A_Bayesian_Model_For_News_Reporting

March 26, 2022

A Bayesian Model for News Reporting

Nicholas Lines

1 Environment Setup

[2]:
```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

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

[4]:
```
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

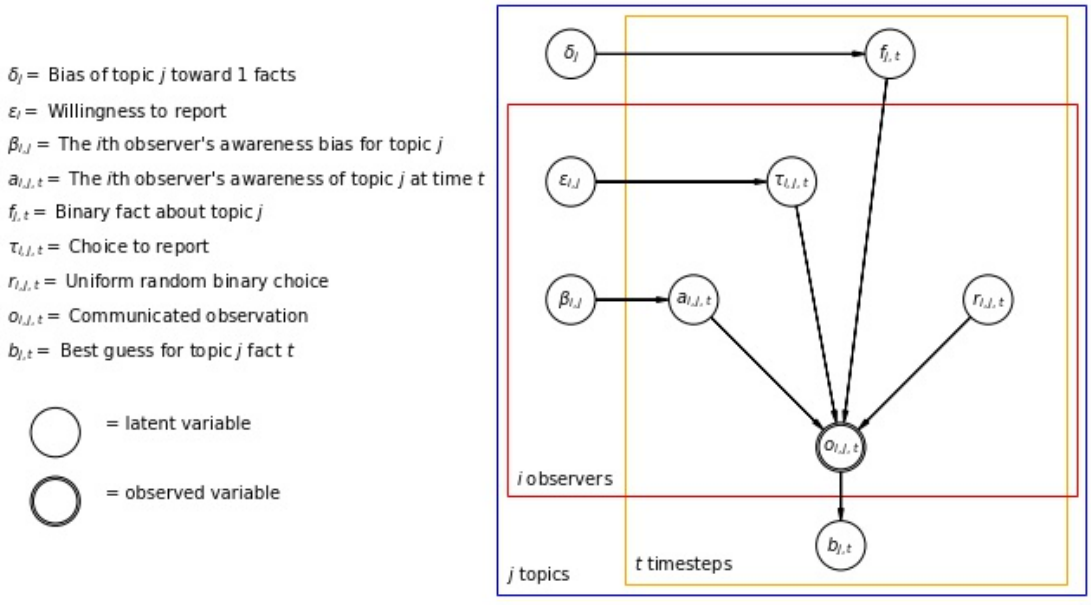
   $$o_{i,j,t} = \tau_{i,j,t}(a_{i,j,t} f_{j,t} + \tilde{a_{i,j,t}} r_{i,j,t}) + 2\tilde{\tau_{i,j,t}},$$

   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.

This model is shown in the diagram below.



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 $\tau_{\bar{i},j,t} = 1$ 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

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

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.

[5]:
```
def bin_not(n):
    '''Just switch 1 and 0.'''
    return 1 - 1*n
```

[260]:
```
True ^ True
```

[260]:
```
False
```

[262]:
```
x=array([1,0,0,1,0])
t = 0.5
print(x<t)
print(x<2)
(x<t) ^ (x<2)
```

```
[False  True  True False  True]
[ True  True  True  True  True]
```

[262]:
```
array([ True, False, False,  True, False])
```

[265]:
```
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])
```

[275]:
```
n_ts = [10, 100, 500, 1000, 5000, 10000]
n_j = 1 # One news topic
n_i = 20 # observers
n_t = 1000 # timesteps
n_d = 10 # number of deltas to try
average_norms = []
average_agreements = []
average_accuracies = []
for n_t in tqdm(n_ts):
#for n_i in tqdm([10,20,50]):
    norms = []
    agreements = []
    accuracies = []
    for d in range(n_d):
        delta = st.uniform.rvs(0,1,n_j)
        epsilon = st.uniform.rvs(0,1,n_i)
        beta = st.uniform.rvs(0,1,n_i)
        f = st.bernoulli(delta[0]).rvs(n_t)
        tau = array([st.bernoulli(epsilon_ij).rvs(n_t) for epsilon_ij in␣
↪epsilon])
        a = array([st.bernoulli(beta_ij).rvs(n_t) for beta_ij in beta])
        r = array([st.bernoulli(0.5).rvs(n_t) for i in range(n_i)])
        o = array([[tau[i][t] * (a[i][t] * f[t] + bin_not(a[i][t])*r[i][t]) + 2␣
↪* bin_not(tau[i][t]) for i in range(n_i)] for t in range(n_t)])
```

```
        #o = array([[1 * (a[i][t] * f[t] + bin_not(a[i][t])*r[i][t]) + 0 *␣
↪bin_not(tau[i][t]) for i in range(n_i)] for t in range(n_t)])
        # We need to drop any timestep where we got absolutely no information.
        drop_rows = argwhere(np.all(o == 2, axis=1)).flatten()
        o = delete(o,drop_rows,0)
        n_t = o.shape[0]
        b = array([st.mode([oit for oit in o[t] if oit!=2])[0][0] for t in␣
↪range(n_t)])
        f = delete(f,drop_rows)
        accuracies.append(1-(abs(b-f).sum()/b.shape[0]))
        #print(f"The approximation b for f is {100*(1-(abs(b-f).sum()/b.
↪shape[0]))}% accurate")
        p_o_i_bar_1 = [(o[:,i]==1).sum()/((o[:,i]!=2).sum()) for i in␣
↪range(n_i)]
        #delta_hat_j = (o==1).sum()/((o!=2).sum())
        delta_hat_j = mean(b)
        beta_hat_j = array([(2 * p_i - 1) / (2*delta_hat_j - 1) for p_i in␣
↪p_o_i_bar_1])
        #beta_hat_j = array([(2 * p_i - 1) / (delta[0] - 1) for p_i in␣
↪p_o_i_bar_1])
        norms.append(norm(beta_hat_j - beta))
        agreements.append(proportionate_threshold_agreement(beta_hat_j, beta, 0.
↪5))
    average_norms.append(mean(norms))
    average_agreements.append(mean(agreements))
    average_accuracies.append(mean(accuracies))
```

```
  0%|          | 0/6 [00:00<?, ?it/s]
```

```
<ipython-input-275-15ce4343c3a5>:32: RuntimeWarning: invalid value encountered
in long_scalars
  p_o_i_bar_1 = [(o[:,i]==1).sum()/((o[:,i]!=2).sum()) for i in range(n_i)]
<ipython-input-275-15ce4343c3a5>:35: RuntimeWarning: divide by zero encountered
in double_scalars
  beta_hat_j = array([(2 * p_i - 1) / (2*delta_hat_j - 1) for p_i in
p_o_i_bar_1])
<ipython-input-275-15ce4343c3a5>:35: RuntimeWarning: invalid value encountered
in double_scalars
  beta_hat_j = array([(2 * p_i - 1) / (2*delta_hat_j - 1) for p_i in
p_o_i_bar_1])
```

[276]:
```
average_norms
```

[276]:
```
[nan, nan, nan, 1.3404982161762702, 0.6238999004082298, 0.364013256298419]
```

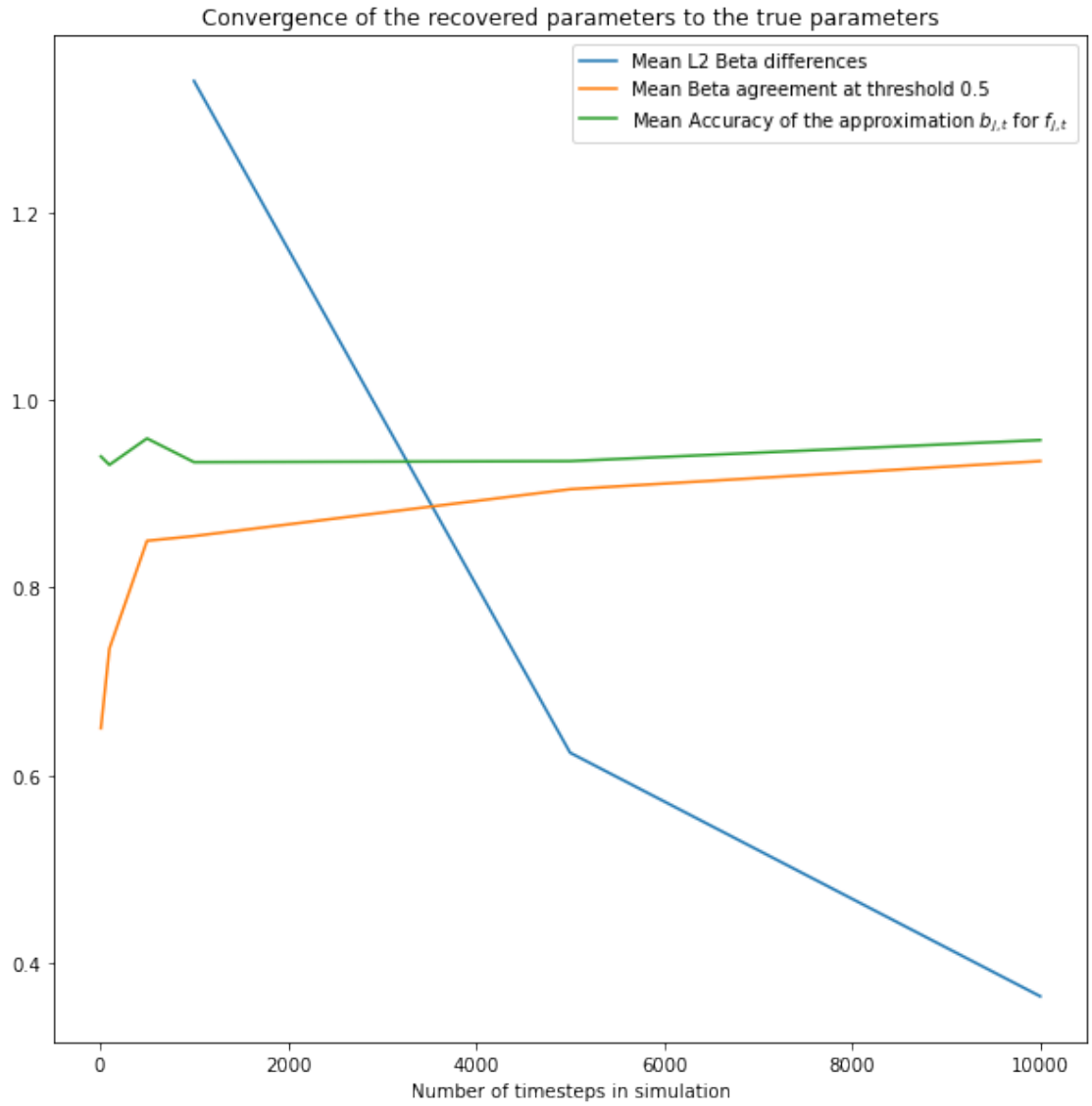
[277]:
```
average_agreements
```

[277]:
```
[0.65,
 0.7349999999999999,
 0.85,
 0.8550000000000001,
 0.9049999999999999,
 0.9350000000000002]
```

[278]:
```
average_accuracies
```

[278]:
```
[0.9399999999999998,
 0.9309999999999998,
 0.9591999999999998,
 0.9339000000000001,
 0.9349999999999999,
 0.9574400000000001]
```

[281]:
```
figure(figsize=(10,10))
plot(n_ts, average_norms, label="Mean L2 Beta differences")
plot(n_ts, average_agreements, label="Mean Beta agreement at threshold 0.5")
plot(n_ts, average_accuracies, label="Mean Accuracy of the approximation␣
 ↪$b_{j,t}$ for $f_{j,t}$")
legend();
xlabel("Number of timesteps in simulation");
title("Convergence of the recovered parameters to the true parameters");
```



5 Next steps

The next steps for this project are as follows.

1. Demonstrate how to prune reporting edges from uninformed observers.
    a. Prune by convergence of $\hat{\beta}$
    b. Prune by thresholding (some minimum number of timesteps)
2. Show that this results in approximations of the fact variable comparable to the unpruned version.

5.1 Pseudocode for the Remaining Work

We'll now describe the same steps using pseudocode.

```
set threshold_for_min_iterations
set threshold_awareness
set threshold_convergence
for delta in deltas:
    set iterations = 0
    set c_pruned_observers = list()
    set t_pruned_observers = list()
    set beta_hats = dict()
    draw epsilons
    draw betas
    for timestep in timesteps:
        iterations += 1
        draw fact
        for observer in observers:
            draw a
            draw tau
            draw r
            form o
            form beta_hat
            update beta_hats[observer]
        create b
        create t_pruned_b
        create c_pruned_b
        for observer in observers:
            if |beta_hat[observer][-1]-beta_hat[observer][-2]| < threshold_convergence:
                c_pruned_observers.append(observer)
            if iterations > threshold_for_min_iterations:
                if beta_hat[observer][-1] < threshold_awareness:
                    t_pruned_observers.append(observer)

        compare b, c_pruned_b and t_pruned_b to f
```