# SURF 2017 INTERIM REPORT 1

V. CHEN
ADVISED BY A. M. STUART AND M. M. DUNLOP

**1. Introduction.** In the context of semi-supervised learning, the *clustering* problem refers to attempting to group a set of data or objects into clusters of similar objects. Applications of clustering include image segmentation, social network analysis, and voter classification, but the reader will readily identify numerous other potential applications. We approach the clustering problem from a graph-theoretical point of view, which assigns a node to each data value and weights on edges between nodes to indicate how related the data points are.
Let $Z$ be a set of nodes $\{1, \ldots N\}$. The feature vectors are given by $x : Z \to \mathbb{R}^d$. $Z' \subset Z$ is a subset of labeled nodes, for which we have label data given by $y : Z' \to \{-1, 1\}$ for **binary** data, and $y : Z' \to \{e_1, \ldots e_k\}$ for $k$**-ary** (multiclass) data.
The classifying function is given by the vector $u$, and the label model is given by $y(\ell) = S(u(\ell))$ for $l \in Z'$ (with possible additional corruption by noise). In the case of binary data, $y(\ell) \in \{\pm 1\}$, and for $v \in \mathbb{R}$,

$$S(v) = \begin{cases} 1 & v > 0 \\ 0 & v = 0 \\ -1 & v < 0. \end{cases}$$

In the case of multiclass data, $y(\ell) \in \{e_1, \ldots e_k\}$, $v \in \mathbb{R}^k$ and

$$S(v) = e_p, \quad p = \operatorname*{argmax}_{r=1,\ldots,k} v_r$$

where $\{e_l\}_{l=1}^k$ denotes the standard basis of $\mathbb{R}^k$.
Given this problem, one goal is to come up with a classification that best fits the data. The research group in which I am working is currently looking at the effectiveness of incorporating Bayesian inference into semi-supervised learning. In the Bayesian approach, the unknown classifying function $u$ is viewed as a random variable, whose distribution is determined by combining the labeling model $y$ with a probability distribution representing our prior beliefs about the classification and derived from the feature vectors. Hierarchical Bayesian methods provide additional flexibility by introducing hyperparameters governing the prior distribution, which are in turn treated as random variables to be inferred.
Our prior beliefs come from *spectral clustering*, which uses the clustering properties of the eigenvalues and eigenvectors of the graph Laplacian, a matrix defined on the similarity graph constructed from the feature vectors as follows:
Let the similarity graph be $G$ with weights $w_{ij} \geq 0$. A larger weight indicate a higher degree of similarity between the two nodes. Define $\delta_i = \sum_{j=1}^N w_{ij}$. Then, define matrices $W, D$ as:

$$W = w_{ij}, \quad D = \begin{cases} \delta_i & i = j \\ 0 & i \neq j. \end{cases}$$

Then, let $L = D - W$ denote the unnormalized graph Laplacian on $\mathbb{R}^N$. Also, denote $L_{\text{sym}} = D^{-1/2} L D^{-1/2}$ as the symmetric normalized graph Laplacian, and $L_{\text{rw}} = D^{-1} L$ as the random walk normalized graph Laplacian. Denote the eigenbasis of $L$ by

$$L q_j = \lambda_j q_j, \quad j = 0, \ldots, N-1. \tag{1}$$

We order the eigenvectors $q_0, \ldots q_{N-1}$ in increasing order of their corresponding eigenvalues. These eigenvalues are always non-negative [6]. The graph Laplacian is important because the eigenvectors corresponding to the smaller eigenvalues are known to hold information about cluster data. In particular, if the graph has "ideal" clusters so that the edges between clusters all have weight 0, then the graph Laplacian would have as many eigenvalues 0 as there are clusters, and corresponding eigenvectors would be indicators for the clusters [6]. Even if the data is not ideal, the eigenvectors corresponding to small eigenvalues should hold clustering information. This includes the important Fiedler vector, the second-smallest eigenvector which is useful for bi-partitioning the graph. The eigenvalues and eigenvectors of the graph Laplacian are used to construct the prior probability distribution about $u$.

For the Bayesian approach, the prior, likelihood and posterior are defined as follows:

$\mathbb{P}(u)$: **Prior** distributed as $\mathsf{N}(0, C)$ where the covariance matrix $C = f(L)^2$ in general, where $f$ is a function that we choose. A sample $u$ from this prior can be made through a realization of the following random sum:

$$u = \sum_{j=0}^{M} f(\lambda_j)\xi_j q_j, \quad \xi_j \sim \mathsf{N}(0,1) \quad \text{i.i.d.} \tag{2}$$

Looking at the form of the sample, it would make sense to impose the restriction that $f$ should be monotonically decreasing so that lower eigenvectors dominate the distribution, a belief consistent with our discussion about graph Laplacian eigenvectors above.

We will be mostly using a fixed function $f$, chosen as $f(\lambda) = (\lambda + \tau^2)^{-\alpha/2}$. In this case, $C = (L + \tau^2 I)^{-\alpha}$, where $\tau$ and $\alpha$ are hyperparameters that govern the prior, and a sample $u$ from the prior can be made through the following:

$$u = \sum_{j=0}^{M} (\lambda_j + \tau^2)^{-\alpha/2}\xi_j q_j, \quad \xi_j \sim \mathsf{N}(0,1) \quad \text{i.i.d.} \tag{3}$$

Note that this $f$ is indeed monotonically decreasing.

Finally, one other possibility is to learn $f$ as well.

$\mathbb{P}(y|u)$: **Likelihood**, given by $\exp(-\Phi(u))$.

If we believe there is uncertainty in the labels, we could represent this uncertainty as additive Gaussian noise. For $\gamma > 0$, $\Phi(u)$ is chosen as

$$\Phi(u) = \frac{1}{2\gamma^2} \sum_{\ell \in Z'} |y(\ell) - S(u(\ell))|^2 \tag{4}$$

as introduced in [4]. This definition can be derived from the **Bayesian level-set** model for label uncertainty, which assumes $y(j) = S(u(j)) + \eta_j, \eta_j \sim \mathsf{N}(0, \gamma^2)$ i.i.d. Then, the conditional distribution $y(j)|u \sim \mathsf{N}(S(u(j)), \gamma^2)$. We can compute the likelihood as follows:

$$\mathbb{P}(y|u) = \prod_{j \in Z'} \mathbb{P}(y(j)|u) \propto \prod_{j \in Z'} \exp\left(-\frac{(y(j) - S(u(j)))^2}{2\gamma^2}\right).$$

So $\Phi(u) = -\log \mathbb{P}(y|u) = \frac{1}{2\gamma^2} \sum_{\ell \in Z'} |y(\ell) - S(u(\ell))|^2$ as desired.

If we believe that the label data is certain, $\Phi(u)$ is defined as

$$\Phi(u) = \begin{cases} 0 & S(u(l)) = y(l) \quad \forall l \in Z' \\ \infty & \text{otherwise.} \end{cases} \tag{5}$$

This case can be viewed as the $\gamma \to 0$ limit of (4) to enforce that $u$ must perfectly respect the labels. Notice that $\exp(-\Phi(u)) = 1$ if and only if the classifying function $u$ respects the label data. Otherwise, $\exp(-\Phi(u)) = 0$.

One other model for the likelihood is the **probit** model, also used in [1], in which we add noise to the classifying function before taking the sign function. Specifically, we assume $y(j) = S(u(j) + \eta_j), \eta_j \sim \mathsf{N}(0, \gamma^2)$ i.i.d.. Denote the normal cumulative distribution function

$$\psi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp\left(-t^2/2\right) dt$$

Then,

$$\begin{aligned}\mathbb{P}(y(j)=1|u)=\mathbb{P}(\mathsf{N}(u(j),\gamma^2)>0)&=\frac{1}{\sqrt{2\pi\gamma^2}}\int_0^\infty \exp\left(\frac{-(z-u(j))^2}{2\gamma^2}\right)dz\\&=\frac{1}{\sqrt{2\pi}}\int_{-\frac{u(j)}{\gamma}}^\infty \exp\left(-t^2/2\right)dt\\&=1-\psi(-u(j)/\gamma)\\&=\psi(u(j)/\gamma)\end{aligned}$$

where we used the fact that $1-\psi(-x)=\psi(x)$.
Similarly, we can compute $\mathbb{P}(y(j)=-1|u)=\psi(-u(j)/\gamma)$. Since the labels $y(j)$ are either 1 or $-1$, we can write $\mathbb{P}(y(j)|u)=\psi(y(j)u(j)/\gamma)$. Finally,

$$\mathbb{P}(y|u)=\prod_{j\in Z'}\mathbb{P}(y(j)|u)=\prod_{j\in Z'}\psi(y(j)u(j)/\gamma)$$

and we can see that we would define

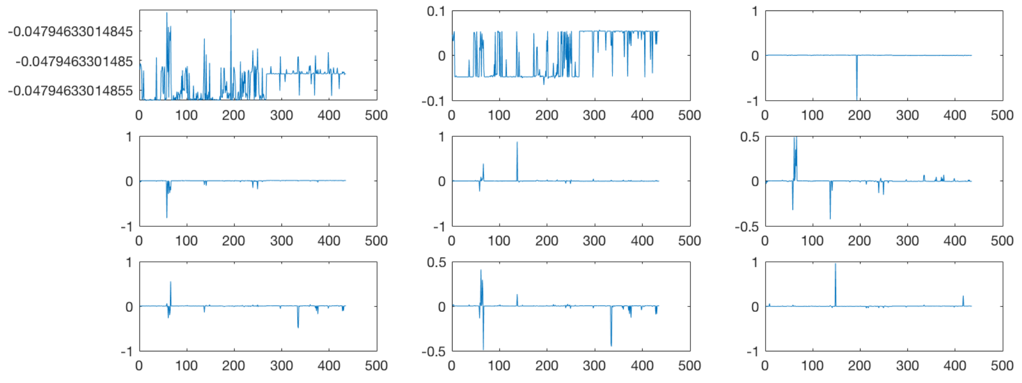
$$\Phi(u)=-\sum_{j\in Z'}\log\left(\psi(y(j)u(j)/\gamma)\right) \tag{6}$$

$\mathbb{P}(u|y)$: **Posterior**, which by Bayes' theorem is proportional to $\mathbb{P}(u)\mathbb{P}(y|u)$

**2. Model problems.** Applications to be considered include the following:

**2.1. Voting records.** A data set of the voting records on 16 votes of 435 individuals from the U.S. House of Representatives. The data is ordered so that all the representatives from the party are contiguous in the indexing. The graph has $N=435$ nodes, with feature vectors that are $d=16$ dimensional. Each component of the feature vectors is either $+1,-1$, or 0 to indicate voting for, voting against, or abstaining on a particular bill, respectively. On this data set, we want to perform binary classification with a subset of labeled data. For this problem, we will use the unnormalized Laplacian with fixed length-scale weights. That is, we will define $w_{ij}=\exp\left(-\frac{d(x_i,x_j)}{2l^2}\right)$, $d_{p,q}(x,y)=||x-y||_p^q$ with $p=2,q=2,l=1$. Figure 1 is part of the spectrum of this choice of Laplacian.

FIG. 1. *Lowest 9 eigenvectors of unnormalized $L$ of voting records.*



**2.2. Two moons.** This is a synthetic data set constructed with two half circles in $\mathbb{R}^2$ with radius one. These are embedded in $\mathbb{R}^{100}$, and data points are sampled from the circles with Gaussian noise distributed as $\mathsf{N}(0,\sigma^2)$ added to each of the 100 dimensions. In this data set, we will be generating realizations of two moons with $N=1000$ or $N=2000$ nodes, each associated with $d = 100$ dimensional feature vectors. This data is again ordered, with the first $N/2$ nodes generated from the first half circle, and the latter $N/2$ from the second. Again, here we want to perform binary classification into the two half circles from which the data was generated. We will be using the self-tuning, symmetric Laplacian for this data set, introduced in [7]. This Laplacian matrix has weights that infer the local spatial scale from the data, removing the need to choose a fixed length-scale parameter. The use of this self-tuning Laplacian in the two-moons data set seems to encode more information in the eigenvectors. Compare Figure 2 with Figure 3.

FIG. 2. *Lowest* 9 *eigenvectors of self-tuning symmetric* $L$ *of two moons,* $\sigma = 0.04$.

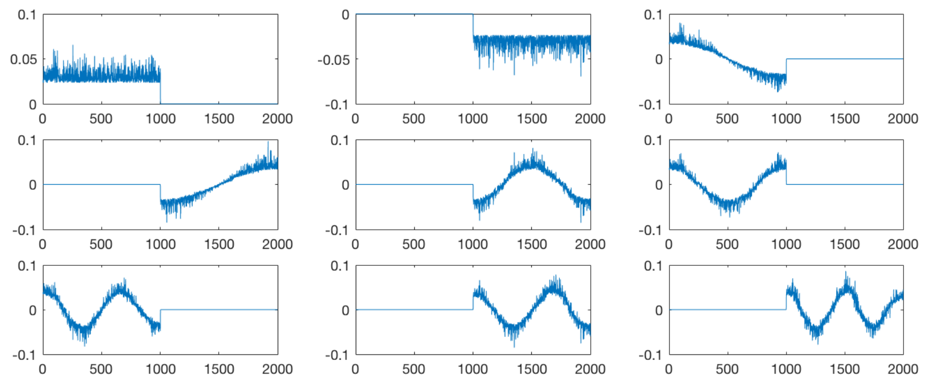
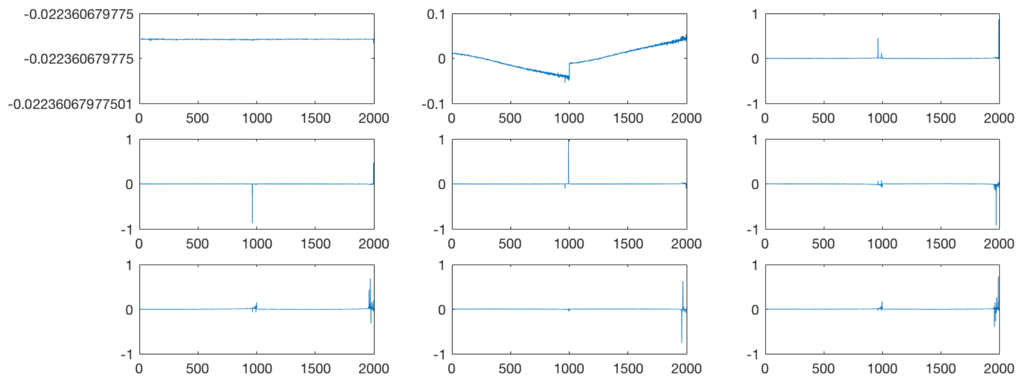


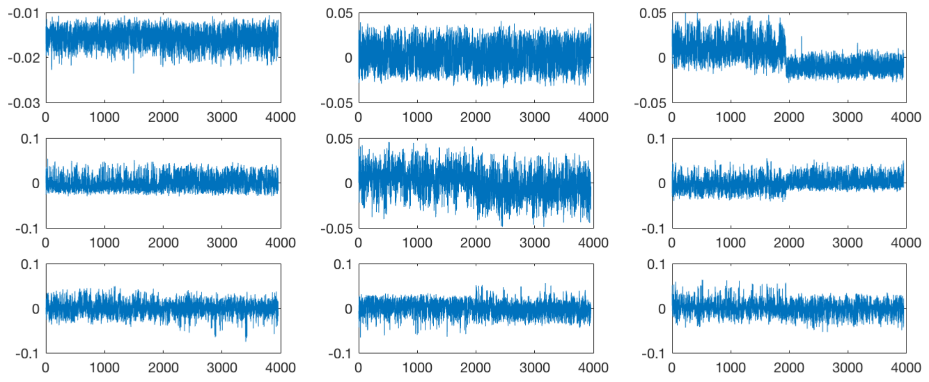
FIG. 3. *Lowest* 9 *eigenvectors of unnormalized fixed length* $L$ *of two moons,* $\sigma = 0.04$.



**2.3. MNIST data sets.** This data set contains 70,000 images of $28 \times 28$ pixels with handwritten digits 0 through 9. The feature vectors are $d = 400$ dimensional and are formed by projection onto the first 50 PCA components. We will focus on binary classification between the digits 4 and 9. We will use the self-tuning, symmetric normalized Laplacian again. See Figure 4.

**3. Models.** My project looks at the effectiveness of different Bayesian hierarchical clustering algorithms. I am implementing these algorithms in MATLAB and testing them on the data sets mentioned. The complexity of the models considered will evolve as follows:
For models (A), (B), and (C), the hyperparameters $\tau, \alpha, M$ refer to the prior in (3)
(A) Use fixed $\tau, \alpha, M = N - 1$ to cluster data.
(B) Learn $\tau, \alpha$; fix $M = N - 1$.
(C) Learn $\tau, \alpha, M$.
For models (D) and (E), samples from a new prior are given by $u = \sum_{j=0}^{M} v_j \xi_j q_j$.

FIG. 4. *Lowest 9 eigenvectors of self-tuning symmetric $L$ of MNIST, digits 4 and 9.*



(D) Learn $\{v_j\}_{j=0}^M$ with $M$ fixed.
(E) Learn $\{v_j\}_{j=0}^M$ and $M$.
Model (F) can apply to all of the previous models.
(F) Multiclass, hierarchical on number of classes.
Model (G) focuses on learning the arbitrary function $f$ defined in (2).
(G) Learn $f(\lambda), M$.
To learn $\tau, \alpha$, we will assume uniform priors on intervals $U(0, c)$ for constants $c$. For $M$, we will assume an exponentially or algebraically decaying prior with the reasoning that larger $M$ being less likely. We will refer to the hyperparameters as $\theta$.

## 4. Approach.

One main goal of this project is to study the effectiveness of different hierarchical models. To do this, I have been understanding the algorithms and implementing them in MATLAB. In MATLAB, it is easy to visualize the algorithm by plotting the traces of the hyperparameters and the running averages. I can test the algorithms on the data sets and change parameters to see how they affect the clustering effectiveness. I have been using the voting records and the two moons data sets so far. I also made movies of the algorithms running on these data sets by printing figures and using a video-making tool. These movies are useful for both debugging and assessing the effectiveness of the algorithms. Metrics that are useful for comparing the algorithms include the percent classification accuracy, the variance, and the convergence time. In particular, in our applications on the voting records, the two moons, and the MNIST data sets, the classification accuracy is well defined because there is a true classification. Assessing uncertainty quantification may be more difficult if there is no clear truth.
The important next step of this project is to write a test suite for the different algorithms that I have implemented so far. We want to create a figure similar to Figure 13 from [1], plotting the classification accuracies of the different algorithms over increasing noise variance in the two moons data set. We would like to see if the hierarchical methods lead to improving classification accuracies.

## 5. Algorithms.

This section discusses the algorithms that I have implemented for the hierarchical models.

### 5.1. Model (A).

For the nonhierarchical model (A), I implemented a pCN algorithm first introduced in [2] and described in Algorithm 1.

### 5.2. Model (B).

To implement model (B), which is hierarchical, we need to sample conditional distributions that govern $u, \tau$, and $\alpha$. When we attempt to sample the posterior distribution in the hierarchical methods, we could use the Gibbs sampler. The basic form of Gibbs sampling has two repeated steps:

- Update $u^{(n+1)} \sim u|\theta^{(n)}, y$
- Update $\theta^{(n+1)} \sim \theta|u^{(n+1)}, y$

**Algorithm 1** General pCN adapted from [3]

```
Select $u^{(0)}$. Select $\tau, \alpha$. Select $\beta \in [0, 1]$
for $k = 0$ to $S$ do
    Sample $v$ from the prior distribution given in (3)
    Set $\hat{u}^{(k)} = (1 - \beta^2)^{1/2} u^{(k)} + \beta v$
    Set $\alpha(u^{(k)} \to \hat{u}^{(k)}) = \min(1, \exp(\Phi(u^{(k)}) - \Phi(\hat{u}^{(k)})))$
    Set $u^{(k+1)} = \hat{u}^{(k)}$ with probability $\alpha(u^{(k)} \to \hat{u}^{(k)})$, and set $u^{(k+1)} = u^{(k)}$ otherwise
end for
return $\{u^{(k)}\}$
```

We use $\theta$ to represent the hyperparameters, which could include $\tau, \alpha$, and $M$. However, we cannot sample the conditional distributions directly, so we replace direct sampling with Markov Chain Monte Carlo (MCMC) indirect sampling methods, which are invariant with respect to each of the exact conditional distributions. With Metropolis-within-Gibbs, at every step we update $u^{(n+1)}, \tau^{(n+1)}$ and $\alpha^{(n+1)}$ each with MCMC to target these conditional distributions. One method is to update $u, \tau, \alpha$ in a block, by proposing $(u, \theta) \to (\hat{u}, \hat{\theta})$ and computing the transition probability for this step.
The algorithms that we will implement will be slightly different, as we will independently propose $u \to \hat{u}, \tau \to \hat{\tau}, \alpha \to \hat{\alpha}, M \to \hat{M}$, and perform these transitions in separate steps. At each step, we fix all but the proposed parameter, and we compute the transition probability. This is the algorithm that we will be using for the following hierarchical algorithms.
Our first of two hierarchical algorithms for model (B) is deemed "centered," as contrasted with the second algorithm described later in this section. Using Metropolis-within-Gibbs, we will require an expression for the posterior, so define $f(u, \tau, \alpha)$ as the joint posterior distribution. By Bayes' theorem,

$$f(u, \tau, \alpha) \propto \exp(-\Phi(u)) \times \mathbb{P}(u, \tau, \alpha) = \exp(-\Phi(u)) \mathbb{P}(u|\theta) \pi_0(\theta).$$

Recall from (3) that the prior is distributed as $\mathsf{N}(0, C)$ with the covariance matrix $C(\theta) = C(\tau, \alpha) = (L + \tau^2 I)^{-\alpha}$. We can write

$$(7) \qquad f(u, \tau, \alpha) \propto \frac{1}{\sqrt{(2\pi)^d \det C(\tau, \alpha)}} \exp\left(-\Phi(u) - \frac{1}{2}\langle u, C(\tau, \alpha)^{-1} u\rangle + \log(\pi_0(\tau, \alpha))\right).$$

The normalization constant $\det(C(\theta))$ depends on $\tau, \alpha$ now and does not cancel out. Using this expression for the posterior, I implemented the Metropolis-within-Gibbs method for model (B) in Algorithm 2.
We also looked at a different parameterization for model (B). Algorithm 3 uses the variable $\xi$ that is related to the classifying function $u$ by

$$(8) \qquad T(\xi, \tau, \alpha) = \sum_{i=0}^{M} \frac{1}{(\lambda_i + \tau^2)^{\alpha/2}} \xi_i q_i = u$$

following (3). This algorithm is "non-centered" compared to the centered parameterization given in Algorithm 2, meaning that the unknown classifying variable $\xi$ and the parameters $\theta = (\tau, \alpha)$ are a priori independent [5].
This model assumes the prior $\xi \sim \mathsf{N}(0, I)$. Similar to the centered case, define $g(\xi, \tau, \alpha)$ as the joint posterior distribution. By Bayes' theorem,

$$g(\xi, \tau, \alpha) \propto \exp(-\Phi(T(\xi, \tau, \alpha))) \times \mathbb{P}(\xi, \tau, \alpha) = \exp(-\Phi(T(\xi, \tau, \alpha))) \mathbb{P}(\xi) \pi_0(\tau, \alpha)$$

Note that $\mathbb{P}(\xi) = \frac{1}{\sqrt{(2\pi)^d \det I}} \exp(-\frac{1}{2}\langle \xi, I\xi\rangle) \propto \exp(-\frac{1}{2}\langle \xi, \xi\rangle)$ and the normalization constants drop out. We obtain:

$$(9) \qquad g(\xi, \tau, \alpha) \propto \exp\left(-\Phi(T(\xi, \tau, \alpha)) - \frac{1}{2}\langle \xi, \xi\rangle + \log(\pi_0(\tau, \alpha))\right).$$

**Algorithm 2** Hierarchical on $\tau, \alpha$

```
Initialize $u^{(0)} = q_1$, the Fiedler vector expressed in the standard basis.
Initialize $\tau^{(0)}, \alpha^{(0)}$. Select $\beta \in [0,1]$
Pick $\epsilon_1, \epsilon_2$, the jump sizes for $\tau, \alpha$ respectively.
for $k = 0$ to $S$ do
    Sample $v$ from the prior distribution and expressed in the eigenbasis        ▷ $u|y, \tau, \alpha$.
    Expressing $u$ in the eigenbasis, set a proposal $\hat{u}^{(k)} = (1-\beta^2)^{1/2}u^{(k)} + \beta v$
    Set $u^{(k+1)} = \hat{u}^{(k)}$ with probability
        $$A(u^{(k)} \to \hat{u}^{(k)}) = \min\left\{1, \exp(\Phi(u^{(k)}) - \Phi(\hat{u}^{(k)}))\right\}$$
    Set a proposal $\hat{\tau}^{(k)} = \tau^{(k)} + \epsilon_1 t$ for $t \sim \mathsf{N}(0,1)$        ▷ $\tau|y, u, \alpha$
    Set $\tau^{(k+1)} = \hat{\tau}^{(k)}$ with probability
        $$A(\tau^{(k)} \to \hat{\tau}^{(k)}) = \min\left\{1, \frac{f(u^{(k+1)}, \hat{\tau}^{(k)}, \alpha^{(k)})}{f(u^{(k+1)}, \tau^{(k)}, \alpha^{(k)})}\right\}$$
(Using the eigenbasis representation of $u$ for computation)        ▷ $f$ defined in (7)
    Set a proposal $\hat{\alpha}^{(k)} = \alpha^{(k)} + \epsilon_2 a$ for $a \sim \mathsf{N}(0,1)$        ▷ $\alpha|y, u, \tau$
    Set $\alpha^{(k+1)} = \hat{\alpha}^{(k)}$ with probability
        $$A(\alpha^{(k)} \to \hat{\alpha}^{(k)}) = \min\left\{1, \frac{f(u^{(k+1)}, \tau^{(k+1)}, \hat{\alpha}^{(k)})}{f(u^{(k+1)}, \tau^{(k+1)}, \alpha^{(k)})}\right\}$$
end for
return $\{u^{(k)}, \tau^{(k)}, \alpha^{(k)}\}$
```

### 5.3. Model (C).

## 6. Voting records experiments.

**6. Voting records experiments.** In each of the following experiments, the labeled data $Z'$ was chosen consistently: members 20-30 was labeled one party and members 280-290 as the other.
For Algorithm 1, fixing $\gamma = 0.0001, \beta = 0.4, \tau = 1, \alpha = 1$, and running for 100000 iterations with a burn-in period of 1000, I could get clustering at about 85% accuracy. One final clustering obtained is shown in Figure 5, with relevant figures Figure 6 and Figure 7 that suggest that the chain has converged.

FIG. 5. *Algorithm 1 final average*

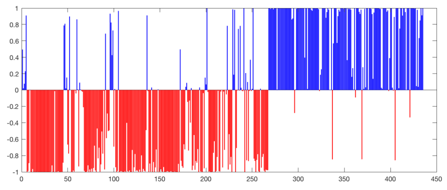
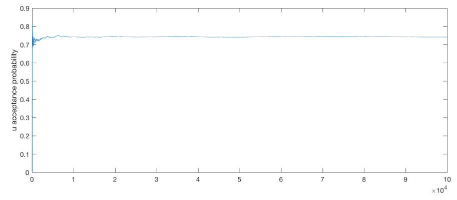


FIG. 6. *Algorithm 1 average u acceptance probability*



Experiments with Algorithm 2 on the voting records data indicated poor mixing in the hyper-parameters by looking at traces of $\tau, \alpha$. We decided that the problem was in part due to the irregularity of the larger eigenvectors, perhaps because of the accuracy of MATLAB's eig solver. Truncating the list of eigenvectors to only consider the first 50 seemed to help the problem, and we were able to observe better clustering. Fixing $\gamma = 0.0001, \beta = 0.4, \tau^{(0)} = 20, \alpha^{(0)} = 20, \tau \in [0.1, 60], \alpha \in [0.1, 60], \epsilon_\tau = 0.1, \epsilon_\alpha = 0.1$ and running 100000 iterations with a burn-in period of 1000, the accuracy is about 83%. See Figure 8, Figure 9, Figure 10, Figure 11.
Finally, the non-centered Algorithm 3 seems to converge faster than Algorithm 2, and truncation

**Algorithm 3** Non-centered parameterization: sampling $\xi, \tau, \alpha$

```
Choose $\xi^{(0)} \in \mathbb{R}^N, \alpha^{(0)}, \tau^{(0)} > 0, \beta \in (0,1]$ and $\epsilon_1, \epsilon_2 > 0$.
for $k = 0$ to $S$ do
    Propose $\hat{\xi}^{(k)} = (1-\beta^2)^{\frac{1}{2}}\xi^{(k)} + \beta\zeta^{(k)}$, $\zeta^{(k)} \sim \mathsf{N}(0, I)$
    Make transition $\xi^{(k)} \to \hat{\xi}^{(k)}$ with probability
        $$A(\xi^{(k)} \to \hat{\xi}^{(k)}) = \min\left\{1, \exp\left(\Phi(T(\xi^{(k)}, \tau^{(k)}, \alpha^{(k)})) - \Phi(T(\hat{\xi}^{(k)}, \tau^{(k)}, \alpha^{(k)}))\right)\right\}$$
                                                        ▷ $T$ defined in (8)
    Propose $\hat{\tau}^{(k)} = \tau^{(k)} + \epsilon_1\rho^{(k)}, \rho^{(k)} \sim \mathsf{N}(0,1)$
    Make transition $\tau^{(k)} \to \hat{\tau}^{(k)}$ with probability
        $$A(\tau^{(k)} \to \hat{\tau}^{(k)}) = \min\left\{1, \frac{g(\xi^{(k+1)}, \hat{\tau}^{(k)}, \alpha^{(k)})}{g(\xi^{(k+1)}, \tau^{(k)}, \alpha^{(k)})}\right\}$$
                                                        ▷ $g$ defined in (9)
    Propose $\hat{\alpha}^{(k)} = \alpha^{(k)} + \epsilon_2\sigma^{(k)}, \sigma^{(k)} \sim \mathsf{N}(0,1)$
    Make transition $\alpha^{(k)} \to \hat{\alpha}^{(k)}$ with probability
        $$A(\alpha^{(k)} \to \hat{\alpha}^{(k)}) = \min\left\{1, \frac{g(\xi^{(k+1)}, \tau^{(k+1)}, \hat{\alpha}^{(k)})}{g(\xi^{(k+1)}, \tau^{(k+1)}, \alpha^{(k)})}\right\}$$
end for
return $\{T(\xi^{(k)}, \tau^{(k)}, \alpha^{(k)}), \tau^{(k)}, \alpha^k\}$
```

FIG. 7. *Running averages of classifications of delegates*

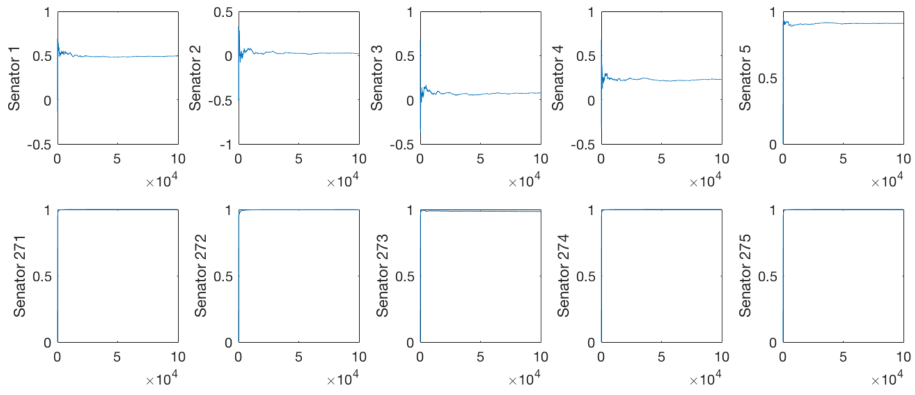


FIG. 8. *Algorithm 2 after truncating eigenvectors, final average*

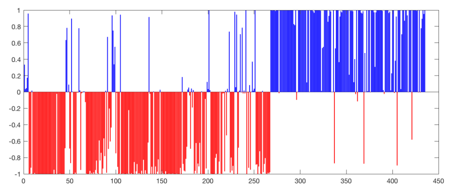
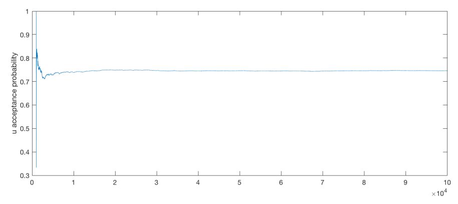


FIG. 9. *Algorithm 2 average $u$ acceptance probability*



of the eigenvectors is not necessary. Fixing $\gamma = 0.0001, \beta = 0.1, \tau^{(0)} = 20, \alpha^{(0)} = 20, \tau \in [0.1, 60], \alpha \in [0.1, 60], \epsilon_\tau = 1, \epsilon_\alpha = 1$ and running 100000 iterations with a burn-in period of 1000, the accuracy is about 87%. See Figure 12, Figure 13, Figure 14, Figure 15.

FIG. 10. *Algorithm 2 trace of $\tau$*

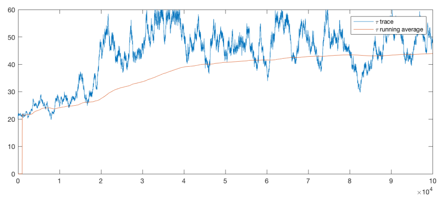


FIG. 11. *Algorithm 2 trace of $\alpha$*

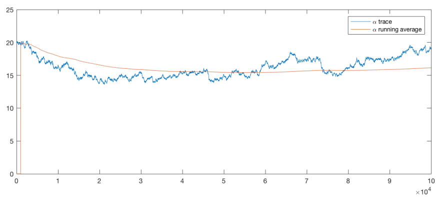


FIG. 12. *Algorithm 3 final average*

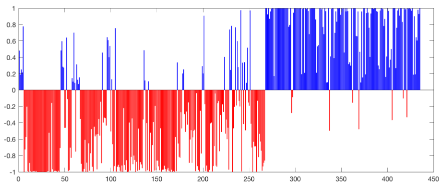


FIG. 13. *Algorithm 3 average $\xi$ acceptance probablity*

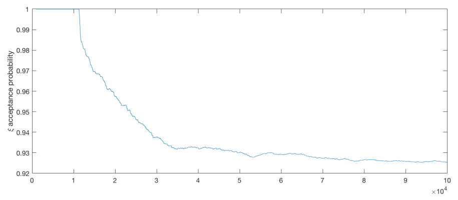


FIG. 14. *Algorithm 3 trace of $\tau$*

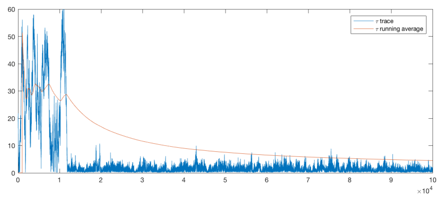
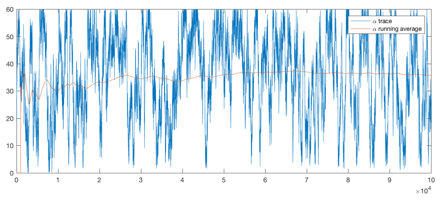


FIG. 15. *Algorithm 3 trace of $\alpha$*



## 7. Two moons experiments.

Using these algorithms, I also ran experiments to cluster the two-moons data set, with $r = 1, N = 1000, d = 100, \sigma = 0.1$. We used the self-tuning Laplacian introduced in [7].

For each of the following experiments, we labeled the same 42 nodes, 21 in each half-moon, approximately uniformly spaced. Running Algorithm 1 on this data set using only the first 50 eigenvectors with $\gamma = 0.0001, \beta = 0.4, \tau = 1, \alpha = 1$ with 100000 iterations and a burn-in period of 1000, we could get around 90% accuracy.

FIG. 16. *Algorithm 1 final average*

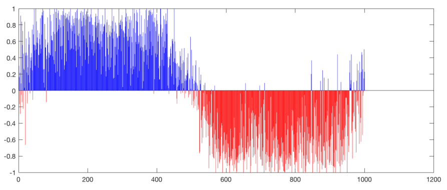
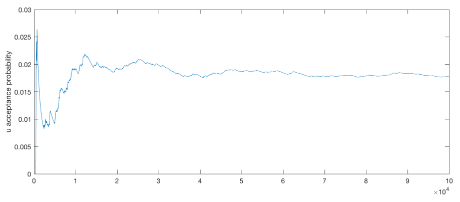


FIG. 17. *Algorithm 1 average u acceptance probability*



We now compare the results from the centered (Algorithm 2) and non-centered (Algorithm 3) algorithms, truncating the number of eigenvectors to 50 in both cases. For both algorithms, we fixed $\gamma = 0.1, \beta = 0.4, \tau^{(0)} = 20, \alpha^{(0)} = 20, \tau \in [0.01, 60], \alpha \in [1, 60], \epsilon_\tau = 0.5, \epsilon_\alpha = 1$ and ran 100000 iterations with a burn-in period of 1000. With these settings, Algorithm 2 achieves around 90% accuracy, while Algorithm 3 achieves around 98%. It still appears that the non-centered algorithm converges faster than the centered algorithm. If the centered algorithm is initialized at $\tau^{(0)} = 1$, the final classification accuracy is very similar to that of the non-centered algorithm. See Figure 18, Figure 19, Figure 20, Figure 21, Figure 22, Figure 23, Figure 24, Figure 25.

Notice the convergence of $\tau$ and its low variance in the samples for the non-centered self-tuning

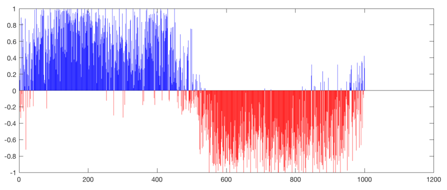


FIG. 18. *Algorithm 2 final average*

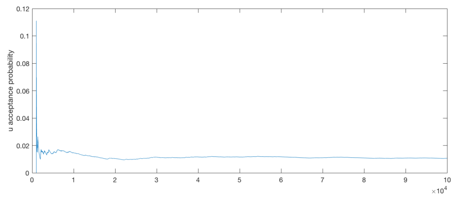


FIG. 19. *Algorithm 2 average $u$ acceptance probability*

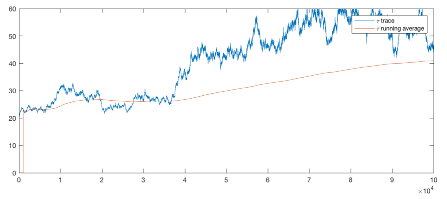


FIG. 20. *Algorithm 2, trace $\tau$*

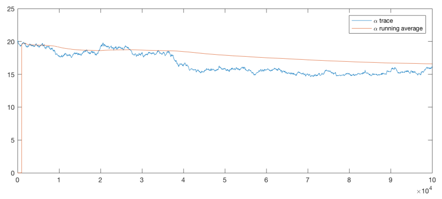


FIG. 21. *Algorithm 2, trace $\alpha$*

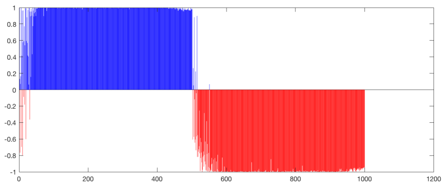


FIG. 22. *Algorithm 3 final average*

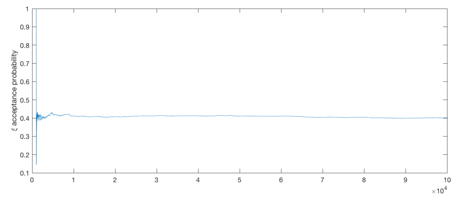


FIG. 23. *Algorithm 3 average $\xi$ acceptance probability*

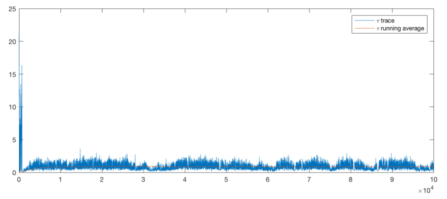


FIG. 24. *Algorithm 3, trace $\tau$*

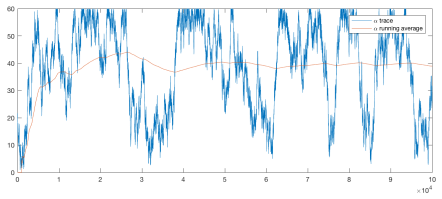


FIG. 25. *Algorithm 3, trace $\alpha$*

algorithm.

**8. Challenges.** Some common issues that I encountered are bugs in my MATLAB code. Some of these bugs included overflow or underflow of ratios, when both numerator and denominator are very large or very small. To improve accuracy, I implemented these ratios as the difference between logs, and took the exp of the result at the end. I became used to using MATLAB's debugger to step through line-by-line and isolate the mistakes.
Another challenge was trying to get Algorithm 2, the centered hierarchical algorithm, to perform well. The first implementation of that algorithm gave traces of the hyperparameters with low mixing and poor clustering of the voting records. Allowing for large ranges for $\tau$ and $\alpha$ did not solve the problem as the MCMC seemed to find one value of $\tau$ and stay stuck at that value. To try to debug the code, I re-implemented the algorithm using more of the linear algebra functionality of MATLAB and improved the readability. This did not solve the problem, and we decided that the problem might have to do with the eig solver of MATLAB failing for larger eigenvalues. We tested this by truncating the list of eigenvectors from 435 down to only 50, and the results, as mentioned above, were better. Since this fix is specific to these data sets, moving forward I anticipate that there may be more challenges with this algorithm. Note that the next step in

the hierarchical models is to be hierarchical about this truncation level, which would tackle this problem.
We have also found that $\tau$ seems to have much lower variance than $\alpha$, and it is difficult to see if $\alpha$ has converged. To understand this problem, I have tried looking at histograms for $\alpha$ and saw that the distribution appears different from the prior distribution, since there seems to be a bias towards larger $\alpha$ (an example of this is in Figure 25). We do not necessarily expect $\alpha$ to converge to a particular value – it could just be that $\alpha$ is not strongly informed by the data. We have also observed high levels of correlation between $\tau$ and $\alpha$.

**9. Additional resources.** My project is mostly done on my computer, my notebook, and a whiteboard. Most likely, I will not require additional resources other than references to papers and books to read to understand the algorithms.

## REFERENCES

[1] A. L. Bertozzi, X. Luo, A. M. Stuart, and K. C. Zygalakis, *Uncertainty quantification in the classification of high dimensional data*, ArXiv e-prints, (2017), https://arxiv.org/abs/1703.08816.

[2] A. Beskos, G. O. Roberts, A. M. Stuart, and J. Voss, *MCMC methods for diffusion bridges*, Stochastics and Dynamics, 8 (2008), pp. 319–350.

[3] S. L. Cotter, G. O. Roberts, A. M. Stuart, and D. White, *MCMC methods for functions: modifying old algorithms to make them faster*, Statistical Science, 28 (2013), pp. 424–446.

[4] M. Iglesias, Y. Lu, and A. M. Stuart, *A Bayesian level set method for geometric inverse problems*, Interfaces and Free Boundaries, 18 (2016), pp. 181–217.

[5] O. Papaspiliopoulos, G. O. Roberts, and M. Sköld, *A general framework for the parametrization of hierarchical models*, Statistical Science, (2007), pp. 59–73.

[6] U. von Luxburg, *A tutorial on spectral clustering*, Statistics and Computing, 17 (2007), pp. 395–416.

[7] L. Zelnik-Manor and P. Perona, *Self-tuning spectral clustering*, in Advances in neural information processing systems, 2005, pp. 1601–1608.