A BAYESIAN MODEL FOR NEWS REPORTING

NICHOLAS A. LINES

Abstract. Learning information through unreliable observers is difficult. In news creation and consumption we encounter problems of this sort, in which information from various observers of differing awareness levels must be fused and processed. We introduce a model for news-related information digestion and consider several methods for weighting or eliminating sources by credibility levels. We implement this in Python and compare PyMC3's approximate parameter inference results with tailored estimators, and discuss variations of this model.

Contents

List of Figures

Date: May 12, 2022.

The author would like to thank Dr. Woolf for advising this project, as well as the author's friend Rod Gomez for his advice regarding PyMC3.

This paper is part of the author's final project for EN.625.692.81.SP22 Probabilistic Graphical Models, for Johns Hopkins University.

1. Modeling Information Collation and Reporting

Anyone with a social media account and internet connection can expect to be bombarded by information, opinions, and news. On an individual level, accumulating important news information is overwhelmingly done through passive online activities. Five years ago the Pew Research Center found that 67% of Americans learned about news on social media, and this year they added that 23% got their news from podcasts [2][10]. Interestingly, this movement toward digitally-fed news has also brought on increased concern about legitimacy: Pew also asked their 2021 respondents if they would support Federal internet censorship, and 48% said yes, compared with 39% in 2018. How to use existing systems to find credible information is a vital question each individual must answer. More generally, any actor involved in distilling news data into factual reports must perform a similar credibility analysis using a network of dubious sources. Self-filtering news data is a difficult challenge, involving both bias in the sources consulted, bias introduced by those sources, and bias in credibility assessment introduced by the actor.

In this paper we propose a probabilistic graphical framework with which we can explore information fusion tasks. This proposed approach is elementary, but it has the potential to develop as a tool to compliment human credibility assessment of news information sources. We note that the goal of our investigation is to improve in recognition of net credibility of sources in terms of *consensus*, not *truth*. We will model a general information fusion task and demonstrate several approaches for reducing input from untrusted sources, and discuss variations of this problem and its applications.

2. Related Work

In recent years, efforts to understand how news information propagates in a network have largely focused on social networks. In [12] we find this problem approached in a way that explicitly admits that ground truth is unavailable, which inspired our consensus-based approach. In many applications, including digestion of news information, ground truth is unavailable, and consensus of a class of observers or experts is considered its best approximation. Consensus-based modeling of one kind or other fills an increasingly important role in medical research and practice [1], blockchain protocols [9], and other multi-agent systems [8].

Social media analysis frequently turns to consensus-based approaches with success. This is evident in applications such as trust exploration [3], and efforts to characterize social dynamics to gauge adoption of new technology, like in [5]. We emphasize that consensus-based approaches have great potential for misinterpretation and abuse in social (particularly news analysis) settings. Discarding disagreements from consensus in these settings may ignore early warnings, marginalized social groups, or expert opinions, and may obscure the true evolution of opinion. We urge that these methods be applied with caution and intentionally used to filter noise under careful supervision.

Analysis of how credibility is changed within a network of communicating agents is often connected to the concept of belief propagation, a term used both for the idea in general of spreading beliefs, but also specifically applied to the sum-product algorithm for this purpose pioneered by Judea Pearl. A useful survey on this subject is provided in [11].

Storing and using knowledge in a probabilistic graphical framework is usually the realm of knowledge graphs. For a fairly thorough source that introduces knowledge graphs and their applications, we refer to [4]. Another name this problem falls under is "information fusion," which has begun to be applied to text-based problems as well (see [6] and [7]).

3. A Bayesian Network Model

To motivate our model and the related inference problems, we will consider the following common situation. A news analyst is tasked with understanding and summarizing news over time, covering several news topics. This analyst is disconnected from the actual chain of events of interest, and relies entirely on a set of first-hand observers to pass on reports about these topics. However, not all observers are alike. Some are more apt to report information than others. Some are more aware of a particular news topic than others. We will make the (significant) assumption that a consensus of observers represents accepted fact for the analyst. The analyst's principal task is to identify observations held in consensus and report these. Secondarily, the analyst wishes to "weed out" observations that are unhelpful. If an observer consistently disagrees with the general consensus on a topic, that observer does not need to be surveyed on that topic, or perhaps their response should be weighted low, proportionate in some way to the trust they have accrued.

We will now introduce a Bayesian network that generatively models this situation with certain constraints. In particular we will only consider binary facts related to each topic.

Definition 1. ***Reporter Model A****: Let n_j be the number of news topics of interest, n_i the number of observers, and n_t the number of timesteps in the modeled timeframe. For each topic $j = 1, \ldots, n_j$ we assume that news topic has a bias*

$$\delta_j \sim \ Uniform\,(0,1)$$

that indicates preference toward a fact value of 1. At each time step $t = 1, \ldots, n_t$ a new fact is generated

$$f_{j,t} \sim \ Bernoulli\,(\delta_j)\,.$$

Each observer $i = 1, \ldots, n_i$ has a bias that indicates how willing they are to report their observations about this topic,

$$\varepsilon_{i,j} \sim \ Uniform\,(0,1)\,,$$

and an awareness level for this topic,

$$\beta_{i,j} \sim \ Uniform\,(0,1)\,.$$

At each timestep they draw a decision about whether to report

$$\tau_{i,j,t} \sim \ Bernoulli\,(\varepsilon_{i,j})$$

and draw to decide if they are aware of the current fact

$$\alpha_{i,j,t} \sim \ Bernoulli\left(\beta_{i,j}\right).$$

We also generate random noise

$$r_{i,j,t} \sim \ Bernoulli\left(\frac{1}{2}\right).$$

The observer then computes

$$o_{i,j,t} = \tau_{i,j,t}\left(\alpha_{i,j,t} f_{j,t} + (1 - \alpha_{i,j,t} r_{i,j,t})\right) + 2\left(1 - \tau_{i,j,t}\right)$$

which is their observation passed on to the analyst. This means that if the observer is aware and willing to report, they pass on the true fact. If they are aware but unwilling to report, they pass on the value 2*, which means nothing was reported. If they are willing to report but unaware of the fact, they report the random noise.*

The analyst at each time step computes the vote $b_{j,t} = \operatorname{mode}\left(\{o_{i,j,t} : o_{i,j,t} \neq 2\}\right)$*, which represents the consensus for fact* $f_{j,t}$*. This model is presented as a plate diagram in Figure 1.*

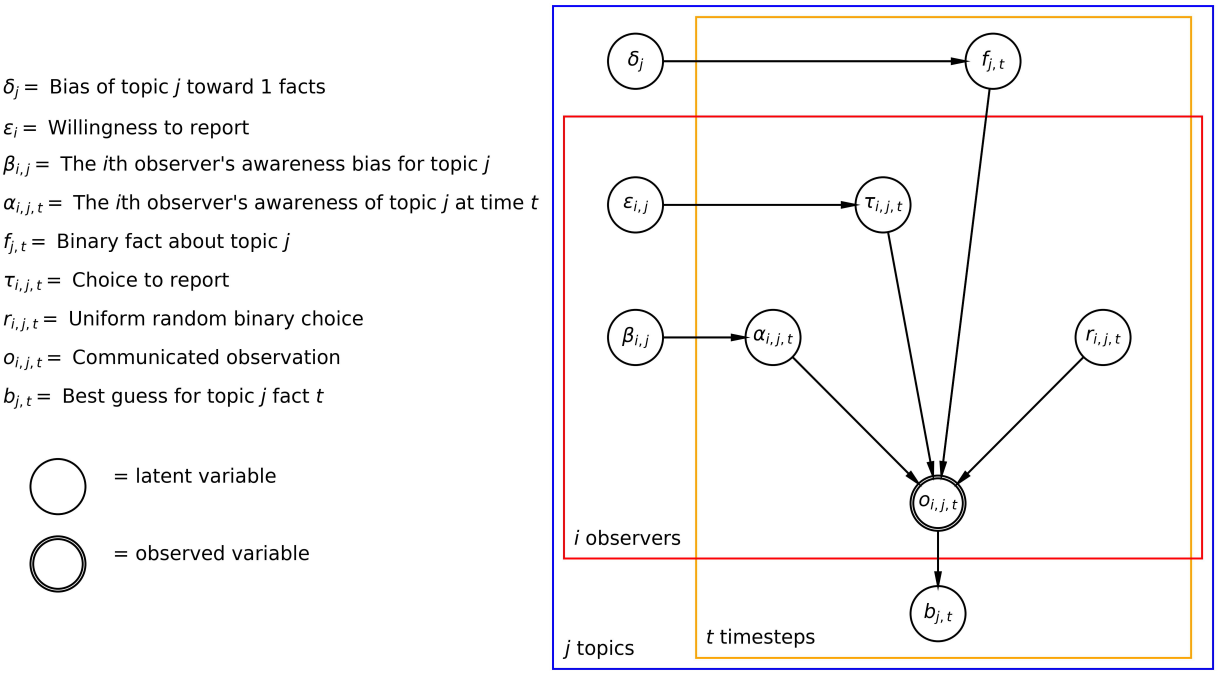


FIGURE 1. Reporter Model A

The deterministic variable $o_{i,j,t}$ accurately models the actual decision to report that each observer must make, but it also presents certain inference challenges, as we will discuss later. This model makes some strict assumptions we should review. The uniform priors for the observer parameters and the topic bias are uninformative, and represent a first investigation. More informative distributions might be available in certain circumstances. We also note that the problem of inferring $\beta_{i,j}$ becomes more difficult if $\left|\delta_j - \frac{1}{2}\right|$ grows too small, since the $f_{j,t}$ input will become indistinguishable from the random noise $r_{i,j,t}$.

We are interested first in the quality of $b_{j,t}$ as an estimator for $f_{j,t}$. Second, we wish to recover or at least bound the awareness biases $\beta_{i,j}$ so we may downweight or ignore unaware observers in the future. The task of inferring $\varepsilon_{i,j}$ is of minimal interest but happens to be particularly simple.

One of the principal advantages to this model is that it breaks easily into independent subgraphs. For example, all variables associated with each topic j are independent of all other topic's variables, so we may perform inference within each topic network separately.

4. Parameter Estimation Via Particle Methods in PyMC3

Before we make any subtle approach at parameter recovery, we consider how well a blunt particle-based parameter estimation method can do. Using PyMC3, we apply the No U-Turns Sampler (NUTS), a Hamiltonian Monte Carlo approach to estimate $\varepsilon_{i,j}, \beta_{i,j}$ and δ_j, while utilizing the Binary Gibbs Metropolis Sampler (BGMS) to recover $f_{j,t}, \alpha_{i,j,t}$, and $\tau_{i,j,t}$. The details are shown in Appendix C. For this experiment, we set the number of observers to $n_i = 20$, the number of timesteps to $n_t = 100$, and restrict our review to a single topic, $n_j = 1$. We ran 1000 samples, with 1000 burn-in (tuning) time steps, which took roughly 1.5 hours on a 4-core Linux machine. The results are summarized in Figure 2 below.

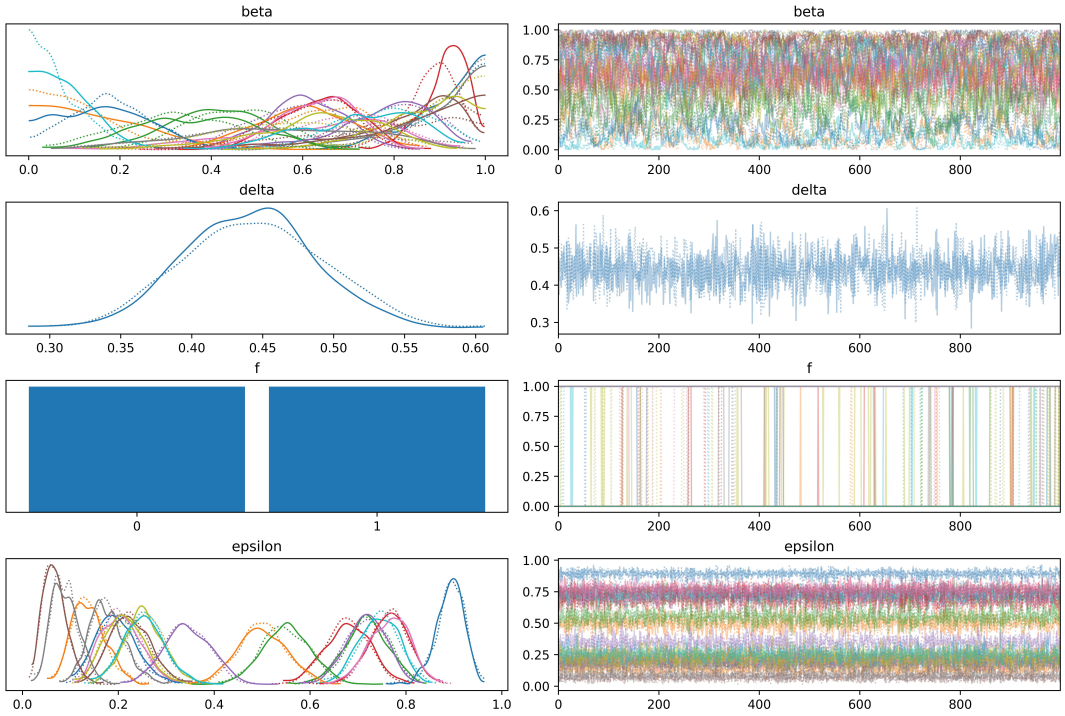


Figure 2. The results of the PyMC3 experiment recovering parameters for Model A. The plots on the left show the empirical densities as histograms, and the plots on the right show values sampled.

These estimations for $f_{j,t}$ are 98% accurate, but the estimates for delta are poor (a mean of $\hat{\delta} = 0.44$ compared to $\delta = 0.35$). Importantly, the estimates for $\hat{\beta}_{i,j}$ performed fairly well. The mean $\hat{\beta}_{i,j}$ values and true $\beta_{i,j}$ are shown in Figure 3.

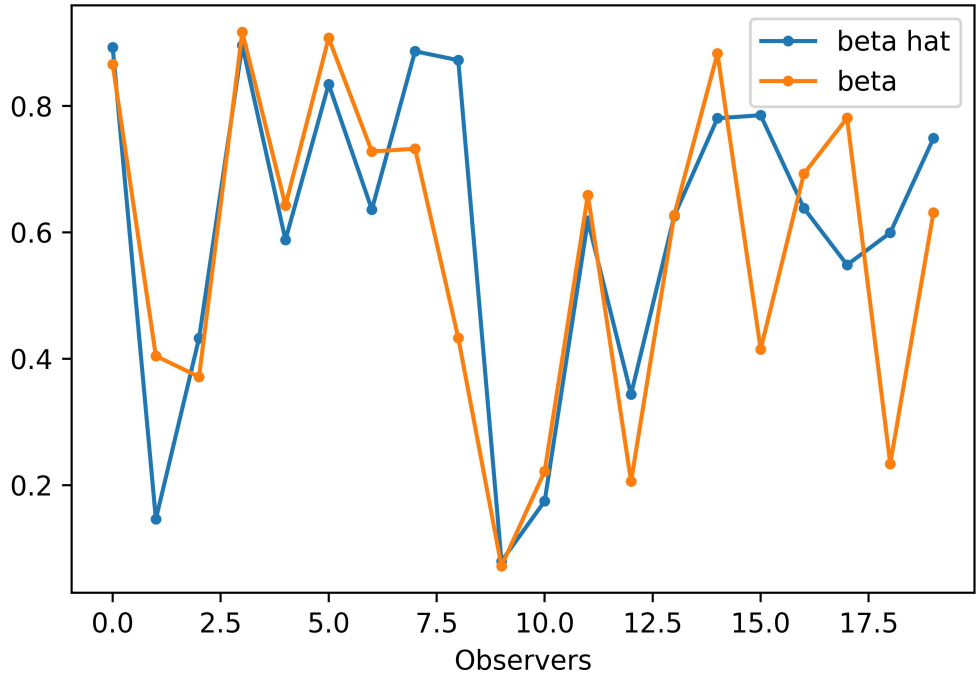


FIGURE 3. Comparison of the mean $\hat{\beta}$ values to the true β values over observers i, from the PyMC3 experimental results.

This particle-based approach represents a baseline for approximate inference that does not take advantage of the details of the model, and we will see that we can improve upon it.

An interesting limitation of PyMC3 is shown in this demonstration: it cannot model observations from deterministic variables. In our case $o_{i,j,t}$ is deterministic in that it is entirely determined as a logical combination of stochastic variables. PyMC3 cannot sample from this, so an unattractive stopgap is commonly used in such cases. We define a new random variable

$$X_{i,j,t} \sim \ Normal\left(\mu = o_{i,j,t}, \sigma = 0.01\right),$$

which we can sample from. This introduces minimal variance, and overcomes the obstacle.

5. Improved Estimators

Let's consider how we could approximate $\beta_{i,j}$ using the assumed distributions. We will begin by simplifying our view of the model incrementally. First, we consider only a single topic's information network, as shown in Figure 4. Next we consider only the contributions of a single observer $\bar{\imath}$, shown in Figure 5.

Finally, we recognize that under these conditions it is easy to filter out the "no report" observations. If we condition on $\tau_t = 1$ we restrict ourselves to a review of

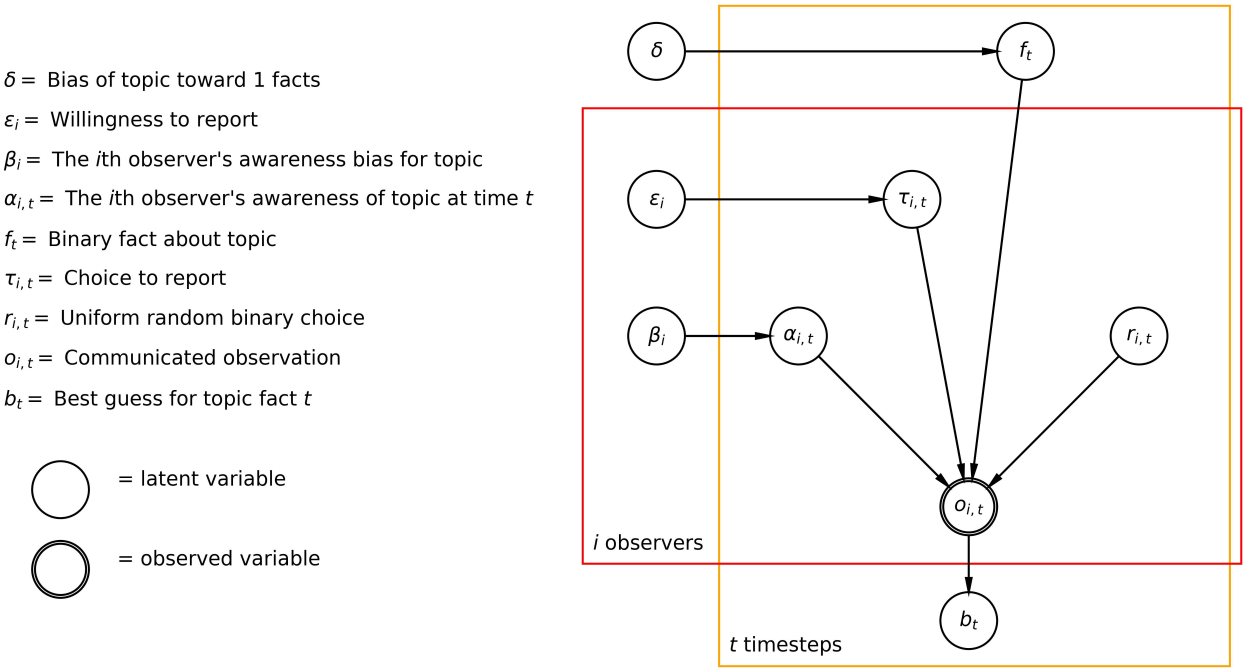


FIGURE 4. Reporter Model A shown for a single topic j.

o_t	α_t	f_t	r_t	$P(\alpha_t)$	$P(f_t)$	$P(r_t)$
0	1	0	0	β	$1-\delta$	0.5
1	1	1	0	β	δ	0.5
1	1	1	1	β	δ	0.5
0	1	0	1	β	$1-\delta$	0.5
1	0	1	1	$1-\beta$	δ	0.5
1	0	0	1	$1-\beta$	$1-\delta$	0.5
0	0	1	0	$1-\beta$	δ	0.5
0	0	0	0	$1-\beta$	$1-\delta$	0.5

TABLE 1. Truth table for computing o_t

the informative reports from this observer, i.e.

$$\{o_t : o_t \neq 2\},$$

and we can write

$$o_t = \alpha_t f_t + (1-\alpha_t)\, r_t.$$

This simplification is shown in Figure 6.

We may comprehensively describe the probability distribution for o_t under these conditions using the probability table, Table 1, since this depends only on binary variables. We recall that the parameter of each Bernoulli random variable represents the probability that variable equals one.

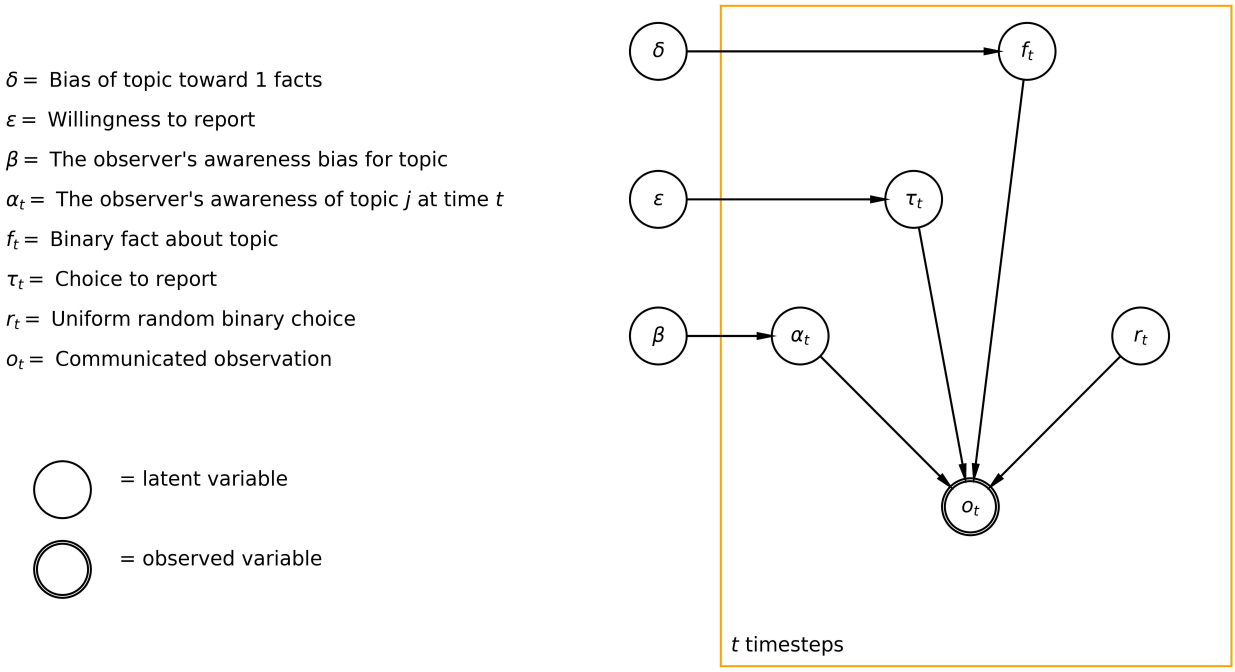


FIGURE 5. Reporter Model A reduced to a single observer and single topic.

Each of β, δ, and r_t are mutually independent variables, so their joint probability is the product of their marginal probabilities. By summing up the joint probabilities of combinations that lead us to $o_t = 1$ we have

$$\begin{aligned} P(o_t = 1) &= 0.5(\beta_j\delta + \delta - \beta\delta + 1 - \beta_t - \delta + \beta_t\delta + \beta_t\delta) \\ &= 0.5(-\beta_t + \beta_t\delta + 1) \end{aligned}$$

We would reach the same result with much less work using conditional probabilities (factors), since

$$\begin{aligned} P(o_{\bar{\imath},j,t} = 1) &= P(o_{\bar{\imath},j,t} = 1 \mid a_{\bar{\imath},j,t} = 0)P(a_{\bar{\imath},j,t} = 0) + P(o_{\bar{\imath},j,t} = 1 \mid a_{\bar{\imath},j,t} = 1)P(a_{\bar{\imath},j,t} = 1) \\ &= 0.5(1 - \beta_{\bar{\imath},j}) + \delta_j\beta_{\bar{\imath},j}. \end{aligned}$$

Solving for β gives us

$$\beta_{\bar{\imath},j,t} = \frac{2P(o_{\bar{\imath},j,t} = 1) - 1}{2\delta_j - 1}.$$

We approximate

$$\delta_j \approx \hat{\delta_j} = \frac{\#\{o_{i,j,t} : o_{i,j,t} = 1\}}{\#\{o_{i,j,t} : o_{i,j,t} \neq 2\}},$$

using the sample mean of the informative observations to approximate the probability that a generated fact equals 1 (an extension of the approximation $b_{j,t} = \hat{f}_{j,t} \approx f_{j,t}$). If we have computed $b_{j,t}$, however, we already have the observed mean

$$\hat{\delta_j} = \bar{b_{j,t}},$$

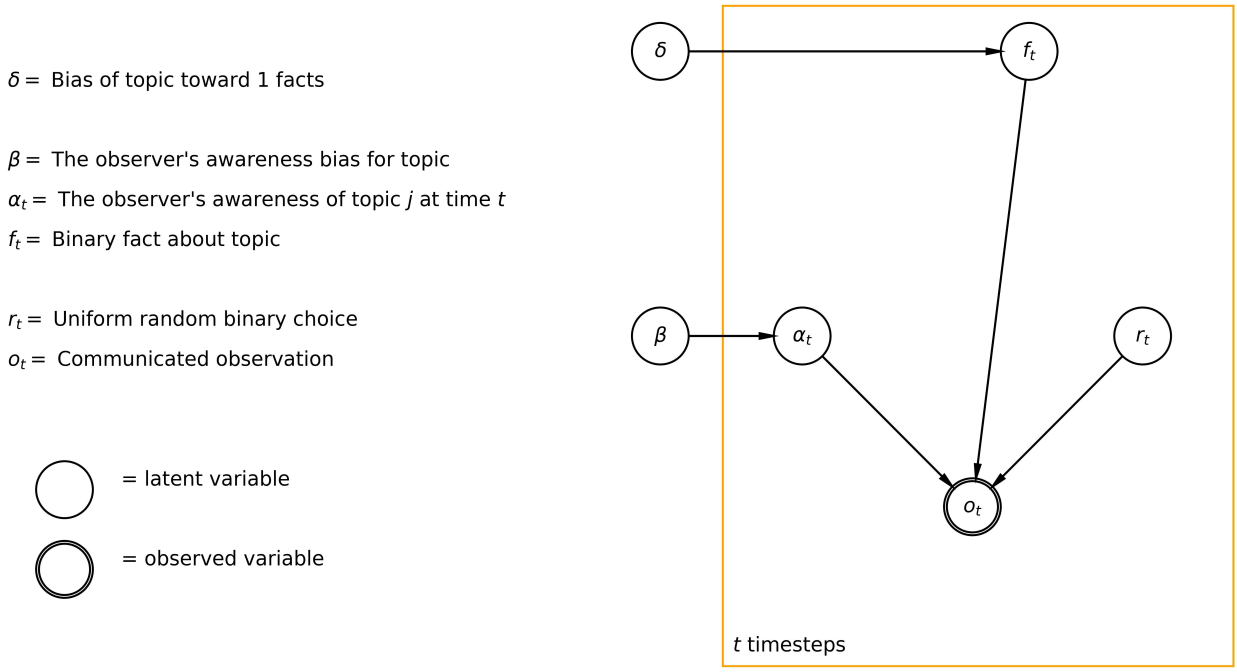


FIGURE 6. The Reporter Model A simplified to a single topic and user, with no-report observations removed.

which immediately fills that role.

Our statistic for $P(o_{\bar{\imath},j,t}=1)$ is the observed probability

$$P(o_{\bar{\imath},j,t}=1)=\frac{\#\{o_{\bar{\imath},j,t}:o_{\bar{\imath},j,t}=1\}}{\#\{o_{\bar{\imath},j,t}:o_{\bar{\imath},j,t}\neq 2\}}.$$

Now we can make the approximation

$$\begin{aligned}\beta_{i,j}\approx\hat{\beta_{i,j}} &= \frac{2\frac{\#\{o_{\bar{\imath},j,t}:o_{\bar{\imath},j,t}=1\}}{\#\{o_{\bar{\imath},j,t}:o_{\bar{\imath},j,t}\neq 2\}}-1}{2\hat{\delta_j}-1}\\ &= \frac{2\frac{\#\{o_{\bar{\imath},j,t}:o_{\bar{\imath},j,t}=1\}}{\#\{o_{\bar{\imath},j,t}:o_{\bar{\imath},j,t}\neq 2\}}-1}{2\frac{\#\{o_{i,j,t}:o_{i,j,t}=1\}}{\#\{o_{i,j,t}:o_{i,j,t}\neq 2\}}-1}.\end{aligned}$$

We note that $\varepsilon_{\bar{\imath},j}$ is best estimated using its Maximum Likelihood Estimator (MLE),

$$\varepsilon_{\bar{\imath},j}\approx\hat{\varepsilon}_{\bar{\imath},j}=\frac{\#\{o_{\bar{\imath},j,t}:o_{\bar{\imath},j,t}\neq 2\}}{\#\{o_{\bar{\imath},j,t}\}}.$$

In Appendix B we show experimentally that these estimators present a much less computationally intensive and more accurate method for approximate parameter estimation, and provide a thorough implementation in Python.

6. Reducing Untrusted Influence

We will now consider and review several methods for reducing the influence of untrustworthy observers in Reporter Model A. These fall into two categories: *pruning*, which entirely eliminates an observer's contributions to the fact approximation for a given topic, and its generalization *reduction*, which decreases our trust level in that observer's contributions, reducing the influence of their vote.

These two approaches highlight the two-fold purpose of the trust concept in our model. We primarily wish to eliminate untrustworthy information streams to optimize the quality of information we process. However, and possibly more importantly, we wish to reduce the overhead cost of approximating each fact in each topic, so we wish to reduce the number of observers whose reports we consider. In some applications, pruning should be even more aggressive, removing even nodes that report facts reliably once erroneous reporters are all removed. We note that in Reporter Model A we do not remove an observer from all topics if they are unreliable in the sense of a single topic. In some settings (such as combatting disinformation) it may be appropriate to prune observers completely from the model for poor performance in a subset of topics. It may also be appropriate in some circumstances to inversely weight some observer votes, e.g. in a case where an observer always reports the opposite of the truth they observe.

Each of the following methods is implemented and explored in Appendix B.

6.1. Pruning Methods.

The simplest form of pruning we consider is to pause after every n_p time steps and construct the approximation $\hat{\beta}_{i,j,t=n_p}$ for each observer i which has not yet been pruned. If this approximation falls below some threshold $\aleph$ around 0.5, i.e.

$$\hat{\beta}_{i,j,t=n_p} \leq \aleph$$

we prune observer i, and do not include any contributions from this observer for this topic in future. This method is extremely efficient and in our experiments produced acceptable results.

We may wish instead to prune only when we are certain that the estimate $\hat{\beta}_{i,j,t}$ has converged. We do so naively by checking if the absolute relative change has dropped below a threshold $h \approx 0.1$. If

$$\left| \frac{\hat{\beta}_{i,j,t} - \hat{\beta}_{i,j,t-1}}{\hat{\beta}_{i,j,t-1}} \right| \leq h \text{ AND } \hat{\beta}_{i,j,t} \leq \aleph$$

hold, we mark the observer i as pruned. This is slow in practice if we compute $\hat{\beta}_{i,j,t}$ at every time step t, but produces clean results. It also gives us a sense of "how" untrustworthy an observer was, since we have a clear sense of when they were pruned: earlier pruning represents more egregiously low awareness.

6.2. Markovian Reduction.

We will now consider a simple approach to reduction that requires minimal memory. We start by choosing a reduction constant $\rho \approx 0.95$. We begin the algorithm at $t = 0$ with a weight of 1 for all observers. At each timestep t we compute the weighted vote of all observers, and mark all observers who disagreed with this consensus. We multiply the weights of all incorrect observers by ρ, and continue to timestep $t + 1$. The result is that consistently poor performers lose their influence over the group vote over time.

The parameter ρ is sensitive: if the parameter is too low, it will result in a single observer emerging as the "most trusted" observer, whose vote is law. If the parameter is too high, the observers will not be separated as quickly. In future work, we may wish to investigate the relationship between choices of ρ, n_i, and n_t, to identify ideal parameter ranges for distinguishing observer awareness.

In our formulation of Reporter Model A we defined the β priors to be static, so there is no reason to think that later errors are more severe than earlier errors. However, in real life situations, it is reasonable to expect that observers would become more aware over time, so later errors would deserve higher distrust penalties. It also is reasonable to expect high-priority topics to induce higher awareness levels in observers. In these settings it might make sense to make ρ a function of time and topic.

7. Toward a Better Pruning Stopping Criterion

We previously discussed pruning when the estimate for $\hat{\beta}_{i,j,t}$ converges, but it would perhaps be preferable if we could stop and prune when $P\left(\beta_{i,j,t} = \hat{\beta}_{i,j,t}\right)$ reaches a threshold. As we consider this possibility, for brevity we will use the convention x^v to represent the random variable assignment $x = v$ where v is in the set of possible values of x.

Let us return to the simplification where one observer over n_t time steps provides observations, with all the "failed to report" data removed. We can show that the following holds via variable elimination.

Theorem 1. *In the simplification of Reporter Model A in which a single observer provides input for a single topic, the observations for any given time step t have the following distribution*

$$\begin{aligned} P\left(o_t^1 \mid a_t, f_t, r_t, \beta, \delta\right) &= P\left(o_t^1 \mid \beta, \delta\right) = \beta\left(\delta - \frac{1}{2}\right) + \frac{1}{2}, \\ P\left(o_t^0 \mid a_t, f_t, r_t, \beta, \delta\right) &= P\left(o_t^0 \mid \beta, \delta\right) = \beta\left(\frac{1}{2} - \delta\right) + \frac{1}{2}. \end{aligned}$$

Furthermore, $P\left(o_t^1\right) = P\left(o_t^0\right) = \frac{1}{2}$.

This is established by Proof 1 in Appendix A.

Thus the probability of reporting outcomes is independent of the time step and all variables contained within that time step. Let

$$E = \{o_t\}_{t=1}^{n_t}$$

be the evidence, the sequence of realizations observed. Let N_0 and N_1 be the numbers of 0's and 1's observed in this sequence,

$$N_v = \#\left\{o_t : o_t^v\right\}_{t=1}^{n_t}.$$

Then Bayes' law gives us

$$\begin{aligned} P\left(\beta=\hat{\beta},\delta=\hat{\delta}\mid o_t^1\right) &= \frac{P\left(o_t^1\mid\beta=\hat{\beta},\delta=\hat{\delta}\right)P\left(\beta=\hat{\beta},\delta=\hat{\delta}\mid E_{t-1}\right)}{P\left(o_t^1\right)} \\ &= \frac{\left(\hat{\beta}\left(\hat{\delta}-\frac{1}{2}\right)+\frac{1}{2}\right)P\left(\beta=\hat{\beta},\delta=\hat{\delta}\mid E_{t-1}\right)}{0.5} \\ &= 2\left(\hat{\beta}\left(\hat{\delta}-\frac{1}{2}\right)+\frac{1}{2}\right)P\left(\beta=\hat{\beta},\delta=\hat{\delta}\mid E_{t-1}\right). \end{aligned}$$

At time $t=0$ we have the uninformative prior,

$$P\left(\beta=\hat{\beta},\delta=\hat{\delta}\mid E_0\right)=P\left(\beta=\hat{\beta}\mid E_0\right)P\left(\delta=\hat{\delta}\mid E_0\right)=1.$$

We can use Bayesian updates at each time step to develop a posterior distribution for β under the condition $\delta=\hat{\delta}_t$. This means that we can halt at time t if $P\left(\beta=\hat{\beta}_t\mid\delta=\hat{\delta}_t,E_{t-1}\right)$ stabilizes (in a Kullback-Leibler sense) and prune this observer if $\hat{\beta}_t\leq\aleph$.

This is computationally expensive compared to our other approaches, and is not yet implemented in our code.

8. Alternate Formulations and Future Work

The Reporter Model A we have described has broad applications, but is not universally suitable, so it behooves us to consider minor alterations that fulfill similar needs. One such alteration we encountered early on when applying Model A is described below. We motivate this model as follows. Suppose our news analyst is interested in a set of topic states $\{\delta_j\}_{j=1}^{n_j}$. For example, each may represent the disaster readiness level of a state in the United States. The observers, rather than all encountering the same binary fact each timestep, now encounter individual facts. This might model a situation where each observer asks yes-or-no-questions of a representative of each state.

Definition 2. ***Reporter Model B:*** *We define this precisely the same as Reporter Model A in Definition 1 except that the facts $f_{i.j.t}$ are now within the observer loop, meaning they are unique to each observer. This model is shown in Figure 7.*

In this model, we are no longer interested in the individual facts $f_{i,j,t}$, but instead wish to recover δ_j. This model has several computational advantages, since it now presents plates (i.e. for-loops) that are strictly ordered as subsets. This is advantageous for some probabilistic languages (including PyMC3).

In our code we have implemented but not exhaustively tested the discussed reduction strategies. In future work we wish to evaluate these methods more thoroughly, including probability-based pruning. An element from real-life which our models do not replicate is the concept of finding new information sources. Finding new sources to replace less informed observers would be extremely useful, and could transform this problem into an explore versus exploit task, much like a multi-armed-bandit problem.

Finally, the most important difference between our model and real-life situations we wish to examine is that news is rarely expressed as binary facts. Applying

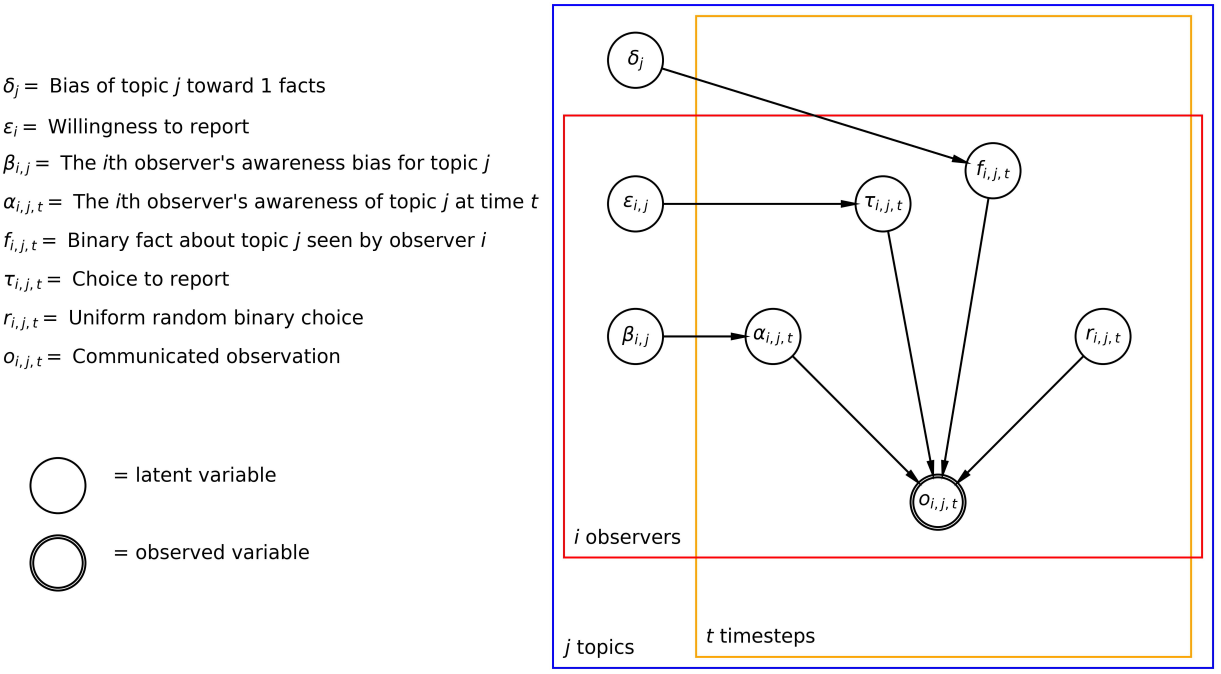


FIGURE 7. Reporter Model B

our work to a real-world situation would require accomplishing numerous fact-extraction and fact-matching tasks within the realm of natural language processing. We hope that these models we discussed are sufficiently portable that such a text-to-fact tool could plug into the models with minimal structural changes.

9. CONCLUSION

The model(s) proposed in this paper represents only a beginning of this type of investigation, and we feel optimistic that more efforts in this vein will follow. The human brain has hitherto been assigned the work of sorting and prioritizing information sources almost exclusively, but cognitive thresholds, if they have not yet been exceeded by news volume, soon will be. In such a setting it will be invaluable for news analysts and news readers alike to become equipped with tools that triage their media exposure using credibility assessments and other features of the media and its sources.

References

1. Arlene Fink, Jacqueline Kosecoff, Mark Chassin, and Robert H Brook, *Consensus methods: characteristics and guidelines for use.*, American journal of public health **74** (1984), no. 9, 979–983.
2. Jeffrey Gottfried and Elisa Shearer, *News use across social media platforms 2016*, (2019).
3. Hendrik Heuer and Andreas Breiter, *Trust in news on social media*, Proceedings of the 10th Nordic conference on human-computer interaction, 2018, pp. 137–147.
4. Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al., *Knowledge graphs*, Synthesis Lectures on Data, Semantics, and Knowledge **12** (2021), no. 2, 1–257.
5. Hans-Peter Kohler, Jere R Behrman, and Susan C Watkins, *The density of social networks and fertility decisions: Evidence from south nyanza district, kenya*, Demography **38** (2001), no. 1, 43–58.
6. Georgiy Levchuk and Erik Blasch, *Probabilistic graphical models for multi-source fusion from text sources*, 2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA), IEEE, 2015, pp. 1–10.
7. Dragomir Radev, *A common theory of information fusion from multiple text sources step one: cross-document structure*, 1st SIGdial workshop on Discourse and dialogue, 2000, pp. 74–83.
8. Wei Ren, Randal W Beard, and Ella M Atkins, *A survey of consensus problems in multi-agent coordination*, Proceedings of the 2005, American Control Conference, 2005., IEEE, 2005, pp. 1859–1864.
9. Lakshmi Siva Sankar, M Sindhu, and M Sethumadhavan, *Survey of consensus protocols on blockchain applications*, 2017 4th international conference on advanced computing and communication systems (ICACCS), IEEE, 2017, pp. 1–5.
10. Mason Walker, *Nearly a quarter of [a]mericans get their news from podcasts*, (2022).
11. Jonathan S Yedidia, William T Freeman, Yair Weiss, et al., *Understanding belief propagation and its generalizations*, Exploring artificial intelligence in the new millennium **8** (2003), no. 236-239, 0018–9448.
12. Reza Zafarani and Huan Liu, *Evaluation without ground truth in social media research*, Communications of the ACM **58** (2015), no. 6, 54–60.

Appendix A. Supplemental Derivations

We first provide a proof for Theorem 1.

Proof 1. Recall that we have eliminated observations with value 2 (the "no report" observations) and are considering one topic and one observer only. Let t be any given timestep. The random variables α_t, f_t, and r_t are Bernoulli, with the parameters β, δ, and $\frac{1}{2}$. The variable o is produced by computing

$$o_t = \alpha_t f_t + (1 - \alpha_t) r_t,$$

which yields the following truth table:

o_t	α_t	f_t	r_t
1	1	1	1
1	1	1	0
1	0	1	1
1	0	0	1
0	1	0	1
0	1	0	0
0	0	1	0
0	0	0	0

In this case, the probability of the observer stating the value 1 is the sum of the probability of this occurring given all possible immediate parent variable combinations,

$$\begin{aligned} P\left(o_t^1 \mid \alpha_t, f_t, r_t, \beta, \delta\right) &= \sum_{i,j,k,l,m} P\left(o_t^1, \alpha_t^i, f_t^j, r_t^k, \beta^l, \delta^m\right) \\ &= P\left(o^1, \alpha^1, f^1, r^1\right) + P\left(o^1, \alpha^1, f^1, r^0\right) + P\left(o^1, \alpha^0, f^1, r^1\right) + P\left(o^1, \alpha^0, f^0, r^1\right) \\ &= P\left(o^1, \alpha^1, f^1\right) + P\left(o^1, \alpha^0, r^1\right) \\ &= P\left(o^1 \mid \alpha^1, f^1\right) P\left(\alpha^1, f^1\right) + P\left(o^1 \mid \alpha^0, r^1\right) P\left(\alpha^0, r^1\right) \end{aligned}$$

Here we may use the independence of the parent variables to simplify,

$$\begin{aligned} P\left(o^1 \mid \beta, \delta\right) &= P\left(o^1 \mid \alpha^1, f^1\right) P\left(\alpha^1\right) P\left(f^1\right) + P\left(o^1 \mid \alpha^0, r^1\right) P\left(\alpha^0\right) P\left(r^1\right) \\ &= (1)(\beta)(\delta) + (1)(1-\beta)\left(\frac{1}{2}\right) \\ &= \beta\delta + \frac{1}{2} - \frac{1}{2}\beta \\ &= \beta\left(\delta - \frac{1}{2}\right) + \frac{1}{2}. \end{aligned}$$

Similarly we find

$$\begin{aligned} P\left(o^0 \mid \beta, \delta\right) &= P\left(o^0 \mid \alpha^1, f^0\right) P\left(\alpha^1\right) P\left(f^0\right) + P\left(o^0 \mid \alpha^0, r^0\right) P\left(\alpha^0\right) P\left(r^0\right) \\ &= (1)(\beta)(1-\delta) + (1)(1-\beta)\left(\frac{1}{2}\right) \\ &= \beta(1-\delta) + \frac{1}{2} - \frac{1}{2}\beta \\ &= \beta\left(\frac{1}{2} - \delta\right) + \frac{1}{2}. \end{aligned}$$

Let us consider the marginalization of this probability given the assumption that β and δ are both uniform random between 0 and 1. This gives us

$$\begin{aligned}
P\left(o^0\right) &= \int_{l,m} P\left(o^0 \mid \beta^l, \delta^m\right) P\left(\beta^l, \delta^m\right) dldm \\
&= \int_m \int_l \left(l\left(\frac{1}{2} - m\right) + \frac{1}{2}\right) P\left(\beta^l\right) P\left(\delta^m\right) dldm \\
&= \int_m \int_l \left(l\left(\frac{1}{2} - m\right) + \frac{1}{2}\right) (1)\,(1)\, dldm \\
&= \int_m \int_l \left(l\left(\frac{1}{2} - m\right) + \frac{1}{2}\right) dldm \\
&= \int_0^1 \int_0^1 \left(l\left(\frac{1}{2} - m\right) + \frac{1}{2}\right) dldm \\
&= \frac{1}{4} lm\left(l - lm + 2\right)\Big|_{l,m=0}^{l,m=1} \\
&= \frac{1}{2}
\end{aligned}$$

□

It turns out that the awareness variable may be estimated using what we just proved. We simply compute

$$\begin{aligned}
P\left(a^1 \mid o^1, \beta, \delta\right) &= \frac{P\left(a^1, o^1 \mid \beta, \delta\right)}{P\left(o^1 \mid \beta, \delta\right)} \\
&= \frac{P\left(o^1 \mid a^1, \beta, \delta\right) P\left(a^1 \mid \beta, \delta\right)}{P\left(o^1 \mid \beta, \delta\right)} \\
&= \frac{\left[P\left(o^1 \mid a^1, f^1, \beta, \delta\right) + P\left(o^1 \mid a^1, f^0, \beta, \delta\right)\right] P\left(a^1 \mid \beta\right)}{P\left(o^1 \mid \beta, \delta\right)} \\
&= \frac{[1+0]\,\beta}{\beta\left(\delta - \frac{1}{2}\right) + \frac{1}{2}} \\
&= \frac{\beta}{\beta\left(\delta - \frac{1}{2}\right) + \frac{1}{2}} \\
&= \frac{1}{\delta - \frac{1}{2} + \frac{1}{2\beta}}.
\end{aligned}$$

And similarly,

$$\begin{aligned} P\left(a^1 \mid o^0, \beta, \delta\right) &= \frac{P\left(a^1, o^0 \mid \beta, \delta\right)}{P\left(o^0 \mid \beta, \delta\right)} \\ &= \frac{P\left(o^0 \mid a^1\right) P\left(a^1 \mid \beta, \delta\right)}{P\left(o^0\right)} \\ &= \frac{\left[P\left(o^0 \mid a^1, f^0\right) + P\left(o^0 \mid a^1, f^1\right)\right] P\left(a^1 \mid \beta\right)}{P\left(o^0\right)} \\ &= \frac{[1+0]\,\beta}{\beta\left(\frac{1}{2}-\delta\right)+\frac{1}{2}} \\ &= \frac{\beta}{\beta\left(\frac{1}{2}-\delta\right)+\frac{1}{2}} \\ &= \frac{1}{\frac{1}{2}-\delta+\frac{1}{2\beta}} \end{aligned}$$

This lends itself to an alternative updating scheme. We wish to consider a Bayesian belief update where we use an observation of o to change our opinion about the value of β. We manipulate

$$\begin{aligned} P\left(a^1 \mid o^0, \beta, \delta\right) &= \frac{P\left(a^1, o^0 \mid \beta, \delta\right)}{P\left(o^0 \mid \beta, \delta\right)} \\ \frac{1}{2\beta} &= \frac{1}{P\left(a^1 \mid o^0, \beta, \delta\right)} - \frac{1}{2} + \delta \\ 2\beta &= \frac{1}{\frac{1}{P(a^1 \mid o^0, \beta, \delta)} - \frac{1}{2} + \delta} \\ \beta &= \frac{1}{2\left(\frac{1}{P(a^1 \mid o^0, \beta, \delta)} - \frac{1}{2} + \delta\right)} \end{aligned}$$

We do not know δ, but using a consensus of multiple observers over multiple draws of f we can approximate it with $\hat{\delta}$. This gives us the estimator

$$\hat{\beta}_t = \frac{1}{2\left(\frac{1}{P\left(a^1 \mid o^0, \hat{\beta}_{t-1}, \hat{\delta}_{t-1}\right)} - \frac{1}{2} + \hat{\delta}_t\right)}$$

We can start with a uninformative prior (uniform over $[0, 1]$) and update this via Bayes theorem.

E-mail address, N. Lines: nicholasalines@gmail.com
URL: https://github.com/linesn/reporter_modeling

Appendix_B

May 13, 2022

Appendix B: Code For Reporter Model A

Nicholas Lines

1 Environment Setup

[1]:
```
%pylab inline
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
```

[2]:
```
from importlib import reload
import reporter
```

[10]:
```
reload(reporter);
```

[3]:
```
import os
print (os.environ['CONDA_DEFAULT_ENV'])
```

```
pymc3
```

[4]:
```
#import arviz as az
import numpy as np
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
#import graphviz
import os
import pymc3 as pm
from pymc3 import Model, Normal, HalfNormal, Bernoulli, Deterministic, Uniform
from pymc3 import find_MAP
```

[5]:
```
import scipy.stats as st
```

[6]:
```
from tqdm.notebook import tqdm
```

2 A Generative Model for News Information

We wish to model the information network by which a news reporter can gather facts about the news topics of interest to them. While we will use vocabulary related to the application of gathering information about news topics, we note that this task is analagous to many other information-gathering-and-fusion processes.

Consider a reporter who must stay informed about n_j news topics, indexed by j. The reporter cannot observe the facts related to these news topics directly, and relies on a network of first-hand observers to inform the reporter. This network consists of n_i observers, indexed by i, who each provide an observation at each time step t of n_t timesteps. These observations, labeled $o_{i,j,t}$ (for observer i speaking about topic j at time t) are the only information provided to the reporter.

Many techniques have been proposed to derive "facts" from text streams and other media, but we will not include these steps. Instead, we will assume that each news topic j produces a single binary "fact" called $f_{j,t}$ at each time step t. (For example, the sports topic might yield facts such as "The tigers beat the rams on Saturday," which can be represented as a 1 or 0 for true or false.) The fact is then observed by each observer. However, we wish to model the fact that not all observers are are equally aware of all newsworthy subjects, and not all observers will pass on their information at each time step. Therefore, we insist on the following dependencies.

1. The binary fact for each topic and each timestep is sampled from a Bernoulli distribution $f_{j,t} \sim Bern(\delta_j)$ where $\delta_j \sim Unif(0,1)$ represents the bias of this news topic toward 1-valued facts.
2. Whether an observer will report their observation or not is represented by the binary variable $\tau_{i,j,t} \sim Bern(\epsilon_{i,j})$ where $\epsilon_{i,j} \sim Unif(0,1)$ represents that observer's bias toward reporting about this news topic.
3. Whether an observer is well-informed about this topic's fact at time t is represented by the binary variable $a_{i,j,t} \sim Bern(\beta_{i,j})$ where $\beta_{i,j} \sim Unif(0,1)$ is a hyperparameter representing how well-informed the observer is on this topic on average. This hyperparameter is the variable of most interest for us.
4. Let $\tilde{x}$ represent `not` x for a binary variable x. We will use a uniform random binary variable $r_{i,j,t} \sim Bern(0.5)$. The observer's report to the reporter is
$$o_{i,j,t} = \tau_{i,j,t}(a_{i,j,t}f_{j,t} + \tilde{a_{i,j,t}}r_{i,j,t}) + 2\tilde{\tau_{i,j,t}},$$
meaning that if the observer is aware and chooses to report at this timestep, they report the fact $f_{j,t}$ with no alteration; if the observer is unaware, the observer reports 1 or 0 with equal probability; and if the observer chooses not to report, a 2 is returned, signifying that no information was passed on.
5. The reporter then constructs
$$b_j = mode_i(\{o_{i,j,t} : o_{i,j,t} \neq 2\})$$
as an approximation to $f_{j,t}$, i.e. the reporter's best guess at the true fact.

We should make a few observations about this naive construction before continuing.

1. First, the approximation $b_j \approx f_{j,t}$ is not really legitimate because it could well be that all observers agreed to lie. Really b_j tells us nothing more than the consensus of observations about the fact. This fact is important in situations where we suspect that there is not consensus or that the consensus of our observers is unreliable.

2. The model as stated leaves each topic's network entirely independent of the other topics, so there is no real need for the outermost plate in the diagram: this was included solely to remind us of this assumption and to indicate that this problem scales in topics j.
3. The purpose for laying out this generative model is to help us strategize how to learn the observer awareness hyperparameter, $\beta_{i,j}$, which will allow us to reduce the network by cutting all edges $o_{i,j,t} - b_{j,t}$ when $\beta_{i,j} < \rho$ for some threshold ρ.
4. We are not interested in recovering any other hyperparameters in this situation. These exist solely to lend verisimilitude to the model.
5. If we allow $|\delta_j - 0.5|$ to grow too small, the inference problem gets much harder.

3 Extremely Simple Inference

Let's consider how we could approximate $\beta_{i,j}$ using the assumed distributions. If we condition on $\tau_{i,j,t} = 1$ we restrict ourselves to a reveiw of the informative reports from a single observer, i.e.

$$\{o_{i,j,t} : o_{i,j,t} \neq 2, i = \bar{i}\},$$

and we can write

$$o_{i,j,t} = a_{i,j,t} f_{j,t} + \tilde{a}_{i,j,t} r_{i,j,t}.$$

We can quickly describe the probability distribution for $o_{i,j,t}$ under these conditions using a probability table (since these are all binary variables). We recall that the parameter of each Bernoulli random variable represents the probability that variable equals one. (For simplicity we will drop the subscripts for the table.)

o	a	f	r	$P(a)$	$P(f)$	$P(r)$
0	1	0	0	β	$1-\delta$	0.5 (always)
1	1	1	0	β	δ	0.5
1	1	1	1	β	δ	0.5
0	1	0	1	β	$1-\delta$	0.5
1	0	1	1	$1-\beta$	δ	0.5
1	0	0	1	$1-\beta$	$1-\delta$	0.5
0	0	1	0	$1-\beta$	δ	0.5
0	0	0	0	$1-\beta$	$1-\delta$	0.5

Each of β, δ, and r are mutually independent variables, so their joint probability is the product of their marginal probabilities. By summing up the joint probabilities of combinations that lead us to $o_{\bar{i},j,t} = 1$ we have

$$\begin{aligned} P(o_{\bar{i},j,t} = 1) &= 0.5(\beta_{\bar{i},j}\delta_j + \delta_j - \beta_{\bar{i},j,t}\delta_j + 1 - \beta_{\bar{i},j,t} - \delta_j + \beta_{\bar{i},j,t}\delta_j + \beta_{\bar{i},j,t}\delta_j) \quad (1) \\ &= 0.5(-\beta_{\bar{i},j,t} + \beta_{\bar{i},j,t}\delta_j + 1) \quad (2) \end{aligned}$$

We would reach the same result with much less work using conditional probabilities (factors), since

$$P(o_{\bar{i},j,t}=1) = P(o_{\bar{i},j,t}=1 \mid a_{\bar{i},j,t}=0)P(a_{\bar{i},j,t}=0) + P(o_{\bar{i},j,t}=1 \mid a_{\bar{i},j,t}=1)P(a_{\bar{i},j,t}=1) \quad (3)$$

$$= 0.5(1-\beta_{\bar{i},j}) + \delta_j\beta_{\bar{i},j}. \quad (4)$$

Solving for β gives us

$$\beta_{\bar{i},j,t} = \frac{2P(o_{\bar{i},j,t}=1)-1}{2\delta_j - 1}.$$

We could approximate

$$\delta_j \approx \hat{\delta_j} = \frac{\#\{o_{i,j,t} : o_{i,j,t}=1\}}{\#\{o_{i,j,t} : o_{i,j,t} \neq 2\}},$$

i.e. we use the sample mean of the informative observations to approximate the probability that a generated fact equals 1 (an extension of the approximation $b_{j,t} = \hat{f}_{j,t} \approx f_{j,t}$). But we already have the observed mean

$$\hat{\delta_j} = \bar{b_{j,t}},$$

which immediately fills that role.

Our statistic for $P(o_{\bar{i},j,t}=1)$ is the observed probability

$$P(o_{\bar{i},j,t}=1) = \frac{\#\{o_{\bar{i},j,t} : o_{\bar{i},j,t}=1\}}{\#\{o_{\bar{i},j,t} : o_{\bar{i},j,t} \neq 2\}}.$$

Now we can make the approximation

$$\beta_{i,j} \approx \hat{\beta_{i,j}} = \frac{2\frac{\#\{o_{\bar{i},j,t}:o_{\bar{i},j,t}=1\}}{\#\{o_{\bar{i},j,t}:o_{\bar{i},j,t}\neq 2\}} - 1}{2\hat{\delta_j} - 1} \quad (5)$$

$$= \frac{2\frac{\#\{o_{\bar{i},j,t}:o_{\bar{i},j,t}=1\}}{\#\{o_{\bar{i},j,t}:o_{\bar{i},j,t}\neq 2\}} - 1}{2\frac{\#\{o_{i,j,t}:o_{i,j,t}=1\}}{\#\{o_{i,j,t}:o_{i,j,t}\neq 2\}} - 1}. \quad (6)$$

4 Simulating Data with this Model

We want to create an artificial dataset so we can observe this model at work, and hopefully show how to recover the desired hyperparameter.

Generate sample data

[8]:
```
n_i = 20
n_t = 1000
data = reporter.generate_data(n_i, n_t, r_p=0.5)
```

4.1 Approximate Parameter Inference

[11]:
```
estimators = reporter.evaluate_estimators_full(data=data, agreement_thresh=0.5,␣
↪printing=True)
```

```
Estimators are delta_hat=0.813, b, beta_hat.
The norm distance between epsilon_hat and epsilon is 0.06284274373013359.
The approximation b for f is 93.7% accurate.
The approximation delta_hat for delta=0.8590461529521575 has relative error
-0.05360148903981176.
The norm distance between beta_hat and beta is 0.5460080903701915.
The beta_hat estimator recognizes betas less than 0.5
100.0% of the time.
```

[12]:
```
(o, delta, f, beta, alpha, epsilon,tau,r) = data
```

5 Dynamically Pruning Unhelpful Observers

5.1 Pruning by convergence of $\hat{\beta}$

[61]:
```
def prune_by_beta_hat_convergence(data, tolerance=.1, agreement_thresh=0.5,␣
↪printing=True):
    """Prune unhelpful observers by beta_hat convergence.

    Args:
        data (tuple): The output from generate_data, a tuple of
            (o, delta, f, beta, alpha, epsilon,tau,r).
        tolerance (float): The relative change level at which to
            declare convergence.
        agreement_thresh (float): The agreement threshold for
            proportionate threshold agreement.
        printing (bool): Indicates whether to print info.
    Returns:

    """
    (o, delta, f, beta, alpha, epsilon,tau,r) = data
    # We need to drop any timestep where we got absolutely no information.
    drop_rows = np.argwhere(np.all(o == 2, axis=1)).flatten()
    o = np.delete(o,drop_rows,0)
    n_t, n_i = o.shape
    pruned_observers = zeros(n_i)
    beta_hats = []
    for t_max in tqdm(range(n_t)):
        # Form beta_hat
        b = np.array([st.mode([oit for oit in o[t] if oit!=2])[0][0] for t in␣
↪range(t_max)])
        delta_hat = np.mean(b)
```

```
        p_o_i_bar_1 = [(o[:t_max,i]==1).sum()/((o[:t_max,i]!=2).sum()) for i in␣
↪range(n_i)]
        beta_hat = np.array([(2 * p_i - 1) / (2*delta_hat - 1) for p_i in␣
↪p_o_i_bar_1])
        beta_hats.append(beta_hat)
        if t_max>0:
            # Check convergence
            c = np.abs(np.divide(beta_hats[t_max]-beta_hats[t_max-1],
                                 beta_hats[t_max-1])) <= tolerance
            u = beta_hats[t_max] <= agreement_thresh
            for i in range(n_i):
                if c[i] and u[i] and not pruned_observers[i]:
                    pruned_observers[i] = t_max
    # compute evaluation results with pruning
    drop_cols = np.nonzero(pruned_observers)
    keep_cols = np.where(pruned_observers==0)[0]
    beta_cut = np.array([beta[i] for i in keep_cols])
    op = np.delete(o,drop_cols,1)
    # Concensus estimator for f
    b = np.array([st.mode([oit for oit in op[t] if oit!=2])[0][0] \
                  if np.any(op[t]!=2) else 0 for t in range(n_t)])
    f = np.delete(f,drop_rows)
    # Judge the quality of the f estimate
    concensus_f_accuracies = 1-(abs(b-f).sum()/b.shape[0])
    # Estimate the probability of observing 1 empirically
    p_o_i_bar_1 = [(o[:,i]==1).sum()/((o[:,i]!=2).sum()) for i in keep_cols]
    # Estimate delta
    delta_hat = np.mean(b)
    # Estimate beta
    beta_hat = np.array([(2 * p_i - 1) / (2*delta_hat - 1) for p_i in␣
↪p_o_i_bar_1])
    # Judge the quality of the beta estimate
    beta_norm = np.linalg.norm(beta_hat - beta_cut)
    beta_agreements = reporter.proportionate_threshold_agreement(beta_hat,
                                                                 beta_cut,
                                                                 ␣
↪agreement_thresh)
    if printing:
        print("Results when pruning by beta_hat convergence:")
        print(f"The observer pruning is \n{pruned_observers}")
        print(f"Estimators are delta_hat={delta_hat}, b, beta_hat.")
        #print(f"The norm distance between epsilon_hat and epsilon is␣
↪{epsilon_norm}.")
        print(f"The approximation b for f is {100*concensus_f_accuracies}%␣
↪accurate.")
        print(f"The approximation delta_hat for delta={delta}")
```

```
        print(f"has relative error {(delta_hat-delta)/delta}.")
        print(f"The norm distance between beta_hat and beta is {beta_norm}.")
        print(f"The beta_hat estimator recognizes betas less than␣
↪{agreement_thresh}")
        print(f"{beta_agreements*100}% of the time.")
    return(pruned_observers, b, delta_hat, beta_hat)
```

[62]:
```
pestimators = prune_by_beta_hat_convergence(data, tolerance=.1,␣
↪agreement_thresh=0.5, printing=True)
```

```
  0%|          | 0/1000 [00:00<?, ?it/s]
```

```
Results when pruning by beta_hat convergence:
The observer pruning is
[ 0.  0.  8. 74.  4.  4. 25. 27.  8.  4.  6.  4. 39.  8.  8.  6.  0.  4.
  0. 20.]
Estimators are delta_hat=0.813, b, beta_hat.
The approximation b for f is 93.9% accurate.
The approximation delta_hat for delta=0.8590461529521575
has relative error -0.05360148903981176.
The norm distance between beta_hat and beta is 0.32500630254090734.
The beta_hat estimator recognizes betas less than 0.5
100.0% of the time.
```

5.2 Pruning after a set number of timesteps

The idea in this case is to run n_p timesteps and then prune any observer i where $\hat{\beta}_{i,t=n_p} < \aleph$ for a given threshold $\aleph \approx 0.5$.

[68]:
```
def prune_at_n_p(data, n_p=100, agreement_thresh=0.5, repeat=True,␣
↪printing=True):
    """Prune unhelpful observers after n_p.

    Args:
        data (tuple): The output from generate_data, a tuple of
            (o, delta, f, beta, alpha, epsilon,tau,r).
        n_p (int): We check for unhelpful observers at timesteps
            that are multiples of n_p. Must be less than n_t.
        agreement_thresh (float): The agreement threshold for
            proportionate threshold agreement.
        repeat (bool): Indicates whether to use multiples of n_p
            or just the one pruning.
        printing (bool): Indicates whether to print info.
    Returns:

    """
    (o, delta, f, beta, alpha, epsilon,tau,r) = data
```

```
    # We need to drop any timestep where we got absolutely no information.
    drop_rows = np.argwhere(np.all(o == 2, axis=1)).flatten()
    o = np.delete(o,drop_rows,0)
    n_t, n_i = o.shape
    pruned_observers = zeros(n_i)
    #beta_hats = []
    n_ps = [n_p]
    if repeat:
        n_ps = range(n_p, n_t, n_p)
    for t_max in tqdm(n_ps):
        # Form beta_hat
        b = np.array([st.mode([oit for oit in o[t] if oit!=2])[0][0] for t in␣
↪range(t_max)])
        delta_hat = np.mean(b)
        p_o_i_bar_1 = [(o[:t_max,i]==1).sum()/((o[:t_max,i]!=2).sum()) for i in␣
↪range(n_i)]
        beta_hat = np.array([(2 * p_i - 1) / (2*delta_hat - 1) for p_i in␣
↪p_o_i_bar_1])
        #beta_hats.append(beta_hat)
        # Check convergence
        u = beta_hat <= agreement_thresh
        for i in range(n_i):
            if u[i] and not pruned_observers[i]:
                pruned_observers[i] = t_max
    # compute evaluation results with pruning
    drop_cols = np.nonzero(pruned_observers)
    keep_cols = np.where(pruned_observers==0)[0]
    beta_cut = np.array([beta[i] for i in keep_cols])
    op = np.delete(o,drop_cols,1)
    # Concensus estimator for f
    b = np.array([st.mode([oit for oit in op[t] if oit!=2])[0][0] \
                  if np.any(op[t]!=2) else 0 for t in range(n_t)])
    f = np.delete(f,drop_rows)
    # Judge the quality of the f estimate
    concensus_f_accuracies = 1-(abs(b-f).sum()/b.shape[0])
    # Estimate the probability of observing 1 empirically
    p_o_i_bar_1 = [(o[:,i]==1).sum()/((o[:,i]!=2).sum()) for i in keep_cols]
    # Estimate delta
    delta_hat = np.mean(b)
    # Estimate beta
    beta_hat = np.array([(2 * p_i - 1) / (2*delta_hat - 1) for p_i in␣
↪p_o_i_bar_1])
    # Judge the quality of the beta estimate
    beta_norm = np.linalg.norm(beta_hat - beta_cut)
    beta_agreements = reporter.proportionate_threshold_agreement(beta_hat,
                                                                 beta_cut,
```

```
                                                                    ␣
↪agreement_thresh)
   if printing:
       if not repeat:
           print(f"Results when pruning once at n_p={n_p}:")
       else:
           print(f"Results when pruning every n_p={n_p} steps:")
       print(f"The observer pruning is \n{pruned_observers}")
       print(f"Estimators are delta_hat={delta_hat}, b, beta_hat.")
       #print(f"The norm distance between epsilon_hat and epsilon is␣
↪{epsilon_norm}.")
       print(f"The approximation b for f is {100*concensus_f_accuracies}%␣
↪accurate.")
       print(f"The approximation delta_hat for delta={delta}")
       print(f"has relative error {(delta_hat-delta)/delta}.")
       print(f"The norm distance between beta_hat and beta is {beta_norm}.")
       print(f"The beta_hat estimator recognizes betas less than␣
↪{agreement_thresh}")
       print(f"{beta_agreements*100}% of the time.")
   return(pruned_observers, b, delta_hat, beta_hat)
```

[69]:
```
npestimators = prune_at_n_p(data, n_p=100, agreement_thresh=0.5, repeat=True,␣
↪printing=True)
```

```
  0%|          | 0/9 [00:00<?, ?it/s]
Results when pruning every n_p=100 steps:
The observer pruning is
[  0.   0. 100. 200. 100. 100. 100. 100. 100. 100. 100.   0.   0. 100.
   0. 100.   0. 200.   0. 100.]
Estimators are delta_hat=0.837, b, beta_hat.
The approximation b for f is 96.7% accurate.
The approximation delta_hat for delta=0.8590461529521575
has relative error -0.025663525616632744.
The norm distance between beta_hat and beta is 0.24712039308903547.
The beta_hat estimator recognizes betas less than 0.5
100.0% of the time.
```

5.3 Markovian Reduction

The idea of this "pruning" variation is that we down-weight each observer's contribution if their previous contribution disagreed with the consensus.

[80]:
```
def markov_reduce(data, reduction=0.9, agreement_thresh=0.5, printing=True):
    """Reduce unhelpful observers by Markov history.

    Args:
```

```
        data (tuple): The output from generate_data, a tuple of
            (o, delta, f, beta, alpha, epsilon,tau,r).
        reduction (float): An incorrect answer reduces trust by
            this multiplicative amount.
        agreement_thresh (float): The agreement threshold for
            proportionate threshold agreement.
        printing (bool): Indicates whether to print info.
    Returns:

    """
    (o, delta, f, beta, alpha, epsilon,tau,r) = data
    # We need to drop any timestep where we got absolutely no information.
    drop_rows = np.argwhere(np.all(o == 2, axis=1)).flatten()
    o = np.delete(o,drop_rows,0)
    conc = []
    n_t, n_i = o.shape
    observer_trust = ones(n_i)
    beta_hats = []
    for t_max in tqdm(range(n_t)):
        # the consensus is now a weighted vote
        consensus = np.argmax(np.bincount(o[t_max], weights=observer_trust)[:2])
        conc.append(consensus)
        # reduce the trust for the elements that voted against consensus
        reduce = ones(n_i) - ((o[t_max]!=consensus) * (1 - reduction))
        observer_trust = np.multiply(observer_trust, reduce)
    # compute evaluation results with reduction
    # Consensus estimator for f
    b = np.array(conc)
    f = np.delete(f,drop_rows)
    # Judge the quality of the f estimate
    concensus_f_accuracies = 1-(abs(b-f).sum()/b.shape[0])
    # Estimate delta
    delta_hat = np.mean(b)
    if printing:
        print(f"Results when using Markov Reduction = {reduction}:")
        print(f"The observer trust weights are \n{observer_trust}")
        print(f"Estimators are delta_hat={delta_hat}, b")
        print(f"The approximation b for f is {100*concensus_f_accuracies}%␣
↪accurate.")
        print(f"The approximation delta_hat for delta={delta}")
        print(f"has relative error {(delta_hat-delta)/delta}.")
    return(observer_trust, b, delta_hat)
```

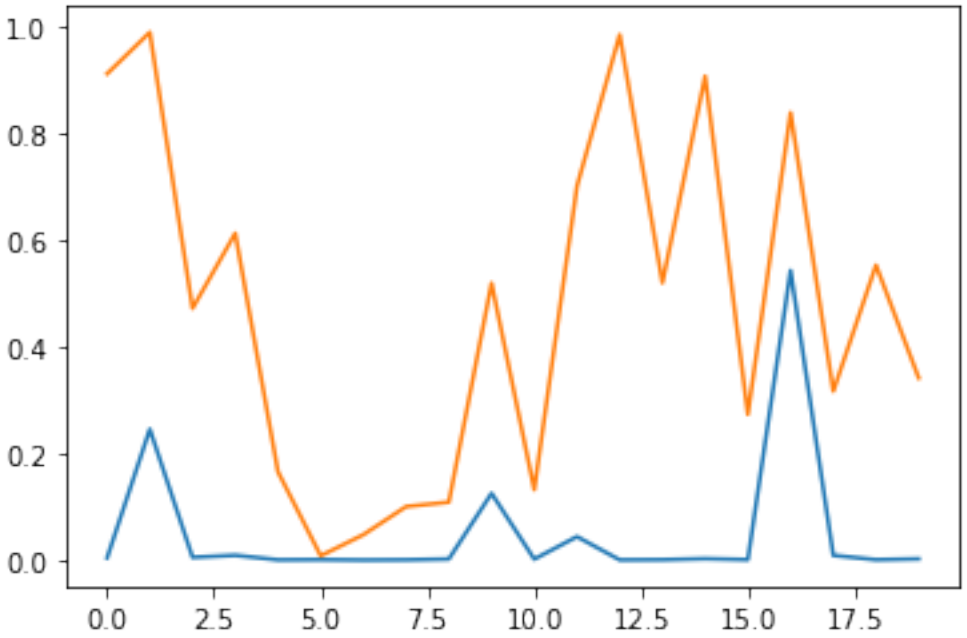
[84]:
```
mrestimates = markov_reduce(data, reduction=0.99, agreement_thresh=0.5,␣
↪printing=True)
```

```
  0%|          | 0/1000 [00:00<?, ?it/s]
```

```
Results when using Markov Reduction = 0.99:
The observer trust weights are
[1.30275764e-03 7.55518341e-02 1.60888329e-03 2.76836936e-03
 1.53165192e-04 2.66210006e-04 5.66297108e-05 1.71069639e-04
 6.19252872e-04 3.85303585e-02 5.48896377e-04 1.35476288e-02
 1.17943806e-04 2.35964522e-04 8.04179448e-04 2.79929329e-04
 1.67133935e-01 2.74068567e-03 2.26666578e-04 7.05684372e-04]
Estimators are delta_hat=0.848, b
The approximation b for f is 97.0% accurate.
The approximation delta_hat for delta=0.8590461529521575
has relative error -0.012858625714342363.
```

[87]:
```
plot(mrestimates[0]/mrestimates[0].sum())
plot(beta)
```

[87]:
```
[<matplotlib.lines.Line2D at 0x7fd835e73a60>]
```



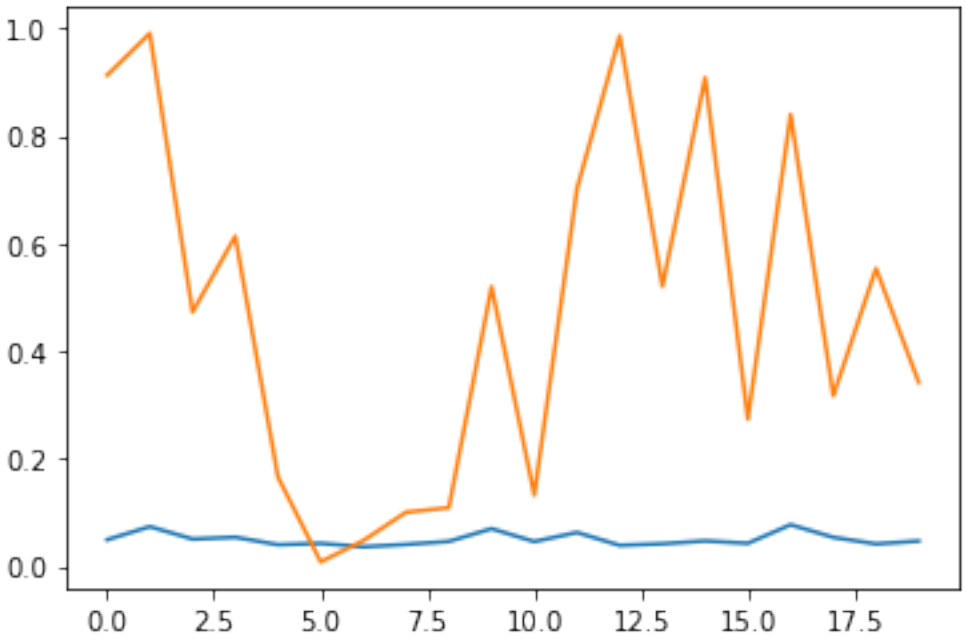
[89]:
```
mrestimates = markov_reduce(data, reduction=0.999, agreement_thresh=0.5,␣
 ↪printing=True)
plot(mrestimates[0]/mrestimates[0].sum())
plot(beta)
```

```
  0%|          | 0/1000 [00:00<?, ?it/s]

Results when using Markov Reduction = 0.999:
The observer trust weights are
[0.51616401 0.77172301 0.52923738 0.56479882 0.41876922 0.44423314
```

```
 0.37814169 0.42425152 0.4831811  0.7311354  0.48028925 0.65756633
 0.40638662 0.43717847 0.49294694 0.44601452 0.80887889 0.56198046
 0.43456195 0.49048713]
Estimators are delta_hat=0.842, b
The approximation b for f is 96.6% accurate.
The approximation delta_hat for delta=0.8590461529521575
has relative error -0.01984311657013712.
```

[89]:
```
[<matplotlib.lines.Line2D at 0x7fd835d68070>]
```

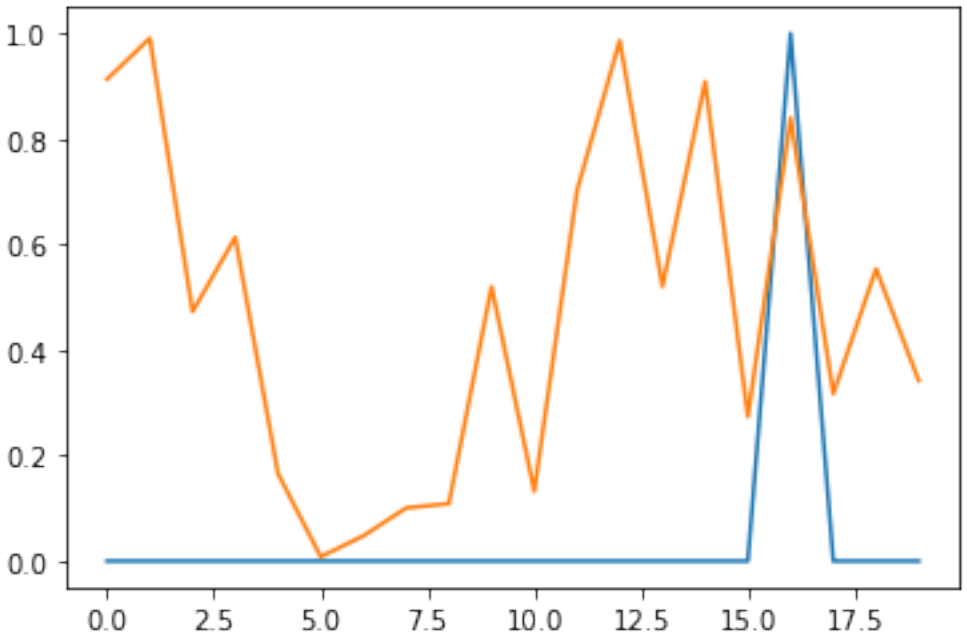


[90]:
```
mrestimates = markov_reduce(data, reduction=0.8, agreement_thresh=0.5,␣
 ↪printing=True)
plot(mrestimates[0]/mrestimates[0].sum())
plot(beta)
```

```
  0%|          | 0/1000 [00:00<?, ?it/s]
Results when using Markov Reduction = 0.8:
The observer trust weights are
[2.29626139e-65 9.16444925e-28 5.22109392e-64 4.57269431e-59
 2.49780565e-85 1.75008561e-80 6.36014997e-95 3.63478561e-84
 3.02280687e-72 7.72103322e-35 2.07725706e-73 6.44219799e-45
 1.97896552e-88 9.62119740e-82 1.34239591e-70 4.27266995e-80
 3.38460656e-14 4.57269431e-59 2.01771134e-82 2.81520827e-71]
Estimators are delta_hat=0.82, b
The approximation b for f is 93.6% accurate.
The approximation delta_hat for delta=0.8590461529521575
```

```
has relative error -0.045452916374717885.
```

[90]:
```
[<matplotlib.lines.Line2D at 0x7fd835cdc580>]
```



[72]:
```
w = weights=ones(o.shape[1])
print(np.bincount(o[0],weights=w))
w[0]=10
print(np.bincount(o[0],weights=w))
```

```
[ 2.  7. 11.]
[ 2.  7. 20.]
```

Appendix_C

May 13, 2022

Appendix C: Using PyMC3 for Approximate Parameter Estimation in Reporter Model A

Nicholas Lines

We will explore in this notebook how well parameter estimation can be performed using PyMC3 (rather bluntly).

1 Environment Setup

[1]:
```
%pylab inline
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
```

[2]:
```
import arviz as az
import numpy as np
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
import graphviz
import os
import pymc3 as pm
from pymc3 import Model, Normal, HalfNormal, Bernoulli, Deterministic, Uniform
from pymc3 import find_MAP
```

[3]:
```
import scipy.stats as st
```

[4]:
```
from tqdm.notebook import tqdm
```

2 Experiment

Generate sample data.

[233]:
```
I = 20
T = 100
hidden_delta = np.random.uniform(high=0.5)
hidden_beta = np.random.uniform(size=I)
hidden_epsilon = np.random.uniform(size=I)
hidden_f_array = np.random.uniform(size=(T))<= hidden_delta
l = hidden_f_array.shape[0]
hidden_f = np.concatenate([hidden_f_array.reshape((l,1))]*I,axis=1)
hidden_alpha = np.random.uniform(size=(T,I)) <= hidden_beta.reshape(1,-1)
hidden_tau   = np.random.uniform(size=(T,I)) <= hidden_epsilon.reshape(1,-1)
hidden_r     = np.random.uniform(size=(T,I)) <= 0.5
observed_data = (hidden_tau*(hidden_alpha * hidden_f +␣
 ↪(1-hidden_alpha)*hidden_r) + 2*(1-hidden_tau))
```

[236]:
```
reporter_model = Model()
with reporter_model:
    delta = Uniform(name="delta", lower=0, upper=1)
    f = Bernoulli(name="f", p=delta,shape=T)
    beta = Uniform(name="beta", lower=0, upper=1, shape=I)
    epsilon = Uniform(name="epsilon", lower=0, upper=1, shape=I)
    alpha = Bernoulli(name="alpha", p=beta,shape=(T,I))
    tau = Bernoulli(name="tau", p=epsilon,shape=(T,I))
    r = Bernoulli(name="r", p=0.5,shape=(T,I))
    o = [Deterministic("o_"+str(t), tau[t,:] * (alpha[t,:] * f[t] + (1-alpha[t,:
 ↪])*r[t,:]) + 2*(1-tau[t,:])) for t in range(T)]
    X = [Normal("X_"+str(t),mu=o[t],sigma=0.01,observed=observed_data[t]) for t␣
 ↪in range(T)]
```

This next cell is handy for smaller models, but overwhelming for one of this size!

[ ]:
```
#pm.model_to_graphviz(reporter_model)
```

[237]:
```
with reporter_model:
    trace = pm.sample(1000)
```

```
Multiprocess sampling (2 chains in 2 jobs)
CompoundStep
>NUTS: [epsilon, beta, delta]
>BinaryGibbsMetropolis: [f, alpha, tau, r]
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

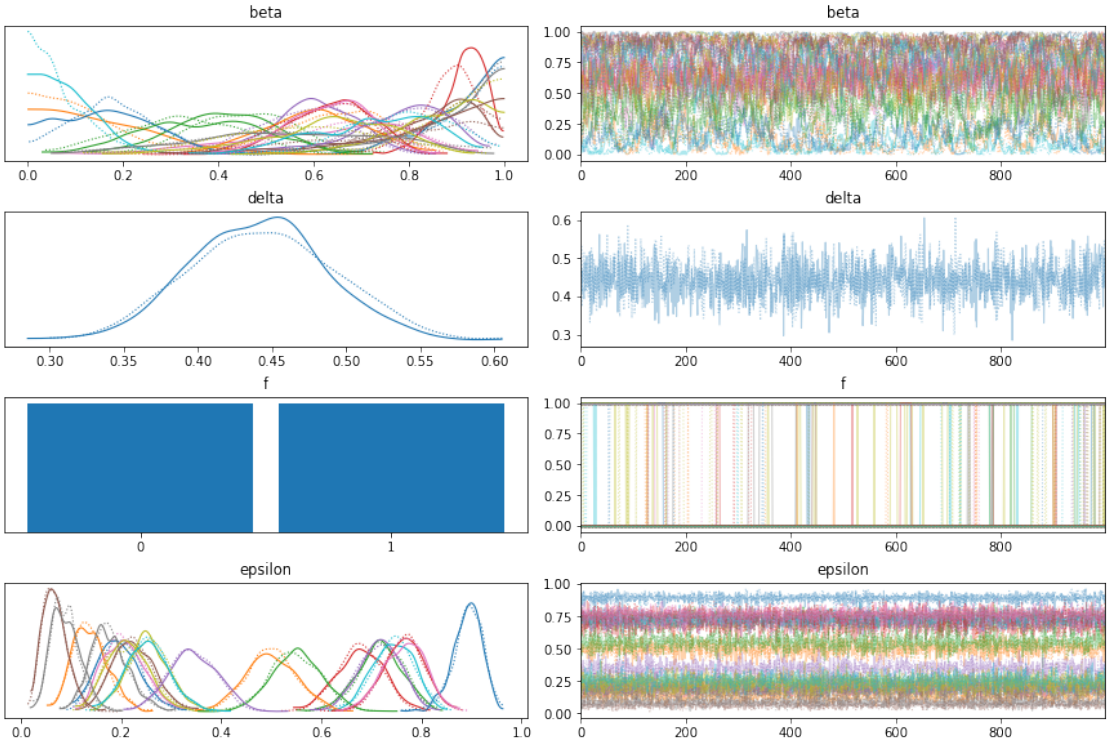
```
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws
total) took 5095 seconds.
The rhat statistic is larger than 1.4 for some parameters. The sampler did not
converge.
The estimated number of effective samples is smaller than 200 for some
```

parameters.

[266]:
```
import pickle
with open("reporter.pkl","wb") as pfile:
    pickle.dump(trace, pfile)
```

[268]:
```
f = figure()
pm.traceplot(trace,var_names=["beta","delta","f","epsilon"]);
savefig("pymc3results.jpg",dpi=700)
```

```
<Figure size 432x288 with 0 Axes>
```



[242]:
```
trace['delta'].mean()
```

[242]: 0.4421892276331697

[243]:
```
hidden_delta
```

[243]: 0.3512355494942204

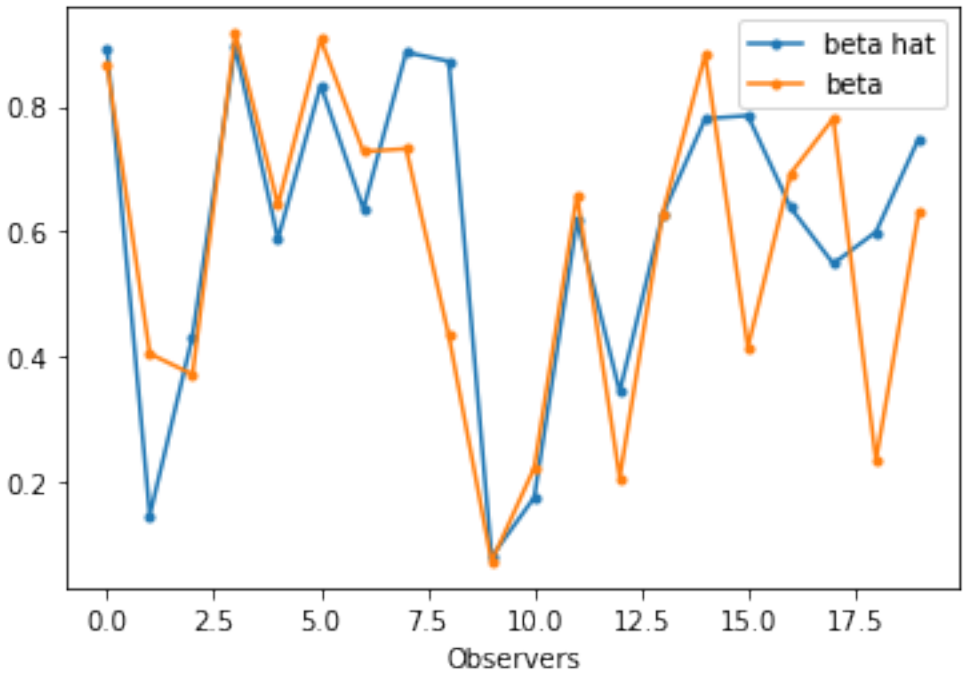
[265]:
```
trace['epsilon']
```

[265]: 0.4245300329268061

[244]:
```
print(trace['beta'].mean(axis=0))
print(hidden_beta)
```

```
[0.89283702 0.14631981 0.43258268 0.89590561 0.58804162 0.83438561
 0.63614918 0.88652525 0.87235246 0.07964526 0.1747446  0.61834232
 0.34392017 0.62604359 0.7805043  0.78534801 0.6383671  0.5485854
 0.5994691  0.74881558]
[0.86583519 0.40446965 0.37147047 0.91689329 0.64265487 0.90750678
 0.72799183 0.73231331 0.43312468 0.07210774 0.22201181 0.65898232
 0.20605286 0.62742296 0.882989   0.41511239 0.69260092 0.78093816
 0.23401369 0.63131669]
```

[280]:
```
f = figure()
plot(trace['beta'].mean(axis=0),'.-',label="beta hat")
plot(hidden_beta, '.-', label='beta');
xlabel("Observers")
legend();
savefig("betahat.jpg",dpi=700)
```

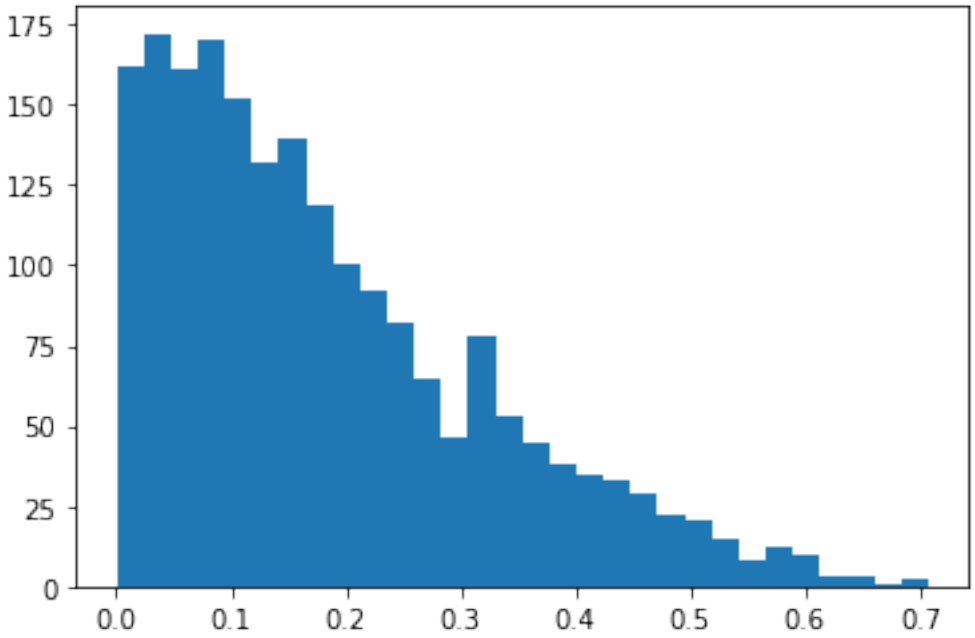


[262]:
```
f_accuracy = (st.mode(trace['f'])[0].flatten()==hidden_f[:,0].flatten()).sum()/
  ↪hidden_f.shape[0]xlabel
```

[263]:
```
print(f_accuracy)
```

```
0.98
```

[230]:
```
guessed_delta = trace['delta'].mean()
hist(trace['delta'],bins=30);
```



Reporter_Code

May 13, 2022

The following Python code is a reproduction of `reporter.py`.

[ ]:
```
"""
Functions and objects to aid in Reporter modeling.

We will use the Google standard for docstring formatting.
"""

import scipy.stats as st
from tqdm.notebook import tqdm
import numpy as np


def bin_not(n):
    '''Just switch 1 and 0.'''
    return 1 - 1*n


def proportionate_threshold_agreement(x,y,t=0.5):
    """Determine how similar two n-long arrays are in a threshold sense.

    This function first labels which entries of x are less than t and which
    entries of y are less than t, then sums up how many disagreements there
    are between these two label vectors, and divides this by the length n.
    That gives us the proportionate disagreement; we subtract from 1 to get
    the proportionate agreement.
    """
    return (1-((x<t)^(y<t)).sum()/x.shape[0])


def generate_data(n_i, n_t, r_p=0.5):
    """Generate data for one topic using Reporter Model A.

    Args:
        n_i (int): The number of observers.
        n_t (int): The number of time steps.
        r_p (float): Noise parameter.
```

```
    Returns:
        o (array): The observed deterministic observations.
        delta (float): The (hidden) true delta (topic bias to 1).
        f (array): The (hidden) true f values (binary facts).
        beta (array): The (hidden) observer awareness biases.
        alpha (array): The (hidden) observer awareness values.
        epsilon (array): The (hidden) observer communication biases.
        tau (array): The (hidden) observer communication choice values.
        r (array): The (hidden) random noise.
    """
    delta = st.uniform.rvs(0,1)
    epsilon = st.uniform.rvs(0,1,n_i)
    beta = st.uniform.rvs(0,1,n_i)
    f = st.bernoulli(delta).rvs(n_t)
    tau = np.array([st.bernoulli(epsilon_ij).rvs(n_t) for epsilon_ij in␣
 ↪epsilon])
    alpha = np.array([st.bernoulli(beta_ij).rvs(n_t) for beta_ij in beta])
    r = np.array([st.bernoulli(r_p).rvs(n_t) for i in range(n_i)])
    o = np.array([[tau[i][t] * (alpha[i][t] * f[t] +␣
 ↪bin_not(alpha[i][t])*r[i][t]) + 2 * \
                  bin_not(tau[i][t]) for i in range(n_i)] for t in range(n_t)])
    return((o, delta, f, beta, alpha, epsilon,tau,r))


def evaluate_estimators_full(data, agreement_thresh=0.5, printing=True):
    """Evaluate Reporter Model A estimators

    Args:
        data (tuple): The output from generate_data, a tuple of
            (o, delta, f, beta, alpha, epsilon,tau,r).
        agreement_thresh (float): The agreement threshold for
            proportionate threshold agreement.
        printing (bool): Indicates whether to print info.

    Returns:
    """
    (o, delta, f, beta, alpha, epsilon,tau,r) = data
    n_t, n_i = o.shape
    # MLE for epsilons
    epsilon_hat = [1-np.argwhere(o[:,i]==2).shape[0]/(n_t) for i in range(n_i)]
    epsilon_norm = np.linalg.norm(epsilon_hat - epsilon)
    # We need to drop any timestep where we got absolutely no information.
    drop_rows = np.argwhere(np.all(o == 2, axis=1)).flatten()
    o = np.delete(o,drop_rows,0)
    n_t, n_i = o.shape
    # Concensus estimator for f
```

```
    b = np.array([st.mode([oit for oit in o[t] if oit!=2])[0][0] for t in␣
↪range(n_t)])
    f = np.delete(f,drop_rows)
    # Judge the quality of the f estimate
    concensus_f_accuracies = 1-(abs(b-f).sum()/b.shape[0])
    # Estimate the probability of observing 1 empirically
    p_o_i_bar_1 = [(o[:,i]==1).sum()/((o[:,i]!=2).sum()) for i in range(n_i)]
    # Estimate delta
    delta_hat = np.mean(b)
    # Estimate beta
    beta_hat = np.array([(2 * p_i - 1) / (2*delta_hat - 1) for p_i in␣
↪p_o_i_bar_1])
    # Judge the quality of the beta estimate
    beta_norm = np.linalg.norm(beta_hat - beta)
    beta_agreements = proportionate_threshold_agreement(beta_hat, beta,␣
↪agreement_thresh)
    if printing:
        print(f"Estimators are delta_hat={delta_hat}, b, beta_hat.")
        print(f"The norm distance between epsilon_hat and epsilon is␣
↪{epsilon_norm}.")
        print(f"The approximation b for f is {100*concensus_f_accuracies}%␣
↪accurate.")
        print(f"The approximation delta_hat for delta={delta} has relative␣
↪error {(delta_hat-delta)/delta}.")
        print(f"The norm distance between beta_hat and beta is {beta_norm}.")
        print(f"The beta_hat estimator recognizes betas less than␣
↪{agreement_thresh}")
        print(f"{beta_agreements*100}% of the time.")
    return(b, delta_hat, beta_hat)
```