BAYESIAN ANALYSIS OF BIG DATA IN INSURANCE PREDICTIVE MODELING USING DISTRIBUTED COMPUTING

BY

YANWEI ZHANG

ABSTRACT

While Bayesian methods have attracted considerable interest in actuarial science, they are yet to be embraced in large-scaled insurance predictive modeling applications, due to inefficiencies of Bayesian estimation procedures. The paper presents an efficient method that parallelizes Bayesian computation using distributed computing on Apache Spark across a cluster of computers. The distributed algorithm dramatically boosts the speed of Bayesian computation and expands the scope of applicability of Bayesian methods in insurance modeling. The empirical analysis applies a Bayesian hierarchical Tweedie model to a big data of 13 million insurance claim records. The distributed algorithm achieves as much as 65 times performance gain over the non-parallel method in this application. The analysis demonstrates that Bayesian methods can be of great value to large-scaled insurance predictive modeling.

KEYWORDS

Bayesian, big data, distributed computing, predictive modeling, ratemaking, Spark.

1. INTRODUCTION

Bayesian methods play a critical role in many areas of casualty actuarial science, such as credibility theories (Bühlmann, 1967) and stochastic loss reserving (Wüthrich and Merz, 2008). The predictive distribution from a Bayesian analysis is particularly useful for insurance forecasting and decision making under uncertainties. However, application of Bayesian methods is still rare in practical insurance predictive modeling (Frees *et al.*, 2014), where insurers are increasingly advancing the use of big data technology to utilize individual or transaction level data. The major challenge in such large-scaled Bayesian analysis is the vast computational time demanded by simulation-based algorithms, such

Astin Bulletin 47(3), 943-961. doi: 10.1017/asb.2017.15 © 2017 by Astin Bulletin. All rights reserved.

as Markov chain Monte Carlo (MCMC) used by Bayesian inference (Gelman *et al.*, 2003).

Various methods for accelerating Bayesian computation have been developed (Zhu *et al.*, 2014; Green *et al.*, 2015). One research stream focuses on adjusting the simulation algorithm to improve sampling efficiency and convergence speed. Examples of such research include Langevin drift (Roberts and Tweedie, 1996), Hamiltonian Monte Carlo (Neal, 2013), adaptive MCMC (Haario *et al.*, 2001) and so on. In addition to fast convergence, Bayesian computation methods need to have high scalability to cope with the computational expenses introduced by the size of big data. Another stream of research focuses on algorithm scalability and employs parallel processing to accelerate Bayesian computation. The main parallelization strategies include parallel computation of data likelihood (Agarwal and Duchi, 2012), stochastic approximation of gradient (Welling and Teh, 2011) and parallel independent MCMC on data partitions (Neiswanger *et al.*, 2014; Wang and Dunson, 2016).

This paper extends the research on parallel MCMC to large-scaled Bayesian analysis in insurance predictive modeling. We implement a distributed MCMC algorithm across a cluster of computer nodes using Apache Spark (Zaharia *et al.*, 2010) and demonstrate the efficiency gain over non-parallel methods. We show that distributed computing makes Bayesian analysis feasible in practical predictive modeling. We then apply a Bayesian hierarchical Tweedie model (Zhang, 2013) to a large data set of personal auto bodily injury claims and demonstrate the many advantages of Bayesian forecasting in such practical problems as insurance ratemaking.

The main contributions of the paper are the following. First, we provide a viable solution for large-scaled Bayesian analysis using distributed computing on Apache Spark. The main challenge in Bayesian posterior computation arises from the evaluation of the data likelihood. A logical method to accelerate such evaluation is through parallel processing using multiple computer cores (Agarwal and Duchi, 2012). That is, we compute the data likelihood efficiently by dividing the large task into smaller pieces that are processed in parallel by a network of computers. We choose the parallelization on data likelihood over other parallelization strategies because of its general applicability — it does not rely on approximation or make additional assumptions of the posterior.[1] We implement the parallelization in a distributed fashion across a cluster of computer nodes on top of the Spark computing engine. Spark is a highly scalable, fast and easy-to-use engine for distributed programming (Zaharia *et al.*, 2010), and thus is well suited for practical large-scaled insurance problems. Our experiment shows that efficient parallelization achieves 65 times performance gain over existing single-thread implementation and even 8 times gain over state-of-the-art likelihood-based methods in one application.

Second, this is the first research to study individual insurance claims using the Bayesian hierarchical Tweedie model. The Tweedie distribution (Tweedie, 1984) is widely adopted to model insurance losses (e.g., Jørgensen and de Souza, 1994; Smyth and Jørgensen, 2002; Wüthrich, 2003; Shi, 2016). Recent research has estimated Tweedie models in the Bayesian setting. Notably, Peters *et al.* (2009) use the Bayesian Tweedie generalized linear model to study loss triangles and Zhang (2013) proposes Bayesian methods for hierarchical Tweedie models. These studies estimate their models on small samples of data. In contrast, efficient distributed computing enables us to estimate the Bayesian hierarchical Tweedie model on large-scaled individual-level claim data with more than 13 million records.

Last, we demonstrate the values of the Bayesian methodology to insurance ratemaking. Whereas there is a growing consensus that the Bayesian methodology is ideal for aggregate loss reserving analysis (see, e.g., de Alba, 2002; Ntzoufras and Dellaportas, 2002; Zhang *et al.*, 2012; Shi *et al.*, 2012; Zhang *et al.*, 2012), Bayesian analysis has not attracted much attention in insurance ratemaking (Bermudez and Karlis, 2011). Our study shows that many of the advantages of Bayesian analysis are indeed of significant value to insurance ratemaking. Among others, we highlight three such advantages. First, Bayesian predictive distributions facilitate loss cost forecast under policy coverage modifiers, such as deductibles and/or limits, avoiding complex numerical integration that would otherwise be needed. Second, Bayesian predictive distributions produce uncertainty measures that appropriately account for group-level variations and uncertainties in model estimation. Third, the Bayesian hierarchical Tweedie model provides a formal mechanism to incorporate the core insight of actuarial credibility in insurance ratemaking.

In the next section, we lay out the details of the Bayesian hierarchical model and develop the distributed MCMC algorithm. We then estimate the model on a large data set of insurance claims, compare the performance of the distributed algorithm to existing non-parallel algorithms, and discuss the benefits of Bayesian ratemaking. Last, we provide concluding remarks.

2. Theoretical development

This section formulates the Bayesian hierarchical model and develops the methodology for distributed Bayesian computing. The focus is on the hierarchical Tweedie model, but other model types can be accommodated similarly.

2.1. The hierarchical Tweedie model

Suppose our data can be abstracted as the tuple $(g_i, y_i, \boldsymbol{x}_i)$ for the ith record. y_i is the response variable (insurance loss), $\boldsymbol{x}_i$ is the vector of predictors (including the intercept) and g_i represents a categorical variable, such as the insured identifier, vehicle type, territory identifier and so on, which partitions the data into groups. Boldface is used to indicate vectors or matrices. The standard hierarchical Tweedie model makes the following assumptions:

$$y_i \sim \mathrm{Tw}(\mu_i, \phi, p), \tag{1}$$

$$\eta(\mu_i) = \boldsymbol{x}_i^T \boldsymbol{\beta} + u_{g_i}, \tag{2}$$

$$u_j \sim N(0, \sigma^2). \tag{3}$$

We use the notation $\mathrm{Tw}(\mu, \phi, p)$ to represent the Tweedie distribution. If $y \sim \mathrm{Tw}(\mu, \phi, p)$, then it follows a Tweedie distribution with the following density:

$$f_{Tw}(y|\mu, \phi, p) = a(y, \phi) \exp\left(\frac{1}{\phi}\left(y\frac{\mu^{1-p}}{1-p} - \frac{\mu^{2-p}}{2-p}\right)\right). \tag{4}$$

In the above, μ is the mean, $\phi > 0$ the scale parameter and $p \in (1, 2)$ the index parameter such that $\mathrm{Var}(y) = \phi\mu^p$. The quantity $a(y, \phi)$ is the normalizing constant and is not analytically tractable. It can be approximated reasonably well using numerical methods (Dunn and Smyth, 2005, 2008).

The mean is linked to the predictors through the function η, which is taken as the logarithmic function in this paper. $\boldsymbol{\beta}$ is the vector of parameters associated with the predictors, and u_{g_i} is the group-level effect. If the ith record belongs to group j, i.e., $g_i = j$, then u_j represents the deviation of group j from the population-level intercept. These group-level effects are assumed to follow a normal distribution with mean zero and variance σ^2. The Bayesian model specification is completed by specifying the priors for all the other parameters, e.g., $\pi(\boldsymbol{\beta}, \phi, p, \sigma^2)$. More details of the Bayesian hierarchical model can be found in Gelman *et al.* (2003).

2.2. Posterior distribution

Inference of the parameters is typically carried out through the MCMC algorithm. In particular, the Metropolis-within-Gibbs algorithm scheme sequentially samples parameters from their lower dimensional full conditional distributions over many iterations. The full conditionals are derived from the joint posterior density. The logarithmic joint posterior density can be expressed as

$$\begin{aligned} \ell &= \log f(\boldsymbol{y}, \boldsymbol{\beta}, \boldsymbol{u}, \sigma^2, \phi, p) \\ &= \underbrace{\sum_i \log f_{Tw}(y_i|\boldsymbol{\beta}, u_{g_i}, p, \phi)}_{\text{heavy computation}} + \underbrace{\sum_j \log f_N(u_j|\sigma^2) + \log \pi(\boldsymbol{\beta}, \phi, p, \sigma^2)}_{\text{light computation}}. \end{aligned} \tag{5}$$

The first part of the logarithmic joint density is the familiar loglikelihood of the data, which sums the log Tweedie density of each observation. The second part is the contribution from the group-level effects and parameters. This is generally low dimensional and requires little computational effort. Most computations

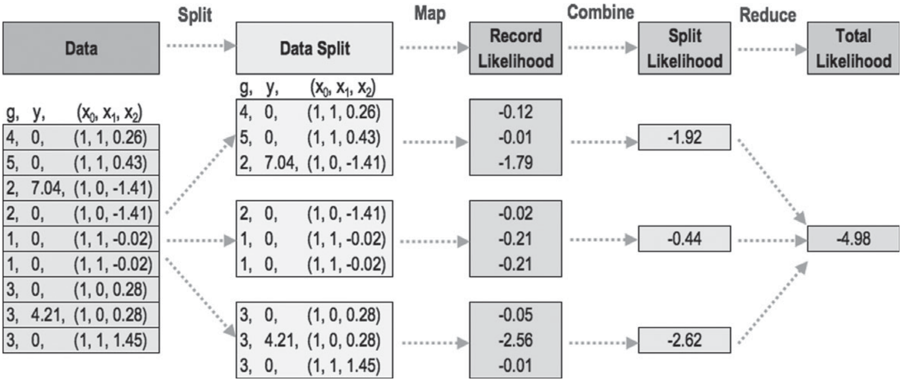

FIGURE 1: Illustration of the parallel computing of the data loglikelihood. All likelihoods are on the logarithmic scale.

occur in the loglikelihood part, and the computational complexity grows linearly with the size of the data. For vast amount of data, this evaluation becomes a significant computational burden. The MCMC algorithm generally runs for a large number of iterations, each of which requires many evaluations of the loglikelihood. Such an excessive number of loglikelihood evaluations essentially makes the MCMC algorithm infeasible for big data analysis. We address this issue using a distributed MCMC algorithm that computes the loglikelihood efficiently by dividing the large task into smaller pieces that are processed in parallel by a network of computers.

2.3. Parallel computation of posterior distributions

The standard approach to computing the data loglikelihood loops over each record of the data *sequentially*. Parallel processing, on the other hand, divides a large task into smaller splits that can be processed *simultaneously* by separate computer processors. The use of a large number of processors allows the job to be completed in much less time than the sequential approach requires (Grama *et al.*, 2003).

Figure 1 illustrates the computation of the loglikelihood using parallel processing. First, the data is partitioned according to the number of splits a user specifies (Data Split), and each data split is processed by a thread separately from the others. The number of partitions determines the level of parallelization. In general, more splits lead to higher level of parallelization and faster processing time. But the performance gain will be overshadowed by the communication overhead when too many splits result in few data for each split to process. In practice, each split is best to hold at least some Megabytes of data.

Second, the loglikelihood of each data record is evaluated using the current estimates of the parameters (Record Loglikelihood). The Tweedie density is evaluated using numerical methods (Dunn and Smyth, 2005) when necessary.

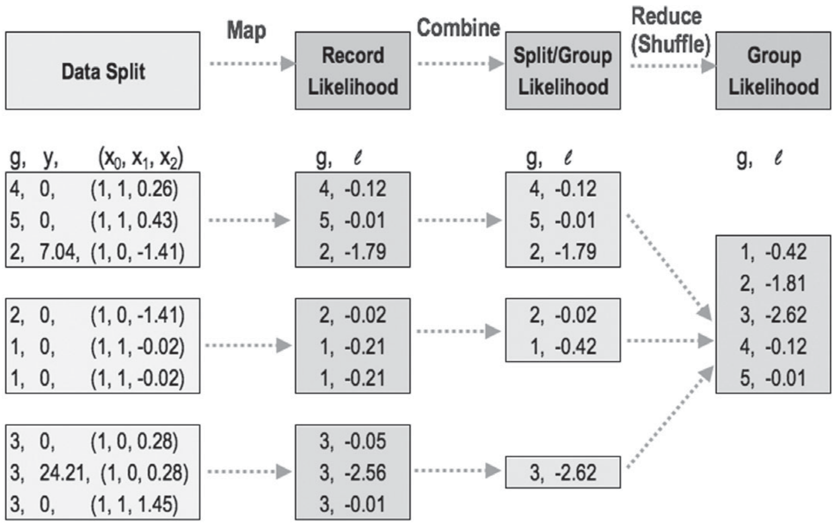

FIGURE 2: Illustration of the parallel computing of the data loglikelihood by group. All likelihoods are on the logarithmic scale.

Since each data split is processed separately from the others, the computation time of this step is the maximum of the processing time of each split, as opposed to the summation of the processing time of each record in sequential processing.

The last stage is the reduction of the individual loglikelihoods to the total loglikelihood. This is also operationalized in a parallel fashion. Each split of the individual loglikelihoods is first combined and aggregated into the “Split Loglikelihood”, independently of the other splits, and then the combined loglikelihoods are reduced and aggregated into the “Total Loglikelihood”.

This is the general scheme for computing the data loglikelihood in parallel. A slight deviation from this approach can further speed up the computation for the group-level effects $\boldsymbol{u}$. In the appendix, we show that the posterior distribution of the group-level effects can be partitioned, that is, the posterior distribution of one group is entirely independent of that of another. This factorization property implies that one sweep through the data enables us to derive the posterior density for *all* group-level parameters. In contrast, we must perform a complete sweep through the data for *each* component of $\boldsymbol{\beta}$ because each posterior depends on all data. Figure 2 illustrates the parallel processing scheme for the group-level effects.

Different from the general scheme, the result of the map phase is a set of key-value pairs (Record Loglikelihood). The key is the group identifier and the value is the associated loglikelihood. The reduce phase aggregates the values by the key. Such reduction is first performed within each split, that is, values with the same key are combined (Split/Group Loglikelihood). This step does not incur network communication cost since all data reside in the same computer node. The final step takes the key-value pairs from the combiner and aggregates the values by key to form the group-level loglikelihood (Group Loglikelihood). Because the data splits could sit in multiple computer nodes, this last step generally involves shipping of data among computers in the network, a process generally known as shuffling.

2.4. Method of parallel processing

When it comes to implementation of the parallel processing scheme, several technologies are available, which offer different values in terms of computation performance, coding complexity and scalability. We briefly discuss some of the prominent technologies and more details can be found in Grama *et al.* (2003) and Reyes-Ortiz *et al.* (2015).

One popular parallel processing method is shared memory programming, such as POSIX Threads and OpenMP. They are fairly easy to use and work well with multiple cores/processors. However, they generally run on a single machine and are therefore not scalable.

Message passing (MPI) is another well-known method for high-performance computing. It is highly scalable since it can work with a cluster of computer nodes. Parallel computing based on a network of computers is often referred to as distributed computing. The performance of optimized C/C++ program using MPI is often hard to beat. However, creating parallel tasks at scale using MPI could be very challenging. For this reason, it is not popular among practitioners and rarely used in commodity data centers.

This research chose Apache Spark as the distributed computing engine (Zaharia *et al.*, 2010). It is the state of the art for high-performance distributed computing, designed to effectively perform iterative procedures that operate on the same data set. Spark runs on a cluster of computer nodes and is thus highly scalable to deal with huge amount of data. It has become the standard for large-scaled machine learning, and many insurance companies have adopted this technology.

3. Empirical analysis

In this section, we implement the distributed MCMC algorithm and compare its performance with standard single-thread algorithms. We also test the Bayesian hierarchical Tweedie model using a real data set and demonstrate the benefits of Bayesian ratemaking.

3.1. Data

We use a public data set from the Allstate claim prediction challenge.[2] It includes 13.2 million samples of vehicle-level claim information from the bodily injury coverage of personal auto insurance. About 99% of the records do not observe a payment, which justifies the use of the Tweedie distribution. There are about 30 variables containing various characteristics of the insured vehicle as well as some non-vehicle characteristics. Both the names and the values of most variables are masked and we cannot infer their meanings. Two labeled fields relevant to this analysis are the vehicle's make and model. We specify vehicle make and vehicle model level effects and analyze how loss propensity varies by vehicle make and model. There are 75 unique vehicle makes and 1,303 unique vehicle models. In addition to the group-level effects, we include nine predictors selected through standard variable selection procedures based on data samples.

The final data set consisting of only the relevant variables amounts to about 1.1 GB. While this may not seem a "big data" based on size alone, it is in fact very "big" in the domain of Bayesian analysis where, due to the use of computationally intensive algorithms, data size of Megabytes could be considered large. Indeed, this data size resembles the magnitude of the real data that insurers, especially commercial insurers, deal with, which is typically in the order of GBs.

3.2. Implementation

In the hierarchical Tweedie model, we include an intercept and the nine selected variables as predictors, where the intercept is allowed to vary by vehicle make and vehicle model. We specify standard non-informative priors for parameters in the model and derive the full MCMC simulation scheme in the appendix.

We implemented the simulation algorithms in Scala, "Scalable Language", an object-oriented and functional programming language that is used to create Spark. We spun up a cloud cluster of 5 computer nodes running the Linux system, each equipped with 50 cores and about 100 GB memory.

3.3. Performance analysis

We ran both the distributed and the single-thread MCMC algorithm and compared their performances on the data. An additional larger data set (2.2 GB) created by stacking two replicates of the real data set was also used for the test. In the distributed algorithm, the heavy-computing job was divided into 100 parallel tasks for the real data set and 200 parallel tasks for the artificial data set. In the single-thread algorithm, the job was executed using a single core, which is the default of most computing software.

Table 1 reports the average execution time based on five independent runs of each algorithm for 100 iterations. Several patterns are noteworthy. First, we notice a striking difference between the running time of the two algorithms on each data set. The distributed algorithm achieved about 40 and 65 times performance gain, respectively, compared to the standard single-thread algorithm. This speed improvement is of significant importance because it makes practical applications of Bayesian methods possible. To put it into perspective, it would take 25.5 days to run 20,000 iterations of the standard single-thread MCMC algorithm on the real data set. This effectively renders the Bayesian methods infeasible. In contrast, the same job using the distributed algorithm took only

TABLE 1

PERFORMANCE COMPARISON OF THE SINGLE-THREAD AND THE DISTRIBUTED MCMC ALGORITHM FOR 100 ITERATIONS.

	Real Data (1.1 GB)		Artificial Data (2.2 GB)	
Algorithm	Time (min)	Improvement	Time (min)	Improvement
Single-thread	183.5	–	375.3	–
Distributed	4.6	**40** times	5.8	**65** times

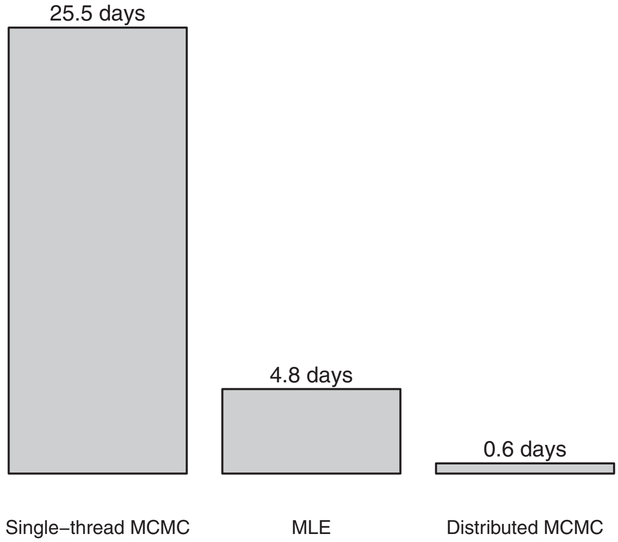

FIGURE 3: Computation time of different algorithms on the claim data. The MCMC algorithms were run for 20,000 iterations to get reliable posterior inference. The maximum likelihood estimation (MLE) was based on Laplace approximation, implemented as a single-thread C program.

14 hours to finish. Such a job could be run over night, causing little disruption to other analytical work. In addition, we also estimated the model using the state-of-the-art likelihood-based method (Zhang, 2013), implemented using a single-thread C program. Interestingly, this direct optimization took 116 hours, or 4.8 days, to finish. This is more than 8 times slower than the distributed MCMC, although still significantly faster than the single-thread MCMC. Figure 3 compares the computation times of these methods graphically.

The second important observation is that the distributed algorithm is highly scalable, thanks to the capability of Spark to scale to large tasks. When the size of the data was doubled, the processing time of the single-thread method was roughly doubled. In comparison, the processing time of the distributed method was only increased by 26%, due to the use of additional computational resource (the number of cores was doubled).

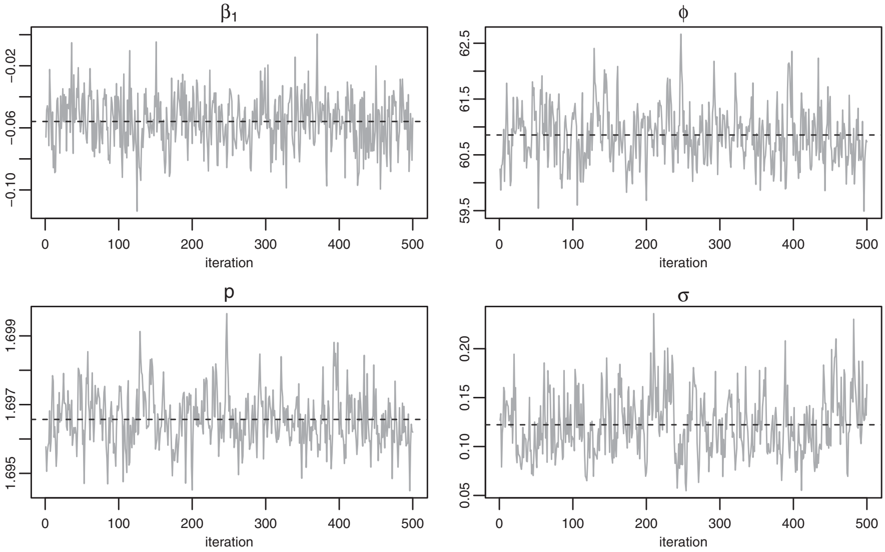

FIGURE 4: The simulated values in the first 500 iterations for four parameters: the parameter for one predictor β_1, the scale parameter of the Tweedie distribution ϕ, the power (index) parameter of the Tweedie distribution p and the vehicle make-level variation σ. The dashed line indicates the posterior mean in each figure.

3.4. Bayesian ratemaking

We carried out a full run of the distributed MCMC algorithm to simulate all parameters from the posterior distributions. We ran one chain for 20,000 iterations, which finished within 14 hours as a scheduled overnight job. In the first 5,000 iterations, we adjusted the variance of the Gaussian proposal distribution for each parameter to get reasonable empirical acceptance rates (Browne and Draper, 2006). At the end of the tuning phase, the acceptance rates of most parameters were between 40% and 60%, which indicate properly tuned proposal distributions for univariate Metropolis simulations (Gelman *et al.*, 2003). The samples drawn during the adaptive phrase were discarded. The tuned proposal variances were used in simulations that followed the adaptive phase without any adjustment. To reduce auto-correlation, we used every fifth iteration of the simulation. This resulted in 3,000 simulated draws for each parameter. Figure 4 shows the simulated values for four representative parameters in the model. These mixing patterns show strong evidence that the chain reached approximate convergence after the tuning phase. Indeed, the potential scale reduction factors (Gelman *et al.*, 2003) were below 1.1 for all parameters.

Predictive Distribution. The fundamental metric to forecasting problems is the predictive distribution given the observed data, which provides the full picture of the forecasted values. Suppose $\boldsymbol{y}$ is all observed historical data and $\boldsymbol{\theta} = (\boldsymbol{\beta}, \boldsymbol{u}, p, \phi, \sigma)$ contains all parameters in the model. A new risk is represented by the tuple $(g_{\text{new}}, \boldsymbol{x}_{\text{new}}, y_{\text{new}})$, where $\boldsymbol{x}_{\text{new}}$ and g_{new} are the observed predictors and group indicator, and y_{new} is the unobserved future loss to be

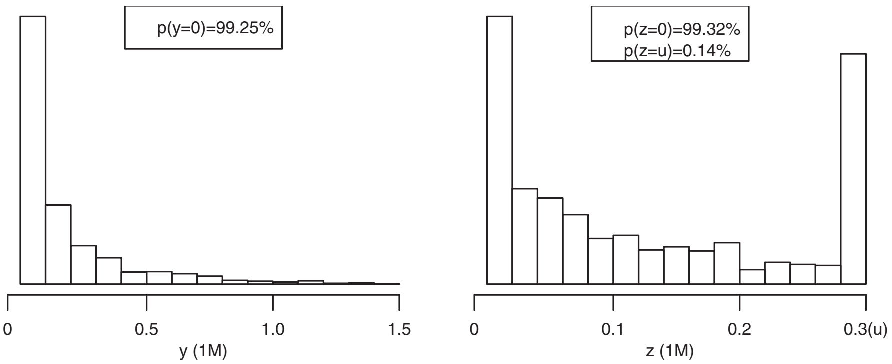

FIGURE 5: The predictive distribution (histogram) of positive losses for a new risk. Zero losses are not included in the histogram due to the disproportionally large frequency. The probability of zero loss is reported at the top. The left plot is the predictive distribution of the ground-up loss. The right plot is the predictive distribution of the loss with a 1,000 deductible and 300,000 limit.

predicted. The predictive distribution of the new risk given the historical data is

$$f(y_{\text{new}}|\boldsymbol{y}) = \int_{\boldsymbol{\theta}} f(y_{\text{new}}|\boldsymbol{\theta}) f(\boldsymbol{\theta}|\boldsymbol{y}) d\boldsymbol{\theta}. \tag{6}$$

That is, the parameters in the conditional distribution $f(y_{\text{new}}|\boldsymbol{\theta})$ are integrated out with respect to their posterior distributions given the data. In the Bayesian setting, this predictive distribution can be computed naturally, by simulating the loss of the new risk based on each posterior draw of the parameters. That is, for the sth posterior draw $\boldsymbol{\theta}^{(s)}$, we simulate the new loss as $y_{\text{new}}^{(s)} \sim Tw(\mu_{\text{new}}^{(s)}, \phi^{(s)}, p^{(s)})$ for $s = 1, \ldots, S$. These simulated values come from the desired predictive distribution and can be used to derive other quantities of interest. The left plot of Figure 5 shows the predictive loss distribution for a new risk.

Predictive Distribution for Modified Loss. In practice, insurance policies are issued with coverage modifiers, such as deductibles and policy limits. These policy clauses lead to modified loss distributions. For example, the loss for a new policy with limit u is $y_{\text{new}} \wedge u = \min(y_{\text{new}}, u)$, and the loss under deductible d and limit u is $z_{\text{new}} = (y_{\text{new}} \wedge u) - (y_{\text{new}} \wedge d)$. The derivation of the predictive distribution of the modified loss variable is straightforward through Monte Carlo sampling. For each predictive draw of the ground-up loss $y_{\text{new}}^{(s)}$, we apply the deductible and limit to obtain the modified loss $z_{\text{new}}^{(s)}$. The right plot of figure 5 shows the predictive loss distribution for a new risk that has \$1,000 deductible and \$300,000 limit.

The availability of the predictive distribution greatly facilitates the assessment of new risks with policy coverage modifiers. Various characteristics of the risk distribution can be summarized through sample statistics. One important statistic is the mean predicted loss cost which is the core component of the premium. It can be estimated as $E(z_{\text{new}}|\boldsymbol{y}) \approx \frac{1}{S}\sum_{s=1}^{S} z_{\text{new}}^{(s)}$. In contrast, derivation of the truncated or limited expected loss in many non-Bayesian methods typically involves numerical integrations. For example, the expected loss under policy limit u is $E(y \wedge u) = \int_0^u S(y)dy$, where $S(y)$ is the survival function of the underlying loss variable (Klugman *et al.*, 2012). In the case of the Tweedie loss distribution, the survival function has no closed-form solution. Numerical approximations are necessary to approximate this integral.

Remark 1. *Bayesian predictive distributions facilitate loss cost forecast under policy coverage modifiers, such as deductibles and policy limits.*

Bayesian Uncertainty Measures. Uncertainty measures of forecasts are important for insurance risk management. The Bayesian approach has many advantages for measuring forecast uncertainty in insurance ratemaking. First, it is a great challenge to derive uncertainty measures of truncated or limited losses using numerical methods. In comparison, risk measures such as the standard deviation, credible interval and value-at-risk can be constructed readily through the sample statistics of the simulated modified losses from the Bayesian model.

Second, it is difficult to accurately estimate the group-level variation in a hierarchical model. Likelihood-based methods generally underestimate the group-level variation and Bayesian methods have been shown to provide more accurate estimates (Browne and Draper, 2006; Zhang, 2013).

Third, an important source of uncertainty when making predictions arises from the uncertainty of model parameters. The Bayesian risk measures inherently take into account the uncertainty in estimating all parameters, as a result of the integration over the posterior distribution of the parameters. Various methods have been proposed to account for the estimation uncertainty in making insurance forecasting. In the area of loss reserving, England and Verrall (1999) developed both analytical and bootstrapping methods to assess the predictive uncertainty. However, these methods face great difficulties when applied to insurance ratemaking. First, it is not clear how the analytical approach, derived via Taylor's approximation, can be adapted to policies with deductibles and limits. Second, the bootstrapping method needs to estimate a large number of Tweedie mixed models to get the sampling distribution of the parameters. However, the estimation of one single model using the likelihood-based method took about 5 days on the current data. Third, bootstrapping cannot resolve the issue of downwardly biased estimate of the group-level variation.

Remark 2. *Bayesian predictive distributions produce uncertainty measures that appropriately account for group-level variations and uncertainties in model estimation.*

Credibility Estimates. The actuarial profession has relied on the credibility weighting procedure to make robust estimation when there is insufficient information from the historical data. In this regard, the Bayesian hierarchical model is a very appealing approach to insurance ratemaking, as it provides a formal mechanism for generating rates that incorporate actuarial credibility. Indeed, Frees *et al.* (1999) demonstrate that various actuarial credibility methods can

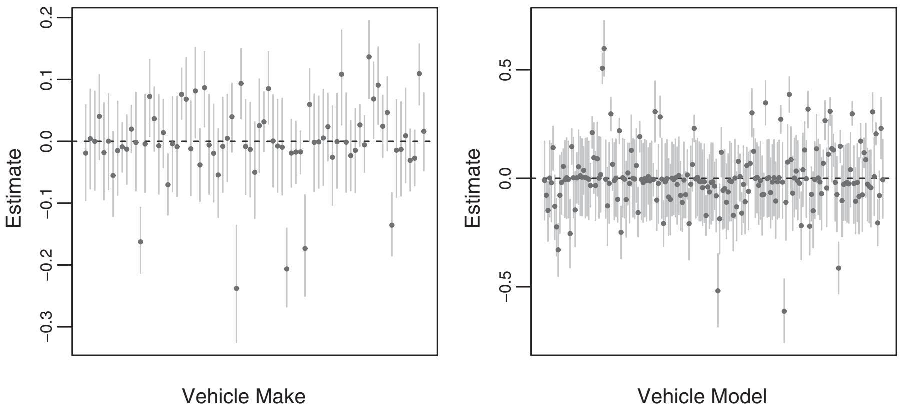

FIGURE 6: The posterior mean and 50% credible interval for the vehicle make-level and vehicle model-level effects. Red dots represent the posterior mean and the gray vertical line corresponds to the 50% interval. A randomly selected 200 vehicle makes are shown to avoid clustered plot.

be expressed as special cases of hierarchical models with Gaussian errors. The Bayesian hierarchical Tweedie model thus extends the core concept of actuarial credibility to insurance ratemaking based on individual loss data. Figure 6 shows the estimated vehicle make-level and vehicle model-level effects, along with the 50% credible intervals. Many of these effects are distributed around zero as a result of shrinking the individual estimates.

Remark 3. *Bayesian hierarchical Tweedie model provides a formal mechanism to incorporate the core insight of actuarial credibility in insurance ratemaking.*

4. CONCLUSIONS AND DISCUSSIONS

The rise and rapid growth of big data analytics undoubtedly brings opportunities to improve information extraction from insurance data, but it also creates a unique set of challenges. One challenge is how to make probabilistic forecasts that can be utilized in business decision making. Another challenge is how to overcome the computational burden and process large volumes of data efficiently. The paper suggests that Bayesian methods, implemented via distributed MCMC, are well suited for large-scaled insurance predictive modeling. The empirical analysis demonstrates that when applied to insurance ratemaking, Bayesian methods excel in several ways. In particular, Bayesian methods facilitate loss cost forecast reflecting policy coverage clauses, produce coherent uncertainty measures and provide a formal mechanism to incorporate the core insight of actuarial credibility through hierarchical modeling.

While rapidly growing, the development of big-data analysis is still in the early stage. For practical implementation of large-scaled Bayesian ratemaking, we recommend the following three-step strategy.

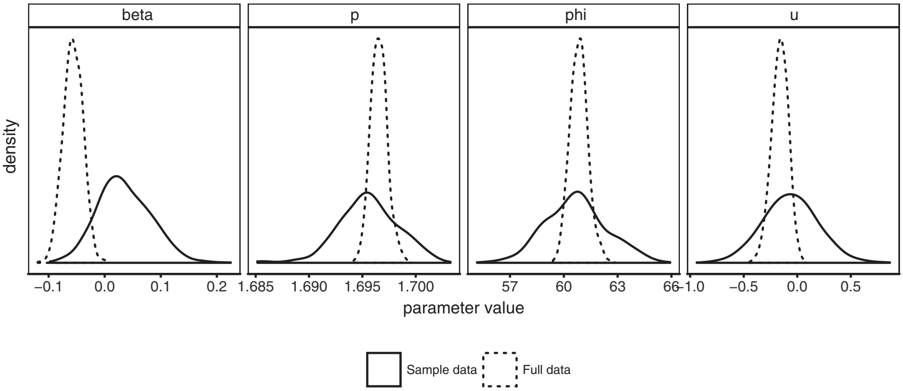

FIGURE 7: The posterior distributions of four representative parameters estimated using a sample of data and the full data, respectively.

1. *Single-thread non-Bayesian analysis using sample data*. Take a representative random sample from the big data and analyze the sample data using standard data analysis and modeling techniques without parallel processing. This helps to gain initial insights from the data and design preliminary modeling strategies quickly.
2. *Distributed non-Bayesian analysis using full data*. Test the data transformation and modeling strategies on the full data. This step requires a big-data platform such as Spark to efficiently implement data manipulation and statistical learning methods. Run additional analyses such as pattern and trend detection, variable selection, model comparison, out-of-sample validation and so on to finalize the model structure.
3. *Distributed Bayesian analysis using full data*. Formulate the chosen model into the Bayesian setting and run distributed MCMC on the full data to obtain parameter estimates as posterior samples.

We recommend starting the analysis using a sample of data. This will help gain initial understanding of the data quality, the necessary variable transformations and so on. However, the modelers must be aware that patterns from the sample may be different from those in the full data due to sampling variations. To illustrate this, we also estimated a model using only 10% of our data and compared the estimated distributions of some representative parameters to those in the full model in Figure 7. Notably, the posterior distribution using the full data is more dense than that using the sample data, because of the use of 10 times more data. This is particularly important for the group-level effect, the shrinkage of which depends on the number of observations in each level. We see that in Figure 7, it can be hard to identify a significant group deviation in small samples of sparse data like insurance loss. The availability of big data provides the opportunity to find more signals in the data.

We recommend running distributed non-Bayesian analysis because many classic learning algorithms are already implemented on major distributed platforms (no support for distributed hierarchical model on Spark at the moment). The full-blown Bayesian analysis typically produces estimates consistent with those from the classical counterparts, but has the additional advantage of getting the full posterior distribution of parameters that is valuable for measuring and managing insurance risks.

Beyond ratemaking, large-scaled Bayesian analysis also provides value in other areas of insurance predictive modeling. For example, in individual loss reserving, the Bayesian predictive distribution can be used to derive ranges for both individual and aggregate reserves. In particular, with the arise of usage-based and ride-sharing insurance, actuaries have the opportunity to perform reserving analysis for each trip, namely a driver going from point A to point B. Trip-level characteristics can be leveraged to infer the likelihood that an accident has occurred on the trip for the purpose of booking the incurred but not reported claims. Another potential application is in targeted insurance marketing, such as customer acquisition or customer retention. The individual-level predictive distribution is critical for successful marketing strategies and targeting decisions, given that the individual-level densities are likely to be non-symmetric or heavy tailed (Rossi *et al.*, 1996).

We believe that Bayesian methods are valuable in many areas of insurance analytics. The current paper brings a viable solution for practical application of Bayesian methods. We hope this will spur the application of Bayesian analysis in insurance predictive modeling, which has the potential to transform the way that actuaries assess and manage insurance risks.

Notes

[1] In contrast, another noteworthy strategy is to partition the data and run independent MCMC in parallel on each data partition (Ormandi *et al.*, 2015; Wang and Dunson, 2016). This method works when the posterior can be factored as products of the individual posterior from each partition. Such a condition, however, is not satisfied for the variance component in the hierarchical model we consider in this paper.

[2] https://www.kaggle.com/c/ClaimPredictionChallenge

References

AGARWAL, A. and DUCHI, J.C. (2012) Distributed delayed stochastic optimization. In *Proceedings of the IEEE 51st Annu. Conf. Decision and Control (CDC)*, IEEE, vol. 83, pp. 5451–5452.

BERMUDEZ, L. and KARLIS, D. (2011) Bayesian multivariate Poisson models for insurance ratemaking. *Insurance: Mathematics and Economics*, **48**, 226–236.

BROWNE, W.J. and DRAPER, D. (2006) A comparison of Bayesian and likelihood-based methods for fitting multilevel models. *Bayesian Analysis*, **1**(3), 473–514.

BÜHLMANN, H. (1967) Experience rating and credibility. *ASTIN Bulletin*, **4**(3), 99–207.

DE ALBA, E. (2002) Bayesian estimation of outstanding claims reserves. *North American Actuarial Journal* **6**(4), 1–20.

DUNN, P.K. and SMYTH, G.K. (2005) Series evaluation of Tweedie exponential dispersion models densities. *Statistics and Computing*, **15**, 267–280.

DUNN, P.K. and SMYTH, G.K. (2008) Evaluation of Tweedie exponential dispersion model densities by Fourier inversion. *Statistics and Computing*, **18**, 73–86.

ENGLAND, P.D. and VERRALL, R.J. (1999) Analytic and bootstrap estimates of prediction errors in claims reserving. *Insurance: Mathematics and Economics*, **25**, 281–293.

FREES, E.W., DERRIG, R.A. AND MEYERS, G. (2014) *Predictive Modeling Applications in Actuarial Science*, vol. 1. New York: Cambridge University Press.

FREES, E.W., YOUNG, V.R. and LUO, Y. (1999) A longitudinal data analysis interpretation of credibility models. *Insurance: Mathematics and Economics*, **24**(3), 229–247.

GELMAN, A., CARLIN, J.B., STERN, H.S. and RUBIN, D.B. (2003) *Bayesian Data Analysis*, 2nd ed. Boca Raton, FL: CRC Press.

GRAMA, A., KARYPIS, G., KUMAR, V. and GUPTA, A. (2003) *Introduction to Parallel Computing*, 2nd ed. Harlow: Pearson.

GREEN, P.J., LATUSZYNSKI, K., PEREYRA, M. and ROBERT, C.P. (2015) Bayesian computation: A summary of the current state, and samples backwards and forwards. *Statistics and Computing*, **25**, 835–862.

HAARIO, H., SAKSMAN, E. and TAMMINEN, J. (2001) An adaptive metropolis algorithm. *Bernoulli*, **7**, 223–242.

JØRGENSEN, B. and DE SOUZA, M.C. (1994) Fitting Tweedie's compound Poisson model to insurance claims data. *Scandinavian Actuarial Journal*, **1**, 69–93.

KLUGMAN, S.A., PANJER, H.H. and WILLMOT, G.E. (2012) *Loss Models: From Data to Decisions*. Hoboken, NJ: Wiley.

NEAL, R. (2013) MCMC using Hamiltonian dynamics. In *Handbook of Markov Chain Monte Carlo* (eds. S. Brooks, A. Gelman, G. Jones and X.L. Meng), pp. 113–162. Boca Raton, FL: Chapman and Hall.

NEISWANGER, W., WANG, C. and XING, E. (2014) Asymptotoically exact, embarassingly parallel MCMC. In *UAI'14 Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, pp. 623–632. Arlington, VA: AUAI Press.

NTZOUFRAS, I. and DELLAPORTAS, P. (2002) Bayesian modelling of outstanding liabilities incorporating claim count uncertainty. *North American Actuarial Journal*, **6**(1), 113–128.

ORMANDI, R., YANG, H. and LU, Q. (2015) Scalable multidimensional hierarchical Bayesian modeling on spark. *JMLR: Workshop and Conference Proceedings*, **41**, 33–48.

PETERS, G. W., SHEVCHENKO, P.V. and WÜTHRICH, M.V. (2009) Model uncertainty in claims reserving within Tweedie's compound Poisson models. *ASTIN Bulletin*, **39**(1), 1–33.

REYES-ORTIZ, J.L., ONETO, L. and ANGUITA, D. (2015) Big data analytics in the cloud: Spark on hadoop vs mpi/openmp on beowulf. *Procedia Computer Science*, **53**, 121–130.

ROBERTS, G. and TWEEDIE, R. (1996) Geometric convergence and central limit theorems for multidimensional hastings and metropolis algorithms. *Biometrika*, **83**, 95–110.

ROSSI, P.E., MCCULLOCH, R.E. and ALLENBY, G.M. (1996) The value of purchase history data in target marketing. *Marketing Science*, **15**(4), 321–340.

SHI, P. (2016) Insurance ratemaking using a copula-based multivariate Tweedie model. *Scandinavian Actuarial Journal*, **3**, 198–215.

SHI, P., SANJIB, B. and MEYERS, G. (2012) A Bayesian log-normal model for multivariate loss reserving. *North American Actuarial Journal* **16**(1), 29–51.

SMYTH, G.K. and JØRGENSEN, B. (2002) Fitting Tweedie's compound Poisson model to insurance claims data: dispersion modelling. *ASTIN Bulletin*, **32**, 143–157.

TWEEDIE, M.C.K. (1984) An index which distinguishes between some important exponential families. In *Statistics: Applications and New Directions, Proceedings of the Indian Statistical Institute Golden Jubilee International Conference* (eds. J.K. Gosh and J. Roy), pp. 579–604. Calcutta: Indian Statistical Institute.

WANG, X. and DUNSON, D.B. (2016) *Parallelizing MCMC via weierstrass sampler*. Working Paper `http://arxiv.org/pdf/1312.4605`.

WELLING, M. and TEH, Y. (2011) Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pp. 681–688.

WÜTHRICH, M. (2003) Claims reserving using Tweedie's compound Poisson model. *Astin Bulletin*, **33**, 331–346.

WÜTHRICH, M.V. and MERZ, M. (2008) *Stochastic Claims Reserving Methods in Insurance*. Hoboken, NJ: Wiley.

ZAHARIA, M., CHOWDHURY, M., FRANKLIN, M.J., SHENKER, S. and STOICA, I. (2010) Spark: Cluster computing with working sets. In *HotCloud'10 Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, p. 10. Berkeley, CA: USENIX Association.

ZHANG, Y. (2013) Likelihood-based and Bayesian methods for Tweedie compound Poisson linear mixed models. *Statistics and Computing*, **23**(6), 743–757.

ZHANG, Y., DUKIC, V. and GUSZCZA, J. (2012) A Bayesian non-linear model for forecasting insurance loss payments. *Journal of the Royal Statistical Society: Series A*, **175**(2), 637–656.

ZHU, J., CHEN, J. and HU, W. (2014) Big learning with Bayesian methods. *National Science Review* nwx044. doi:10.1093/nsr/nwx044.

YANWEI ZHANG (Corresponding author)
Uber Technologies
1455 Market Street
San Francisco, CA 94103, USA
E-Mail: actuary_zhang@hotmail.com

APPENDIX: DERIVATION OF THE MCMC ALGORITHM

We derive the Metropolis-within-Gibbs sampler used to simulate samples from the posterior of the parameters. In particular, we derive the sampling scheme for the model specified in (1)–(3). Other variations of the hierarchical structure including nested or crossed effects can be derived likewise. The posterior distribution is proportional to the joint distribution of the data and the priors. In the Metropolis-within-Gibbs sampler, we sequentially sample each parameter from its full conditional distribution given all the other parameters and the data. If the full conditional cannot be easily simulated from, we resort to the Metropolis algorithm. The logarithmic joint posterior density corresponding to this model is

$$\begin{aligned}\ell &= \log f(\boldsymbol{y}, \boldsymbol{\beta}, \boldsymbol{u}, \sigma^2, \phi, p)\\ &= \underbrace{\sum\nolimits_i \log f_{Tw}(y_i|\boldsymbol{\beta}, u_{g_i}, p, \phi)}_{\text{data likelihood}} + \underbrace{\sum\nolimits_j \log f_N(u_j|\sigma^2)}_{\text{group-level distribution}} + \underbrace{\log \pi(\boldsymbol{\beta}, \phi, p, \sigma^2)}_{\text{prior distribution}}.\end{aligned}$$

We now describe in detail the random-walk Metropolis algorithm for the kth component of $\boldsymbol{\beta}$, denoted β_k. First, we propose a new sample from a normal distribution with the current value β_k as the mean and a pre-specified proposal variance σ_p^2.

Second, we compute the log joint density ℓ_{new} and ℓ_{old} using the newly sampled value and the value from the last iteration, respectively. In this step, the data loglikelihood is computed in parallel across multiple cores from a network of computer nodes.

Last, we decide whether to accept the new sample. Let $A = \exp(\ell_{\text{new}} - \ell_{\text{old}})$. If $A > 1$, we accept the new sample. Otherwise, we accept it with probability A. This finishes one simulation for β_k, and we continue the Gibbs step and move to β_{k+1}. This algorithm is summarized in Algorithm 1.

The simulation of ϕ and p follows the above algorithm similarly. However, the simulation is based on transformations of these parameters. Specifically, we simulate on the scale of $\log(\phi)$ and $\text{logit}(p - 1)$. These transformations are made so that the new variables span the

```
Initialize: $\ell_{\text{new}} = 0$, $\ell_{\text{old}} = 0$, and the proposal variance $\sigma_p^2$
Module updateBetak:                                          // function to update $\beta_k$
    Set $\boldsymbol{\beta}_{\text{old}} = \boldsymbol{\beta}$;                                          // store current value
    Simulate new sample $\beta_k \leftarrow N(\beta_k, \sigma_p^2)$;     // simulate new value & update
    parallel for $i$ :                                        // parallel computation
        $\ell_{\text{new}} \leftarrow \ell_{\text{new}} + \log f(y_i, \boldsymbol{\beta}, \boldsymbol{u}, \sigma^2, \phi, p)$;  // update new loglikelihood
        $\ell_{\text{old}} \leftarrow \ell_{\text{old}} + \log f(y_i, \boldsymbol{\beta}_{\text{old}}, \boldsymbol{u}, \sigma^2, \phi, p)$; // update old loglikelihood
    $\ell_{\text{new}} \leftarrow \ell_{\text{new}} + \log \pi(\boldsymbol{\beta})$ ;                              // prior contribution
    $\ell_{\text{old}} \leftarrow \ell_{\text{old}} + \log \pi(\boldsymbol{\beta}_{\text{old}})$ ;                            // prior contribution
    Set $A = \exp(\ell_{\text{new}} - \ell_{\text{old}})$;
    if $A > 1$ then return $\boldsymbol{\beta}$;                              // accept new sample
    else
        if $Unif(0, 1) < A$ then return $\boldsymbol{\beta}$;                  // accept new sample
        else return $\boldsymbol{\beta}_{\text{old}}$;                                // reject new sample
    end
```

Algorithm 1: Simulation of the kth element of $\boldsymbol{\beta}$

real line, enabling the use of the random-walk Metropolis algorithm. Otherwise, truncated Normal distributions could be chosen as the proposal.

For the simulation of the group-level effects $\boldsymbol{u}$, we notice that the full conditional of each component is independent. For example, the full conditional of u_j is

$$\sum_{i:g_i=j} \log f_{Tw}(y_i|\boldsymbol{\beta}, u_{g_i}, p, \phi) - \frac{1}{2}u_j^2/\sigma^2.$$

That is, the part of loglikelihood calculation only involves observations in group j. The Metropolis step can be simplified as follows. First, given the current values $\boldsymbol{u}_{\text{old}}$, simulate $\boldsymbol{u}_{\text{new}} \sim N(\boldsymbol{u}_{\text{old}}, \tau \boldsymbol{I})$, where τ is a pre-specified proposal variance. Second, compute in parallel the **by-group** logarithmic joint density $\boldsymbol{\ell}_{\text{new}}$ and $\boldsymbol{\ell}_{\text{old}}$ using $\boldsymbol{u}_{\text{new}}$ and $\boldsymbol{u}_{\text{old}}$, respectively. These logarithmic joint densities are now vectors. Third, let $\boldsymbol{A} = \exp(\boldsymbol{\ell}_{\text{new}} - \boldsymbol{\ell}_{\text{old}})$, and for each component of $\boldsymbol{A}$, apply the Metropolis decision rule as above.

Finally, the sampling of the variance component is straightforward. The posterior distribution of the variance component σ^2 is an inverse-Gamma distribution

$$\sigma^{-2} \sim \text{Gamma}\left(\frac{J}{2} + C_1, \frac{1}{2}\sum\nolimits_{j=1}^{J} u_j^2 + C_2\right),$$

if we specify a conjugate inverse Gamma prior with C_1 and C_2 as the shape and scale parameter, and J is the number of groups.

Below, we summarize some additional implementation details.

- The prior distributions we used are $\beta_k \sim N(0, 100^2)$, $\log(\phi) \sim N(0, 100^2)$, $\text{logit}(p-1) \sim N(0, 100^2)$ and $\sigma^{-2} \sim \text{Gamma}(0.01, 0.01)$.
- Note that the full conditionals of ϕ and p depend on the normalizing quantity $a(y, \phi)$ in (4). We implement the numerical approximation method in Dunn and Smyth (2005) in Scala to calculate this quantity. The code is provided below. It is also to be noted that this

approximation is not needed for the other parameters because their full conditionals do not depend on the normalizing quantity.

- To avoid numerical issues, we operate on the logarithmic scale. Standard procedure is applied when calculating the log sum of exponentials to avoid overflow/underflow issues. This is implemented in the Tweedie density evaluation below.

```
/**
 * Compute the log density for the Tweedie compound Poisson distribution.
 *
 * @param y the observation
 * @param mu the mean parameter
 * @param phi the dispersion parameter
 * @param p the index parameter
 *
 */
 def dtweedie(y: Double, mu: Double, phi: Double, p: Double): Double = {

  val TWEEDIE_DROP = 37.0
  val TWEEDIE_INCRE = 5
  val TWEEDIE_NTERM = 20000
  val p1: Double = p - 1.0
  val p2: Double = 2.0 - p

  if (y == 0.0) return(-pow(mu, p2)/(phi * p2))

  val a: Double = -p2/p1
  val a1: Double = 1.0/p1
  val jmax = max(1.0, pow(y, p2)/(phi * p2))
  val logz = -a * log(y) - a1 * log(phi) + a * log(p1) - log(p2)

  val cc = logz + a1 + a * log(-a)   // locate upper bound
  val w = a1 * jmax
  var j: Double = jmax + TWEEDIE_INCRE
  while (j * (cc - a1 * log(j)) >= (w - TWEEDIE_DROP))
   j += TWEEDIE_INCRE
  val jh = ceil(j);

  j = jmax - TWEEDIE_INCRE           // locate lower bound
  while (!((j < 1 || j * (cc - a1 * log(j)) < w - TWEEDIE_DROP)))
   j -= TWEEDIE_INCRE
  val jl = max(1, floor(j))
  val jd = jh - jl + 1;

  val nterms = min(jd, TWEEDIE_NTERM).toInt
  var ww = Array.fill(nterms)(0.0)
  for (k <- 0 until nterms){
   j = k + jl ;
   ww(k) = j * logz - lgamma(1 + j) - lgamma(-a * j);
  }
  val ww_max = ww.reduceLeft(_ max _)
  val sum_ww = ww.map(x => exp(x - ww_max)).sum   //overflow
  return(-y/(phi * p1 * pow(mu, p1)) - log(y) + log(sum_ww)
        + ww_max -pow(mu, p2)/(phi * p2))
 }
```