MODELLING FATALITIES CAUSED BY CONFLICT EVENTS IN BANGLADESH

FEARGUS E. JAMIESON BALL

Abstract. Understanding conflict events is intrinsic to understanding disorder in society, with fatalities being one of the main impacts of conflict. Governments, journalists and researchers would be aided in their responses to violent conflict by understanding how events cause fatalities. We present different models for understanding how fatalities are linked to conflict events and find that it is important to ignore time periods with zero events, which are the majority, when building fatality models. We also show that individual events are not independent of one another and models need to reflect this. Finally, the strengths and weaknesses of the models are placed in context to provide suggestions to guide further modelling.

1. Introduction

An understanding of conflict is essential to understanding the impact of disorder in society, and one of the main impacts of any conflict event is the fatalities that it causes. Work has already been done to understand the location and timing of conflict events in four South Asian countries as a discrete time Hawkes process[4]. This is a form of self-exciting spatio-temporal point process; i.e. a process in which events occurring at a particular location in space and time can spark other events at future times and in different locations. It would be of use to governments, journalists and researchers to understand how fatalities are caused by conflict events, especially in times of unrest, so that these organisations can better respond to crisis.

We here present models to understand how fatalities are caused by conflict events in Bangladesh, one of the four South Asian countries whose conflicts have already been investigated in prior research. Bangladesh has the eighth-highest population in the world and one of the highest population densities. One seventh of the population live in Dhaka, the capital city. Bangladesh is divided into eight divisions at the first administrative level and 64 districts on the second level.

ACLED - the Armed Conflict Location and Event Data Project - collect data about the location and timing of conflict events globally[1]. The types of events collected and recorded by them are:

- Battles: violence between two organised armed groups;

Date: August 2023.

- Explosions/Remote violence: violence conducted from afar, limiting the ability of the target to respond;
- Protests: nonviolent action, usually against a political entity;
- Riots: violent events where demonstrators engage in violence against property or people;
- Strategic developments: nonviolent activity by agents otherwise involved in conflict, which can include looting, recruitment and arrests;
- Violence against civilians: events where an organised armed group deliberately inflicts violence on unarmed non-combatants.

The dataset used is adapted from the ACLED dataset and provides information on the number of events of each of the six conflict types in each district for each day of 2010-2021, along with information on the number of fatalities. Information such as the population and area of each district is also provided, as well as the latitude and longitude of its centre.

To simplify the modelling, the response variable is taken to be the presence of a fatality in a certain temporal window, as opposed to the number of fatalities. This statistic is also more useful to interested organisations, as it is more valuable to have a strong idea of whether fatalities will occur than it is to have a weaker idea of a number of fatalities. Furthermore, the fatality data collected by ACLED is less accurate than their conflict data, as it is gathered from local news reports and can be heavily influenced by actors in the conflict[1]. Using the binary indicator reduces the effect of data which have been manipulated in this way, and simplifies the modelling.

What is therefore of interest is the probability of at least one fatality occurring in a given temporal window: that is, the probability that the binary indicator F_i takes the value 1 in temporal window i. We present analysis in this report which uses the weekly level. This is a level which reduces the number of data points enough so that computation does not take too long. Furthermore, it reduces the effect of any manipulation that may have happened to the fatality dataset, as whilst a record for an individual event of one or two fatalities may be inaccurate enough that no fatalities actually occurred, it is more likely in an individual week that if there are any fatalities recorded, that fatalities did actually happen.

This report investigates different models for F_i with different covariates. The primary covariate of interest is the number of events in week i. A natural way to model a binary response is with a logistic regression, so this is explored first on the number of events in a week, but also expanded to encompass other covariates. Some issues raised by the logistic regression are removed from the model with a discretisation. However, previous work models a continuous number of events, so it would be of use to be able to incorporate this into the model. For this reason, a different continuous model is also explored.

The final question of interest is the independence of events. For this reason, a final model is built which assumes that the events are completely independent of one another. This model is compared to the other models to investigate how it performs, and conclusions are drawn about the independence of events.

2. Methodology

2.1. **Models considered.**

2.1.1. *Basic Logistic Regression.* The first model we considered is a logistic regression on the number of events in a week. This has reponse variable $F_i \in (0,1)$ and covariate S_i, which is the number of events in week i. The intuition behind this model is that a higher number of events in a week should result in a higher probability of a fatality occurring. Given S_i events in week i, the logistic regression equation expresses the probability of at least one fatality occuring as

$$p_j := \mathbb{P}\left(F_i = 1 | S_i = j\right) = \frac{\exp\left(\alpha + \beta S_i\right)}{1 + \exp\left(\alpha + \beta S_i\right)} \tag{1}$$

for parameters α and β. Note that p_j is not the probability of at least one fatality in week j, but the probability of at least one fatality given that there have been j events. This notation will be used consistently throughout the report. The likelihood can then be expressed as

$$p(F|\alpha,\beta) = \prod_{i=1}^{N} p_{S_i}^{F_i} \left(1 - p_{S_i}\right)^{1-F_i} \tag{2}$$

where F is the vector of all fatality values, N is the number of weeks, and p_{S_i} is given as in Equation 1. This expression is easily derived since $F_i \sim \text{Bernoulli}\left(p_{S_i}\right)$.

2.1.2. *Further Logistic Regression.* The second model we considered is another logistic regression model which explores the effect of fatalities recorded in the previous week. This is presented as the following relationship:

$$p_{j,k} := \mathbb{P}\left(F_i = 1 | S_i = j, F_{i-1} = k\right) = \frac{\exp\left(\alpha + \beta S_i + \gamma F_{i-1}\right)}{1 + \exp\left(\alpha + \beta S_i + \gamma F_{i-1}\right)}. \tag{3}$$

It is realistic to expect that the presence of fatalities in previous weeks may have an effect on the fatalities observed in the current week. For example, fatalities having occurred previously may lead to a lack of capacity in emergency health services, which in turn may lead to more fatalities from subsequent events. That is why this model is singled out from the more general logistic regression presented below.

2.1.3. *Expanded Logistic Regression.* The final of the logistic regression models, this model was built for exploring the relationship between F_i and a much wider set of explanatory variables:

- the number of events in the district, S_i;
- the number of events in the wider division, excluding the district, R_i;
- the fatality indicator for the wider division, excluding the district, G_i.

Similar to the γ parameter in the Further Logistic Regression, this model can incorporate the effect of S_{i-1}, R_{i-1} and G_{i-1} on the binary response variable. This model is not inevstigated here, but has been mentioned as code has been written to investigate it in further work. The restriction imposed by this model is that it can only consider F_i for a district - that is, the administrative level two region - to enable the modelling of variables from the wider level one division.

2.1.4. *Discretisation and Ordering.* Even when the number of events recorded in a week is 0, the logistic regression models tend to produce $\mathbb{P}\left(F_i = 1 | s_i = 0\right) > 0$, which not only goes against intuition but is also not a valid result - if there are no events, there are no fatalities. One solution for this issue is to exclude weeks with no events from the modelling, and see how this affects the logistic regression. This solution is explored later in the report.

Another solution to this problem is to find the posterior distributions for p_j independently for each j. The posterior distributions for each p_j are then conjugate to a Beta prior. However, the issue here is that some summary statistics can be unordered - for example, we had in the posterior means that $\widehat{p_3} < \widehat{p_2}$ when considering Battle-type events in the district of Dhaka. This also is contrary to intuition.

To avoid results like this, we used an ordering system, which assigns a maximum event probability p_K for either $S_i = K = \max_i \{S_i\}$ or $S_i \in K = [K, \infty)$. For $j \in \{1, \ldots, K-1\}$, we then defined

$$p_j = \mathbb{P}\left(F_i = 1 | S_i = j\right) = p_K \prod_{k=j}^{K-1} q_k \tag{4}$$

for model parameters $p_K, q_{K-1}, \ldots, q_1 \in (0, 1)$. This forces in each sample the ordering $p_1 < p_2 < \cdots p_K$ which is conferred to summary statistics and in some sense to the posterior distributions overall.

2.1.5. *Cumulative Probability.* As discussed in the introduction, whilst in theory the discretisation and ordering model is set up to model a blend of observed data and real life very well, it does not integrate well into previous work on modelling the ACLED data which produces an expected number of events in a given time interval[4]. This model is an attempt to blend the flexibility of the logistic regression models with the intuitive advantages of the discretisation and ordering model.

Letting p_1 as usual represent the probability that a week with one event has a fatality, it is thus clear that the probability that a week with one event does not have a fatality is $1 - p_1$. It might therefore be expected that the probability of a week with two events not having a fatality is $(1 - p_1)^2$. Having set p_1, let

$$p_j = \mathbb{P}\left(F_i = 1 | S_i = j\right) = 1 - (1 - p_1)^j \tag{5}$$

which allows the likelihood function to be built in the usual way.

The advantage of this model is built on the discrete data and provides $p_0 = 0$ for all models, regardless of the data. The computational resources are reduced by having the one parameter to sample and when using this model only the one parameter need be known to have the full information about the behaviour of fatalities. Furthermore, this model integrates well with previous work on modelling conflict by allowing a continuous number of weekly events. However, this model does not allow covariates other than the number of events in a week to be investigated, which is some thing that the logistic regression models allow.

2.1.6. *A Null Hypothesis.* The Cumulative Probability Model relies on the assumption that the probability an event causes a fatality is independent of the number of events within a week. However, the value of p_1, which is the only parameter in the model, is influenced by the likelihood function considering all weeks with varying numbers of event counts in the sampling of p_1. To test the hypothesis that a model considering events to be independent performs just as well as a different model presented here (and thus that the probability an event causes a fatality is independent between events) it is possible to sample p_1 based solely on the number of events. The same likelihood as the Cumulative Probability model is then used to find the marginal liklelihood of the null hypothesis, which can be compared to the marginal likelihoods of other models.

2.2. **Model sampling.** Two methods were used to sample from the posterior distributions of the models: an implementation of the Metropolis-Hastings algorithm and the No U-Turn Sampler (NUTS) sampler provided by the Stan software package[6]. The code used for sampling the models can be found in a GitHub repository containing the sampling functions so that further work may make use of them [2].

The implementation of the Metropolis-Hastings sampler was written in Python for the three logistic regression models, for Discretisation and Ordering and for Cumulative Probability. For the latter two the proposal distribution was transformed from a normal proposal with a large variance in $\mathbb{R}$ to the interval $(0, 1)$ using a logistic link, which improved the capability of the sampler to sample. This was after attempting a method which rejected any samples falling outside $(0, 1)$ from a normal proposal with a small variance, which was unable to sample at all from the posterior distributions. The code used for the implementations can be found in the appendix, with code used for the likelihoods available in the GitHub repository associated with this report [3].

2.3. **Marginal Likelihood.** A key aim when building these models is to be able to test them against one another which is naturally done with Bayes Factors, for which marginal likelihoods must be calculated. We considered a few estimators for this. The first was to evaluate the likelihood at a large number of prior samples and find the arithmetic mean. However, this method is inefficient and can be influenced by a small number of samples in areas of high posterior probability. The second is to evaluate the likelihood at a large number of posterior samples and find their harmonic mean. However, this method can be influenced by a small number of samples in the tails of the posterior distribution.

Noting that these problems are in some sense the opposite to one another, we used an iterative method suggested by Newton and Raftery[5]. This method uses only posterior samples, but uses a parameter δ to treat some of them in the likelihood estimate as though they are prior samples. It is suggested that this parameter is kept low. Given samples from the posterior distribution, $\theta_t \overset{\text{i.i.d.}}{\sim} \pi(\cdot|X)$, $t = 1, \ldots, T$, we initialise an estimate for the marginal likelihood $m_0 \in \mathbb{R}$. The following iteration is then used:

$$\hat{m}_{j+1}(X) = \frac{\frac{\delta T}{1-\delta} + \sum_{t=1}^{T} \frac{p(X|\theta_t)}{\delta \hat{m}_j(X) + (1-\delta) p(X|\theta_t)}}{\frac{\delta T}{1-\delta} \hat{m}_j(X) + \sum_{t=1}^{T} \left(\delta \hat{m}_j(X) + (1-\delta) p(X|\theta_t)\right)^{-1}}. \tag{6}$$

Since this is an iterative method, some termination criterion is required. In results shown in this report, the iteration is terminated when the difference between two successive estimates falls below a predetermined value ε. Discussion and justification of this termination criterion can be found in the results section.

This estimator has a much lower variance than the harmonic mean estimator, so can be used with more confidence when selecting models.

3. Results

3.1. **Posterior Sampling.** In this section we present the results of sampling four of the models for Battle-type events in the district of Dhaka using both Metropolis-Hastings MCMC and Stan. Numerical results about the posterior distributions are presented later; this section is intended to show the results of using a self-written implementation of the Metropolis-Hastings algorithm, so detailed results about modelling can be found in the next section of the report.

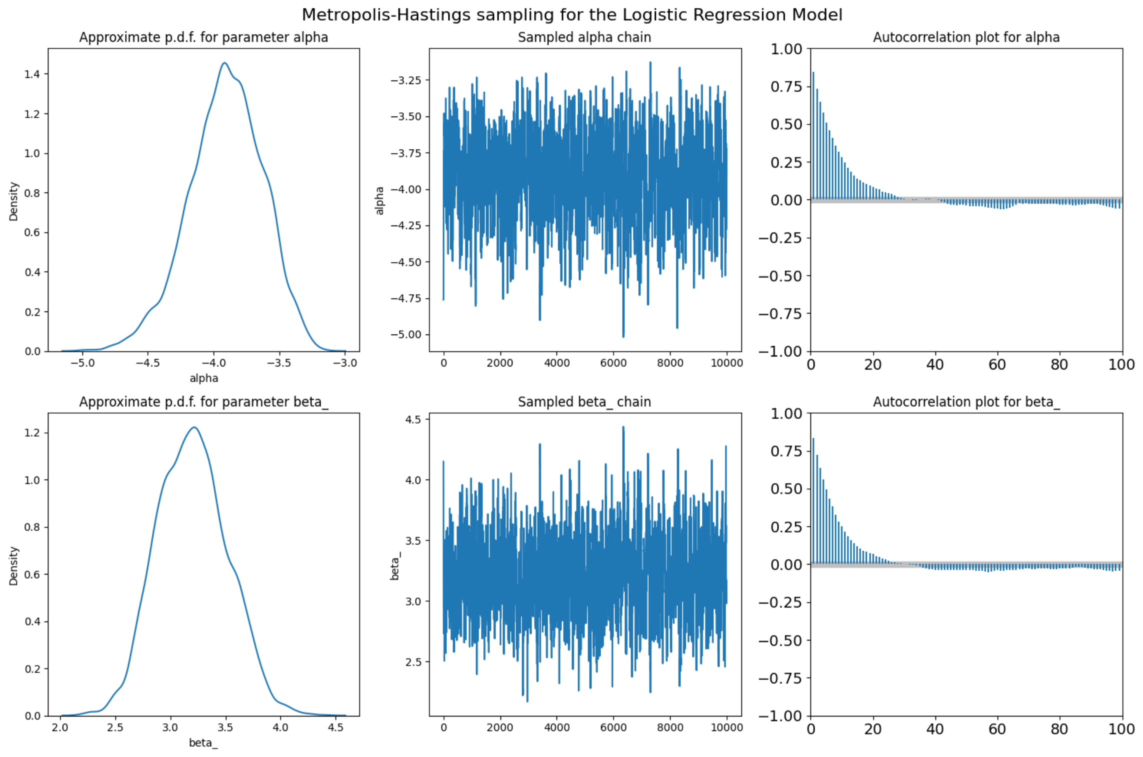

Figure 1. Metropolis Hastings output for a logistic model with two parameters.

3.1.1. *Logistic Regression - two parameters.* For the first Logistic Regression model, 10000 samples were found of α and β, with proposal acceptance rates of 0.3135 and 0.3587 respectively. The effective sample sizes were 492 and 512. As shown in the trace plots, the sampler was able to traverse the parameter space well. Improvements to this could make attempts to improve the proposal variances, but the ones used seemed to work well. The posterior distributions have moved significantly from the standard normal priors, showing the effect of the likelihood on the posterior distribution.

3.1.2. *Logistic Regression - three parameters.* For the second Logistic Regression model, which took as a covariate the fatality indication of the previous weeks, 10000 samples were found for α, β and γ with proposal acceptance rates of 0.3165, 0.3591 and 0.4677 respectively. The effective sample sizes were 671, 691 and 1728, which again could be improved by investigating different proposal variances. The trace plots show that the sampler traversed the parameter space well and the posterior distributions have been affected by the likelihood.

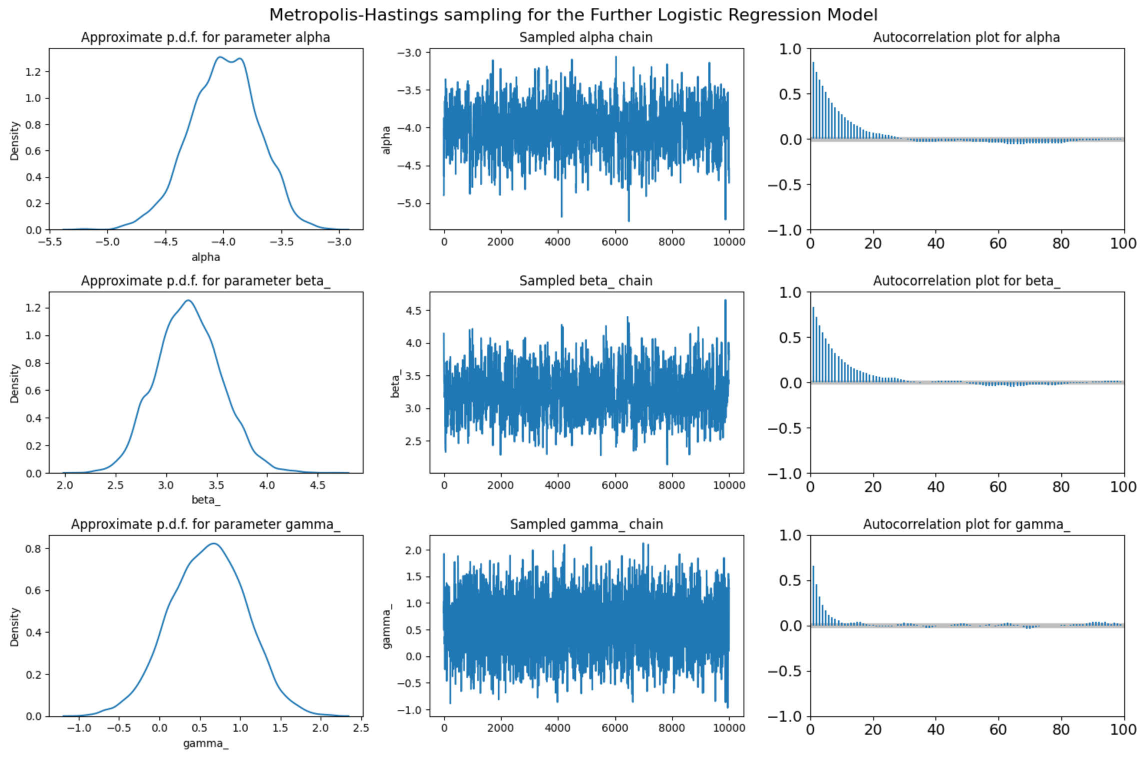

FIGURE 2. Metropolis Hastings output for a logistic model with three parameters.

3.1.3. *Discretisation and Ordering.* The Discretisation and Ordering model was sampled using an adjusted Metropolis Hastings algorithm which transformed a normal proposal with a wide variance from $\mathbb{R}$ to $(0, 1)$ using a logistic link. However, as shown in the trace plots, the sampler was unable to traverse the parameter space for q_2 even with these adjustments, which in turn has affected the sampling of q_1.

3.1.4. *Cumulative Probability.* Of the four models whose sampling is presented here, this is the model that was sampled the most effectively, which is down to its simplicity. The proposal acceptance rate was 0.3308 with an effective sample size of 2002 from 10000 samples. From the trace plot it can be seen that the sampler effectively traversed the parameter space, and the posterior distribution clearly shows the effect of the likelihood.

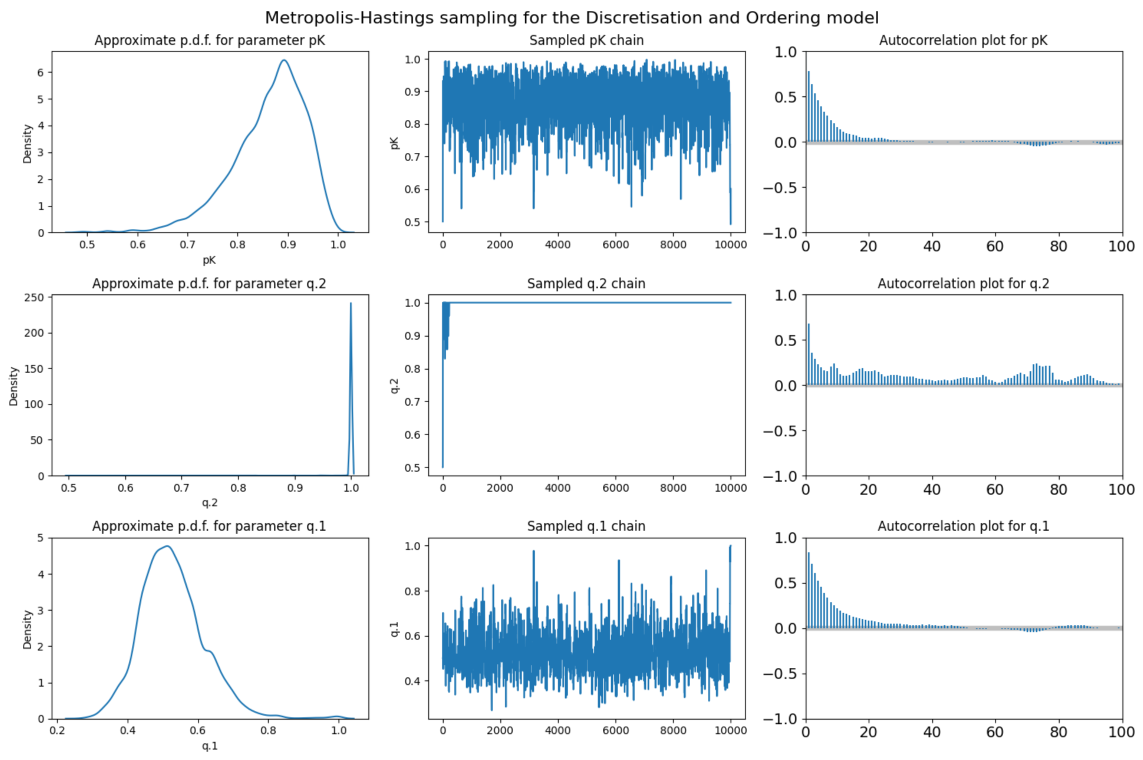

FIGURE 3. Metropolis Hastings output for the Discretisation and Ordering model.

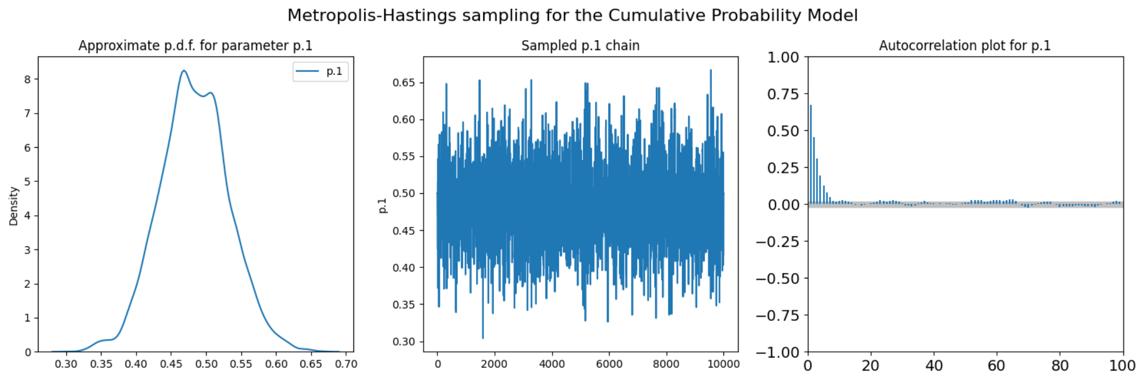

FIGURE 4. Metropolis Hastings output for the Cumulative Probability model.

3.2. **Model Results.** Spatially aggregating over the level-one divisions, four of the models were sampled using Stan. These four models were Logistic Regression, Discretisation and Ordering, Cumulative Probability, and the Null Model. In reality, five models were considered, as the Logistic Regression was modelled both including and excluding weeks which had no events. The graphical results of the analysis are presented below, with further information on parameter means in the following tables.

For some of the logistic regression models, the posterior mean of β is negative, which means that the probability a week has a fatality reduces for higher event counts. This is not something that would be observed in real life, so when finding these posterior means either the top and bottom ten percent of samples could be ignored, or any values of $\beta < 0$ could be ignored. The full results are presented here for completeness.

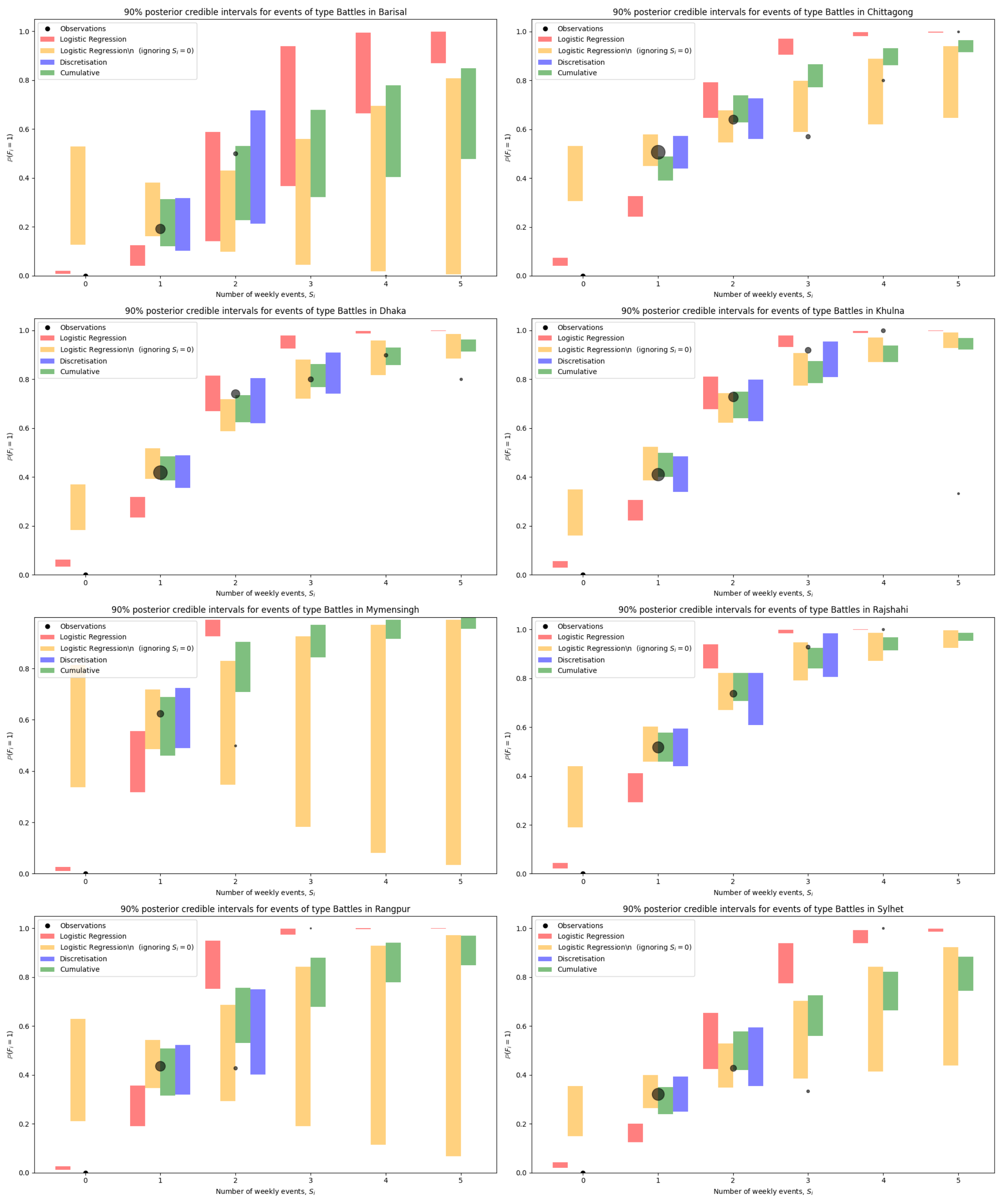

FIGURE 5. 90% credible intervals for Battles in each division. Black points correspond to the observed values in the data set; for $S_i > 0$ the size of the point represents the number of datapoints for that S_i.

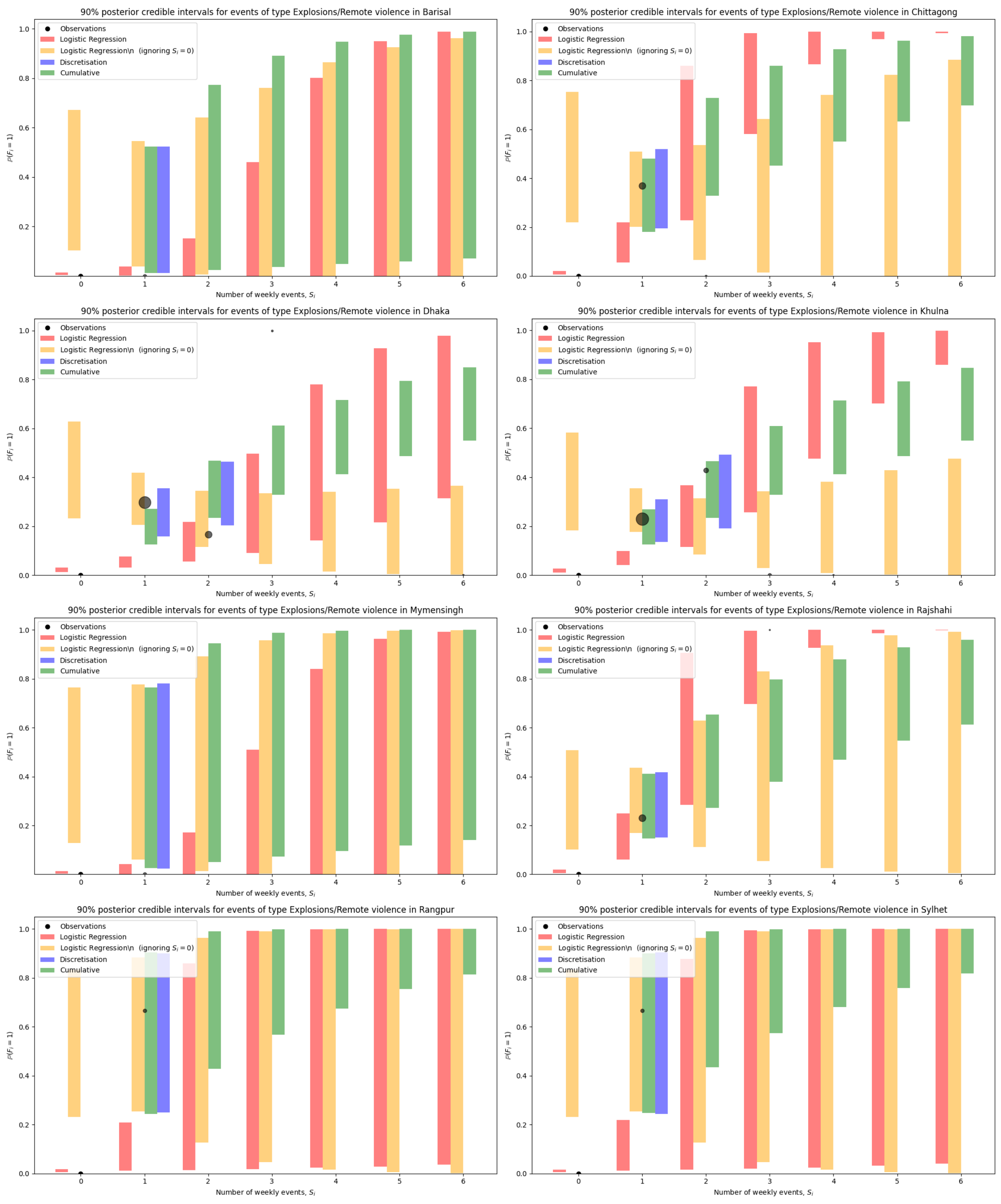

FIGURE 6. 90% credible intervals for Explosions/Remote violence in each division. Black points correspond to the observed values in the data set; for $S_i > 0$ the size of the point represents the number of datapoints for that S_i.

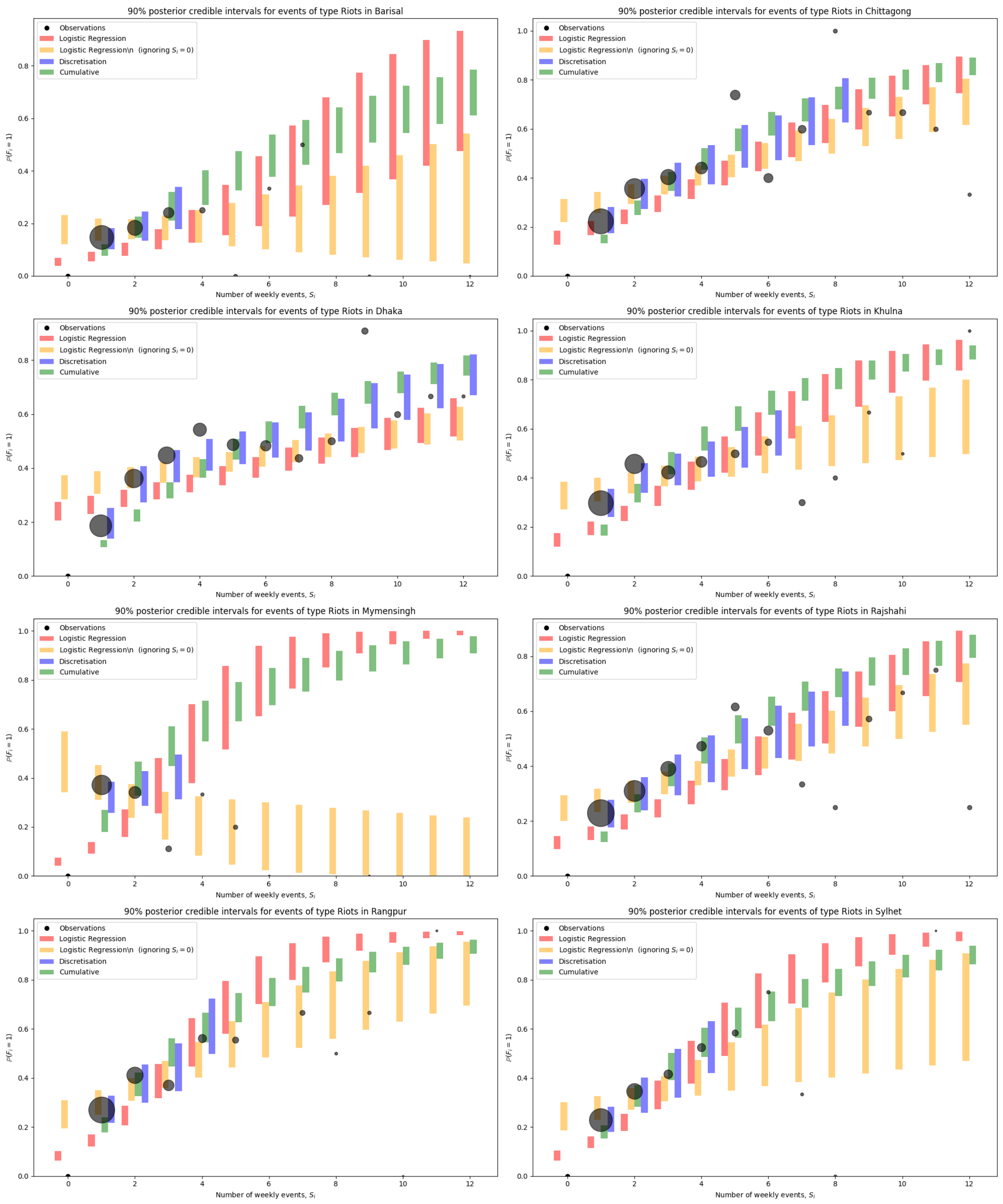

FIGURE 7. 90% credible intervals for Riots in each division. Black points correspond to the observed values in the data set; for $S_i > 0$ the size of the point represents the number of datapoints for that S_i.

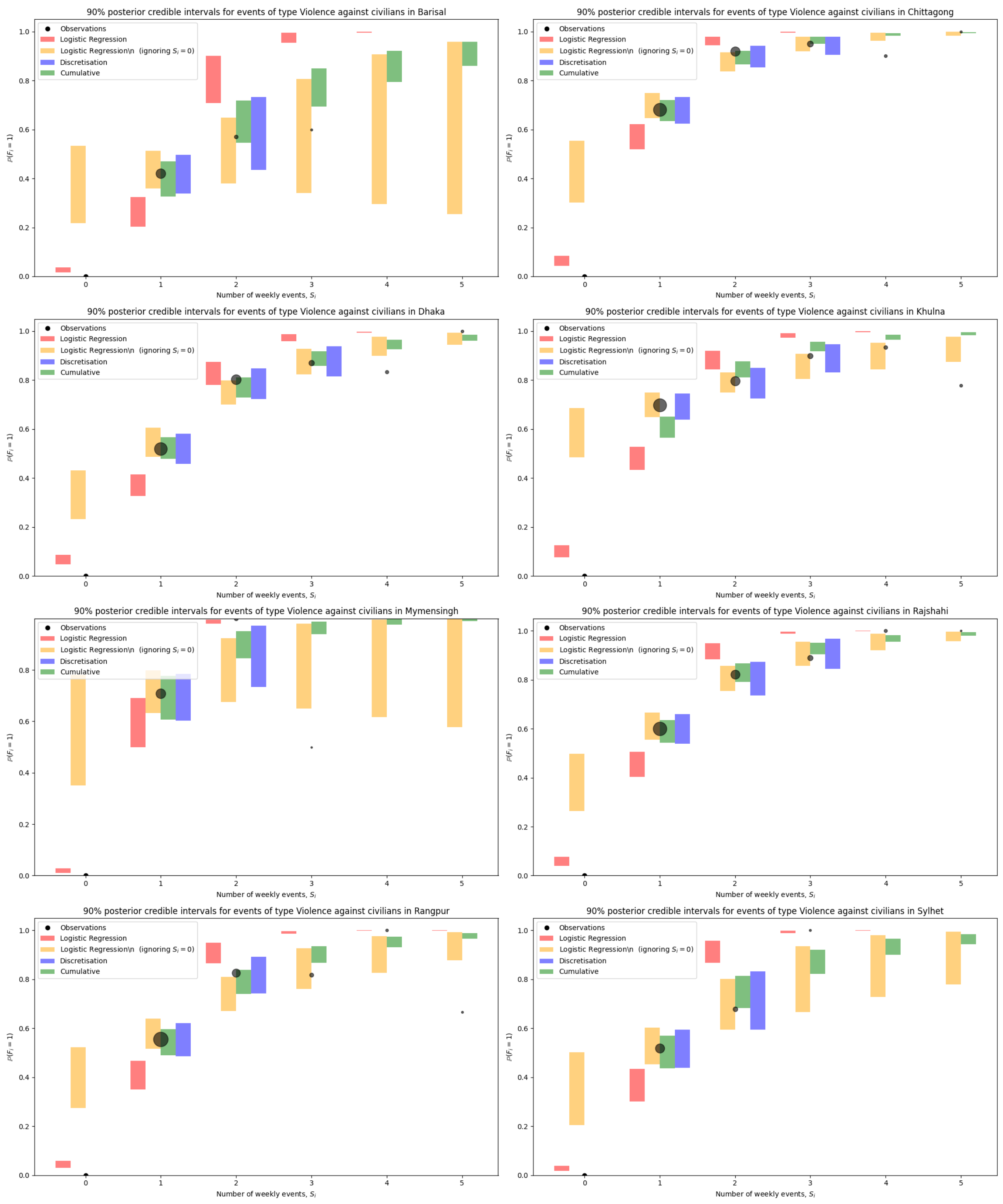

FIGURE 8. 90% credible intervals for Violence against civilians in each division. Black points correspond to the observed values in the data set; for $S_i > 0$ the size of the point represents the number of datapoints for that S_i.

Event	Division	α	β	α (ig. 0)	β (ig. 0)	p - Cumulative	p - Null
Battles	Barisal	-4.34	1.82	-0.91	-0.14	0.21	0.21
	Chittagong	-2.83	1.89	-0.34	0.4	0.44	0.51
	Dhaka	-3.04	2.06	-1.0	0.82	0.44	0.42
	Khulna	-3.17	2.13	-1.14	0.96	0.45	0.41
	Mymensingh	-4.11	3.85	0.41	0.03	0.58	0.62
	Rajshahi	-3.42	2.81	-0.85	0.98	0.52	0.52
	Rangpur	-4.03	3.02	-0.39	0.17	0.41	0.44
	Sylhet	-3.48	1.82	-1.16	0.45	0.29	0.32
Expl.	Barisal	-4.92	-0.04	-0.71	-0.72	0.2	0.2
	Chittagong	-4.39	2.35	-0.07	-0.59	0.32	0.38
	Dhaka	-3.83	0.88	-0.36	-0.47	0.19	0.31
	Khulna	-4.02	1.36	-0.59	-0.47	0.19	0.24
	Mymensingh	-4.92	0.01	-0.36	-0.35	0.33	0.33
	Rajshahi	-4.51	2.6	-1.07	0.16	0.27	0.25
	Rangpur	-4.63	1.76	0.22	0.22	0.6	0.6
	Sylhet	-4.64	1.78	0.22	0.22	0.6	0.6
Riots	Barisal	-2.9	0.35	-1.59	0.02	0.1	0.15
	Chittagong	-1.71	0.28	-1.02	0.16	0.15	0.23
	Dhaka	-1.16	0.13	-0.72	0.08	0.12	0.19
	Khulna	-1.78	0.35	-0.73	0.12	0.19	0.3
	Mymensingh	-2.81	0.75	-0.16	-0.34	0.22	0.38
	Rajshahi	-1.99	0.29	-1.13	0.15	0.14	0.23
	Rangpur	-2.43	0.65	-1.11	0.25	0.21	0.27
	Sylhet	-2.42	0.57	-1.16	0.19	0.18	0.23
V.A.C.	Barisal	-3.64	2.59	-0.58	0.32	0.4	0.42
	Chittagong	-2.74	3.03	-0.3	1.14	0.68	0.68
	Dhaka	-2.66	2.13	-0.73	0.92	0.52	0.52
	Khulna	-2.21	2.13	0.36	0.49	0.61	0.7
	Mymensingh	-3.99	4.39	0.33	0.63	0.69	0.7
	Rajshahi	-2.83	2.64	-0.53	0.98	0.59	0.6
	Rangpur	-3.13	2.76	-0.43	0.75	0.54	0.55
	Sylhet	-3.6	3.04	-0.66	0.77	0.5	0.52

Table 1. Posterior means for model parameters, by event type and division.

Event	Division	p.1	p.2	p.3	p.4	p.5	p.6	p.7	p.8	p.9	p.10	p.11	p.12
Battles	Barisal	0.20	0.42										
	Chittagong	0.51	0.64										
	Dhaka	0.42	0.72	0.83									
	Khulna	0.41	0.72	0.89									
	Mymensingh	0.61											
	Rajshahi	0.52	0.72	0.91									
	Rangpur	0.42	0.56										
	Sylhet	0.32	0.47										
Expl.	Barisal	0.20											
	Chittagong	0.35											
	Dhaka	0.25	0.32										
	Khulna	0.22	0.32										
	Mymensingh	0.33											
	Rajshahi	0.28											
	Rangpur	0.60											
	Sylhet	0.60											
Riots	Barisal	0.14	0.19	0.25									
	Chittagong	0.23	0.33	0.39	0.45	0.53	0.56	0.63	0.72				
	Dhaka	0.19	0.34	0.41	0.45	0.48	0.50	0.53	0.58	0.63	0.66	0.70	0.75
	Khulna	0.30	0.40	0.43	0.47	0.52	0.58						
	Mymensingh	0.32	0.35	0.40									
	Rajshahi	0.23	0.30	0.36	0.42	0.48	0.52	0.57	0.65				
	Rangpur	0.27	0.37	0.44	0.61								
	Sylhet	0.23	0.33	0.41	0.52								
V.A.C.	Barisal	0.42	0.58										
	Chittagong	0.68	0.90	0.95									
	Dhaka	0.52	0.79	0.88									
	Khulna	0.69	0.79	0.89									
	Mymensingh	0.70	0.87										
	Rajshahi	0.60	0.81	0.91									
	Rangpur	0.55	0.82										
	Sylhet	0.52	0.72										

TABLE 2. Posterior means for Discretisation and Ordering parameters, by event type and division

3.3. **Marginal Likelihoods.** The Newton-Raftery iterative estimator was tested with some care. The first potential issue is with the choice of stopping criterion. In our implementation, the iterator terminates as soon as the difference between two successive estimates is lower than some ε. This is not a guarantee that the iterator has converged, if the difference between successive iterations is not monotonically decreasing. However, running the estimator for a much larger number of iterations than required for the termination criterion did not affect the marginal likelihood estimate. The difference between successive estimates was found to be monotonically decreasing in all examples investigated, suggesting that the termination criterion selected is valid. This termination criterion is also computationally inexpensive, making it a suitable choice for this implementation.

A second issue is the behaviour of the estimator as the posterior sample size increases. As the posterior sample size increased from 10 to 500 thousand, the estimate seemed to converge. However, the difference between the estimate with 500000 iterations and that with 10000 was less than 0.01, which was the value used in the stopping criterion, so the sample size can be kept low to increase computation speed without unintended influence of the marginal likelihood.

A third issue is the behaviour of the estimator as the value of δ is increased. The purpose of δ is to control the mixing of samples treated as though they are prior and samples treated as though they are posterior, and represents the mixing between the Monte Carlo and the Harmonic Mean marginal likelihood estimators. A higher value of δ results in a lower variance of the iterative estimator. However, it leads to a higher bias, as a high value of δ represents a higher proportion of posterior samples being treated as prior samples in the estimator. This introduces the issue of a small number of "prior" samples with high posterior density having a large effect on the estimate of the marginal likelihood. We did a small amount of testing and found that for δ not very close to zero the difference in marginal likelihoods between two models remains relatively constant. However, more testing of this should be done as we only investigated this for two models built on fatalities in Dhaka district for Battle-type events.

For each model, event type and division, the model marginal likelihood was calculated using the Newton-Raftery iterative estimator, with $\delta = 0.1$ and stopping criterion parameter $\varepsilon = 0.01$. This allows us to see that the logistic regression model is preferred in most of the divisions for most event types, when weeks with zero events are ignored. The different scales of values for the marginal likelihoods are due to differing numbers of events occurring in each division for each event type; at least one of the divisions saw no fatalities for certain event types which means some of the models are not able to be properly estimated.

The estimated marginal likelihood values indicate that the model that performs best is Logistic Regression, when weeks with no events are ignored. This is especially

Event	Division	Logistic	Log. ig. 0	Disc.	Cumulative	Null	Preferred	Evidence
Battles	Barisal	-25.96	-21.89	-25.67	-21.1	-21.45	Cumulative	0.34
	Chittagong	-179.77	-151.34	-170.63	-153.95	-159.53	Log. ig. 0	2.6
	Dhaka	-163.88	-147.27	-202.85	-145.69	-146.7	Cumulative	1.01
	Khulna	-151.07	-137.82	-206.3	-136.26	-139.19	Cumulative	1.56
	Mymensingh	-37.76	-30.12	-29.9	-31.03	-31.5	Inconclusive	< 0.3
	Rajshahi	-119.69	-106.91	-145.32	-106.4	-107.03	Cumulative	0.51
	Rangpur	-51.37	-44.03	-53.48	-44.05	-44.47	Inconclusive	< 0.3
	Sylhet	-107.83	-94.6	-113.01	-94.63	-96.0	Inconclusive	< 0.3
Expl.	Barisal	-0.03	-1.01	-0.89	-0.88	-0.88	Logistic	0.85
	Chittagong	-19.34	-13.65	-13.92	-14.66	-15.25	Inconclusive	< 0.3
	Dhaka	-48.28	-30.88	-41.44	-34.37	-40.17	Log. ig. 0	3.49
	Khulna	-44.1	-33.35	-43.04	-34.87	-36.48	Log. ig. 0	1.52
	Mymensingh	-0.01	-0.57	-0.54	-0.52	-0.54	Logistic	0.51
	Rajshahi	-16.07	-16.2	-16.0	-15.17	-15.21	Inconclusive	< 0.3
	Rangpur	-6.84	-2.2	-2.23	-2.25	-2.24	Inconclusive	< 0.3
	Sylhet	-6.83	-2.2	-2.25	-2.24	-2.24	Inconclusive	< 0.3
Riots	Barisal	-133.99	-121.34	-197.09	-125.56	-138.53	Log. ig. 0	4.22
	Chittagong	-293.36	-281.61	-874.81	-286.59	-326.78	Log. ig. 0	4.98
	Dhaka	-346.26	-338.66	-1480.86	-352.03	-419.42	Log. ig. 0	7.6
	Khulna	-277.24	-255.04	-638.46	-270.92	-315.79	Log. ig. 0	15.88
	Mymensingh	-138.83	-101.49	-177.64	-116.74	-138.16	Log. ig. 0	15.25
	Rajshahi	-247.79	-232.1	-754.01	-240.01	-281.56	Log. ig. 0	7.91
	Rangpur	-203.16	-182.25	-331.46	-185.44	-196.92	Log. ig. 0	3.2
	Sylhet	-210.19	-191.7	-347.46	-193.62	-204.67	Log. ig. 0	1.91
V.A.C.	Barisal	-92.56	-78.8	-92.22	-78.82	-79.45	Log. ig. 0	0.02
	Chittagong	-181.04	-163.88	-213.96	-162.96	-163.56	Cumulative	0.6
	Dhaka	-207.18	-189.44	-260.29	-188.17	-189.19	Cumulative	1.02
	Khulna	-222.01	-181.77	-230.67	-188.77	-199.52	Log. ig. 0	7.0
	Mymensingh	-51.27	-44.41	-48.7	-44.42	-44.61	Inconclusive	< 0.3
	Rajshahi	-186.53	-167.83	-219.29	-167.34	-168.14	Cumulative	0.48
	Rangpur	-153.34	-132.87	-150.88	-132.17	-132.75	Cumulative	0.58
	Sylhet	-106.36	-94.61	-107.77	-94.18	-94.67	Cumulative	0.42

Table 3. Estimated marginal likelihoods for models, by event type and division.

true for event types which occur more frequently, such as Explosions/Remote violence and Riots. For other event types, the Cumulative Probability model performs as well as Logistic Regression with the better of the two models being inconclusive, suggesting that Battle and Violence against civilians events can be treated as independent, as long as the likelihood function covers all event types. In none of the situations was the null hypothesis the preferred model, showing that in general conflict events are not independent of one another in the way that they cause fatalities.

4. Discussion

By considering different models for the occurrence of a fatality, this report enables conclusions to be drawn about how the data should be handled when looking to model this. The main conclusion is that modelling should ignore weeks of zero events, as these can be assumed to have zero fatalities. Furthermore, it is evident that different models work better for different event types. Where the number of events in a given week has a larger range, logistic regression provides a better model than discretisation of the data points.

There is also evidence that the probability of a fatality from an individual event cannot be taken as independent from other events happening in that week. We have shown evidence that the probability that an individual event causes a fatality is not independent of the number of events in a week; in none of the 32 different event type/division combinations was the true null model preferred. Furthermore, for only one event type was the Cumulative model, which implies an independence between event types in its likelihood function, preferred for a majority of divisions.

The increasing weakness of the Discretisation and Ordering model for greater ranges of event counts is an indication that the discretisation procedure employed was not adequate. A direction that further study could go is to create and implement a more robust binning procedure to improve the modelling in this regard.

For some of the models, there was a lack of data, with some of the divisions experiencing no fatalities over the entire dataset for some event types. This makes modelling difficult. Furthermore, some of the events had a distinctly lower probability of causing a fatality, and some of the divisions saw a lower number of fatalities. This suggests that a useful direction of further study would be to have factor covariates for each division and each event type. Alternatively, some spatial metric could be used, such as the area of the division or its distance from large population centres. In addition, demographic covariates could be considered, such as the population or the population density of the division. This has not been investigated at all here, but intuitively it seems likely that the population of a region will affect the way that conflict events cause fatalities. This would develop the analysis presented here and enable further information to be found about the causation of fatalities by conflict events.

References

[1] ACLED. *Armed Conflict Location and Event Data Project (ACLED) Codebook*. 2019. URL: `https://www.acleddata.com/wp-content/uploads/dlm_uploads/2019/04/General-User-Guide_FINAL.pdf` (visited on 08/31/2023).

[2] Feargus J. Ball. *ACLED Fatality Modelling*. 2023. URL: `https://github.com/FeargusB/ACLED_fatality_modelling` (visited on 09/29/2023).

[3] Feargus J. Ball. *Oxford ACLED Internship*. 2023. URL: `https://github.com/FeargusB/Oxford_ACLED_Internship` (visited on 09/29/2023).

[4] Raiha Tui Taura Browning. "Bayesian Approaches for Modelling Discrete-Time Self-Exciting Processes and Their Applications". PhD thesis. Queensland University of Technology, 2023.

[5] Michael A. Newton and Adrian E. Raftery. "Approximate Bayesian Inference with the Weighted Likelihood Bootstrap". In: *Journal of the Royal Statistical Society, Series B (Methodological)* 56.1 (1994), pp. 3–48.

[6] Stan Development Team. *Stan Modeling Language User's Guide and Reference Manual*. Version 2.32. 2023. URL: `https://mc-stan.org` (visited on 09/03/2023).

Appendix A. Metropolis Hastings Algorithms in Python

A.1. **Logistic Regression.** The following code is an implementation of the Metropolis Hastings algorithm in Python for a logistic regression model.

```
import numpy as np
import pandas as pd

def logit_mh_mcmc(responses, observed_covariates,
                  parameter_names, initial_parameters, parameter_stepsizes,
                  hyperparameters, proposal_dists,
                  target_dist, iterations=10000):
    '''
    A Metropolis-Hastings algorithm for a logistic model with
        a free number of parameters
    Assumes a symmetric proposal distribution for simplicity

    Arguments:
    responses - a vector of N observed responses
    observed_covariates - an array of N observed covariate values (rows)
        for M covariates (columns)
    parameter_names - a list of K string parameter names
    initial_parameters - a list of K float initial parameter values
```

```
parameter_stepsizes - a list of K float initial parameter stepsizes
parameter_prior_parameters - a list of K float parameters for
    the prior distributions
proposal_dists - a list of K functions, representing proposal
    distributions, taking the current parameter value
    and a stepsize as arguments
target_dist - a function to be approximated by the MH MCMC algorithm
iterations - an integer number of iterations to perform

Returns:
a pandas data frame with K columns of iterations+1 sampled parameter
    values from the target distribution
'''

## INITIAL SETUP

# find the number of parameters
num_params = len(parameter_names)

# create a matrix to store the sampled values
params_chain = np.zeros((iterations+1, num_params))
params_chain[0,:] = initial_parameters

# initialise parameters and acceptance rates
current_params = initial_parameters
param_acceptance_count = np.zeros(num_params)

## MAIN CHAIN SECTION

# perform the required number of iterations
for i in range(iterations):
    # update the parameters sequentially
    for idx, param in enumerate(current_params):
        # propose the same parameters, changing the appropriate parameter only
        # this is a messy method but I'd not realised that lists aren't immutable
        proposed_param = proposal_dists[idx](param, parameter_stepsizes[idx])
        proposed_params = np.zeros(num_params)
        proposed_params[:idx] = current_params[:idx]
        proposed_params[idx] = proposed_param
        if idx != num_params-1:
                proposed_params[idx+1:] = current_params[idx+1:]

            # find the target log densities under the current and proposed param
            log_post_current = target_dist(current_params, hyperparameters,
                responses, observed_covariates)
            log_post_proposed = target_dist(proposed_params, hyperparameters,
                responses, observed_covariates)
            acceptance_ratio = log_post_proposed - log_post_current

            if acceptance_ratio > np.log(np.random.rand()):
                # if accepted, record the new parameter and update the parameter
                params_chain[i+1, idx] = proposed_param
                current_params = proposed_params
                param_acceptance_count[idx] += 1

            else:
                params_chain[i+1, idx] = param

    for idx, param in enumerate(parameter_names):
        print(f'Proposal acceptance rate of {param_acceptance_count[idx]/iteration

    params_chain_frame = pd.DataFrame(params_chain, columns = parameter_names)

    return params_chain_frame
```

A.2. **Discretisation and Ordering.** The following code is an implementation of the Metropolis Hastings algorithm in Python for the Discretisation and Ordering model, with proposed parameters transformed between parameter spaces.

```
import numpy as np
import pandas as pd

def mod4_mh_mcmc(iterations, stepsizes, target, initial_params, param_names, hyper
    '''
    Model 4 (stick-breaking) sampling algorithm

    Transforms a normal proposal on the real line to a proposal in [0,1] to
    reflect the parameter space

    Arguments:
    iterations - an integer number of iterations to perform
```

```
stepsizes - a list of stepsizes for each parameter
target - the target distribution
initial_params - the initial state of the chain
param_names - names for the parameters to form the data frame
hyperparams - hyperparameters for the Beta prior distributions
observations - an array of observed event counts
responses - an array of binary response variables

Returns a data frame of the sampled parameters
'''
num_params = len(initial_params)
samples = np.zeros((iterations+1, num_params))
samples[0,:] = initial_params
current_params = initial_params
acceptance_count = np.zeros(num_params)

for iter in range(iterations):
    for idx, param in enumerate(current_params):
        # propose the same parameters, changing the appropriate parameter only
        # this is a messy method but I'd not realised that lists aren't immutable
        # propose in the proposal space, transform to the parameter space
        current_unconstrained = inv_logistic(param)
        proposed_unconstrained = current_unconstrained +
            np.random.normal(scale=abs(np.log(stepsizes[idx])))
        proposed_param = logistic(proposed_unconstrained)
        proposed_params = np.zeros(3)
        proposed_params[:idx] = current_params[:idx]
        proposed_params[idx] = proposed_param
        if idx != num_params-1:
            proposed_params[idx+1:] = current_params[idx+1:]

        # find the target log densities under the current and proposed parameters,
        log_post_current = target(current_params, responses,
                                  observations, hyperparams)
        log_post_proposed = target(proposed_params, responses,
                                   observations, hyperparams)
        acceptance_ratio = log_post_proposed - log_post_current

        if acceptance_ratio > np.log(np.random.rand()):
            # if accepted, update the parameters
```

```
                current_params = proposed_params
                acceptance_count[idx] += 1
                samples[iter+1, idx] = proposed_param

            else:
                samples[iter+1, idx] = param

    for idx, param in enumerate(param_names):
        print(f'Proposal acceptance rate of {acceptance_count[idx]/iterations} for

    return_frame = pd.DataFrame(samples, columns = param_names)
    return return_frame
```

A.3. **Cumulative Probability.** The following code is an implementation of the Metropolis Hastings algorithm in Python for the Cumulative probability model, once again transforming the proposed parameters between parameter spaces.

```
def mod6_mh_mcmc(iterations, stepsize, target, initial_param, param_name, hyperpar
    '''
    Model 6 sampling algorithm

    Transforms a normal proposal on the real line to a proposal in [0,1] to
    reflect the parameter space
    Specifically for use with the one dimension of this model

    Arguments:
    iterations - an integer number of iterations to perform
    stepsizes - the stepsize for the parameter
    target - the target distribution
    initial_param - the initial state of the chain
    param_name - the name of the parameter
    hyperparams - hyperparameters for the Beta prior distribution
    observations - an array of observed event counts
    responses - an array of binary response variables

    Returns a data frame of the sampled parameters
    '''
    samples = np.zeros(iterations+1)
    samples[0] = initial_param
    current_param = initial_param
    acceptance_count = 0
```

```
for iter in range(iterations):
        # propose the same parameters, changing the appropriate parameter only
        # this is a messy method but I'd not realised that lists aren't immutable
        # propose in the proposal space, transform to the parameter space
    current_unconstrained = inv_logistic(current_param)
    proposed_unconstrained = current_unconstrained +
        np.random.normal(scale=abs(np.log(stepsize)))
    proposed_param = logistic(proposed_unconstrained)

        # find the target log densities under the current and proposed parameters,
    log_post_current = target(current_param, responses,
                              observations, hyperparams)
    log_post_proposed = target(proposed_param, responses,
                               observations, hyperparams)
    acceptance_ratio = log_post_proposed - log_post_current

    if acceptance_ratio > np.log(np.random.rand()):
            # if accepted, update the parameters
        current_param = proposed_param
        acceptance_count += 1
        samples[iter+1] = proposed_param

    else:
        samples[iter+1] = current_param

print(f'Proposal acceptance rate of {acceptance_count/iterations} for parameter {par

return_frame = pd.DataFrame(samples, columns = param_name)
return return_frame
```