Analyzing Methods for Aggregate-Disaggregate Data Fusion

By

Vinay Kasat

An Undergraduate Thesis submitted in partial fulfillment of the requirements for the

JOSEPH WHARTON SCHOLARS

Faculty Advisor:

Peter S. Fader

Frances and Pei-Yuan Chia Professor, Marketing

THE WHARTON SCHOOL, UNIVERSITY OF PENNSYLVANIA

MAY 2020

1 Abstract

This thesis examines methods for aggregate-disaggregate data fusion. In the presence of both accurate, aggregate-level data which often lacks granularity, and disaggregate but potentially biased individual-level data, marketers and statisticians alike struggle in determining the appropriate weight to assign to each piece of information. Through simulation, this paper tests the effectiveness of the use of a multivariate normal approximation to aggregate-level data. An alternative algorithm to impute the missing data, subject to aggregate characteristics, and then estimate parameters is posed and evaluated against the initial method. The paper finds that the multivariate normal approximation not only outperforms the proposed algorithm but also quite accurately estimates model parameters.

Table of Contents

2 Introduction

This thesis examines methods for aggregate-disaggregate data fusion. Traditionally, access to consumer behavior data has been limited to either small, non-representative samples of disaggregate data, or aggregate summary statistics which lack granularity. However, both access to and affordability of data have increased drastically in recent years. Determining how best to fuse aggregate and disaggregate sources of data poses a challenge to marketers and statisticians alike.

Section 3 of this paper reviews the literature surrounding data fusion. Section 4 details both an existing and a proposed method for tackling the data fusion problem and explains how the two will be evaluated. Sections 5 and 6 explore the existing and proposed methods respectively and discuss extensions.

3 Literature Review

Modeling and understanding consumer behavior are important tasks for marketers, statisticians, and researchers. Traditionally, research in consumer behavior has had one of two focuses: 1) using aggregate data to estimate information about individuals or cohorts of individuals, or 2) using a sample of disaggregate, individual-level data to make inferences about a population (Chen and Yang 2007). The former approach benefits from the widespread and often publicly available nature of aggregate level data but suffers from a lack of granularity (McCarthy and Oblander 2020). The latter approach provides such granularity but faces problems with both data availability and representativeness.

The difficulty in incorporating aggregate data into models fit on disaggregate data has been explored before. Two main classifications of approaches have been explored in prior literature: non-likelihood-based approaches and likelihood-based approaches (Chen and Yang 2007). Non-likelihood-based approaches often suffer from arbitrary definitions of how well a model conforms to the given aggregate data. Many methods for fitting models to aggregate data have been explored before, including minimizing discrepancy between observed and predicted values (Boyd and Mellman 1980; Cardell and Dunbar 1980) or using constrained optimization techniques with somewhat arbitrarily defined functions (Berry 1994). The fact that these methods may be arbitrary is not to say they perform poorly, but rather that they lack favorable statistical properties which can be derived from likelihood-based approaches.

The alternative approach, specifying a likelihood function of the aggregate data, has also been explored in prior research (Bodapati and Gupta 2004). However, these approaches often result in functions which are not easily manageable or computationally tractable. Thus, much of the research taking this or a similar approach relies on simulation and imputation of data (Bodapati and Gupta 2004; Feit, Wang, Bradlow, and Fader 2013). However, these approaches can also be time consuming or costly. Issues in these approaches often stem from the difficulty in forcing imputed data to behave in accordance with the observed aggregate data.

Incorporating disaggregate data from samples of the population into models fit on aggregate data also poses several problems. Most notably, dealing with selection bias and the potential non-representability of disaggregate data result in difficulties for those studying consumer behavior.

The problems explored above demonstrate the need for methods to fuse aggregate and disaggregate data. Availability of data has increased drastically over the past several years. As a result, research has turned to the use of aggregate and disaggregate data to augment each other. However, where these approaches succeed within their respective disciplines or data paradigms, they lack generalizability or other desirable characteristics, as noted by McCarthy and Oblander (2020). The problem of aggregate-disaggregate data fusion is important and serves as the basis for this analysis.

4 Methodology

4.1 The Data Paradigm

This paper explores methods for aggregate-disaggregate data fusion and evaluates their effectiveness. The specific data paradigm I examine has individual level data given by:

$$Y_i \sim D_{\boldsymbol{\theta}}, 1 \leq i \leq N$$

where D is some underlying distribution or model which generates the data and $\boldsymbol{\theta}$ is a vector of parameters. Each Y_i is taken to be independent and identically distributed. However, only a subset of this data is observable, along with an aggregate function over the entire data. Thus, the observable data is:

$$Y_i \sim D_{\boldsymbol{\theta}}, 1 \leq i \leq k \text{ and } f(\boldsymbol{Y})$$

where $\boldsymbol{Y}$ is a vector of the population data and f is some function of this data. In this paper, f is taken to be a function computing the average of the data, meaning the observable data is a disaggregate sample of size k and the population mean. Given these two pieces of information, one can equivalently compute the mean over just the aggregated data, meaning the observable data can be expressed as:

$$Y_i \sim D_{\boldsymbol{\theta}}, 1 \leq i \leq k \text{ and } \mu_{agg} = {\textstyle\sum_{i=k+1}^{N} Y_i} / {N-k}$$

To ensure the robustness of the presented results against different model formulations, this paper explores two distributions for $D_{\boldsymbol{\theta}}$: the Gamma distribution and a Normal Mixture Model. The density of each is given respectively by:

$$g(y) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} * y^{\alpha-1} * e^{-\beta y} \ , \alpha > 0, \beta > 0$$

$$g(y) = p\frac{1}{\sqrt{(2\pi)\sigma_1}} e^{{-(y-\mu_1)^2}/{2\sigma_1^2}} + (1-p)\frac{1}{\sqrt{(2\pi)\sigma_2}} e^{{-(y-\mu_2)^2}/{2\sigma_2^2}}, \sigma_1 > 0, \sigma_2 > 0, 0 \leq p \leq 1$$

Computation, simulation, and optimization were done in R. Care was given to robustly checking optimization to ensure optimality.

4.2 Measure of Fit

To measure the accuracy of parameter estimation techniques, this paper uses the Kullback-Leibler divergence (KL Divergence), also called the relative entropy. KL Divergence measures how one probability distribution differs from a reference distribution. For continuous distributions, it is defined as:

$$KL(g_1(y)||g_2(y)) = \int_S g_1(y) * \log(\frac{g_1(y)}{g_2(y)})dy$$

where S is the support of the densities. The KL divergence between two Gamma distributions is well documented and has a closed form solution. Between two Normal Mixture Models, the KL divergence is difficult to directly compute; however, there are a variety of methods to obtain very precise approximations. This paper approximates the statistic by approximating each Normal Mixture Model density as a discrete distribution with small bucket width and computing the integral above as a sum. Given the small interval size used, as well as the well-behaved nature of the Normal Mixture Model[1], this approximation likely results in negligible error.

4.3 Procedural Outline

The following sections will investigate two methods for aggregate-disaggregate data fusion: approximation of the distribution of the aggregate data using a multivariate normal distribution, and a proposed modification to the Expectation Maximization algorithm. Section 5 explores the multivariate normal approximation and its performance. Section 6 derives and evaluates the modified Expectation Maximization algorithm for aggregate-disaggregate data fusion.

[1] The tails of the distribution converge rapidly to 0 and the distribution is smooth

5 Data Fusion: The Multivariate Normal Approximation

5.1 The Multivariate Normal Approximation

This section examines the multivariate normal approximation for aggregated data proposed by McCarthy and Oblander (2020). Presented with the full disaggregated data, the log-likelihood is:

$$L(\boldsymbol{\theta}|\boldsymbol{y}) = \sum_{i=1}^{N} log\big(P(Y_i = y_i|\boldsymbol{\theta})\big)$$

However, when confronted with the data paradigm as detailed above, the likelihood is instead:

$$L(\boldsymbol{\theta}|\boldsymbol{y}) = \sum_{i=1}^{k} log\big(P(Y_i = y_i|\boldsymbol{\theta})\big) + \log\Big(P\big(\textstyle\sum_{i=k+1}^{N} Y_i = (N-k) * \mu_{agg}\big|\boldsymbol{\theta}\big)\Big)$$

$$= \sum_{i=1}^{k} log\big(P(Y_i = y_i|\boldsymbol{\theta})\big) + \log\left(P\left(\Big.\textstyle\sum_{i=k+1}^{N} Y_i \Big/ (N-k) = \mu_{agg}\right|\boldsymbol{\theta}\Big)\right)$$

Computing the second term requires a high-dimensional convolution and is not computationally tractable. However, under the conditions of the central limit theorem, the second term asymptotically converges to a multivariate normal distribution. Furthermore, given the assumption of independence, the distribution of the average asymptotically converges to a normal distribution with a mean parameter equal to the mean under the model parameters, and a variance equal to the variance under the model parameters divided by $N-k$. Thus, the log-likelihood can be approximated by:

$$\sum_{i=1}^{k} log\big(P(Y_i = y_i|\boldsymbol{\theta})\big) + \log\left(\frac{1}{\sqrt{2\pi}\sigma_{\boldsymbol{\theta}}} * e^{-(\mu_{\boldsymbol{\theta}}-\mu_{agg})^2 \Big/ 2\sigma_{\boldsymbol{\theta}}^2}\right)$$

where $\mu_\theta = E_\theta[Y_i], \sigma_\theta^2 = \frac{Var_\theta[Y_i]}{(N-k)}$. Maximizing over this function, dubbed the proxy likelihood, yields consistent estimators of the unknown parameters. Under the first model used to generate data, the Gamma distribution with parameters α and β, these parameters for the normal approximation are:

$$\mu_\theta = \frac{\alpha}{\beta}, \sigma_\theta^2 = \frac{\alpha}{\beta^2 * (N-k)}$$

Under the Normal Mixture Model, the parameters for the normal approximation are:

$$\mu_\theta = p * \mu_1 + (1-p) * \mu_2, \sigma_\theta^2 = \frac{p * \sigma_1^2 + (1-p) * \sigma_2^2 + p * (1-p) * (\mu_1 - \mu_2)^2}{N-k}$$

Given this methodology, several questions can be posed. This paper investigates two: 1) what, if any, is the improvement in model fit achieved by the incorporation of the multivariate normal approximation to the aggregate data, and 2) how much disaggregate data is necessary in addition to the aggregate data, to obtain a sufficiently good fit? The following sections explore these questions.

5.2 Data Fusion vs. Disaggregate Models

First tested is the improvement in model fit achieved by incorporating the multivariate normal approximation to the aggregate data. To do this, for a fixed sub-sample of size *k* taken from the entire data, the maximum likelihood parameters using just the disaggregate data are computed. Using the data fusion approach outlined above, a second set of parameters is obtained by maximizing over the proxy likelihood function, considering both the disaggregate data and the mean of the remaining data. These computations yield two sets of parameters, $\boldsymbol{\theta}_{\boldsymbol{MLE}}^{\boldsymbol{k}}$ and $\boldsymbol{\theta}_{\boldsymbol{MPL}}^{\boldsymbol{k}}$ respectively. These parameters are compared to the maximum likelihood parameters obtained using the entire data, $\boldsymbol{\theta}_{\boldsymbol{MLE}}^{\boldsymbol{N}}$[2]. The maximum likelihood parameters using the entire data represent the maximum possible information that can be derived from the observed data[3], and thus comparing the closeness of $\boldsymbol{\theta}_{\boldsymbol{MLE}}^{\boldsymbol{k}}$ and $\boldsymbol{\theta}_{\boldsymbol{MPL}}^{\boldsymbol{k}}$ with $\boldsymbol{\theta}_{\boldsymbol{MLE}}^{\boldsymbol{N}}$, the improvement in model fit achieved through the use of maximum proxy likelihood is determined.

[2] Note, given this approach to data fusion, $\boldsymbol{\theta}_{\boldsymbol{MLE}}^{\boldsymbol{N}} = \boldsymbol{\theta}_{\boldsymbol{MPL}}^{\boldsymbol{N}}$. This conceptually makes sense, as when one can observe the entire data, any aggregation will result in information loss and thus an inferior model fit.

[3] This is not necessarily true, but for the purposes of model evaluation it will not be a restrictive or limiting assumption. It is similar to the Strong Likelihood Principle

The goodness of the computed parameters, relative to the optimal ones, is measured by computing KL divergences[4]. The relevant statistics are then $KL(g(y|\boldsymbol{\theta}^{\boldsymbol{k}}_{\boldsymbol{MLE}})||g(y|\boldsymbol{\theta}^{\boldsymbol{N}}_{\boldsymbol{MLE}}))$ and $KL(g(y|\boldsymbol{\theta}^{\boldsymbol{k}}_{\boldsymbol{MPL}})||g(y|\boldsymbol{\theta}^{\boldsymbol{N}}_{\boldsymbol{MLE}}))$. As $k \to N$, both statistics approach 0. The formulas for computing each under the two focal models are detailed above in Section 4.

To test this, $N = 500$ draws from a Gamma distribution with parameters $\alpha = 1, \beta = 1$ are simulated. For each $2 \leq k \leq N$, $\boldsymbol{\theta}^{\boldsymbol{k}}_{\boldsymbol{MLE}}$ and $\boldsymbol{\theta}^{\boldsymbol{k}}_{\boldsymbol{MPL}}$ are computed by maximizing over their respective likelihood functions, and the two KL divergences from the estimates to the maximum likelihood estimates on the full data are compared. This process was repeated for the Normal Mixture, with parameters $\mu_1 = -2, \sigma_1^2 = 1, \mu_2 = 4, \sigma_2^2 = 2, p = .8$. The results are below:

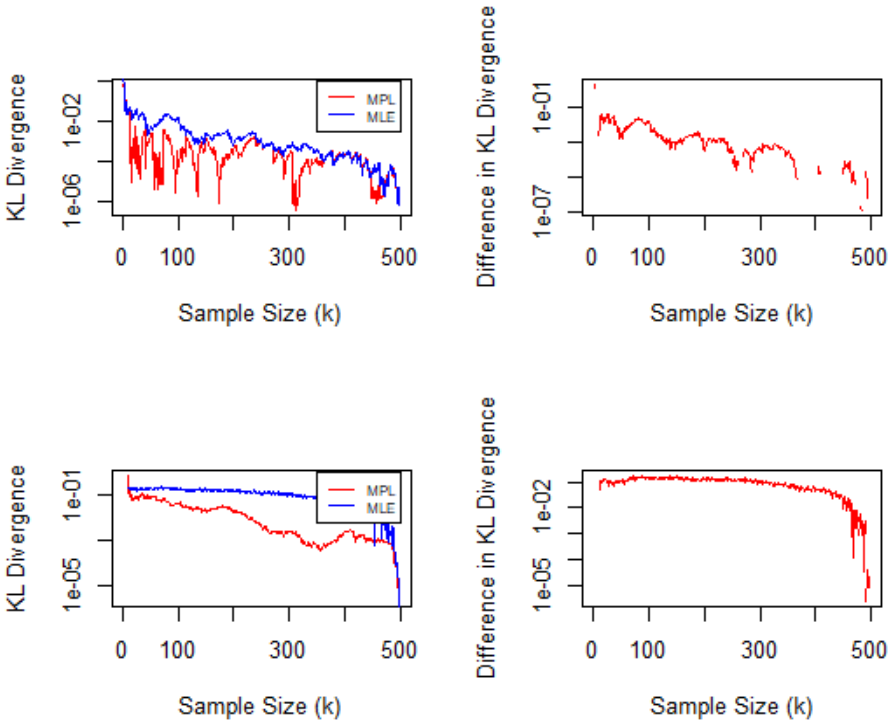

Figure 1: MPL vs MLE

[4] It is incorrect to directly compare the likelihood under maximum likelihood estimation to the likelihood under maximum proxy likelihood. Both approaches are computed on different 'scales', so direct comparisons have little meaning. Thus, a different measure of similarity must be used to compare distributions.

The set of graphs on top refer to the Gamma distribution and the graphs on the bottom refer to the Normal Mixture Model. For smaller values of k, maximum proxy likelihood clearly outperforms maximum likelihood on solely the aggregate data. However, as expected, the difference in KL divergence decreases as k approaches N.

To more robustly demonstrate this result, this procedure is repeated with $N = 10{,}000$ for both models, examining the difference in KL divergence when $k = 100, k = 1{,}000$, or 1% and 10% of the data is disaggregated. This simulation is repeated 100 times. The results are depicted below.

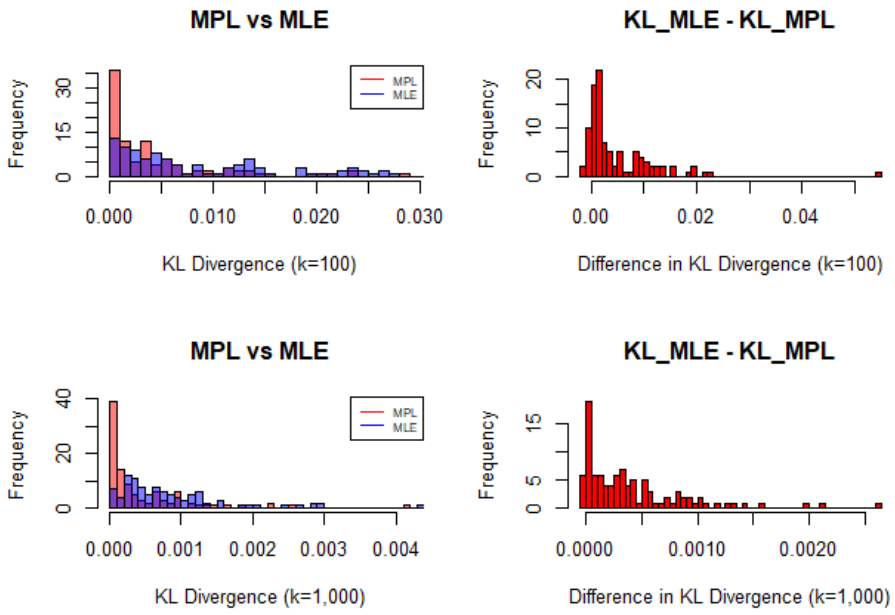

Figure 2: MPL vs MLE for k=100, 1000 Under Gamma

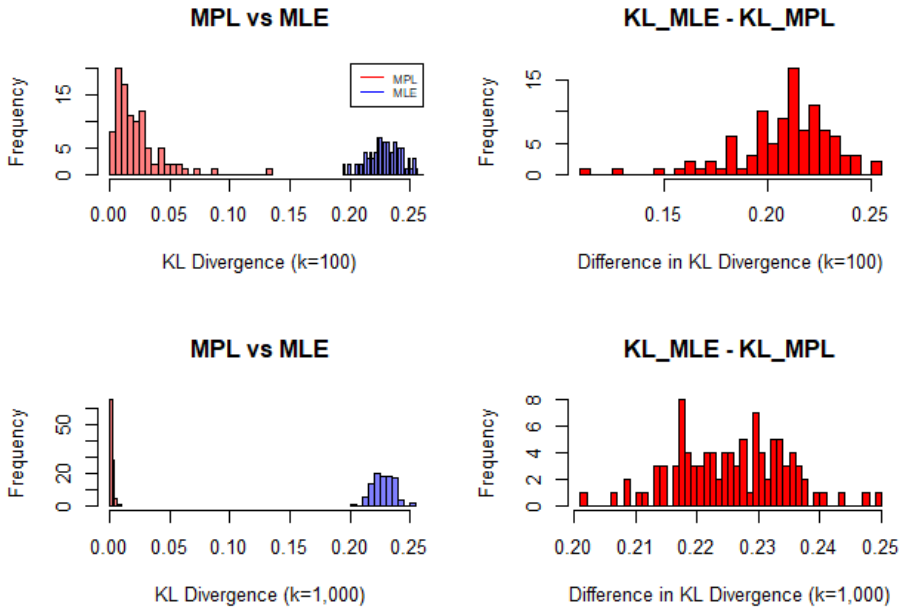

Figure 3: MPL vs MLE for k=100, 1000 Under Normal Mixture

Figures 2 and 3 clearly show that in most cases, the model fit under data fusion performs better than the model fit using the disaggregate data alone. While in the case of the Gamma, the distributions of KL Divergence overlap, for the more complicated Normal Mixture Model specification, maximum proxy likelihood far outperforms maximum likelihood estimation. For each sample, both $KL(g(y|\boldsymbol{\theta}^{\boldsymbol{k}}_{\boldsymbol{MLE}})||g(y|\boldsymbol{\theta}^{\boldsymbol{N}}_{\boldsymbol{MLE}}))$ and $KL(g(y|\boldsymbol{\theta}^{\boldsymbol{k}}_{\boldsymbol{MPL}})||g(y|\boldsymbol{\theta}^{\boldsymbol{N}}_{\boldsymbol{MLE}}))$ were computed, displayed in the histograms on the left, as well as $KL(g(y|\boldsymbol{\theta}^{\boldsymbol{k}}_{\boldsymbol{MLE}})|\left|g(y|\boldsymbol{\theta}^{\boldsymbol{N}}_{\boldsymbol{MLE}})\right) - KL(g(y|\boldsymbol{\theta}^{\boldsymbol{k}}_{\boldsymbol{MPL}})||g(y|\boldsymbol{\theta}^{\boldsymbol{N}}_{\boldsymbol{MLE}}))$, displayed in the histograms on the right.

Based on the data collected, a paired t-test was performed to determine if the model fit using data fusion was better than the model fit using just the disaggregate data. The results of the tests are summarized in the table below:

Model	Disaggregate Sample Size	Method	Mean of KL Divergence (μ_{KL})	Variance of KL Divergence (σ^2_{KL})	T-statistic for $\mu^{MPL}_{KL} \leq \mu^{MLE}_{KL}$	p-value ($\alpha = .05$)
Gamma Distribution	100	MPL	.00464	3.692e-05	6.57	1.19e-09
		MLE	.00956	9.291e-05		
	1,000	MPL	4.473e-04	4.232e-07	8.660	4.530e-14
		MLE	8.728e-04	7.619e-07		
Normal Mixture Model	100	MPL	.02252	3.97e-04	86.94	0.00
		MLE	.2305	2.83e-04		
	1,000	MPL	.00187	2.11e-06	251.62	0.00
		MLE	.2274	8.18e-05		

Figure 4: Results from MPL vs MLE

These simulations depict that there is a clear improvement in model fit achieved by the incorporation of the multivariate normal approximation to the aggregate data under the tested model specifications. To ensure robustness, these simulations were replicated on data generated with different parameters, with similarly strong results. The next section investigates how much disaggregate data is necessary in addition to the aggregate data, to obtain a sufficiently good fit.

5.3 Marginal Benefits to Increased Disaggregate Sample Size

While it is evident that regardless of the parameter estimation method chosen, an increased sample size will yield a more accurate fit, it is not clear how meaningful these incremental expected improvements in fit are. To test this, this paper investigates whether $KL(g(y|\boldsymbol{\theta}^{\boldsymbol{k}}_{\boldsymbol{MPL}})||g(y|\boldsymbol{\theta})) > KL(g(y|\boldsymbol{\theta}^{\boldsymbol{k^\star}}_{\boldsymbol{MPL}})||g(y|\boldsymbol{\theta}))$, where $\boldsymbol{\theta}$ represents the true model parameters and $k^\star > k$. This test compares parameters fit over two different disaggregate sample sizes with the true underlying parameters of the model used. Furthermore, the change in parameter estimates under data fusion via maximum proxy likelihood behave as the disaggregate sample size increases is estimated.

To test this, $N = 10{,}000$ draws from a Gamma distribution with parameters $\alpha = .1, \beta = .1$ are simulated. For $k = 100, k = 1{,}000, k = 10{,}000$, $\boldsymbol{\theta}^{\boldsymbol{k}}_{\boldsymbol{MPL}}$ is computed, along with the KL divergence from these estimates to the actual model parameters. This process was repeated for the Normal Mixture Model, with parameters $\mu_1 = 0, \sigma_1^2 = 1, \mu_2 = 10, \sigma_2^2 = 5, p = .4$. The entire procedure was repeated 100 times. As would be expected, the model performance, as measured by KL divergence from the parameter estimates to the true parameters, does significantly

improve as sample size increases. Paired t-tests confirm this result. However, this result is to be expected[5] based on the prior investigation. Therefore, this paper turns to investigate whether the differing parameter estimates have meaningful implications for the model fits we obtain.

To test this, a modified version of a posterior predictive check is used. Rather than compare simulated data from MPL parameter estimates with the actual underlying data, functions of the parameter estimates with functions of the true parameters are compared. The most meaningful and interesting of these tests occurs when comparing the variance implied by the estimated parameters to the variance implied by the actual parameters[6]. The results from this test are depicted below.

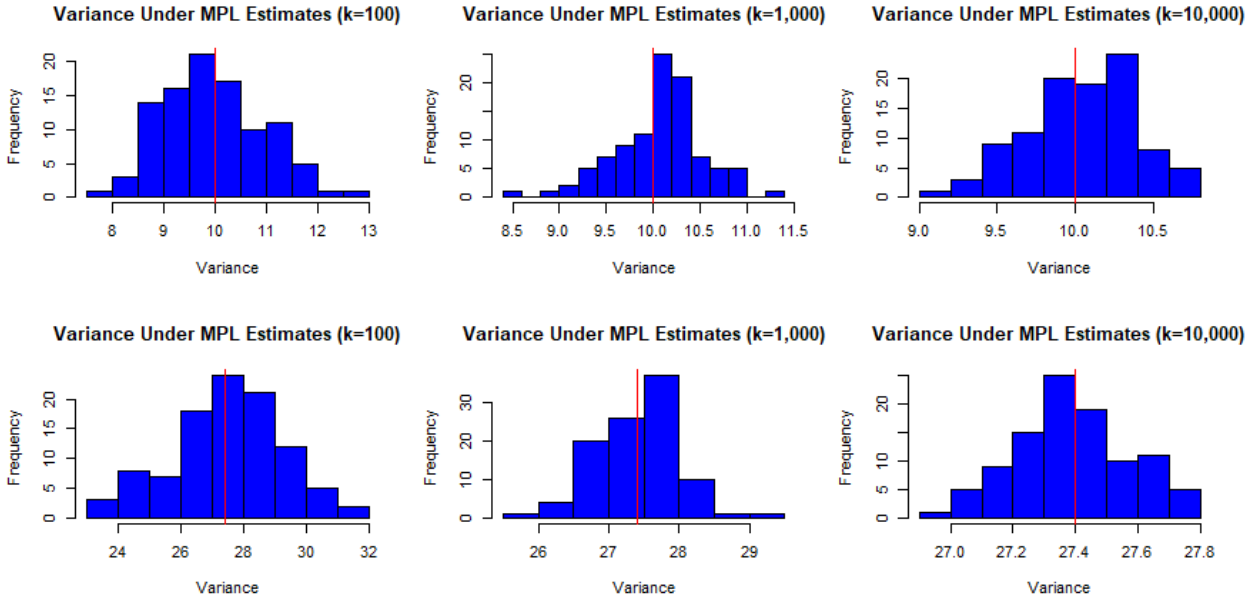

Figure 5: Variance Under MPL Estimates vs True Variance

The top three histograms refer to the Gamma, whereas those on the bottom refer to the Normal Mixture. They display, for a given disaggregate sample size k, the variance that the estimated model parameters imply, against the theoretical true variance, depicted by a red line.

[5] And is also somewhat uninteresting

[6] Comparing the mean under the parameter estimates to the mean under the true parameters does not make conceptual sense, as MPL indirectly fits the mean of the entire data.

As the histograms depict, even at small disaggregate sample sizes, the MPL estimates imply quite similar model variances, with no clear biases. The 'p-values' implied by this test, the proportion of test statistics over the parameter estimates which exceed the theoretical test statistic, were equally good across all values of k. A parametric bootstrap procedure was also performed, by comparing both the parameter estimates and functions of these parameters for a given k to their counterparts for a different value $k^{\star}$. These tests found no significant differences between the parameter estimates at the three chosen levels of k. This is with the exception of comparing parameter estimates under the Normal Mixture Model when $k = 100$ to when $k = N = 10{,}000$, where the latter was found to have a significantly smaller spread of implied variances.

These results are quite positive, demonstrating that in some cases, just one percent of the full data, in addition to an aggregate summary statistic, can be enough to obtain a strong model fit. The next section develops and examines an alternative proposed method for data fusion.

6 Data Fusion: The Modified EM Algorithm

6.1 Developing The Algorithm

The difficulty in computing a likelihood over the aggregate data stems from the computation of a high-dimensional convolution. While when using simple distributions, convolutions often result in known distributions, with more complicated models, high dimensional convolutions have no closed form and are not computationally tractable. As this problem scales to larger datasets, estimating these convolutions can be both timely and inaccurate. Several methods have been explored to tackle this or similar problems. This paper aims to develop an efficient method for imputing data from a distribution that conform to aggregate constraints, similarly to in Feit et al. (2013) and Dong and Taylor (1995).

Ideally, one would be able to directly compute the log-likelihood of the entire data, given by:

$$L(\boldsymbol{\theta}|\boldsymbol{y}) = \sum_{i=1}^{N} log\big(P(Y_i = y_i|\boldsymbol{\theta})\big)$$

As explained in section 4, in the data paradigm under consideration this reduces to:

$$L(\boldsymbol{\theta}|\boldsymbol{y}) = \sum_{i=1}^{k} log\big(P(Y_i = y_i|\boldsymbol{\theta})\big) + \log\left(P\left({\textstyle\sum_{i=k+1}^{N} Y_i}\Big/{(N-k)} = \mu_{agg}\middle|\boldsymbol{\theta}\right)\right)$$

However, while the observations $Y_i, i > k$ cannot be directly observed, they can be simulated. By imputing this missing data, one could then estimate the log-likelihood as:

$$L(\boldsymbol{\theta}|\boldsymbol{y}) = \sum_{i=1}^{k} log\big(P(Y_i = y_i|\boldsymbol{\theta})\big) + \sum_{i=k+1}^{N} log\left(P\big(\hat{Y}_i = y_i|\boldsymbol{\theta}\big)\right)$$

where $\hat{Y}_i$ are simulated data.

The simulation of the 'missing' data poses several additional problems. Primarily, this data must be simulated under the restriction that they sum to a fixed value, $(N-k) * \mu_{agg}$. This alone can be done easily. However, the simulated data under this constraint are very unlikely to be representative of the underlying distribution, as their simulation was done without regard to the underlying distribution of the data. Likewise, simulating data from the underlying distribution

can be done easily. However, there is no guarantee this simulated data will sum to the correct value. Thus, the combination of the linear restriction and the underlying distribution together make simulation difficult. In fact, to properly simulate from the underlying distribution, while ensuring the values sum to a fixed constant, one needs to compute the distribution of the sum of these variables. This brings the conversation back to the issue of computing a high-dimensional convolution.

A method for simulating the data should be evaluated on several criteria: whether it sums to the fixed constant, how representative it is of the underlying distribution, and how long it takes to simulate. The first of these criteria can easily be evaluated. The second can be evaluated by computing the maximum likelihood parameters on $Y = \{Y_i, 1 \leq i \leq k\} \cup \{\widehat{Y_i}, k+1 \leq i \leq N\}$ and comparing them to the true parameters used to generate the underlying distribution, via the KL divergence. The final criteria can be analyzed based on asymptotic runtime; ideally, this simulation should run in $O(N)$ time.

With these criteria in mind, the following algorithm is proposed:

ModifiedEMAlgorithm ← function $(y_{1:k}, \mu_{agg}, N, k)$
    $\theta_{cur} \leftarrow argmax_\theta\{MLE(y_{1:k})\}$ *//initial parameters computed through MLE on disaggregate data*
    $\hat{y}_{k+1:N} \leftarrow \mu_{agg}$
    repeat j times {
        //expectation step
        for pairs (y_1, y_2) in $\hat{y}_{k+1:N}$ {*//generate pairs of imputed datapoints without replacement*
            $total \leftarrow y_1 + y_2$
            $(y_1, y_2) \leftarrow$ Convolution $((y_1, y_2), total, \theta_{cur})$ *//Simulate new values for the two imputed datapoints under the current parameters, subject to their sum not changing*
        }
        //maximization step
        $\theta_{cur} \leftarrow argmax_\theta\{MLE(y_{1:k} \cup \hat{y}_{k+1:N})\}$ *//perform MLE on union of disaggregate and simulated aggregate data to obtain new parameters*
    }
    return (θ_{cur})

There are several features of this algorithm worth discussing. The algorithm's ultimate goal is to return parameters which are consistent with the observed disaggregate and aggregate data. It initializes the parameter estimates to the maximum likelihood estimates over just the disaggregate data. It additionally imputes all the initial 'missing' data as μ_{agg}. This initial imputation ensures the data sums to the correct value. Next, it repeats for a fixed number of

iterations a modified version of the Expectation Maximization (EM) algorithm[7]. The expectation step conceptually serves to update the imputed data values $\hat{y}_{k+1:N}$ to be consistent with the current parameters, while ensuring they keep the same sum. Doing this generally would require a high-dimensional convolution, as discussed before. Alternatively, however, one could repeatedly select small subsets of the data and sample those, while ensuring they have the same sum as initially. This would ensure that the total sum stayed constant, while only requiring a lower-dimensional convolution. Therefore, this algorithm simulates a $N - k$ dimensional convolution through repeated two-dimensional convolutions. This comes at the tradeoff of slower convergence, as the value of any given imputed value is highly correlated with its value in the previous iteration[8]. However, by sampling different pairs of the imputed data values every time, the algorithm can still converge relatively quickly.

The degree of randomness presented by resampling from the convolution, along with the issue of autocorrelation, mean that the algorithm will constantly be jumping about the parameter estimates, rather than converging to them. Therefore, this paper uses a modification of the algorithm derived above; rather than take the final set of parameters, it chooses the set of parameters with the highest associated log-likelihood from each of the j iterations. The log-likelihood for each set of parameters is computed in the expectation step. An initial burn-in period is discarded to allow the algorithm to first reach stable values.

Therefore, this algorithm allows for the estimation of parameters which are consistent with the sum. However, it still must be evaluated on the basis of fit and runtime. Fit will be evaluated in the next section. Runtime is trivially $O(N * j)$, where j is the number of iterations of the algorithm. An important question for further study would be the degree to which j depends on N, if at all.

[7] In some sense the algorithm is more like Gibbs sampling than EM, as Gibbs sampling is a randomized algorithm whereas EM is deterministic. Nonetheless, EM is often used when there are unobservable latent variables, which is why this algorithm is dubbed a modified version of EM.

[8] This is because, in the i-th iteration, an imputed data value $\widehat{y_1}$ will be summed with another imputed value to compute $total$. While these two values are resampled, their sum must remain constant from step to step, to ensure the global sum is constant. This slows the rate of convergence.

Before implementing the algorithm, the distribution of two-fold convolutions under each model assumption must be computed. Regardless of model assumption, however, low-dimensional convolutions are very easy to compute or simulate. More precisely, the model needs to sample values Y_1 and Y_2 from an underlying distribution, subject to their sum remaining constant. This is equivalent to sampling a new value $\hat{y}_1$ for Y_1, and setting $Y_2 = total - \hat{y}_1$, where $total$ was the initial sum. Under the Gamma distribution, this evaluates to:

$$P_\theta(Y_1 = \hat{y}_1 | Y_1 + Y_2 = total) = (P_\theta(Y_1 = \hat{y}_1) * P_\theta(Y_2 = total - \hat{y}_1)) \Big/ P_\theta(Y_1 + Y_2 = total) =$$

$$\frac{\frac{\beta^\alpha}{\Gamma(\alpha)} \widehat{y_1}^{\alpha-1} e^{-\beta \widehat{y_1}} * \frac{\beta^\alpha}{\Gamma(\alpha)} (total - \widehat{y_1})^{\alpha-1} e^{-\beta(total - \widehat{y_1})}}{\frac{\beta^{2\alpha}}{\Gamma(2\alpha)} total^{2\alpha-1} e^{-\beta * total}} = \frac{\Gamma(2\alpha) \left(\frac{\hat{y}_1}{total}\right)^{\alpha-1} \left(1 - \frac{\hat{y}_1}{total}\right)^{\alpha-1}}{\Gamma(\alpha)^2 * total}$$

However, this is simply proportional to a Beta distribution, with both parameters equal to α. Thus, to sample new values Y_1 and Y_2 from a Gamma distribution with parameters α and β, subject to their sum being equal to $total$, a value can be sampled from a Beta distribution with both parameters α, and this value can be multiplied by $total$ to yield $\hat{y}_1$. Then, it is easy to compute $\hat{y}_2 = total - \hat{y}_1$, generating both sampled values[9].

For the Normal Mixture Model, a similar procedure can be followed, although there is no solution in terms of known density functions for the distribution. Similarly to the KL Divergence computations, these convolutions are approximated by approximating the mixture model with a discrete analog. As discussed, any error induced by this step is close to negligible.

6.2 Performance of the Algorithm

To evaluate the performance of this algorithm, $N = 1{,}000$ data points were simulated from a Gamma distribution with parameters $\alpha = .5, \beta = .1$. Only $k = 100$ of the data points were given to the algorithm as disaggregate data; the rest were presented in an aggregate fashion. The

[9] This procedure relied on the fact that the sum of two Gamma distributions with parameters α and β is a Gamma distribution with parameters 2α and β. Of course, given that this is known, it is natural to wonder why one would not directly sample from a $N - k$-fold convolution of Gammas. This paper does not use this fact as its goal is to evaluate the performance of the modified EM algorithm.

algorithm was run $j = 100$ iterations and a burn-in period of 20 observations were discarded. Figure 6 below depicts the produced chain of parameters with and without the burn-in.

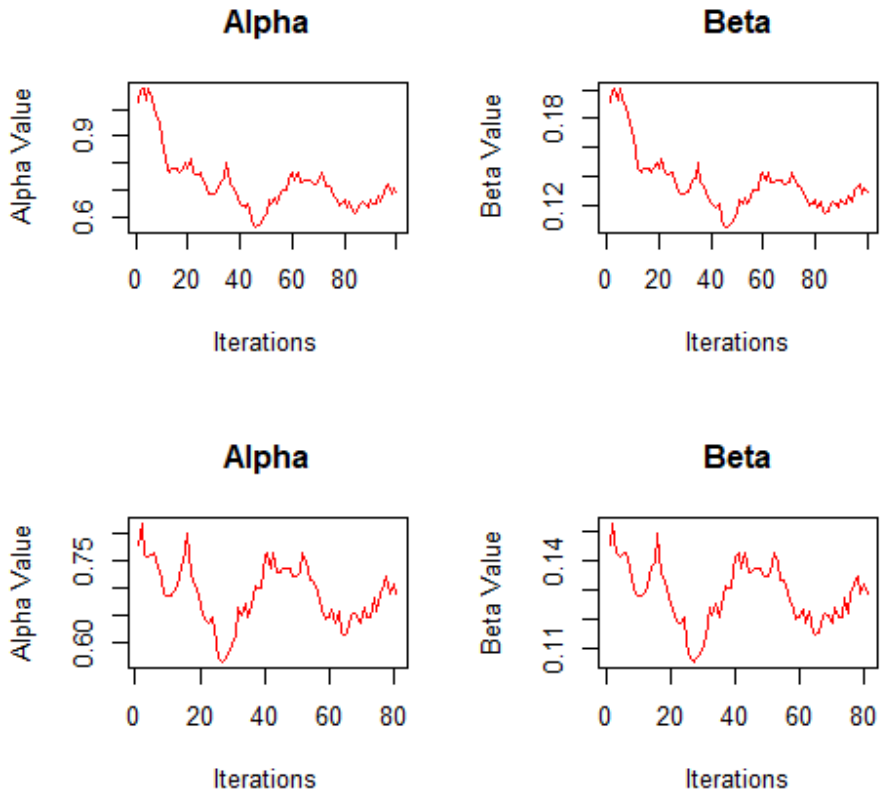

Figure 6: Chain of Parameters for Gamma Distribution

It is clear the algorithm performs well, quickly converging to the correct range of values for both parameters. However, it tends to overestimate both α and β slightly. Given the shape of the Gamma distribution implied by α, however, the algorithm performs quite well.

Figure 7 displays the likelihood and KL divergence at each iteration, first with then without the burn-in period.

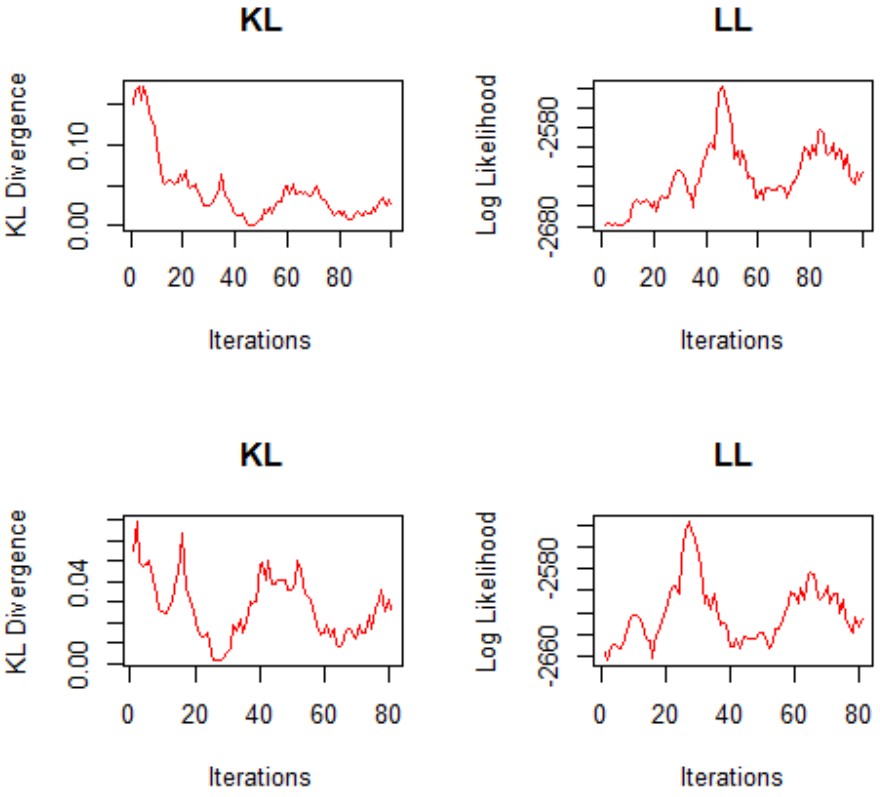

Figure 7: KL Divergence and Log-Likelihood for Gamma Distribution

As can be seen by the above plots, the minimum KL divergence of the algorithm parameters onto the true parameters is achieved close to the highest values of log-likelihood. Therefore, the modification of taking the set of parameters associated with the highest log-likelihood is likely beneficial.[10]

The procedure above was repeated under the Normal Mixture Model, with parameters $\mu_1 = 0, \sigma_1^2 = 1, \mu_2 = 4, \sigma_2^2 = .1, p = .9$. The algorithm was run with $j = 250$ iterations. The results of the algorithm, post a burn-in period of 50 iterations, are depicted below:

[10] In practice, one would never be able to identify the true model parameters, and thus the KL divergence of the estimates onto these parameters. Because of this, given the goal of estimating these parameters as closely as possible, or achieving a KL divergence of 0, it is reassuring to know that the approximation of log-likelihood by the modified EM algorithm closely tracks KL divergence.

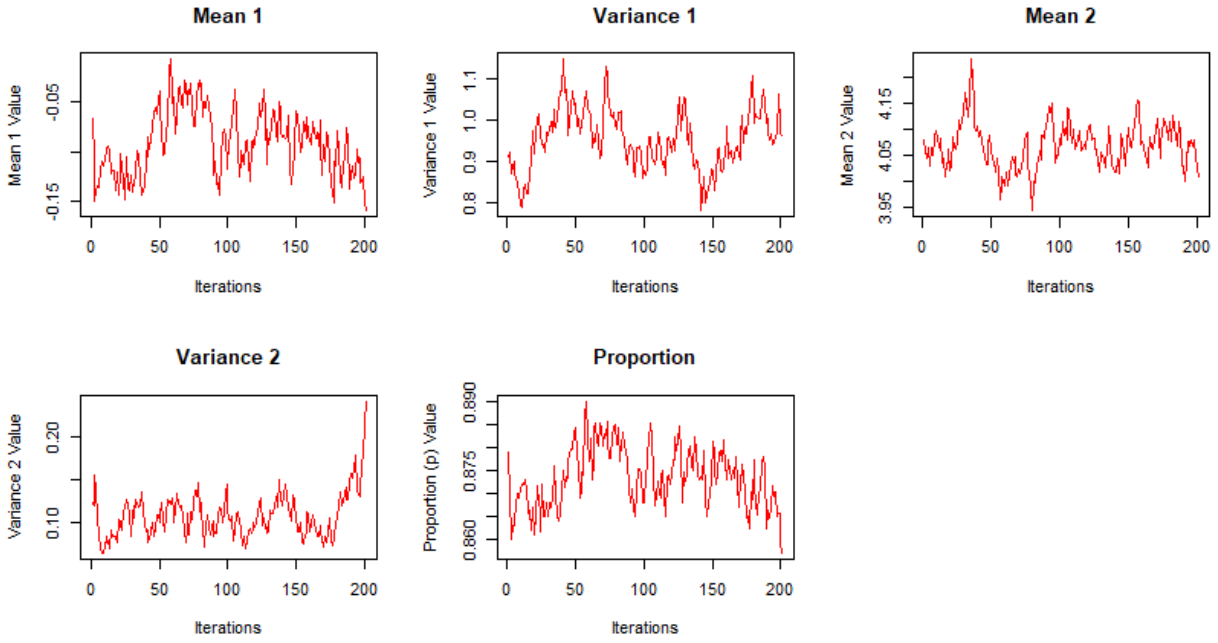

Figure 8: Chain of Parameters for Normal Mixture Model

The parameters are all relatively close to their true values once the algorithm converges. Figure 9 below depicts the KL Divergence and log-likelihoods:

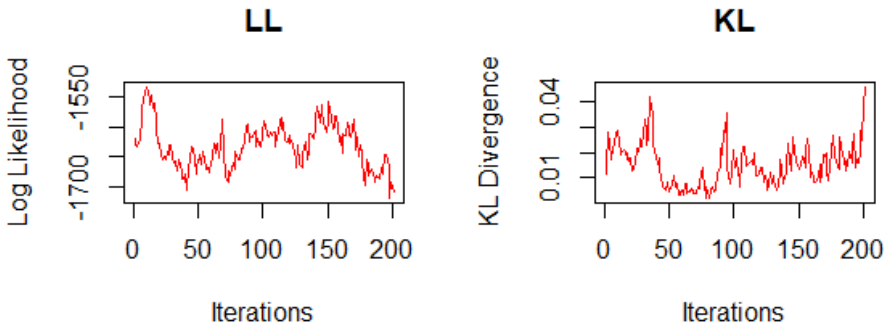

Figure 9: KL Divergence and Log-Likelihood for Normal Mixture Model

Under these plots, the relationship between log-likelihood and KL divergence is no longer so clear. Plotting the two against each other reveals no apparent trend, and the correlation between the two is modestly positive (.08). This suggests that perhaps choosing parameters with the minimum log-likelihood may not be the best criteria in more complicated model settings. The above simulations were repeated 100 times and very similar chains and results were obtained for

each. However, when comparing the results under the modified EM algorithm with the MVN approximation explored in Section 5, the MVN approximation outperformed in each trial. A paired T-test confirmed these results to be significant, with a p-value of 0.

Using these simulations, under the Normal Mixture Model, a modified posterior predictive check was done to compare the variance under the estimated parameters to the true model variance. The algorithm systematically overpredicts the true variance, as depicted in Figure 10.

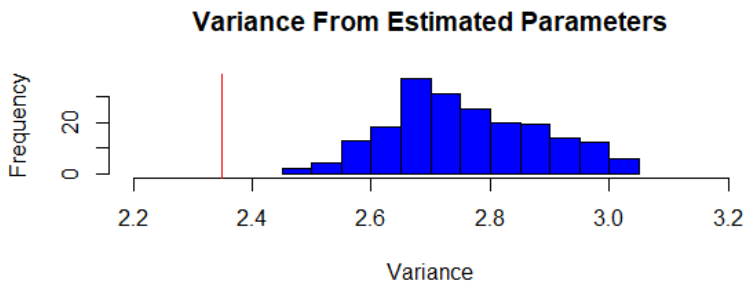

Figure 10: Variance From Estimated Parameters vs. True Variance

6.3 Discussion

Ultimately, while the proposed modified EM algorithm performed well in parameter estimation, it failed when compared to the MVN approximation approach to data fusion. In addition, it displayed systematic biases in its estimation of variance. The MPL approach to data fusion was found to be quite strong, relative to maximum likelihood estimation over the disaggregate data.

There are several important avenues for further study. With respect to the modified EM algorithm, sampling from a two-fold convolution likely induced significant error and slowed the rate of convergence drastically. While higher-dimensional convolutions may not be possible to evaluate, replacing this with a three or four-fold convolution may significantly decrease error. Furthermore, the algorithm could be extended to incorporate potential non-representativeness of the disaggregate sample.

7 References

Berry, Steven (1994), “Estimating Discrete-Choice Models of Product Differentiation,” RAND Journal of Economics, 25 (2), 242–62.

Bodapati, Anand V. , and Gupta, Sachin (2004), “The Recoverability of Segmentation Structure from Store-Level Aggregate Data,” Journal of Marketing Research, 41 (August), 351–64.

Boyd, J. Hayden and Robert E. Mellman (1980), “The Effect of Fuel Economy Standards on the U.S. Automotive Market: An Hedonic Demand Analysis,” Transportation Research, 14A, 367–78.

Cardell, N. Scott and Frederick C. Dunbar (1980), “Measuring the Societal Impacts of Automobile Downsizing,” Transportation Research, 14A, 423–34

Chen Y, Yang S (2007) Estimating disaggregate models using aggregate data through augmentation of individual choice. Journal of Marketing Research 44(4):613{621.

Dong, K. K., & Taylor, J. M. G. (1995). The restricted EM algorithm for maximum likelihood estimation under linear restrictions on the parameters. Journal of the American Statistical Association, 90, 707–716.

Durrieu, J.L.; Thiran, J.P.; Kelly, F. Lower and upper bounds for approximation of the Kullback-Leibler divergence between Gaussian Mixture Models. In Proceedings of the IEEE International Conference on

Feit EM, Wang P, Bradlow ET, Fader PS (2013) Fusing aggregate and disaggregate data with an application to multiplatform media consumption. Journal of Marketing Research 50(3):348{364.

Gupta, Sachin , Chintagunta, Pradeep , Kaul, Anil , and Wittink, Dick (1996), “Do Household Scanner Data Provide Representative Inferences from Brand Choices: A Comparison with Store Data,” Journal of Marketing Research, 33 (November), 383–98.

McCarthy, Daniel and Oblander, Elliot Shin, Scalable Data Fusion with Selection Correction: An Application to Customer Base Analysis (March 16, 2020). Columbia Business School Research Paper, Forthcoming.

Zenor, Michael J. , and Srivastava, Rajendra (1993), “Inferring Market Structure with Aggregate Data: A Latent Segment Logit Approach,” Journal of Marketing Research, 30 (August), 369–79.

8 Appendix

8.1 Code

```
rm(list = ls())
##### Gamma #####
### Generate Data ###
alpha_true = 1
beta_true = 1
n <- 500
data <- rgamma(n, alpha_true, beta_true)

### Compute LL ###
compute_ll_fused <- function(parameters, k) {
  alpha_hat <- parameters[1]
  beta_hat <- parameters[2]
  if (k > 0 && k < n) {
    ll_disaggregate <-
      sum(dgamma(data[1:k], alpha_hat, beta_hat, log = TRUE))
    average <- mean(data[(k + 1):n])
    mean_theoretical <- alpha_hat / beta_hat
    var_theoretical <-
      (alpha_hat / beta_hat ^ 2) / (n - (k + 1) + 1)
    ll_aggregate <-
      dnorm(average, mean_theoretical, sqrt(var_theoretical), log = TRUE)
    ll_total <- ll_disaggregate + ll_aggregate
    return(ll_total)
  }
  if (k == n) {
    return(sum(dgamma(data[1:k], alpha_hat, beta_hat, log = TRUE)))
  }
  if (k == 0) {
    average <- mean(data[(k + 1):n])
    mean_theoretical <- alpha_hat / beta_hat
    var_theoretical <-
      (alpha_hat / beta_hat ^ 2) / (n - (k + 1) + 1)
    return(dnorm(average, mean_theoretical, sqrt(var_theoretical), log = TRUE))
  }
}

compute_ll_disaggregate <- function(parameters, k) {
  alpha_hat <- parameters[1]
  beta_hat <- parameters[2]
  ll_total <- sum(dgamma(data[1:k], alpha_hat, beta_hat, log = TRUE))
  if(alpha_hat <=0 || beta_hat<=0) {
    return(-1000000000)
  }
  if(!is.finite(ll_total)) {
    View(data[1:k])
  }
  return(ll_total)
}

KL.gamma <- function(scale1_inv, shape1, scale2_inv, shape2) {
  scale1 <- 1 / scale1_inv
  scale2 <- 1 / scale2_inv
  (shape1 - 1) * digamma(shape1) - log(scale1) - shape1 - lgamma(shape1) +
    lgamma(shape2) + shape2 * log(scale2) - (shape2 - 1) * (digamma(shape1) +
                                                log(scale1)) + scale1 * shape1 / scale2
}

find_optimum <- function(use_fused_ll, k) {
  if (use_fused_ll) {
    fun <- compute_ll_fused
  } else {
    fun <- compute_ll_disaggregate
  }
```

```
  fit1 <- optim(c(1, 1), fun, k = k, control = list(fnscale = -1))
  fit2 <- optim(c(100, 100),
                fun,
                k = k,
                control = list(fnscale = -1))
  fit3 <- optim(c(.01, .01),
                fun,
                k = k,
                control = list(fnscale = -1))
  fit4 <- optim(fit1$par, fun, k = k, control = list(fnscale = -1))
  fit5 <- optim(fit2$par, fun, k = k, control = list(fnscale = -1))
  fit6 <- optim(fit3$par, fun, k = k, control = list(fnscale = -1))
  fit_ll <-
    c(fit1$value,
      fit2$value,
      fit3$value,
      fit4$value,
      fit5$value,
      fit6$value)
  ind_best <- which(fit_ll == max(fit_ll))
  fit_best <-
    if (ind_best == 1) {
      fit1
    } else if (ind_best == 2) {
      fit2
    } else if (ind_best == 3) {
      fit3
    } else if (ind_best == 4) {
      fit4
    } else if (ind_best == 5) {
      fit5
    } else if (ind_best == 6) {
      fit6
    }
  fit <- optim(fit_best$par,
               fun,
               k = k,
               control = list(fnscale = -1))
  return(fit)
}

### Test ###
KL_fused <- rep(0, n - 1)
KL_disaggregate <- rep(0, n - 1)
fit_opt <- find_optimum(TRUE, n)
alpha_fit_opt <- fit_opt$par[1]
beta_fit_opt <- fit_opt$par[2]
for (i in 2:n) {
  fit_fused <- find_optimum(TRUE, i)
  KL_fused[i - 1] <-
    KL.gamma(fit_fused$par[2], fit_fused$par[1], beta_fit_opt, alpha_fit_opt)
  fit_disaggregate <- find_optimum(FALSE, i)
  KL_disaggregate[i - 1] <-
    KL.gamma(fit_disaggregate$par[2],
             fit_disaggregate$par[1],
             beta_fit_opt,
             alpha_fit_opt)
  print(i)
}
par(mfrow = c(2, 2))
plot(KL_fused,
     type = "l",
     log = 'y',
     col = 'red', xlab='Sample Size (k)', ylab='KL Divergence')
lines(KL_disaggregate, col = 'blue')
legend(x=350, y=1, legend=c('MPL', 'MLE'), col=c('red', 'blue'), lty=1,cex=.6)
plot(
  KL_disaggregate - KL_fused,
  type = "l",
  log = 'y',
```

```
  col = 'red',
  xlab = 'Sample Size (k)', ylab = 'Difference in KL Divergence')

### Question 1 ###
alpha_true = 1
beta_true = 1
n <- 10000
reps <- 100
KL_fused <-
  matrix(rep(0, (floor(log(
    n, base = 10
  )) - 1) * reps), nrow = reps, ncol = floor(log(n, base = 10)) - 1)
KL_disaggregate <-
  matrix(rep(0, (floor(log(
    n, base = 10
  )) - 1) * reps), nrow = reps, ncol = floor(log(n, base = 10)) - 1)
for (i in 1:reps) {
  data <- rgamma(n, alpha_true, beta_true)
  fit_opt <- find_optimum(TRUE, n)
  alpha_fit_opt <- fit_opt$par[1]
  beta_fit_opt <- fit_opt$par[2]
  for (c in 10 ^ (2:log(n / 10, base = 10))) {
    fit_fused <- find_optimum(TRUE, c)
    KL_fused[i, log(c, base = 10)] <-
      KL.gamma(fit_fused$par[2],
               fit_fused$par[1],
               beta_fit_opt,
               alpha_fit_opt)
    fit_disaggregate <- find_optimum(FALSE, c)
    KL_disaggregate[i, log(c, base = 10)] <-
      KL.gamma(fit_disaggregate$par[2],
               fit_disaggregate$par[1],
               beta_fit_opt,
               alpha_fit_opt)
  }
  print(i)
}
c <- 3
par(mfrow = c(2, 2))
hist(KL_fused[, c], col = rgb(1, 0, 0, 0.5), breaks = 40, xlab = 'KL Divergence (k=1,000)', ylab = 'Frequency', main = 'MPL vs MLE')
hist(
  KL_disaggregate[, c],
  col = rgb(0, 0, 1, 0.5),
  breaks = 40,
  add = TRUE
)
legend("topright", c("MPL", "MLE"), col=c("red", "blue"), lty=1, cex=.6)
hist(KL_disaggregate[, c] - KL_fused[, c],
     breaks = 40,
     col = 'red', xlab = 'Difference in KL Divergence (k=1,000)', ylab='Frequency', main='KL_MLE - KL_MPL')
colMeans(KL_fused)
colMeans(KL_disaggregate)
sd(KL_disaggregate[,3]-KL_fused[,3])

### Question 2 ###
alpha_true = .1
beta_true = .1
n <- 10000
reps <- 100
KL_fused <- matrix(rep(0, reps*3), ncol=3, nrow=reps)
alpha_vec <- matrix(rep(0, reps*3), ncol=3, nrow=reps)
beta_vec <- matrix(rep(0, reps*3), ncol=3, nrow=reps)
for (c in 1:reps) {
  data <- rgamma(n, alpha_true, beta_true)
  for (i in c(100,1000,10000)) {
    fit_fused <- find_optimum(TRUE, i)
    KL_fused[c,log(i,base=10)-1] <- KL.gamma(fit_fused$par[2], fit_fused$par[1], beta_true, alpha_true)
    alpha_vec[c, log(i,base=10)-1] <- fit_fused$par[2]
    beta_vec[c, log(i,base=10)-1] <- fit_fused$par[1]
  }
```

```
  print(c)
}
par(mfrow = c(2, 3))
var_vec <- alpha_vec/beta_vec^2
hist(var_vec[,1], breaks=10, xlab = 'Variance', main = 'Variance Under MPL Estimates (k=100)', col='blue')
abline(v=10, col='red')
hist(var_vec[,2], breaks=10, xlab = 'Variance', main = 'Variance Under MPL Estimates (k=1,000)', col='blue')
abline(v=10, col='red')
hist(var_vec[,3], breaks=10, xlab = 'Variance', main = 'Variance Under MPL Estimates (k=10,000)', col='blue')
abline(v=10, col='red')
backup_alpha_vec <- alpha_vec
backup_beta_vec <- beta_vec
### Question 3 ###
alpha_true = .5
beta_true = .1
n <- 1000
data_true <- rgamma(n, alpha_true, beta_true)
k <- 100
data_disaggregate <- data_true[1:k]
mean_aggregate <- mean(data_true[(k+1):n])
data <- data_disaggregate
fit_init <- find_optimum(FALSE, k)
data <- c(data, rep(mean_aggregate,(n-(k+1)+1)))
expectation_step <- function(alpha_hat, beta_hat) {
  samp <- sample(seq(k+1,n,1),replace=FALSE)
  for(i in seq(1,n-k,2)) {
    x1 <- data[samp[i]]
    x2 <- data[samp[i+1]]
    samples <- convolution(alpha_hat, beta_hat, x1, x2)
    data[samp[i]] <- samples[1]
    data[samp[i+1]] <- samples[2]
  }
  return(data)
}
convolution <- function(alpha_hat, beta_hat, x1, x2) {
  total <- x1+x2
  x1 <- total*rbeta(1, alpha_hat, alpha_hat)
  x2 <- total - x1
  if(x1 == 0 || x2 == 0) {
    return(c(total/2, total/2))
  }
  return(c(x1, x2))
}
maximization_step <- function() {
  return(find_optimum(FALSE, n))
}
alpha_hat <- fit_init$par[1]
beta_hat <- fit_init$par[2]
reps <- 100
alpha_vec <- rep(0, reps)
beta_vec <- rep(0, reps)
ll_vec <- rep(0, reps)
KL_vec <- rep(0, reps)
for(i in 1:reps) {
  data <- expectation_step(alpha_hat, beta_hat)
  fit_new <- maximization_step()
  alpha_hat <- fit_new$par[1]
  beta_hat <- fit_new$par[2]
  alpha_vec[i] <- alpha_hat
  beta_vec[i] <- beta_hat
  ll_vec[i] <- fit_new$value
  KL_vec[i] <- KL.gamma(alpha_hat, beta_hat, alpha_true, beta_true)
  print(i)
}
burnin <- 20
alpha_vec <- alpha_vec[burnin:reps]
beta_vec <- beta_vec[burnin:reps]
ll_vec <- ll_vec[burnin:reps]
KL_vec <- KL_vec[burnin:reps]
ind <- which(ll_vec==max(ll_vec))
```

```
alpha_hat <- alpha_vec[ind]
beta_hat <- beta_vec[ind]
KL <- KL_vec[ind]
par(mfrow=c(2,2))
plot(KL_vec, type='l', ylab='KL Divergence',xlab='Iterations',main='KL', col='red')
plot(ll_vec, type='l', ylab='Log Likelihood',xlab='Iterations',main='LL', col='red')

##### Normal-Normal Mixture #####
rm(list = ls())
### Generate Data ###
mu1 = -2
sigsq1 = 1
mu2 = 4
sigsq2 = 2
p <- .8
n <- 500
groups <- rbinom(n, 1, p)
group1 <- rnorm(n, mu1, sqrt(sigsq1))
group2 <- rnorm(n, mu2, sqrt(sigsq2))
data <- groups * group1 + (1 - groups) * group2

### Compute LL ###
compute_ll_fused <- function(parameters, k) {
  mu1_hat <- parameters[1]
  sigsq1_hat <- parameters[2]
  mu2_hat <- parameters[3]
  sigsq2_hat <- parameters[4]
  p_hat <- parameters[5]
  if(p_hat > 1 || p_hat < 0 || sigsq1_hat <= 0 || sigsq2_hat <= 0) {
    return(-100000000000)
  }
  if (k > 0 && k < n) {
    ll_disaggregate <-
      sum(log(
        p_hat * dnorm(data[1:k], mu1_hat, sqrt(sigsq1_hat)) + (1 - p_hat) * dnorm(data[1:k], mu2_hat, sqrt(sigsq2_hat))
      ))
    average <- mean(data[(k + 1):n])
    mean_theoretical <- p_hat * mu1_hat + (1 - p_hat) * mu2_hat
    var_theoretical <-
      (p_hat * sigsq1_hat + (1 - p_hat) * sigsq2_hat + p_hat * (1 - p_hat) *
         (mu1_hat - mu2_hat) ^ 2) / (n - (k + 1) + 1)
    ll_aggregate <-
      dnorm(average, mean_theoretical, sqrt(var_theoretical), log = TRUE)
    ll_total <- ll_disaggregate + ll_aggregate
    if (!is.finite(ll_total)) {
      ll_total <- -100000000000
    }
    return(ll_total)
  }
  if (k == n) {
    ll_total <- sum(log(
      p_hat * dnorm(data, mu1_hat, sqrt(sigsq1_hat)) + (1 - p_hat) * dnorm(data, mu2_hat, sqrt(sigsq2_hat))))
    if (!is.finite(ll_total)) {
      ll_total <- -100000000000
    }
    return(ll_total)
  }
  if (k == 0) {
    average <- mean(data)
    mean_theoretical <- p_hat * mu1_hat + (1 - p_hat) * mu2_hat
    var_theoretical <-
      (p_hat * sigsq1_hat + (1 - p_hat) * sigsq2_hat + p_hat * (1 - p_hat) *
         (mu1_hat - mu2_hat) ^ 2) / (n - (k + 1) + 1)
    ll_total <- dnorm(average, mean_theoretical, sqrt(var_theoretical), log = TRUE)
    if (!is.finite(ll_total)) {
      ll_total <- -100000000000
    }
    return(ll_total)
  }
}
```

```
compute_ll_disaggregate <- function(parameters, k) {
  mu1_hat <- parameters[1]
  sigsq1_hat <- parameters[2]
  mu2_hat <- parameters[3]
  sigsq2_hat <- parameters[4]
  p_hat <- parameters[5]
  ll_total <- sum(log(p_hat * dnorm(data[1:k], mu1_hat, sqrt(sigsq1_hat)) + (1 - p_hat) * dnorm(data, mu2_hat, sqrt(sigsq2_hat))))
  if (!is.finite(ll_total)) {
    ll_total <- -100000000000
  }
  return(ll_total)
}

find_optimum <- function(use_fused_ll, k) {
  if (use_fused_ll) {
    fun <- compute_ll_fused
  } else {
    fun <- compute_ll_disaggregate
  }
  fit1 <-
    optim(
      c(-1, 1, 1, 1, .5),
      fun,
      k = k,
      control = list(fnscale = -1),
      lower = c(-Inf, 0,-Inf, 0, 0),
      upper = c(Inf, Inf, Inf, Inf, 1),
      method = "L-BFGS-B"
    )
  fit2 <-
    optim(
      c(0, 5, 0, 1, .5),
      fun,
      k = k,
      control = list(fnscale = -1),
      lower = c(-Inf, 0,-Inf, 0, 0),
      upper = c(Inf, Inf, Inf, Inf, 1),
      method = "L-BFGS-B"
    )
  fit3 <-
    optim(
      c(-1, 2, 1, 1, .1),
      fun,
      k = k,
      control = list(fnscale = -1),
      lower = c(-Inf, 0,-Inf, 0, 0),
      upper = c(Inf, Inf, Inf, Inf, 1),
      method = "L-BFGS-B"
    )
  fit4 <-
    optim(
      c(-1, 1, 1, 2, .9),
      fun,
      k = k,
      control = list(fnscale = -1),
      lower = c(-Inf, 0,-Inf, 0, 0),
      upper = c(Inf, Inf, Inf, Inf, 1),
      method = "L-BFGS-B"
    )
  fit5 <-
    optim(
      c(10, 5,-5, 10, .3),
      fun,
      k = k,
      control = list(fnscale = -1),
      lower = c(-Inf, 0,-Inf, 0, 0),
      upper = c(Inf, Inf, Inf, Inf, 1),
      method = "L-BFGS-B"
    )
```

```
 fit6 <-
  optim(
    c(5, 10,-10, 5, .7),
    fun,
    k = k,
    control = list(fnscale = -1),
    lower = c(-Inf, 0,-Inf, 0, 0),
    upper = c(Inf, Inf, Inf, Inf, 1),
    method = "L-BFGS-B"
  )

 fit_ll <-
  c(fit1$value,
    fit2$value,
    fit3$value,
    fit4$value,
    fit5$value,
    fit6$value)
 ind_best <- which(fit_ll == max(fit_ll))
 fit_best <-
  if (ind_best == 1) {
    fit1
  } else if (ind_best == 2) {
    fit2
  } else if (ind_best == 3) {
    fit3
  } else if (ind_best == 4) {
    fit4
  } else if (ind_best == 5) {
    fit5
  } else if (ind_best == 6) {
    fit6
  }
 fit <- optim(fit_best$par,
              fun,
              k = k,
              control = list(fnscale = -1))
 return(fit)
}

dx <- .01
KL.norm <- function(params1, params2) {
 mu1_1 <- params1[1]
 sigsq1_1 <- params1[2]
 mu2_1 <- params1[3]
 sigsq2_1 <- params1[4]
 p_1 <- params1[5]
 mu1_2 <- params2[1]
 sigsq1_2 <- params2[2]
 mu2_2 <- params2[3]
 sigsq2_2 <- params2[4]
 p_2 <- params2[5]
 lb <-
  min(mu1_1 - 6 * sigsq1_1,
      mu1_2 - 6 * sigsq1_2,
      mu2_1 - 6 * sigsq2_1,
      mu2_2 - 6 * sigsq2_2)
 ub <-
  max(mu1_1 + 6 * sigsq1_1,
      mu1_2 + 6 * sigsq1_2,
      mu2_1 + 6 * sigsq2_1,
      mu2_2 + 6 * sigsq2_2)
 print(c(lb, ub))
 support <- seq(lb, ub, by = dx)
 dens1 <-
  p_1 * dnorm(support, mu1_1, sqrt(sigsq1_1)) + (1 - p_1) * dnorm(support, mu2_1, sqrt(sigsq2_1))
 dens2 <-
  p_2 * dnorm(support, mu1_2, sqrt(sigsq1_2)) + (1 - p_2) * dnorm(support, mu2_2, sqrt(sigsq2_2))
 dens1 <- dens1 * dx
 dens2 <- dens2 * dx
```

```
  ratio <- dens1/dens2
  ratio[!is.finite(ratio)] <- -1
  largest <- max(ratio)
  ratio[ratio == -1] <- largest
  return(sum(dens1 * log(ratio)))
}

### Test ###
KL_fused <- rep(0, n - 10)
KL_disaggregate <- rep(0, n - 10)
fit_opt <- find_optimum(TRUE, n)
for (i in 11:n) {
  fit_fused <- find_optimum(TRUE, i)
  KL_fused[i - 1] <-
    KL.norm(fit_fused$par, fit_opt$par)
  fit_disaggregate <- find_optimum(FALSE, i)
  KL_disaggregate[i - 1] <-
    KL.norm(fit_disaggregate$par, fit_opt$par)
  print(i)
}
par(mfrow = c(2, 1))

plot(KL_fused,
     type = "l",
     log = 'y',
     col = 'red', xlab='Sample Size (k)', ylab='KL Divergence')
lines(KL_disaggregate, col = 'blue')
legend(x=350, y=1, legend=c('MPL', 'MLE'), col=c('red', 'blue'), lty=1,cex=.6)
plot(
  KL_disaggregate - KL_fused,
  type = "l",
  log = 'y',
  col = 'red',
  xlab = 'Sample Size (k)', ylab = 'Difference in KL Divergence')

### Question 1 ###
mu1 = -2
sigsq1 = 1
mu2 = 4
sigsq2 = 2
p <- .8
n <- 10000
reps <- 100
KL_fused <-
  matrix(rep(0, (floor(log(
    n, base = 10
  )) - 1) * reps), nrow = reps, ncol = floor(log(n, base = 10)) - 2)
KL_disaggregate <-
  matrix(rep(0, (floor(log(
    n, base = 10
  )) - 1) * reps), nrow = reps, ncol = floor(log(n, base = 10)) - 2)
for (i in 1:reps) {
  groups <- rbinom(n, 1, p)
  group1 <- rnorm(n, mu1, sqrt(sigsq1))
  group2 <- rnorm(n, mu2, sqrt(sigsq2))
  data <- groups * group1 + (1 - groups) * group2
  fit_opt <- find_optimum(TRUE, n)
  for (c in 10 ^ (2:log(n / 10, base = 10))) {
    fit_fused <- find_optimum(TRUE, c)
    KL_fused[i, log(c, base = 10)-1] <-
      KL.norm(fit_fused$par, fit_opt$par)
    fit_disaggregate <- find_optimum(FALSE, c)
    KL_disaggregate[i, log(c, base = 10)-1] <-
      KL.norm(fit_disaggregate$par, fit_opt$par)
  }
  print(i)
}
c <- 2
par(mfrow = c(2, 2))
```

```
hist(KL_fused[, c], xlim=c(0,.25), col = rgb(1, 0, 0, 0.5), breaks = 4, xlab = 'KL Divergence (k=1,000)', ylab = 'Frequency', main = 'MPL vs
MLE')
hist(
  KL_disaggregate[, c],
  col = rgb(0, 0, 1, 0.5),
  breaks = 10,
  add = TRUE
)
legend("topright", c("MPL", "MLE"), col=c("red", "blue"), lty=1, cex=.6)
hist(KL_disaggregate[, c] - KL_fused[, c],
     breaks = 40,
     col = 'red', xlab = 'Difference in KL Divergence (k=1,000)', ylab='Frequency', main='KL_MLE - KL_MPL')
colMeans(KL_fused)
colMeans(KL_disaggregate)

### Question 2 ###
mu1 = 0
sigsq1 = 1
mu2 = 10
sigsq2 = 5
p <- .4
n <- 10000
reps <- 100
KL_fused <- matrix(rep(0, reps*3), ncol=3, nrow=reps)
mu1_vec <- matrix(rep(0, reps*3), ncol=3, nrow=reps)
sigsq1_vec <- matrix(rep(0, reps*3), ncol=3, nrow=reps)
mu2_vec <- matrix(rep(0, reps*3), ncol=3, nrow=reps)
sigsq2_vec <- matrix(rep(0, reps*3), ncol=3, nrow=reps)
p_vec <- matrix(rep(0, reps*3), ncol=3, nrow=reps)
for (c in 1:reps) {
  groups <- rbinom(n, 1, p)
  group1 <- rnorm(n, mu1, sqrt(sigsq1))
  group2 <- rnorm(n, mu2, sqrt(sigsq2))
  data <- groups * group1 + (1 - groups) * group2
  for (i in c(100,1000,10000)) {
    fit_fused <- find_optimum(TRUE, i)
    KL_fused[c, log(i, base=10)-1] <- KL.norm(fit_fused$par, c(mu1, sigsq1, mu2, sigsq2, p))
    mu1_vec[c, log(i, base=10)-1] <- fit_fused$par[1]
    sigsq1_vec[c, log(i, base=10)-1] <- fit_fused$par[2]
    mu2_vec[c, log(i, base=10)-1] <- fit_fused$par[3]
    sigsq2_vec[c, log(i, base=10)-1] <- fit_fused$par[4]
    p_vec[c, log(i, base=10)-1] <- fit_fused$par[5]
  }
  print(c)
}
var_vec <- p_vec*sigsq1_vec+(1-p_vec)*sigsq2_vec+(p_vec)*(1-p_vec)*(mu1_vec-mu2_vec)^2
hist(var_vec[,1], breaks=10, xlab = 'Variance', main = 'Variance Under MPL Estimates (k=100)', col='blue')
abline(v=27.4, col='red')
hist(var_vec[,2], breaks=10, xlab = 'Variance', main = 'Variance Under MPL Estimates (k=1,000)', col='blue')
abline(v=27.4, col='red')
hist(var_vec[,3], breaks=10, xlab = 'Variance', main = 'Variance Under MPL Estimates (k=10,000)', col='blue')
abline(v=27.4, col='red')

### Question 3 ###
mu1_true = 0
sigsq1_true = 1
mu2_true = 4
sigsq2_true = .1
p_true <- .9
n <- 1000
groups <- rbinom(n, 1, p_true)
group1 <- rnorm(n, mu1_true, sqrt(sigsq1_true))
group2 <- rnorm(n, mu2_true, sqrt(sigsq2_true))
data_true <- groups * group1 + (1 - groups) * group2
k <- 200
data_disaggregate <- data_true[1:k]
mean_aggregate <- mean(data_true[(k+1):n])
data <- data_disaggregate
fit_init <- find_optimum(FALSE, k)
data <- c(data, rep(mean_aggregate,(n-(k+1)+1)))
```

```
expectation_step <- function(parameters) {
  samp <- sample(seq(k+1,n,1),replace=FALSE)
  for(i in seq(1,n-k,2)) {
    x1 <- data[samp[i]]
    x2 <- data[samp[i+1]]
    samples <- convolution(parameters, x1, x2)
    data[samp[i]] <- samples[1]
    data[samp[i+1]] <- samples[2]
  }
  return(data)
}
dx <- .01
convolution <- function(parameters, x1, x2) {
  total <- x1+x2
  mu1_hat <- parameters[1]
  sigsq1_hat <- parameters[2]
  mu2_hat <- parameters[3]
  sigsq2_hat <- parameters[4]
  p_hat <- parameters[5]
  lb1 <- min(mu1_hat - 6*sigsq1_hat, mu2_hat - 6*sigsq2_hat)
  ub1 <- max(mu1_hat + 6*sigsq1_hat, mu2_hat + 6*sigsq2_hat)
  lb <- min(lb1, total - ub1)
  ub <- max(ub1, total-lb1)
  support <- seq(lb, ub, dx)
  p_x1_equals_x1 <- p_hat*dnorm(support, mu1_hat, sqrt(sigsq1_hat)) + (1-p_hat)*dnorm(support, mu2_hat, sqrt(sigsq2_hat))
  p_x2_equals_kminusx1 <- p_hat*dnorm(rev(support), mu1_hat, sqrt(sigsq1_hat)) + (1-p_hat)*dnorm(rev(support), mu2_hat, sqrt(sigsq2_hat))
  numerator <- p_x1_equals_x1*p_x2_equals_kminusx1
  denominator <- p_hat^2*dnorm(total,2*mu1_hat,sqrt(2*sigsq1_hat))+(1-p_hat)^2*dnorm(total,2*mu2_hat,sqrt(2*sigsq2_hat))+2*p_hat*(1-
p_hat)*dnorm(total,mu1_hat+mu2_hat, sqrt(sigsq1_hat+sigsq2_hat))
  dens <- numerator/denominator
  dens <- dens/sum(dens)
  x1 <- sample(support, size=1, prob=dens)
  x2 <- total - x1
  return(c(x1, x2))
}
maximization_step <- function() {
  return(find_optimum(FALSE, n))
}
mu1_hat <- fit_init$par[1]
sigsq1_hat <- fit_init$par[2]
mu2_hat <- fit_init$par[3]
sigsq2_hat <- fit_init$par[4]
p_hat <- fit_init$par[5]
reps <- 250
mu1_vec <- rep(0, reps)
sigsq1_vec <- rep(0, reps)
mu2_vec <- rep(0, reps)
sigsq2_vec <- rep(0, reps)
p_vec <- rep(0, reps)
ll_vec <- rep(0, reps)
KL_vec <- rep(0, reps)
for(i in 1:reps) {
  data <- expectation_step(c(mu1_hat, sigsq1_hat, mu2_hat, sigsq2_hat, p_hat))
  fit_new <- maximization_step()
  mu1_hat <- min(fit_new$par[1], fit_new$par[3])
  sigsq1_hat <- max(fit_new$par[2], fit_new$par[4])
  mu2_hat <- max(fit_new$par[1], fit_new$par[3])
  sigsq2_hat <- min(fit_new$par[2], fit_new$par[4])
  p_hat <- max(fit_new$par[5], 1-fit_new$par[5])
  mu1_vec[i] <- mu1_hat
  sigsq1_vec[i] <- sigsq1_hat
  mu2_vec[i] <- mu2_hat
  sigsq2_vec[i] <- sigsq2_hat
  p_vec[i] <- p_hat
  ll_vec[i] <- fit_new$value
  KL_vec[i] <- KL.norm(c(mu1_hat, sigsq1_hat, mu2_hat, sigsq2_hat, p_hat), c(mu1_true, sigsq1_true, mu2_true, sigsq2_true, p_true))
  print(c(mu1_hat, sigsq1_hat, mu2_hat, sigsq2_hat, p_hat))
  print(i)
}
burnin <- 50
```

```
mu1_vec <- mu1_vec[burnin:reps]
sigsq1_vec <- sigsq1_vec[burnin:reps]
mu2_vec <- mu2_vec[burnin:reps]
sigsq2_vec <- sigsq2_vec[burnin:reps]
p_vec <- p_vec[burnin:reps]
ll_vec <- ll_vec[burnin:reps]
KL_vec <- KL_vec[burnin:reps]
ind <- which(ll_vec==max(ll_vec))
mu1_hat <- mu1_vec[ind]
sigsq1_hat <- sigsq1_vec[ind]
mu2_hat <- mu2_vec[ind]
sigsq2_hat <- sigsq2[ind]
p_hat <- p_vec[ind]
ll_hat <- ll_vec[ind]
KL <- KL_vec[ind]
par(mfrow=c(1,1))
plot(ll_vec, type='l', ylab='Log Likelihood',xlab='Iterations',main='LL', col='red')
plot(KL_vec, type='l', ylab='KL Divergence',xlab='Iterations',main='KL', col='red')
plot(mu2_vec, type='l', ylab='Mean 2 Value',xlab='Iterations',main='Mean 2', col='red')
plot(sigsq2_vec, type='l', ylab='Variance 2 Value',xlab='Iterations',main='Variance 2', col='red')
plot(p_vec, type='l', ylab='Proportion (p) Value',xlab='Iterations',main='Proportion', col='red')
hist(var_vec, breaks=10, main='Variance From Estimated Parameters', xlab='Variance', xlim=c(2.2,3.2), col='blue')
abline(v=var_true, col='red')
```