\mathbf{0}, \frac{1}{n}\Sigma\right).$$

When n is sufficiently large, our R package use the normal distribution to calculate our probability by demand of users.

3.3 Simulation-Based Method

One can simulate I_i from multinomial distribution and then compute X. Repeat this many times to generate enough samples for X. Then use the sample distribution to approximate the true distribution. To be specific,

Step 1 randomly generate I_i with given p_i using multinomial distribution.

Step 2 repeat step 1 to generate $I_1, \ldots, I_n$. Calculate $X = (X_1, \ldots, X_m)'$, where $X_j = \sum_{i=1}^{n} I_{ij}, j = 1, \ldots, m$.

Step 3 repeat step 1 and step 2 for T times, and calculate the frequency of X as to form our distribution density.

Example 3.1. Suppose we are given $n = 3$, $m = 2$

$$P_{n\times m} = P_{3\times 1} = \begin{pmatrix} 0.1 & 0.9 \\ 0.5 & 0.5 \\ 0.3 & 0.7 \end{pmatrix}.$$

We first generate I_1 by using multinomial distribution with respect to probability vector $(0.1, 0.9)$, we get $I_1 = (0, 1)$,then we generate I_2 using multinomial distribution with probability vector $(0.5, 0.5) \rightarrow I_2 = (1, 0)$, and also generate I_3 by using prob vector $(0.3, 0.7) \rightarrow I_3 = (0, 1)$. Now we have a simulation result $X = I_1 + I_2 + I_3 = (1, 2)$. We repeat above process 10^4 times to get 10^4 results, finally if we want to know probability of (1,2), we can calculate the frequency of it in the 10^4 results.

We design a C based algorithm to perform this simulation process. Our algorithm automatically calculate probabilities of all possible results and it can be called from R.

In our package, as n and m get large, the total number of density points $(n+1)^{m-1}$ will be extensively great. For time efficiency purpose, we only calculate the probability as customer demand. User will be asked to input the resulting vector, and the probability for the vector will be calculated.

4 Method Comparisons

In this section, we compare the three methods in terms of numerical accuracy and the time efficiency, we also give out recommendations of under what condition which method will be preferred. At the first three subsections, we show the accuracy of the methods we mention earlier. Here the major accuracy criterion we use is maximum absolute error(MAE). Suppose

we have an PMD matrix with $n \times m$ dimension. Hence there will be $(n+1)^{m-1}$ density points notated as a set $X = \{x_1, \ldots, x_N\}$, where $N = (n+1)^{m-1}$. By this way, MAE is defined as following,

$$\text{MAE} = \max_X |p(x) - p_{\text{true}}(x)|$$

which is the max value of the differences between probability densities calculated by our methods and true ones. We also use total absolute error (TAE) as a supplemental criterion where

$$\text{TAE} = \sum_X |p(x) - p_{\text{true}}(x)|$$

For MD-DFT method, we test its accuracy by using small PMD matrix with given inputs due to the complexity of calculation. For other methods, let the densities calculated by MD-DFT to be the true densities and compare them with our goal method. The PMD matrices we use here are generated randomly.

4.1 Accuracy

4.1.1 Accuracy of MD-DFT Method

As we know already, MD-DFT is a analytic proved method. For large n and $m = 2$, which is the Binomial Possion Distribution, the accuracy is justified by (Hong, 2013b). Thus for convenience and incapability of calculating the true density of large m, here we show the accuracy of MD-DFT by using given small (n, m) pairs. For those pairs, we can work out their probability densities by hand, and compare them with the results computed by MD-DFT method. As we see from Table 1, the MAE are all less or equal to machine epsilon.

(n, m)	$\min(p_{ij})$	$\max(p_{ij})$	MAE
(2,2)	0.14	0.86	$\leq 10^{-16}$
(4,3)	0.09	0.56	$\leq 10^{-16}$
(5,3)	0.02	0.6	$\leq 10^{-16}$
(6,3)	0.1	0.59	$\leq 10^{-16}$
(4,4)	0.08	0.41	$\leq 10^{-16}$
(5,4)	0.005	0.39	$\leq 10^{-16}$
(6,4)	0.007	0.44	$\leq 10^{-16}$

Table 1: Accuracy of MD-DFT method

This shows us that MD-DFT is accurate enough and the error is only due to the computation limit of devices. Therefore, it is convincing to use MD-DFT as true probability densities in the following subsections.

4.2 Accuracy of Normal Approximation

Figure 1 lists all (n, m) pairs we will test in this section and next section, every (n, m) combination will be tested if it is covered in blue bars. The respecting bar height for $m = 3, 4, 5, 6, 7, 8$ is $100, 83, 43, 25, 16, 17$. We use density points computed by MD-DFT as true probabilities and compare those to the ones approximated by normal distribution(NA) to compute MAE and TAE. Each PMD matrix is generated randomly using uniform distribution, thus we must consider the variation caused by randomness. To reduce the randomness effect, for each (n, m) pair, we generate N PMD matrices and compute all of their MAE. Consequently, we be able to get the average MAE of them and use it as our final MAE. We use the same scheme for TAE.

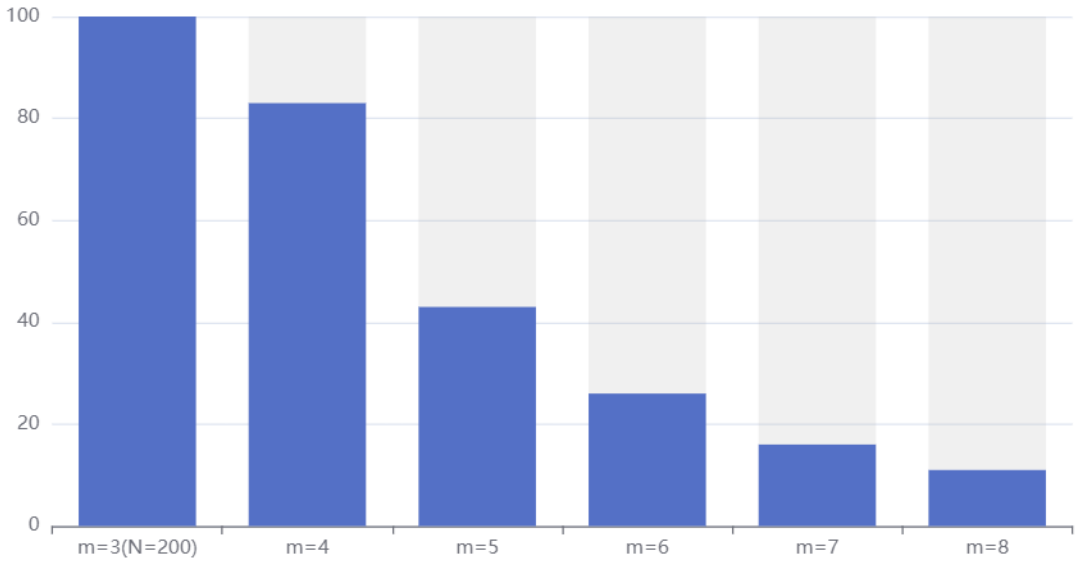
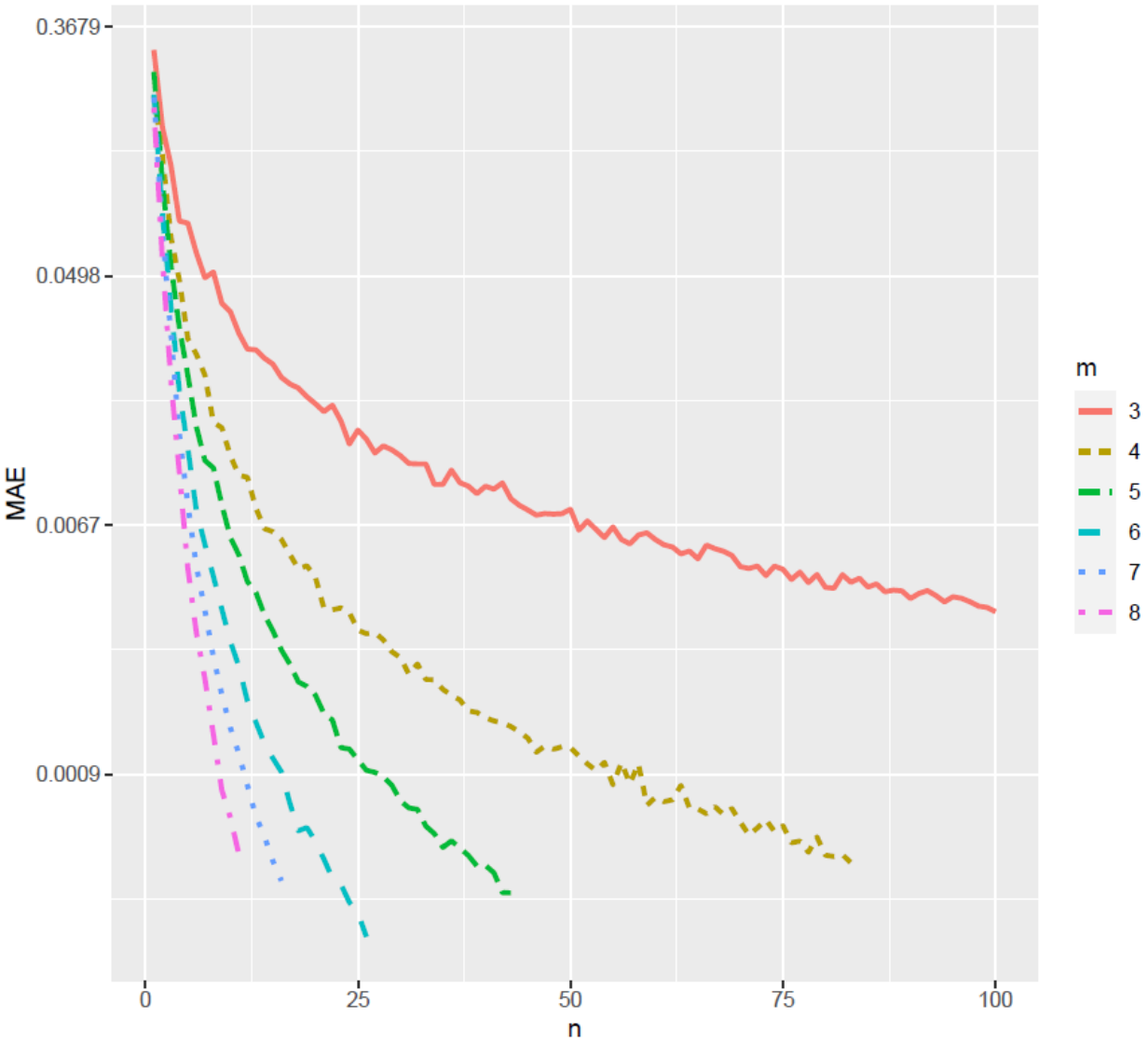


Figure 1: (n,m) pairs, y axis value represents n. For instance, when m=3, we test the pairs from $(n = 1, m = 3)$ to $(n = 100, m = 3)$, the second bar means we testing from $(n = 1, m = 4)$ to $(n = 83, m = 4)$

Figure 2 shows the accuracy test results for normal approximation(NA). The x axis is n and different colors represent different m values. As n increasing, MAE decreases for the same m value. We should also consider the effect of sparsity. As n and m raise, there will be a boost of number of density points. Hence every density point will be given less value averagely. Therefore, no matter what approximation method we apply here, the MAE will decrease as the number of density points grows.



accuracy.png

Figure 2: Different colors and types of lines representing different m. The TAE is under log transformation

4.2.1 Accuracy of simulation method

While NA and MD-DFT can cover most cases, we introduce an other method for approximating PMDs', the simulation approximation method(SA). The idea behind this method is simple, as we know a PMD is a generalized multinomial distribution. Suppose each row of PMD matrix is a probability vector of a multinomial distribution, thus a $n \times m$ PMD matrix can be seen as n multinomial distributions with respect to n m-category probability vectors. Then we can have the following algorithm to approximate each density point of PMD.

Algorithm 1: Simulation approximation algorithm(SA)

Result: Write here the result

Set the repeating time equals to N;

for $t = 1$ **to** N **do**

  for $i = 1$ **to** n **do**

    generate 1 multinomial sample wrt probability vector = i^{th} row of $P_{n\times m}$

  end

  sum up all samples generated in last loop and record the result as the t^{th} row of matrix R.

end

- compute the frequency of every result that stored in R as our density point value.
- compare density calculated above and computed by MD-DFT to get MAE and TAE.

We also calculated the error bound of the simulation method. Suppose for ith repeat, we consider random variable matrix $[X_{1i}, X_{2i}, \ldots, X_{ni}]'$, each X is a vector of length m representing a voter's voting result. We repeat R times to simulate the voting process to get $X_{j1}, \ldots, X_{jR}$ for each $j = 1, \ldots, n$, then

$$
\begin{aligned}
U_1 &= \frac{\sum_{i=1}^{R} X_{1i}}{R} \longrightarrow N\left(p_1, \frac{1}{R}\left(Diag(p_1) - p_1 p_1'\right)\right) \\
U_2 &= \frac{\sum_{i=1}^{R} X_{2i}}{R} \longrightarrow N\left(p_2, \frac{1}{R}\left(Diag(p_2) - p_2 p_2'\right)\right) \\
&\vdots \\
U_n &= \frac{\sum_{i=1}^{R} X_{ni}}{R} \longrightarrow N\left(p_n, \frac{1}{R}\left(Diag(p_n) - p_n p_n'\right)\right)
\end{aligned}
$$

Therefore

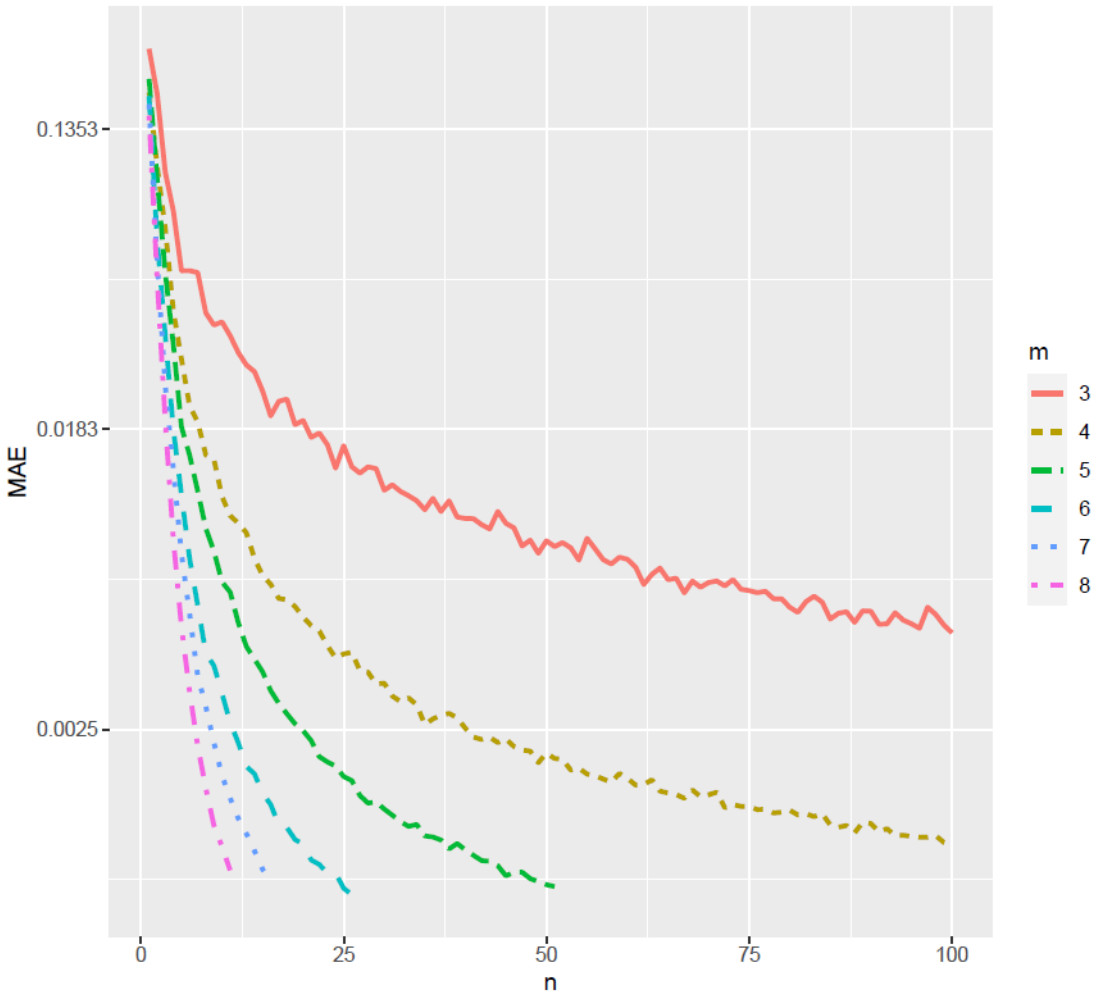
$$
U_1 + \cdots + U_n \longrightarrow N\left(\sum_{i=1}^{n} p_i, \frac{1}{R}\sum_{i=1}^{n}\left(Diag(p_i) - p_i p_i'\right)\right)
$$

For each element in the vector p_i, say p_{ij}

$$
p_{ij} - p_{ij}^2 \leq \frac{1}{4}
$$

So the error of simulation will be controlled within

$$
\frac{1}{R} \times \frac{n}{4} = \frac{n}{4R}
$$



accuracy.png

Figure 3: Simulation accuracy test result, the repeating time set as 10^4

Figure 3 illustrates the performance of SA method. For a fixed m and t, as n increases MAE decreases. Due to sparsity effect if we fix n, we could find MAE also drops while m increasing.

4.3 Time Efficiency

In this section, we show computation efficiency of simulation method and MD-DFT for calculating all density points as (n, m) gets large. We generate random matrices with respect to given (n,m), record the calculating time of each method. As for normal approach, it is a asymptotic method, we only need to calculate the asymptotic normal distribution. Study the computing of the three methods.

(n, m)	$MD - DFT$	Simulation based(10^6)
(4,3)	0.023	0.542
(6,3)	0.002	0.852
(8,3)	0.003	1.177
(10,3)	0.003	1.634
(20,3)	0.014	4.532
(50,3)	0.098	22.349
(100,3)	0.49	80.243
(200,3)	2.834	307.350
(1000,3)	249.449	7992.507

Table 2: Accuracy of simulation method

4.4 Recommendations

For small m and moderate n, the MD-DFT can be used.

For large m and moderate n, the simulation-based method can be used.

For large n, the NA-based method can be used.

5 Examples of Using PMD in Heart Disease Data

Here we illustrate how to apply PMD in real data analysis by using hear disease data from UCI machine learning repository. The data was collected from multiple locations and here we only use the data from Cleveland.

The data contains 14 variables and 297 observations after removal of missing values, our response variable is the diagnosis of heart disease, it has 5 levels coded with 0, 1, 2, 3, 4. There are other 13 variables such as age, sex, exang(exercise induced angina) etc. In order to apply our methods of calculating PMD, we first divided the data into 61 smaller groups by 5 variables(chest pain type, sex, exercise induced angina, resting electrocardiographic results and the slope of the peak exercise ST segment), reader can also divide the data in other reasoning. We use 5 variables as our covariates as shown in the following table. Among the groups, n varies from 2 to 62, which are affordable for MD-DFT method. A group of data is shown in Table 7.

The total number for each level of our response variable in each group is known. WWe have 5 levels of response refer to 5 categories. In the example group shown in Table 7, the outcome vector is $(2, 0, 0, 0, 0)$ because both observations has response of level 0.

age	trestbps	chol	thalach	oldpeak
69	140	239	151	1.8
60	150	24	171	0.9

Table 3: Selected Group of Heart Disease Data from Cleveland

The first step is to estimate P matrix for each group. Suppose the likelihood of each group is $\ell_1, \ldots, \ell_{61}$, where ℓ_i = the probability of getting the result of group i. We use softmax function to find 61 P_i matrices for each one of the 61 groups. Then we can calculate ℓ_i via P_i, details are following.

Initiate values for parameters $\boldsymbol{\beta} = [\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_5]'$, where $\boldsymbol{\beta}_i = [\beta_{i1}, \ldots, \beta_{i5}]$ for our 5 categories and 5 covariates. Suppose we have n_i observations in group i, the respecting P_i matrix should be of dimension $n_i \times 5$, $P_i = \left(p_{jk}^{(i)}\right), j = 1, \ldots, n_i, k = 1, 2, 3, 4, 5$. Our observations are X_i, $i = 1, \ldots, n_i$, each $X_i = [x_{i1}, x_{i2}, \ldots, x_{i5}]$, where $x_{i1}, \ldots, x_{i5}$ representing the covariates information.

Then we have the following P_i matrix for each i by using softmax function,

$$p_{jk}^{(i)} = \frac{e^{\boldsymbol{\beta}_k' X_j}}{\sum_{h=1}^{5} e^{\boldsymbol{\beta}_h' X_j}}$$

After we get our P_i matrices, apply MD-DFT or simulation or normal method, we get our likelihoods ℓ_i for $i = 1, \ldots, 61$.

The total likelihood $L = \prod_{i=1}^{61} \ell_i$, we can use numerical methods eg. gradient descent, Newton method to find the $\boldsymbol{\beta}$ that maximized L, then we get our final P_i matrices.

There are 3 categories and 2 observations, thus the P matrix is 2×3. We calculate the P matrix by using softmax function. The weights should be 4×5 dimension if we set the last category as baseline. By using MD-DFT we get the likelihood for the group getting result $(2, 0, 0)$. As we have 61 groups, we will have 61 likelihoods $l_1, \ldots, l_61$ so that we can get a total likelihood $L = \prod_{i=1}^{61} l_i$.

We estimated the 61 P matrices of 61 groups by maximizing L. Two of the P matrices

are

$$P_1 = \begin{bmatrix} 0.29725 & 0.23646 & 0.236462 & 0.236462 & 0.00058 \\ 0.29749 & 0.23401 & 0.23401 & 0.23401 & 0.00047 \end{bmatrix}$$

$$P_{61} = \begin{bmatrix} 0.28968 & 0.23646 & 0.23646 & 0.23646 & 0.00093 \\ 0.31585 & 0.22798 & 0.22798 & 0.22798 & 0.00021 \\ 0.29853 & 0.23343 & 0.23343 & 0.23343 & 0.00119 \\ 0.30647 & 0.23093 & 0.23093 & 0.23093 & 0.00075 \end{bmatrix}$$

Therefore we hereby using our MD-DFT method again to calculate $P(X_1 = 2, X_2 = 0, X_3 = 0, X_4 = 0, X_5 = 0) = 0.08842$ and $P(X_1 = 0, X_2 = 1, X_3 = 2, X_4 - 1, X_5 = 0) = 0.03487$

6 Illustrations of the R Package

Illustrate the use of major functions in the R package.

7 Concluding Remarks

We develop algorithm that can be useful for computing the pmf of the PMD distribution, which is challenging to compute but useful in many application scenarios.

Acknowledgments

*References

Akter, L. A., Moon, I., & Kwon, G.-R. (2019). Double random phase encoding with a poisson-multinomial distribution for efficient colorful image authentication. *Multimedia Tools and Applications*, *78*(11), 14613–14632.

Cheng, Y., Diakonikolas, I., & Stewart, A. (2017). Playing anonymous games using simple strategies. In *Soda.*

Daskalakis, C., Kamath, G., & Tzamos, C. (2015). On the structure, covering, and learning of poisson multinomial distributions. *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, 1203-1217.

Diakonikolas, I., Kane, D. M., & Stewart, A. (2016). The fourier transform of poisson multinomial distributions and its algorithmic applications. In *Proceedings of the forty-eighth annual acm symposium on theory of computing* (pp. 1060–1073).

Hong, Y. (2013a). On computing the distribution function for the poisson binomial distribution. *Computational Statistics & Data Analysis*, *59*, 41–51.

Hong, Y. (2013b). On computing the distribution function for the poisson binomial distribution. *Computational Statistics & Data Analysis*, *59*, 41-51. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0167947312003568` doi: https://doi.org/10.1016/j.csda.2012.10.006

Kamath, G. G. C. (2014). *On learning and covering structured distributions.* Unpublished doctoral dissertation, Massachusetts Institute of Technology.

Zhang, M., Hong, Y., & Balakrishnan, N. (2018). The generalized poisson-binomial distribution and the computation of its distribution function. *Journal of Statistical Computation and Simulation*, *88*(8), 1515–1527.