# Fast MLE Computation for the Dirichlet Multinomial

Max Sklar
Foursquare Labs
max@foursquare.com

March 19, 2023

**Abstract**

Given a collection of categorical data, we want to find the parameters of a Dirichlet distribution which maximizes the likelihood of that data. Newton's method is typically used for this purpose but current implementations require reading through the entire dataset on each iteration. In this paper, we propose a modification which requires only a single pass through the dataset and substantially decreases running time. Furthermore we analyze both theoretically and empirically the performance of the proposed algorithm, and provide an open source implementation.

## 1 Introduction

The Dirichlet distribution has long been used as a conjugate prior for the multinomial distribution, giving statisticians access to analytically tractable priors. Since the Dirichlet is so central to Bayesian statistics, it is important to have a simple computation to estimate its parameters. Here we consider the maximum likelihood estimate.

### 1.1 Motivation

One use for the Dirichlet prior is to avoid overfitting the estimate of a multinomial distribution from a small amount of data. For example, consider a recommendation engine that aggregates user-generated ratings. With only a few users weighing in on a given item, the maximum likelihood estimate of the rating distribution can be very inaccurate. The Dirichlet prior prevents the estimate from being overfit when only one or two users have left an opinion.

In an example of higher dimension, the relative popularity of a venue on Foursquare at a given time of week is estimated using a Dirichlet prior[12]. The week is divided in to 168 buckets, each representing one hour in the week. The timeliness distribution of a venue starts out with a Dirichlet prior, and is updated whenever new data is received. The Dirichlet prior is originally derived from using many similar Foursquare venues, using an algorithm similar to the one described in this paper.

Of course, not all applications of the Dirichlet prior are new. In the 1960s, James Mosimann found applications in analyzing biological data[9]. In one example, there were 4 types of pollen found in pollen cores

at different levels. For each core, only 100 grains of pollen were counted, giving relatively sparse data for any given core. However, even with only 73 cores analyzed, a lot more could be said about the data after modeling it as a Dirichlet-multinomial[8, pg 8][9].

A Dirichlet distribution is much simpler to estimate if the data comes in the form of multinomial distributions drawn from that Dirichlet. But what if we only have access to a few samples from each multinomial? In fact, if we already had the full multinomial, or had sampled it so heavily that the maximum likelihood estimate is accurate, there would be no need for a Dirichlet prior in the first place.

Perhaps the largest motivation for this effort is that the Dirichlet and Dirichlet-multinomial distributions can be used as part of a larger system. Consider that the K-means algorithm cannot perform well without an efficient calculation for the mean of a set of points. Likewise, a mixture of Gaussians cannot be computed without first computing one Gaussian. If one intends to build a mixture model of Dirichlet distributions by using the Expectation-Maximization algorithm, having a fast computation for individual Dirichlet multinomials is very important.

Dirichlet distributions are also widely used in topic models for Natural Language Processing. For example, the latent Dirichlet allocation requires a computation of the MLE for a Dirichlet[1][4][13]. As the datasets grow larger and the models more complex, the demand for computational efficiency increases.

## 1.2 Previous Work

In 1989, Gerd Ronning published a paper on estimating a Dirichlet from a set of multinomial-valued samples[11]. This paper contains a proof that the objective function is globally concave and therefore the Newton-Raphson method will converge to a solution [8, pg 73]. In 1990, Narayanan published a paper on the MLE computation of the Dirichlet including an implementation in Fortran[7]. He cites various examples of Dirichlet distributions being used for statistical models going back to the 1960s and 1970s, including Mosiman's work on serum proteins in white Pekan ducklings[8, pg 8].

In 2000, Thomas Minka[6] wrote about the MLE for Dirichlet distributions, and later wrote an implementation in Matlab.

Much work has also been done in finding an initialization for the Newton-Raphson method [8, pg 74-54]. In 1980, Dishon and Weiss[2] proposed using the simple method of moments for the Beta distribution [1]. In 1989, Ronning[11] produced a new initialization and proved that it was guaranteed to stay within the bounds for the first Newton-Raphson step. In 2003, Hariharan and Vela[3] proposed a new, simpler, method for initialization. And finally in 2008, Wicker et al[14], proposed a Maximum Likelihood Approximation method which was shown to work more efficiently with some standard data sets.

All of that work assumes direct access to the multinomial distribution, and not from just count data. Minka[6] tackles this problem in 2000 and describes the computations for finding the MLE of a Dirichlet multinomial. His algorithms include the Newton-Raphson method and a fixed-point iteration. He also includes a Matlab implementation. In 2008, Hanna Wallach reviews Minka's algorithms, and develops a new fixed-point iteration based on a mathematical transformation of the digamma function leading to a precomputation. Ng et al[8] in 2011 also describe the mathematics, and say that "in general, Newton's method provides the fastest means for finding the MLE".

This paper builds on the work of Minka and Wallach to compute the MLE of a Dirichlet-multinomial

[1] the 2-dimensional case of the dirichlet distribution

distribution. The proposed algorithm uses the Newton-Raphson method, and arrives at the same value as other MLE methods[2]. It takes advantage of a precomputation that can be made using the properties of the digamma and trigamma function, not unlike Wallach's approach in the fixed-point case. The proposed algorithm is faster in a wide range of circumstances, particularly in the case where count data for each multinomial instance is scarce.

## 2 Definitions and Conventions

The *gamma function* is a positive-valued function whose domain is the positive real numbers.[3]

$$\Gamma(x) = \int_0^\infty t^{x+1} e^{-x} dx$$

The gamma function of a positive integer n is equivalent to n minus 1 factorial:

$$\Gamma(n) = (n-1)!$$

For this paper, define the dual-input gamma function of a positive number and a whole number, $n$. This function is non-standard, but it is useful because of it's computational property; it is equivalent to an $n$-factor multiplication problem.

$$\Gamma(a, n) = \frac{\Gamma(a+n)}{\Gamma(a)} = \prod_{i=0}^{n-1} (a+i)$$

The dual-input log-gamma function is equivalent to the log of the previous function, and can be computed as an $n$-step addition problem. This function is also not standard, but is defined for the purpose of this paper to make equations simpler and more clear.

$$L\Gamma(a, n) = \sum_{i=0}^{n-1} \ln(a+i)$$

A *$K$-dimensional multinomial distribution* is a probability distribution over K discrete and mutually exclusive outcomes. A multinomial distribution is represented by a $K$-dimensional vector $\mathbf{p}$, where all the values sum to 1.

$$\sum_{i=0}^{K-1} p_i = 1$$

[2]In fact, given the same initial conditions, every iteration of Newton-Raphson is untouched.

[3]It is typically generalized to complex numbers but only positive values are considered in this paper.

Suppose that a multinomial distribution is sampled n times. These $n$ pieces of data can be compressed into a K-dimensional count vector $\mathbf{c}$, where $c_k$ is the number of times category $k$ appears in the data set. The counts all sum to $n$.

$$\sum_{k=1}^{K-1} c_k = n$$

A *$K$-dimensional Dirichlet distribution* is a family of probability distributions over the space of K-dimensional multinomial distributions. It is parameterized by a K-dimensional vector $\alpha$.

$$\mathcal{D}(\mathbf{p}, \alpha) = \frac{\prod_{k=0}^{K-1} \Gamma(\alpha_i)}{\Gamma\left(\sum_{k=0}^{K-1} \alpha_k\right)} \prod_{k=0}^{K-1} {p_i}^{\alpha_k - 1}$$

The Dirichlet distribution is a *conjugate prior* for the multinomial distribution. If the Dirichlet distribution represents the prior probability over a multinomial $\mathbf{p}$, and $n$ samples are drawn from $\mathbf{p}$ yielding a count vector $\mathbf{c}$, the posterior distribution is also a Dirichlet.

If the prior distribution is $P(\mathbf{p}) = \mathcal{D}(\mathbf{p}, \alpha)$ then the posterior distribution is $P(\mathbf{p}|\mathbf{c}) = \mathcal{D}(\mathbf{p}, \alpha + \mathbf{c})$.

The *expected value of a Dirichlet distribution* is equal the vector $\alpha$ normalized.

$$\mathbb{E}(\mathcal{D}(\mathbf{p}, \alpha)) = \frac{\alpha}{\sum_{i=0}^{K-1} \alpha_i}$$

An intuitive notion of the Dirichlet distribution is that each $\alpha$ value puts some weight on each category before the data is collected. As more data is taken into account, the initial $\alpha$ values become less relevant and the data is allowed to speak for itself.

The probability distribution over the possible configurations of a count vector given a Dirichlet distribution is the *Dirichlet-multinomial distribution.* It is derived as follows:

$$P(\mathbf{c}|\alpha) = \int_{\mathbf{p}} P(\mathbf{c}|\mathbf{p}) P(\mathbf{p}|\alpha)$$

$$P(\mathbf{c}|\alpha) = \frac{\prod_{i=0}^{K-1} \Gamma(\alpha_i, c_i)}{\Gamma\left(\sum_{i=0}^{K-1} \alpha_i, \sum_{i=0}^{K-1} c_i\right)} \frac{\Gamma\left(1, \sum_{i=0}^{K-1} c_i\right)}{\prod_{i=0}^{K-1} \left(\Gamma(1, x_i)\right)}$$

Finally, the *maximum likelihood estimate (MLE)* for a parameterized probability distribution are those parameters which maximize the likelihood of the data.

$$MLE(\alpha) = \arg\max_{\alpha} \; P(\mathbf{c}|\alpha)$$

# 3 Overview of the Problem

Suppose that $N$ samples are drawn from a Dirichlet distribution, and we want to find the MLE of the parameter $\alpha$. There are two cases to consider.

In the Dirichlet problem, there is direct access to the multinomial distribution of each sample. The data is shown in Table 1, where each row is a sample, and each column represents the probability of one of the $K$ categories.

| sample | $k=0$ | $k=1$ | $k=2$ | sum |
|---|---|---|---|---|
| 0 | 0.21 | 0.37 | 0.42 | 1 |
| 1 | 0.56 | 0.15 | 0.29 | 1 |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $n-1$ | 0.14 | 0.33 | 0.53 | 1 |

Table 1: Dataset in the Pure Dirichlet Problem

In the *Dirichlet-multinomial problem*, the multinomial distributions are not directly observed. Instead, a sample is recorded from each one. This is far more common in practice, since counts are directly observable. This dataset still has N rows and K columns, but each value now represents a count rather than a probability.

| sample | $k=0$ | $k=1$ | $k=2$ | sum |
|---|---|---|---|---|
| 0 | 3 | 14 | 0 | 17 |
| 1 | 1 | 16 | 3 | 20 |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $n-1$ | 6 | 3 | 10 | 19 |

Table 2: Dataset in the Dirichlet Multinomial Problem

In both cases, the Newton-Raphson method can be used to accurately compute the MLE[6]. The pure Dirichlet case is relatively fast because a *sufficient statistic* compressing all the data into a single $K$-dimensional vector can be precomputed. Unfortunately, the Dirichlet-multinomial problem cannot be reduced in the same way, making it an order of magnitude slower.

The logical improvement is therefore to find an alternative pre-computation on the Dirichlet-multinomial data to speed up the algorithm. Instead of compressing the data into a $K$-dimensional vector, the Dirichlet-multinomial data can in fact be compressed into a $K$ x $M$ matrix, and an $M$-dimensional vector, where $M$ is the maximum number of categories sampled in any particular row.

# 4 The Fast Dirichlet Solution

This is equivalent to the solution documented by Minka[6] (2000), Blei[1, A.4.2] (2003) and Ng[8, pg 72-73] (2011). The dataset for the Dirichlet problem is an $N$ x $K$ matrix $D$ in which each row represents a sample from a single Dirichlet. Let $d_{n,k}$ represent the nth row and kth column of matrix $D$. Given a Dirichlet parameter $\alpha$, the probability of the dataset is given as follows:

$$P(D|\alpha) = \prod_{n=0}^{N-1} \Gamma\left(\sum_{k=0}^{K-1} \alpha_k\right) \prod_{k=0}^{K-1} \Gamma(\alpha_k)^{-1} d_{n,k}^{\alpha_k - 1}$$

We want to maximize this probability with respect to $\alpha$. One way to do this is to build and maximize a function $F$, which is related to the log of the probability.

$$\ln(P(D|\alpha)) = \sum_{n=0}^{N-1} \ln\Gamma\left(\sum_{k=0}^{K-1}\alpha_k\right) + \sum_{n=0}^{N-1}\sum_{k=0}^{K-1}\left[-\ln\Gamma(\alpha_k) + (\alpha_k - 1)\ln\left(d_{n,k}\right)\right] \tag{1}$$

We can distribute the summations and remove the constant terms.[4]

$$F(\alpha) = \ln\Gamma\left(\sum_{k=0}^{K-1}\alpha_k\right) - \sum_{k=0}^{K-1} ln\Gamma(\alpha_k) + \sum_{k=0}^{K-1}\alpha_k\left(\frac{1}{N}\sum_{n=0}^{N-1}\ln\left(d_{n,k}\right)\right) \tag{2}$$

In this form, only one term involves the data. This term is called the *sufficient statistic* and will be set as the vector $\mathbf{v}$. The sufficient statistic can be computed with one pass through the data.

$$v_k = \frac{1}{N}\sum_{n=0}^{N-1}\ln\left(d_{n,k}\right)$$

The distinction separating the Dirichlet and the Dirichlet multinomial cases is that the Dirichlet case has a sufficient statistic of constant dimension. In fact, this is true of all exponential distributions[10, pg 116]. The function to maximize now looks like this:

$$F(\alpha) = \ln\Gamma(\sum_{k=0}^{K-1}\alpha_k) - \sum_{k=0}^{K-1}\ln\Gamma(\alpha_k) + \sum_{k=0}^{K-1}\alpha_k v_k$$

In order to use the Newton-Raphson method, we need to compute the gradient vector $\mathbf{g}$ and the Hessian matrix $H$. The formula for a single step is as follows:

$$\alpha_{new} = \alpha_{old} - H^{-1}\mathbf{g}$$

The values for $H$ and $\mathbf{g}$ also require only the vector $\mathbf{v}$, and not the full dataset.

We will use the following derivatives:

$$\Psi(x) = \tfrac{d}{dx}\ln\Gamma(x) \quad \Psi'(x) = \tfrac{d}{dx}\Psi(x)$$

These functions, along with gamma are available in most numeric libraries. The Python code linked to this paper uses the Numpy library. In the past Narayanan[7] and Minka[6] relied on Fortran and Matlab implementations respectively.

The gradient is given as follows:

$$g_k = \Psi(\sum_{k=0}^{K-1}\alpha_k) - \Psi(\alpha_k) + v_k$$

[4]Note that we divided the entire equation by $N$. The number of samples does not factor in to the MLE, but it does control the variance around that value.

The Hessian matrix consists of a constant matrix, plus a diagonal. Let $\mathbb{I}_{i,j}$ be the indicator function that is equal to 1 if $i = j$ and 0 otherwise.

$$h_{i,j} = \Psi'(\sum_{k=0}^{K-1} \alpha_k) - \Psi'(\alpha_i)\mathbb{I}_{i,j}$$

There is a formula for efficiently inverting this type of matrix, and it is a special case of the matrix inversion lemma[15]. The technique is also used by Minka[6] (2000) and Blei[1, A.2](2003) for Dirichlet-based machine learning. Let the vector representing the diagonal be $\mathbf{d}$, and the scalar constant in the constant matrix be $c$. The formula computing the inverse Hessian is as follows:

$$H = I\mathbf{d} + \mathbf{1}\mathbf{1}^T c$$

$$H^{-1} = I\mathbf{d}^{-1} - \frac{\mathbf{d}^{-1}\mathbf{1}\mathbf{1}^T\mathbf{d}^{-1}}{c^{-1} + \sum_{k=0}^{K-1} d_k^{-1}} \tag{3}$$

**Algorithm 1** Algorithm for 1 Newton Step

```
function STEP(α, g, d, c)
    x_k ← g_k − α_k d_k
    Z ← 0
    for k = 0 to K − 1 do
        Z ← Z + α_k / x_k
    end for
    Z ← Z ∗ c
    S ← 0
    for k = 0 to K − 1 do
        S ← S + 1/(1 + Z)/x_k
    end for
    for k = 0 to K − 1 do
        δ_k ← S + x_k ∗ (1 − c ∗ α_i ∗ S)
    end for
    return δ
end function
```

Algorithm 1 calculates one Newton step given the gradient $\mathbf{g}$, the Hessian diagonal vector $\mathbf{d}$, and the Hessian constant $c$. It is derived from equation (3).

# 5 The Dirichlet-Multinomial Solution

The technique for the fast Dirichlet multinomial solution starts along the same lines. Again, the data is an $N$ x $K$ matrix, except now each entry is a count, representing the number of times that category has been seen while sampling that row. Typically some rows will have more samples than others. If a row has all zeros, no data was gathered from this sample, and it will not affect the estimate of the Dirichlet.

Start with the probability of seeing the data matrix $D$ given a set of alphas.

$$P(D|\alpha) = \prod_{n=0}^{N-1} \prod_{k=0}^{K-1} \Gamma\left(\alpha_k, d_{n,k}\right) \Gamma\left(\sum_{k=0}^{K-1} \alpha_k, \sum_{k=0}^{K-1} d_{n,k}\right)^{-1} \Gamma\left(1, \sum_{k=0}^{K-1} d_{n,k}\right) \prod_{k=0}^{K-1} \Gamma\left(1, d_{n,k}\right)^{-1} \tag{4}$$

In order to maximize this, first take the log.

$$\ln(P(D|\alpha)) = \sum_{n=0}^{N-1} (\sum_{k=0}^{K-1} L\Gamma(\alpha_k, d_{n,k}) - L\Gamma(\sum_{k=0}^{K-1} \alpha_k, \sum_{k=0}^{K-1} d_{n,k}) + L\Gamma(1, \sum_{k=0}^{K-1} d_{n,k}) - \sum_{k=0}^{K-1} L\Gamma(1, d_{n,k})) \tag{5}$$

The last two terms are constant with respect to $\alpha$ and can be omitted in the maximization function.

$$F(\alpha) = \sum_{n=0}^{N-1} \sum_{k=0}^{K-1} L\Gamma(\alpha_k, d_{n,k}) - \sum_{n=0}^{N-1} L\Gamma(\sum_{k=0}^{K-1} \alpha_k, \sum_{k=0}^{K-1} d_{n,k}) \tag{6}$$

Unfortunately, the computation of $F$ requires a full read through the data matrix. Because the probability distribution is not in exponential form, there cannot be a sufficient statistic of constant dimension.[10, pg 116] Even so, we can do much better by using the formulas for the dual-input log gamma function and rearranging the terms.

$$F(\alpha) = \sum_{n=0}^{N-1} \sum_{k=0}^{K-1} \sum_{i=0}^{d_{n,k}-1} \ln\left(\alpha_k + i\right) - \sum_{n=0}^{N-1} \sum_{i=0}^{\Sigma_k d_{n,k}-1} \ln\left(\sum_{k=0}^{K-1} \alpha_k + i\right) \tag{7}$$

All of the terms in equation (7) have a similar form. In the first summation block, they each involve the log of an alpha plus a count. In the second block, it's the sum of the alphas plus a count. Some of these will be seen more than once, so it's natural to collect the terms. Let $M$ be the highest possible value of the index $i$, which will be equal to the highest row total.

$$M = \max_n \Sigma_k d_{n,k}$$

We can simplify the formula for $F$ by constructing an $M$-dimensional vector $\mathbf{v}$, and a $K$ x $M$ dimensional matrix, $U$. Let $u_{k,m}$ index the matrix $U$, and $v_m$ index the vector $\mathbf{v}$.

$$F(\alpha) = \sum_{k=0}^{K-1} \sum_{m=0}^{M-1} u_{k,m} \ln(\alpha_k + m) - \sum_{m=0}^{M-1} v_m \ln(\Sigma_k \alpha_k + m) \tag{8}$$

There is an intuitive meaning to the matrix $U$ and vector $\mathbf{v}$. The value $u_{k,m}$ represents the number of rows in the data where the count in column $k$ exceeds $m$. The value $v_m$ represents the number of rows in the data where the total sample in the row exceeds $m$.

$$u_{k,m} = \textstyle\sum_{n=0}^{N-1} \mathbb{I}_{d_{n,k} > m} \quad v_m = \textstyle\sum_{n=0}^{N-1} \mathbb{I}_{\sum_{k=0}^{K-1} d_{n,k} > m}$$

Algorithm 2 takes care of the key pre-computation. We can again look to find the gradient and Hessian of F. The formula for the gradient is as follows:

$$g_k = \sum_{m=0}^{M-1} u_{k,m} (\alpha_k + m)^{-1} - \sum_{m=0}^{M-1} v_m (\Sigma_k \alpha_k + m)^{-1} \tag{9}$$

Again, the Hessian matrix is a constant plus a diagonal. Algorithm 1 can be applied to the results to perform an iteration.

$$h_{i,j} = \sum_{m=0}^{M-1} v_m (\Sigma_k \alpha_k + m)^{-2} - \sum_{m=0}^{M-1} u_{i,m} (\alpha_i + m)^{-2} \mathbb{I}_{i,j} \tag{10}$$

**Algorithm 2** Algorithm for computing $U$ and $\mathbf{v}$

```
U ← 0
for k = 0 to K − 1 do
    v_k ← 0
end for
for m = 0 to M − 1 do
    C ← 0
    for k = 0 to K − 1 do
        C ← c + D[m][k]
        for i = 0 to D[m][k] do
            U[k][i] ← U[k][i] + 1
        end for
    end for
    for i = 0 to C do
        v[i] ← v[i] + 1
    end for
end for
```

# 6 Theoretical Runtime Analysis

In this section, we derive the order of magnitude of the running time for the proposed algorithm, and compare it to Minka's version.

## 6.1 Minka's Algorithm

If the function $F$ were calculated from the summations in equation (7), then the running time would be $O(MNK)$. However, there exists algorithms for the log-gamma, digamma, and trigamma functions which are more efficient than simply adding up the logarithmic terms[13]. We assume that these can be run in constant time. From the double summation in equation (6), the running time becomes $O(NK)$, or the size of the data set.

If the $F$ can be computed in $O(NK)$, so can the gradient and the Hessian components needed to run algorithm 2. If the total number of Newton steps needed to find the MLE within a given precision is $s$, the final running time is $O(sNK)$.

## 6.2 Proposed Algorithm

Algorithm 2 describes the initial sweep through the dataset to calculate $U$ and $\mathbf{v}$. This will have a running time of $O(MNK)$. For the maximization function, we are now looking at equation 8. The calculation of this function is $O(MK)$ and therefore so is the calculation of the gradient, the Hessian, and the entire Newton-Raphson step from algorithm 2.

Again assuming that we require $s$ steps in Newton's Method, the running time for that phase of the calculation will be of $O(sMK)$. Putting it together, the calculation of Newton's method plus the precomputation of $U$ and $\mathbf{v}$ is $O(NMK + sMK)$ or $O((N + s)MK)$.

If $s$ is an order of magnitude lower than $N$, then the bottleneck is reading in the data, which makes the running time effectively $O(NMK)$.[5] When the Newton updates are the bottleneck, the algorithm will have a running time of $O(sMK)$. In that case, the proposed algorithm does not have a running-time benefit over Minka's version.

# 7 Running Time Experiments

The evaluation of the proposed algorithm uses the Fastfit library[6], written by Thomas Minka at Microsoft Research[6]. Fastfit is a Matlab toolbox and contains two functions for evaluating a Dirichlet-multinomial. A third function based on the proposed algorithm was written in order to compare running times on the same platform (See section 8.2). The initialization and stopping criteria were left untouched in order to make a fair comparison[7].

According to the runtime analysis, the proposed algorithm will perform very well when M is small and N grows large. Samples were generated with a 3-dimensional Dirichlet parameter $[3, 1, 2]$, and a constant $M = 10$. The value for $N$ starts at 100 and doubles on each round.

This analysis also includes Minka's fixed-point (FP) solution and a new Matlab version using $U$ and $\mathbf{v}$. The fixed-point algorithm in Fastfit is given as follows:

$$\alpha_k^* = \alpha_k \frac{\sum_{n=0}^{N-1} \Psi(d_{n,k} + \alpha_k) - \Psi(\alpha_k)}{\sum_{n=0}^{N-1} \Psi(\sum_{k=0}^{K-1} d_{n,k} + \alpha_k) - \Psi(\sum_{k=0}^{K-1} \alpha_k)} \tag{11}$$

The proposed version makes use of the parameters $U$ and $\mathbf{v}$ and has the advantage of not having to run through the dataset multiple times.

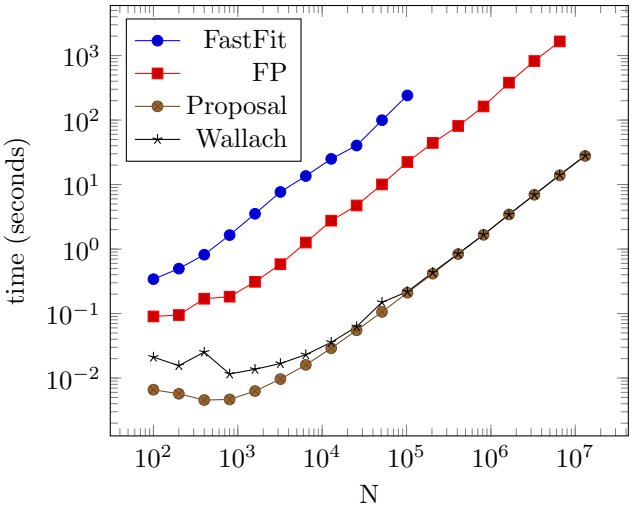
$$\alpha_k^* = \alpha_k \frac{\sum_{m=0}^{M-1} u_{k,m}(\alpha_k + m)^{-1}}{\sum_{m=0}^{M-1} v_m(\sum_{k=0}^{K-1} \alpha_k + m)^{-1}} \tag{12}$$

This is equivalent to Wallach's algorithm using the digamma recurrence relation[13]. The Fastfit fixed-point algorithm has an advantage over the Fastfit Newton algorithm because the bottleneck is reading the data on each iteration, and Newton's method requires more computations per data point. While Newton's method requires computation of both the gradient and the Hessian, the fixed point algorithm needs only the terms from the gradient. With the precalculation of $U$ and $\mathbf{v}$, that advantage for fixed-point iteration becomes less important.

[5] Fortunately, $U$ and $\mathbf{v}$ are additive, so this processes can be parallelized across multiple processors in map-reduce job. More about this in the future work section.

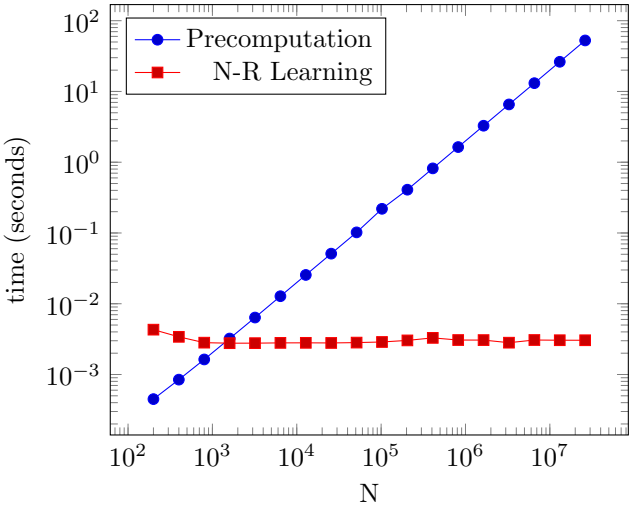
[6] http://research.microsoft.com/en-us/um/people/minka/software/fastfit/

[7] This implementation stops when the absolute value of the gradient is less than $10^{-16}$




The running time for increasing $N$ on the proposed algorithm is hinge-shaped; there is a section where it is constant, and a section where it is increasing linearly. This is in agreement with the running time $O(NK + sMK)$. The constant section is dominated by the term $O(sMK)$, and the linear section by the term $O(NK)$. While any MLE algorithm which reads all of the data has to be at least linear time for increasing $N$, the proposed algorithm reduces running time by a constant factor of over 1000 in this case. For high $N$, Algorithm 2 dominates the running time for both the proposed algorithm and Wallach's algorithm.

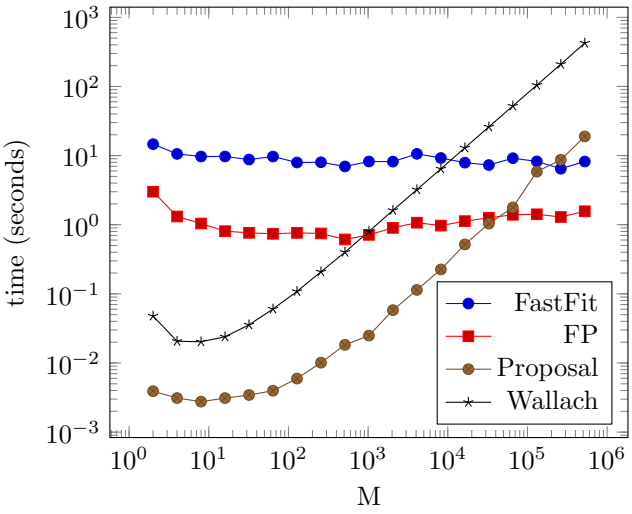
Since the proposed algorithm divides into two parts, the next graph compares the time for precomputation with the time for executing Newton-Raphson.




The running time for the Newton-Raphson phrase becomes negligible with respect to the precomputation for large enough $N$. Suppose that one wanted to find the current estimate after every 1000 rows we add to $U$ and $\mathbf{v}$. This would not add significant overhead and would allow for an implicit online learning mode where increasingly accurate intermediate answers are produced as the data is processed.

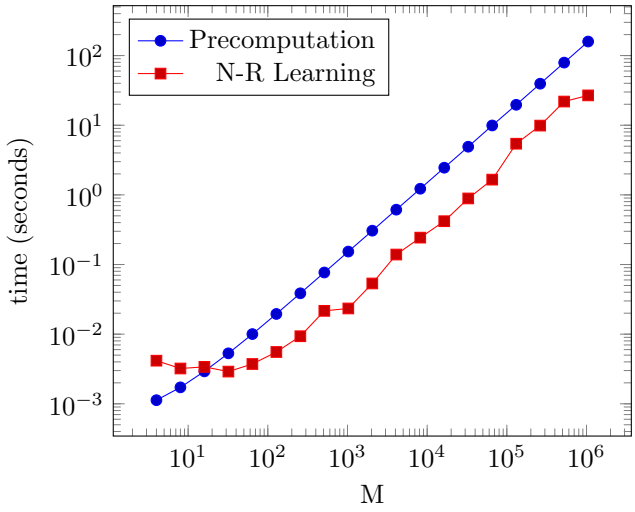
## 7.1 Limitations of the Proposed Algorithm

When $M$ is large and $N$ is held constant, the proposed algorithm is not expected to run as well. Here the same Dirichlet parameter was used to generate the dataset and $N$ was held constant at 5000.

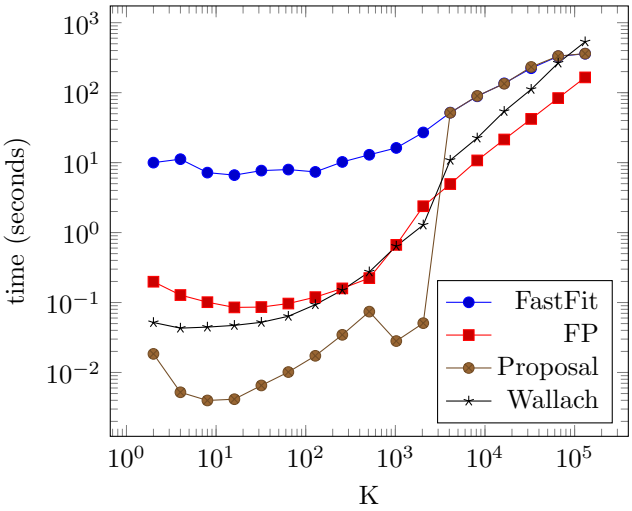



While eventually the proposed algorithm performs worse than the Fastfit version, this only occurs when $M$ is close to $10,000$. In many circumstances, it is not neccesary to have this many samples per row. If the multinomial for that row is already known with such high precision, it might make sense to use the fast Dirichlet solution (section 4) or subsample the rows to a reasonable size. Note that as $N$ increases, the lines will cross at higher and higher values of $M$.

The next graph shows the difference between the two phases with ever increasing $M$. Since the matrix $U$ has $M$ columns, the Newton-Raphson step can no longer remain constant. However, it still settles at around 80% the speed of the precomputation step.



Finally, the algorithms were tested with different values for $K$, the number of categories. The constants were set at $M = 50$, $N = 5000$, and $\alpha_k = 1/K$. The benefits of the proposed algorithm are evident through 2048 dimensions. The negative results for very high $K$ contradict our running-time analysis, and may be caused by issues related to memory allocation and the internal data structures.




# 8 Conclusions

Theoretical analysis and experiments suggest that the proposed Newton-Raphson algorithm for computing the MLE for a Dirichlet multinomial has improved running times over typical implementations where $M$ is small and $N$ is large.

The proposal also provides a separation between the precomputation step and the Newton-Raphson algorithm. In the precomputation step, the results from each row can be added independently and therefore it is easy to parellelize, stop early, or add more data. The Newton-Raphson step, not having to read the entire dataset, runs in constant time as the number of rows increases.

## 8.1 Future Work

In many Dirichlet multinomial datasets, the amount of data in each row follows a power law distribution. In other words, a small number of rows contain a large number of samples, and a large number of rows (the *long tail*) contain a small number of samples. There are two possibilities for handling this case.

First, if the highly sampled rows could be subsampled, the value of $M$ would decrease and the proposed algorithm would run faster. Some analysis would have to be done on exactly how valuable that data is with respect to the MLE for the Dirichlet, but intuitively that would be the least valuable data to sample away.

Another possibility is to pick a desired value for $M$ and split the dataset into two parts. The first part contains all the rows that have more than $M$ samples, and the second will be used to compute $U$ and $\mathbf{v}$. Both of these datasets will be far smaller than the original. Minka's technique will be applied to the first set

to find the gradient and Hessian, and the proposed technique will be used for the second set with the results added together. This hybrid approach would perform better than either of the two approaches independently. The value for $M$ may not even need to be predetermined. A data structure could keep track of free rows, $U$ and $\mathbf{v}$. This data structure would accept new rows and only go back and subsume those rows into $U$ and $\mathbf{v}$ when it shrinks the size of the representation.

Another application of this technique is to use it to compute mixture models, which are far more powerful than a single Dirichlet multinomial model. In a simple mixture model it is assumed that each row in the dataset was produced by a multinomial coming from a single Dirichlet model in the mixture. Such a model could favor Dirichlets with high weights so that it looks like a fuzzier version of the multinomial mixture model. The Latent Dirichlet Allocation is more complex, since different points in each row could be produced by different distributions. The proposed algorithm, along with the expectation-maximization algorithm, could be used to build these.

Finally, a parallelized map-reduce version of algorithm 2 (the precomputation) can be built. For accurately estimating a Dirichlet multinomial, a large dataset is overkill. But suppose that a mixture model is being computed with many Dirichlet clusters to choose from over a sharded dataset. In this case, a parellelized version of algorithm 2 becomes far more important.

### 8.2 The Code

Two implementations for the MLE estimate of the Dirichlet-multinomial have been developed: a Python version and a Matlab version. The main implementation is in Python. A Dirichlet MLE estimate relies on NumPy, but the Dirichlet-Multinomial uses only functions from the math package. This could be found in github in both an actively developed repository[8] and a repository specifically for this paper that will not change much except for corrections or to make it more relevant to this paper[9]. An implementation in Matlab which extends the FastFit library was developed for the experiments and is kept in the same repository[10].

## References


[1] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. the Journal of machine Learning research, 3, 993-1022.

[2] Dishon, M., & Weiss, G. H. (1980). Small sample comparison of estimation methods for the beta distribution. Journal of Statistical Computation and Simulation, 11(1), 1-11.

[3] Hariharan, H. S., & Velu, R. P. (1993). On estimating dirichlet parameters—a comparison of initial values. Journal of statistical computation and simulation, 48(1-2), 47-58.

[4] Heinrich, G. (2005). Parameter estimation for text analysis. Web: http://www. arbylon. net/publications/text-est. pdf.

[5] Hijazi, R. H., & Jernigan, R. W. (2009). Modelling compositional data using Dirichlet regression models. Journal of Applied Probability & Statistics, 4, 77-91.

[6] Minka, T. (2000). Estimating a Dirichlet distribution. Technical report, M.I.T., 2000.


[8] https://github.com/maxsklar/BayesPy/tree/master/ConjugatePriorTools
[9] https://github.com/maxsklar/research/tree/master/2014_05_Dirichlet/python
[10] https://github.com/maxsklar/research/tree/master/2014_05_Dirichlet/matlab

[7] Narayanan, A. (1991). Algorithm AS 266: maximum likelihood estimation of the parameters of the Dirichlet distribution. Journal of the Royal Statistical Society. Series C (Applied Statistics), 40(2), 365-374.

[8] Ng, K. W., Tian, G. L., & Tang, M. L. (2011). Dirichlet and Related Distributions: Theory, Methods and Applications (Vol. 888). John Wiley & Sons.

[9] Mosimann, J. E. (1962). On the compound multinomial distribution, the multivariate $\beta$-distribution, and correlations among proportions. Biometrika, 49(1-2), 65-82.

[10] Robert, C. (2007). The Bayesian choice: from decision-theoretic foundations to computational implementation. Springer.

[11] Ronning, G. (1989). Maximum likelihood estimation of Dirichlet distributions. Journal of statistical computation and simulation, 32(4), 215-221.

[12] Sklar, M., Shaw, B., & Hogue, A. (2012, September). Recommending interesting events in real-time with foursquare check-ins. In Proceedings of the sixth ACM conference on Recommender systems (pp. 311-312). ACM.

[13] Wallach, H. M. (2008). Structured topic models for language. Unpublished doctoral dissertation, Univ. of Cambridge.

[14] Wicker, N., Muller, J., Kalathur, R. K. R., & Poch, O. (2008). A maximum likelihood approximation method for Dirichlet's parameter estimation. Computational Statistics & Data Analysis, 52(3), 1315-1322.

[15] Woodbury, M. A. (1950). Inverting modified matrices. Memorandum report, 42, 106.