Comparison of Bayesian and Non-Bayesian Methods for Clustering Binary Data with Bernoulli Mixture Models

Abstract Clustering is an unsupervised learning technique that seeks "natural" groupings in data. One form of data that has not been widely studied in the context of clustering is binary data. A rich statistical framework for clustering binary data is the Bernoulli mixture model for which there exists both Bayesian and non-Bayesian approaches. This paper conducts a simulation study to compare the efficacy of these methods on clustering binary data when (i) the number of clusters is known *a priori* and (ii) when the number of clusters must additionally be inferred from the data.

1 Introduction

Clustering binary data with Bernoulli mixture models has been used to classify text documents by topic (Wang and Kabán, 2005; Li, 2006), recognize handwritten digits (Grim et al., 2000; Bishop, 2006), and identify bacterial taxa (Gyllenberg et al., 1997). Available clustering methods can be broadly classified as classical (non-Bayesian) or Bayesian. The former provides a familiar framework built on maximum likelihood estimation and hypothesis testing, while the latter allows for the incorporation of prior beliefs to facilitate clustering at the expense of increased computational demands. Here, we compare their ability to accurately cluster binary data and discern the true number of clusters. The paper proceeds as follows. Section 2 formulates the Bernoulli mixture model and Section 3 briefly introduces the methods to be compared. Section 4 presents three factors that are likely to influence clustering performance. Section 5 outlines the design of the simulation and Section 6 presents its results. Section 7 discusses the implications of the results on clustering binary data. References and `R` code follow.

2 Bernoulli Mixture Model

Let $\boldsymbol{X} = (\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_N)$ be a random sample of D-dimensional binary vectors. A vector $\boldsymbol{X}_n$ is assumed to arise from a finite mixture density, $f(\boldsymbol{X}_n|K, \boldsymbol{p}, \boldsymbol{\Theta}) = \sum_{k=1}^{K} p_k q(\boldsymbol{X}_n|\boldsymbol{\theta}_k)$,

where K is the number of components, q is the mixture component density, $\boldsymbol{\Theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K)$ are mixture parameters with $\boldsymbol{\theta}_k$ specific to component $q(\cdot|\boldsymbol{\theta}_k)$, and $\boldsymbol{p} = (p_1, \ldots, p_K)$ are the mixing weights, $\sum_{k=1}^{K} p_k = 1$, $p_k > 0, k = 1, \ldots, K$. For convenience, let $\boldsymbol{\Psi} = (\boldsymbol{p}, \boldsymbol{\Theta})$ and $\boldsymbol{\Psi}_K = (K, \boldsymbol{\Psi})$. As $\boldsymbol{X}_n$ is binary and its components assumed independent, i.e. $\boldsymbol{X}_n = (X_{n1}, \ldots, X_{nD})$ with $X_{nd} \in \{0, 1\}, d = 1, \ldots, D$ and $X_{nd_1} \perp\!\!\!\perp X_{nd_2}$, a natural choice for q is the multivariate Bernoulli distribution with independent components. Letting $\boldsymbol{\theta}_k = (\theta_{k1}, \ldots, \theta_{kD})$ for $k = 1, \ldots, K$ with $0 \leq \theta_{kd} \leq 1, d = 1, \ldots, D$, the Bernoulli mixture model (BMM) from which vectors $\boldsymbol{X}_n$ are independently drawn is given by

$$f(\boldsymbol{X}_n|\boldsymbol{\Psi}_K) = \sum_{k=1}^{K} p_k \prod_{d=1}^{D} \theta_{kd}^{X_{nd}} (1 - \theta_{kd})^{1 - X_{nd}}. \tag{1}$$

Moreover, consider the existence of latent (i.e. unobserved) vectors $\boldsymbol{Z}_n$ that record the membership (or allocation) of $\boldsymbol{X}_n$ to one of k components. Let $\boldsymbol{Z} = (\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_N)$ such that $\boldsymbol{Z}_n = (Z_{n1}, \ldots, Z_{nK})$ with $Z_{nk} = 1$ if $\boldsymbol{X}_n$ belongs to component k and 0 otherwise. Assume that $\boldsymbol{Z}_n$'s are independent conditional on K and $\boldsymbol{p}$ and express $Pr(Z_{nk} = 1|K, \boldsymbol{p}) = p_k$ so that $f(\boldsymbol{Z}_n|K, \boldsymbol{p}) = \prod_{k=1}^{K} p_k^{Z_{nk}}$. Additionally, $Z_{nk} = 1$ in $\boldsymbol{Z}_n$ records the membership of $\boldsymbol{X}_n$ to component k, so $f(\boldsymbol{X}_n|\boldsymbol{Z}_n, K, \boldsymbol{\Theta}) = \prod_{k=1}^{K} \left[\prod_{d=1}^{D} \theta_{kd}^{X_{nd}} (1 - \theta_{kd})^{1 - X_{nd}} \right]^{Z_{nk}}$. Combining these two densities together with $f(\boldsymbol{X}_n, \boldsymbol{Z}_n|\boldsymbol{\Psi}_K) = f(\boldsymbol{X}_n|\boldsymbol{Z}_n, K, \boldsymbol{\Theta}) f(\boldsymbol{Z}_n|K, \boldsymbol{p})$ for all n, the complete-data likelihood is given by

$$L_C(\boldsymbol{\Psi}_K|\boldsymbol{X}, \boldsymbol{Z}) = f(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\Psi}_K) = \prod_{n=1}^{N} \prod_{k=1}^{K} \left[p_k \prod_{d=1}^{D} \theta_{kd}^{X_{nd}} (1 - \theta_{kd})^{1 - X_{nd}} \right]^{Z_{nk}}, \tag{2}$$

which will prove useful for the estimation methods considered here.

3 Estimation Methods

We present four different estimation methods. The former two methods assume K is known and estimate the parameters of the mixture model and the unobserved $\boldsymbol{Z}$ using maximum likelihood

methods and MCMC sampling. The latter two methods operate with K unknown and attempt to infer K from the data. The classical (non-Bayesian) approach uses likelihood ratio tests to choose the best K, while fully Bayesian approaches place a prior on K and select K by its marginal posterior probabilities.

3.1 EM Algorithm

The EM algorithm (Dempster et al., 1977) seeks maximization of the likelihood (product of (1) over $n = 1, \ldots, N$) using (2) and an iterative Expectation (E-step) and Maximization (M-step) procedure. Starting from some initial conditions $\boldsymbol{\Psi}^{(0)}$, at every iteration $t \geq 1$ until convergence is reached, $\boldsymbol{Z}_n^{(t)}$ is updated by $E_{\boldsymbol{\Psi}^{(t-1)}}(\boldsymbol{Z}_n|\boldsymbol{X}) =$

$$\left[\boldsymbol{p}^{(t-1)} \prod_{d=1}^{D} (\theta_{kd}^{(t-1)})^{X_{nd}} (1-\theta_{kd}^{(t-1)})^{1-X_{nd}}\right] \left[\sum_{l=1}^{K} p_l^{(t-1)} \prod_{d=1}^{D} (\theta_{kd}^{(t-1)})^{X_{nd}} (1-\theta_{kd}^{(t-1)})^{1-X_{nd}}\right]^{-1}$$

and, for all k, $p_k^{(t)}$ and $\boldsymbol{\theta}_k^{(t)}$ are updated respectively by $N^{-1}u_k^{(t)}$ and $[u_k^{(t)}]^{-1} \sum_{n=1}^{N} Z_{nk}^{(t)} \boldsymbol{X}_n$ where $u_k^{(t)} = \sum_{n=1}^{N} Z_{nk}^{(t)}$. This procedure yields $\hat{\boldsymbol{\Psi}}$ and $\hat{\boldsymbol{Z}}$, maximizers of the likelihood function. Because the EM algorithm converges only to a local maximum, however, it is recommended one initialize the algorithm at many different starting values, iterate to convergence, and select the best maximizer of the likelihood as one's estimates.

3.2 Gibbs Sampler

A Bayesian approach treats $\boldsymbol{\Psi}$ as random. With conjugate priors $\boldsymbol{p} \sim Dirichlet(\alpha_1, \ldots, \alpha_K)$ and $\theta_{kd} \sim Beta(\gamma_{kd}, \delta_{kd})$ the posterior distribution of $\boldsymbol{\Psi}$ is given by

$$\pi(\boldsymbol{\Psi}|\boldsymbol{X}, \boldsymbol{Z}) \propto f(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\Psi})\pi(\boldsymbol{\Psi}) = \pi(\boldsymbol{p})f(\boldsymbol{Z}|\boldsymbol{p})\pi(\boldsymbol{\Theta})f(\boldsymbol{X}|\boldsymbol{Z}, \boldsymbol{\Theta}) \tag{3}$$

$$\propto \prod_{k=1}^{K} \left[p_k^{u_k+\alpha_k-1} \prod_{d=1}^{D} \theta_{kd}^{v_{kd}+\gamma_{kd}-1} (1-\theta_{kd})^{u_k-v_{kd}+\delta_{kd}-1}\right] \tag{4}$$

where $u_k = \sum_{n=1}^{N} Z_{nk}$ and $v_{kd} = \sum_{n=1}^{N} Z_{nk} X_{nd}$. Joint posterior samples are drawn from (4) via Markov chain Monte Carlo (Gelfand and Smith, 1990) with the mixture model Gibbs sampler (GS)

of Diebolt and Robert (1994). For the simulations here, we assume there is no prior information distinguishing the K components from one another and select $\alpha_1 = \cdots = \alpha_K = 1$ and $\gamma_{11} = \cdots = \gamma_{KD} = \delta_{11} = \cdots = \delta_{KD} = 1$. To address the label-switching problem, we employ the post-processing Kullback-Leibler divergence approach described in Stephens (2000).

3.3 Likelihood Ratio Tests & Bootstrapping

The standard solution to finding the number of clusters in a classical setting is to conduct a series of likelihood ratio tests (LRTs) of the sufficiency of k clusters vs. $k+1$ clusters to adequately group the data. Concretely, for $k = 1, 2, \ldots$, hypotheses $H_0 : K = k$ vs. $H_1 : K = k + 1$, and LRT test statistic λ, we examine $-2\log\lambda = 2[\log L(\hat{\boldsymbol{\Psi}}_{k+1}|\boldsymbol{X}) - \log L(\hat{\boldsymbol{\Psi}}_k|\boldsymbol{X})]$, where $\hat{\boldsymbol{\Psi}}_{k+1}$ and $\hat{\boldsymbol{\Psi}}_k$ denote the maximum likelihood estimates of $\boldsymbol{\Psi}_K$ obtained by the EM algorithm under H_1 and H_0 respectively. Unfortunately, $-2\log\lambda$ violates regularity conditions (Titterington et al., 1985), so one must use a parametric bootstrapping approach to approximate the distribution of $-2\log\lambda$ under H_0 (McLachlan, 1987). The set of T values $\{-2\log\lambda^{(1)}, \ldots, -2\log\lambda^{(T)}\}$ obtained in this way approximate the true null distribution of $-2\log\lambda$ and allow for calculation of an approximate p-value for H_0 by $\sum_{t=1}^{T} I(-2\log\lambda < -2\log\lambda^{(t)})/T$. If H_0 is rejected, k is incremented by one and the LRT + Bootstrap procedure is repeated; otherwise, k is deemed sufficient to group the observed data and $K = k$ is adopted.

3.4 Allocation Sampler

In a Bayesian setting, (3) is modified to include a prior $\pi(K)$ placed on K. Nobile and Fearnside (2007) propose obtaining estimates of K by integrating out $\boldsymbol{\Psi}$ from the posterior and drawing joint posterior samples on $\boldsymbol{Z}$ and K only. With conjugate priors on $\boldsymbol{p}$ and θ_{kd}, $\pi(K|\boldsymbol{X}, \boldsymbol{Z})$ is given by

$$\int \pi(K, \boldsymbol{\Psi}|\boldsymbol{X}, \boldsymbol{Z})d\boldsymbol{\Psi} \propto \pi(K) \int f(\boldsymbol{Z}|K, \boldsymbol{p})\pi(\boldsymbol{p}|K)d\boldsymbol{p} \int f(\boldsymbol{X}|\boldsymbol{Z}, K, \boldsymbol{\Theta})\pi(\boldsymbol{\Theta}|K)d\boldsymbol{\Theta}$$

$$\propto \frac{e^{-\beta}\beta^K}{K!} \cdot \frac{\Gamma(\alpha_0)}{\Gamma(\alpha_0 + N)} \prod_{k=1}^{K} \frac{\Gamma(\alpha_k + u_k)}{\Gamma(\alpha_k)} \cdot \prod_{k=1}^{K} \left[\prod_{d=1}^{D} \frac{\Gamma(\gamma_{kd} + \delta_{kd})}{\Gamma(\gamma_{kd})\Gamma(\delta_{kd})} \frac{\Gamma(v_{kd} + \gamma_{kd})\Gamma(u_k - v_{kd} + \delta_{kd})}{\Gamma(u_k + \gamma_{kd} + \delta_{kd})} \right]$$

for $\alpha_0 = \sum_{k=1}^{K} \alpha_K$ and $K \sim Poisson(\beta)$ with $\beta = 1$ as in Nobile and Fearnside (2007). The Allocation sampler (AS) conducts five main moves at random: Absorb/Eject (AE) changes K by reducing/adding a cluster, Gibbs sampling (GS) changes $\boldsymbol{Z}$ by drawing $\boldsymbol{Z}_n$ sequentially from their full conditional distribution, and three Metropolis-Hastings steps (M1, M2, M3) additionally change $\boldsymbol{Z}$ by redistributing observations between components in varying ways.

4 Factors

This simulation study investigates the effect of three key factors (each with three levels) on the accuracy of clustering (or classification rate) of the EM algorithm and Gibbs sampler and on the ability of LRTs and the Allocation sampler to adequately identify the true number of clusters K when it is unknown.

4.1 Factors of Interest

Number of Clusters K is chosen to vary over $3, 4$, and 5.

Mixture Component Parameters Mixture components $k = 1, \ldots, K$ are assumed to follow independent multivariate Bernoulli distributions with mixture parameters $\boldsymbol{\theta}_k = (\theta_{k1}, \ldots, \theta_{kD})$. Each $\theta_{kd} = Pr(X_{nd} = 1)$ represents the probability that feature d is present in a binary observation $\boldsymbol{X}_n = (X_{n1}, \ldots, X_{nD})$ drawn from component k. We choose to generate feature probabilities $\theta_{k1}, \ldots, \theta_{kD}$ by evaluating normal distribution density functions over discrete values $d = 1, \ldots, D$ and scaling the values so that no feature probability exceeds 0.8.. The standard deviation selected for these normal distributions influences cluster similarity, with higher standard deviations leading to more similar feature probabilities between clusters and thus making it more difficult to distinguish observations as arising from one cluster over another. More formally, if we define Δ to be the distance between any two centers, then we might consider investigating three levels of feature similarity[1]: $\frac{3}{4}\Delta$ (Weak feature similarity), Δ (Moderate), and $\frac{5}{4}\Delta$ (Strong). For example, Figure 1a depicts feature probabilities θ_{kd} (y-axis) for $d = 1, \ldots, D = 25$ dimensions (x-axis) generated for $k = 1, 2, 3, K = 4$ clusters centered equidistantly at $d = 5, 10, 15, 20$ respectively with standard

deviations determined by Moderate feature similarity (i.e., $sd = \Delta = 5$).

Mixing Weights The vector $\boldsymbol{p} = (p_1, \ldots, p_K)$ of mixing weights defines the probability p_k an observation is drawn from component k in (1), $k = 1, \ldots, K$. Accuracy of the estimation of $\boldsymbol{p}$ can have a large effect on the fitting of the mixture model and subsequent clustering (McLachlan and Basford, 1988, Section 3.2). Thus, we consider three distinct weighting schemes: Even, Symmetric, and Skewed (Figure 1b). Observations are equally likely to originate from any of the K clusters under Even weighting, while they are more likely to originate from the middle cluster(s) under Symmetric weighting (i.e. $\boldsymbol{p}$ is $(0.2, 0.6, 0.2), (0.2, 0.4, 0.4, 0.2), (0.1, 0.2, 0.4, 0.2, 0.1)$ for $K = 3, 4, 5$, respectively). Under Skewed weighting, a large portion of observations are drawn from the first cluster, and observations are less likely to be drawn from more dissimilar clusters (i.e. $\boldsymbol{p}$ is $(0.6, 0.2, 0.2), (0.5, 0.2, 0.2, 0.1), (0.4, 0.2, 0.2, 0.1, 0.1)$).

4.2 Fixed Factors

Additional factors that may further influence the accuracy of the estimation methods are the dimensionality D and sample size N. In the interest of feasible computation time for the 27 proposed factor combinations, however, we choose to fix $D = 25$ and $N = 200$. This dimensionality is chosen as a balance between low- and high-dimensional data, with the former too simplistic for the purposes of this study and the latter likely necessitating the incorporation of feature selection methods (Wang and Kabán, 2005; Bouguila, 2010). As for sample size, $N = 200$ is large enough so that even rare clusters (as specified by the weighting scheme) can expect to contribute at least a handful of observations to $\boldsymbol{X}$, but it is not so large as to be too optimistic for real world settings.

5 Simulation Set-Up

A data set $\boldsymbol{X}$ is generated in the following way:

1. One of K components is selected with probability in $\boldsymbol{p}$ defined by the weighting scheme.

[1] Standard deviations lower than $\frac{3}{4}\Delta$ and higher than $\frac{5}{4}\Delta$ tended to be largely uninteresting as they led to clusters with feature probabilities that were either too distinct or too similar, respectively.

2. For this component, say k, every feature d of an observation $\boldsymbol{X}_n$ is selected to be present (1) vs. absent (0) with probability θ_{kd} defined by the feature similarity scheme.
3. The resulting D-dimensional binary vector $\boldsymbol{X}_n$ marks a single observation from component k and its allocation is recorded by $\boldsymbol{Z}_n$ with $Z_{nk} = 1$ and 0 otherwise.
4. Steps 1, 2, and 3 are repeated $N - 1$ more times to produce N binary data vectors.

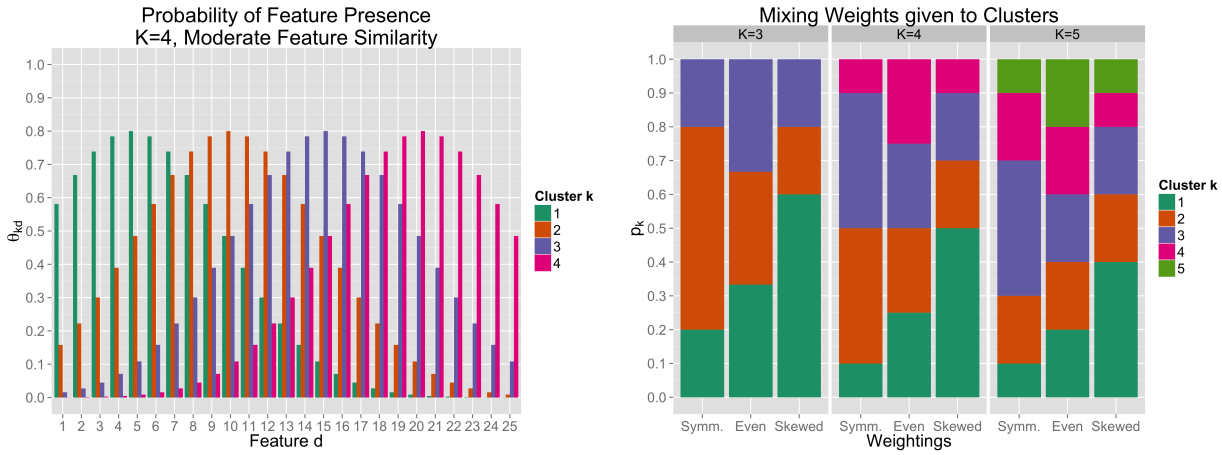
At every factor combination (27 total), five $\boldsymbol{X}$ are generated and analyzed by:

EM: Starting from 100 random initial values, the algorithm runs to convergence and selects the best maximizer ($\hat{\boldsymbol{\Psi}}$, $\hat{\boldsymbol{Z}}$) of the log-likelihood.

GS: Four chains begin from random initial values and iteratively draw 10,000 posterior samples, with the first 5,000 from each chain removed for burn-in. Chains are combined and the label-switching problem is resolved by expensive post-processing computation (Stephens, 2000). $\hat{\boldsymbol{\Psi}}$ is taken as its posterior mean, $\hat{\boldsymbol{Z}}$ by its posterior mode.

LRT: For $k = 1, 2, \ldots, 6$, $H_0 : K = k$ vs. $H_1 : K = k + 1$ is tested and approximate p-value obtained via parametric bootstrapping.

AS: Four chains begin at $K^{(0)} = 1$ and draw 20,000 posterior samples, with the first 10,000 from each chain removed for burn-in. Of interest is the marginal posterior distribution of K.



(a) $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \boldsymbol{\theta}_4$ under Moderate Feature Similarity. (b) Three weighting schemes under $K = 3, 4, 5$.

Figure 1: Visualizations of several factor combinations.

6 Simulation Results

6.1 Classification Rate

Classification rates are very high and nearly identical between EM and GS for $K = 3$ clusters (Table 1). For $K = 4$ clusters, they are also comparable except for instances of Strong feature similarity under Even and Symmetric weighting schemes where the GS performs poorly. This is likely due to the reluctance of the sampler to explore beyond regions of high probability. For $K = 5$ this reluctance is only exacerbated; the posterior surface has $5! = 120$ modes and lower classification rates by GS suggest it does not handle this multi-modality well as compared to the EM. Regardless, EM and GS achieve similar lower classification rates under Strong feature similarity for $K = 5$.

		$K = 3$		$K = 4$		$K = 5$	
Weight	Similarity	EM	GS	EM	GS	EM	GS
Even	Weak	0.97 (0.01)	0.97 (0.01)	0.97 (0.01)	0.96 (0.01)	0.89 (0.03)	0.66 (0.06)
	Moderate	0.96 (0.02)	0.95 (0.01)	0.93 (0.01)	0.92 (0.02)	0.83 (0.08)	0.68 (0.09)
	Strong	0.85 (0.06)	0.81 (0.12)	0.80 (0.07)	0.72 (0.09)	0.63 (0.07)	0.59 (0.07)
Symm.	Weak	0.98 (0.00)	0.98 (0.00)	0.96 (0.01)	0.96 (0.01)	0.81 (0.11)	0.70 (0.05)
	Moderate	0.95 (0.01)	0.95 (0.01)	0.91 (0.02)	0.91 (0.02)	0.88 (0.04)	0.67 (0.05)
	Strong	0.88 (0.04)	0.89 (0.02)	0.72 (0.08)	0.60 (0.15)	0.54 (0.06)	0.54 (0.05)
Skewed	Weak	0.98 (0.00)	0.98 (0.01)	0.96 (0.02)	0.97 (0.02)	0.87 (0.03)	0.81 (0.06)
	Moderate	0.95 (0.01)	0.95 (0.01)	0.91 (0.02)	0.90 (0.05)	0.91 (0.02)	0.84 (0.01)
	Strong	0.88 (0.02)	0.89 (0.02)	0.75 (0.11)	0.75 (0.06)	0.71 (0.06)	0.70 (0.04)

Table 1: Average classification rates of EM algorithm and Gibbs sampler (GS) over 5 generated data sets across all 27 factor combinations. Standard errors of rates are enclosed in parentheses.

6.2 Cluster Identification

Table 2 records approximate LRT p-values on the sufficiency of k vs. $k + 1$ clusters to model the data and marginal posterior probabilities of $\pi(K|\boldsymbol{X}, \boldsymbol{Z})$ as collected by the Allocation sampler (AS). Both methods perform well in identifying $K = 3$ clusters. For $K = 4$, AS accurately identifies four clusters for all weighting and similarity combinations, but it is a close call with three clusters under Symmetric-Strong and Skewed-Strong. In these two situations, LRT selects three clusters as most likely. The results for $K = 5$ are perplexing. Across all weighting schemes, LRT identifies different numbers of clusters for each feature similarity scheme: five under Moderate,

four under Weak, and three under Strong. AS performance is disappointing. Beyond identifiying four clusters under Skewed-Strong, it overwhelmingly identifies three clusters for all remaining combinations. The preference of AS for less clusters when K is large is consistent with Nobile and Fearnside (2007) for $N = 200$ (albeit for Gaussian mixtures) where preference is, at least, shown to attenuate for larger N.

7 Discussion

When K is known, the EM algorithm and Gibbs sampler are the standard non-Bayesian and Bayesian approaches to clustering in the mixture model framework. When the number of clusters is low and feature similarity between clusters is not strong, EM and GS produce comparable classification rates, though the latter requires extended computation due to the label-switching problem if symmetric priors are selected. For greater numbers of clusters ($K = 4, 5$), GS performance can suffer compared to EM as it is prone to trapping itself within posterior modes. For both methods, Strong feature similarity is most detrimental to classification, particularly under Symmetric weighting.

When K is unknown, LRTs and the Allocation sampler estimate K with hypothesis testing and posterior sampling, respectively. The methods do fairly well in identifying $K = 3$ and $K = 4$ clusters, though Strong-Symmetric and Strong-Skewed situations cause some trouble. Neither method performs well for $K = 5$ clusters, however. It may be the case that dimensionality $D = 25$ is too restrictive for these methods to properly discern five distinct clusters in the data set. By the same token, $N = 200$ may simply by too small a sample to get a "good read" on five clusters, particularly in the more extreme weighting schemes where one can expect only about 20 observations from some clusters. Future studies may consider investigating the effects of N and D on cluster identification for binary data under the factor combinations studied here.

			$K=3$					$K=4$					$K=5$				
	Weights	Similarity	2	**3**	4	5	6	2	3	**4**	5	6	2	3	4	**5**	6
LRT	Even	Weak	0.00	**0.51**	0.57	0.71	0.74	0.00	0.00	**0.45**	0.41	0.73	0.00	0.00	**0.45**	0.41	0.73
		Moderate	0.00	**0.41**	0.53	0.70	0.46	0.00	0.00	**0.29**	0.58	0.61	0.00	0.00	0.02	**0.20**	0.41
		Strong	0.01	**0.30**	0.64	0.82	0.78	0.00	0.03	**0.44**	0.46	0.63	0.00	**0.12**	0.54	0.54	0.63
	Symm.	Weak	0.00	**0.39**	0.56	0.65	0.56	0.00	0.00	**0.31**	0.67	0.62	0.00	0.00	**0.17**	0.48	0.46
		Moderate	0.00	**0.48**	0.64	0.78	0.70	0.00	0.00	**0.30**	0.54	0.56	0.00	0.00	0.01	**0.32**	0.49
		Strong	0.00	**0.50**	0.64	0.68	0.56	0.01	**0.29**	0.58	0.58	0.64	0.00	**0.17**	0.24	0.29	0.49
	Skewed	Weak	0.00	**0.42**	0.58	0.72	0.88	0.00	0.00	**0.32**	0.54	0.73	0.00	0.00	**0.10**	0.47	0.54
		Moderate	0.00	**0.45**	0.72	0.54	0.55	0.00	0.00	**0.33**	0.38	0.75	0.00	0.00	0.04	**0.49**	0.51
		Strong	0.00	**0.38**	0.56	0.54	0.62	0.00	**0.30**	0.26	0.58	0.66	0.00	**0.06**	0.43	0.43	0.50
	Std. Err.	Average	0.00	0.29	0.25	0.19	0.19	0.00	0.07	0.24	0.27	0.22	0.00	0.05	0.14	0.19	0.21
		Std. Dev.	0.01	0.07	0.09	0.07	0.07	0.01	0.12	0.09	0.04	0.09	0.00	0.08	0.10	0.07	0.06
AS	Even	Weak	0.00	**0.79**	0.19	0.02	0.00	0.00	0.18	**0.69**	0.12	0.01	0.00	**0.68**	0.28	0.04	0.00
		Moderate	0.00	**0.63**	0.28	0.08	0.02	0.00	0.00	**0.76**	0.21	0.03	0.00	**0.85**	0.13	0.01	0.00
		Strong	0.09	**0.29**	0.25	0.21	0.16	0.00	0.19	**0.32**	0.30	0.19	0.00	**0.50**	0.34	0.13	0.03
	Symm.	Weak	0.00	**0.84**	0.14	0.02	0.00	0.00	0.26	**0.61**	0.12	0.01	0.00	**0.87**	0.12	0.01	0.00
		Moderate	0.00	**0.56**	0.32	0.11	0.02	0.00	0.17	**0.60**	0.20	0.04	0.00	**0.86**	0.13	0.01	0.00
		Strong	0.00	**0.51**	0.33	0.11	0.04	0.16	0.31	**0.34**	0.16	0.04	0.00	**0.59**	0.27	0.12	0.03
	Skewed	Weak	0.00	**0.77**	0.19	0.03	0.01	0.00	0.31	**0.58**	0.11	0.01	0.00	**0.51**	0.41	0.07	0.01
		Moderate	0.00	**0.68**	0.26	0.05	0.01	0.00	0.33	**0.52**	0.14	0.02	0.00	**0.85**	0.13	0.01	0.00
		Strong	0.00	0.35	**0.37**	0.22	0.07	0.00	0.30	**0.33**	0.26	0.11	0.00	0.33	**0.46**	0.16	0.04
	Std. Err.	Average	0.02	0.12	0.08	0.06	0.04	0.03	0.31	0.22	0.10	0.04	0.00	0.17	0.12	0.05	0.01
		Std. Dev.	0.07	0.09	0.06	0.04	0.08	0.09	0.14	0.13	0.04	0.04	0.00	0.18	0.15	0.05	0.01

Table 2: Approximate p-values (LRT) and posterior probabilities (AS) related to identifying the number of clusters to be $k = 2, 3, \ldots, 6$ when the true number of clusters is K. Bold values highlight the most likely number of clusters identified by the methods. In LRT, this corresponds to the first insignificant p-value ($> \alpha = 0.05$) encountered in sequential tests of $H_0 : K = k$ vs. $H_1 : K = k + 1$. In AS, the bolded value is simply the k with highest posterior probability.

References

Bishop, C. (2006), *Pattern Recognition and Machine Learning*, Information Science and Statistics, Springer.

Bouguila, N. (2010), "On multivariate binary data clustering and feature weighting," *Computational Statistics & Data Analysis*, 54, 120–134.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977), "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society. Series B*, 39, 1–38.

Diebolt, J. and Robert, C. P. (1994), "Estimation of Finite Mixture Distributions through Bayesian Sampling," *Journal of the Royal Statistical Society. Series B (Methodological)*, 56, pp. 363–375.

Gelfand, A. E. and Smith, A. F. (1990), "Sampling-based approaches to calculating marginal densities," *Journal of the American statistical association*, 85, 398–409.

Grim, J., Pudil, P., and Somol, P. (2000), "Multivariate structural Bernoulli mixtures for recognition of handwritten numerals," in *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, vol. 2, pp. 585–589 vol.2.

Gyllenberg, M., Koski, T., and Verlaan, M. (1997), "Classification of Binary Vectors by Stochastic Complexity," *Journal of Multivariate Analysis*, 63, 47 – 72.

Li, T. (2006), "A Unified View on Clustering Binary Data," *Machine Learning*, 62, 199–215.

McLachlan, G. J. (1987), "On Bootstrapping the Likelihood Ratio Test Stastistic for the Number of Components in a Normal Mixture," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 36, pp. 318–324.

McLachlan, G. J. and Basford, K. E. (1988), "Mixture models. Inference and applications to clustering," *Statistics: Textbooks and Monographs, New York: Dekker, 1988*, 1.

Nobile, A. and Fearnside, A. (2007), "Bayesian finite mixtures with an unknown number of components: The allocation sampler," *Statistics and Computing*, 17, 147–162.

Stephens, M. (2000), "Dealing with label switching in mixture models," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 62, 795–809.

Titterington, D., Smith, A., and Makov, U. (1985), *Statistical analysis of finite mixture distributions*, Wiley series in probability and mathematical statistics: Applied probability and statistics, Wiley.

Wang, X. and Kabán, A. (2005), "Finding Uninformative Features in Binary Data," in *Intelligent Data Engineering and Automated Learning - IDEAL 2005*, eds. Gallagher, M., Hogan, J., and Maire, F., Springer Berlin Heidelberg, vol. 3578 of *Lecture Notes in Computer Science*, pp. 40–47.

Appendix

```
###  Written Prelim, Sim Study
###  "Clustering Binary Data"
###  known_K.R
###  Neal Grantham
###  06/18/14

rm(list=ls())  # fresh workspace

### Load libraries
library(gtools)  # rdirichlet, permutations
library(parallel)  # mclapply, mcmapply

### Define functions
source("functions.R")

### Define factors
# Mixing weights
p.factors <- c("even", "central", "skewed")

# Mixing parameters, Theta
Theta.factors <- c("weak", "moderate", "strong")

# Number of clusters/components
K.factors <- c(3, 4, 5)

## Fix
# sample size
N <- 200
# dimensionality
D <- 25

pct.correctly.clustered.EM <- array(0, c(3, 3, 3, 5))
pct.correctly.clustered.Gibbs <- array(0, c(3, 3, 3, 5))

for (i in 1:3) {
  for (j in 1:3) {
    for (k in 1:3) {
      seeds <- get_seed(i, j, k, r = 1:5)  # generate 5 seeds
      weighting <- p.factors[i]  # specify weighting scheme
      noise <- Theta.factors[j]  # specify feature similarity scheme
      K <- K.factors[k]          # specify number of clusters
      if (weighting == "even") {
        p <- rep(1, K)/K
      } else if (weighting == "central") {
        if (K == 3) p <- c(0.2, 0.6, 0.2)
        if (K == 4) p <- c(0.1, 0.4, 0.4, 0.1)
        if (K == 5) p <- c(0.1, 0.2, 0.4, 0.2, 0.1)
      } else if (weighting == "skewed") {
        if (K == 3) p <- c(0.6, 0.2, 0.2)
        if (K == 4) p <- c(0.5, 0.2, 0.2, 0.1)
        if (K == 5) p <- c(0.4, 0.2, 0.2, 0.1, 0.1)
```

```
      }
      modes <- floor(quantile(1:D, probs=(1:K)/(K+1)))
      if (noise == "weak") {
        sd <- ceiling((modes[2] - modes[1])/1.25)
      } else if (noise == "moderate") {
        sd <- ceiling((modes[2] - modes[1])/1)
      } else if (noise == "strong") {
        sd <- ceiling((modes[2] - modes[1])/0.75)
      }
      Theta <- t(dnorm(matrix(rep(1:D, K), nrow=K, byrow=TRUE),
                       mean = modes, sd = sd))
      Theta <- Theta/(1.25*max(Theta)) ## max feature probability is 0.8
      dat   <- lapply(seeds, generate_data, N=N, p=p, Theta=Theta, K=K)
      pct.correct <- mclapply(dat, run_EM_and_Gibbs, mc.cores=20)
      pct.correct <- do.call(rbind, pct.correct)
      pct.correctly.clustered.EM[i, j, k, ] <- pct.correct[, 1, drop=TRUE]
      pct.correctly.clustered.Gibbs[i, j, k, ] <- pct.correct[, 2, drop=TRUE]
      print(paste("Done with", i, j, k, "..."))
    }
  }
}

R.info <-sessionInfo()
save.image(file = "EM_and_Gibbs.RData")
```

```
###  Written Prelim, Sim Study
###  "Clustering Binary Data"
###  unknown_K.R
###  Neal Grantham
###  06/18/14

rm(list=ls())  # fresh workspace

### Load libraries
library(gtools)  # rdirichlet, permutations
library(parallel)  # mclapply, mcmapply

### Define functions
source("functions.R")

### Define factors
# Mixing weights
p.factors <- c("even", "central", "skewed")

# Mixing parameters, Theta
Theta.factors <- c("weak", "moderate", "strong")

# Number of clusters/components
K.factors <- c(3, 4, 5)

## Fix
# sample size
N <- 200
# dimensionality
D <- 25

possible.K.LRT <- array(0, c(3, 3, 3, 6, 5))
possible.K.AS <- array(0, c(3, 3, 3, 6, 5))

for (i in 1:3) {
  for (j in 1:3) {
    for (k in 3:3) {
      seeds <- get_seed(i, j, k, r = 1:5)  # generate 5 seeds
      weighting <- p.factors[i]  # specify weighting scheme
      noise <- Theta.factors[j]  # specify feature similarity scheme
      K <- K.factors[k]          # specify number of clusters
      if (weighting == "even") {
        p <- rep(1, K)/K
      } else if (weighting == "central") {
        if (K == 3) p <- c(0.2, 0.6, 0.2)
        if (K == 4) p <- c(0.1, 0.4, 0.4, 0.1)
        if (K == 5) p <- c(0.1, 0.2, 0.4, 0.2, 0.1)
      } else if (weighting == "skewed") {
        if (K == 3) p <- c(0.6, 0.2, 0.2)
        if (K == 4) p <- c(0.5, 0.2, 0.2, 0.1)
        if (K == 5) p <- c(0.4, 0.2, 0.2, 0.1, 0.1)
      }
```

```
      modes <- floor(quantile(1:D, probs=(1:K)/(K+1)))
      if (noise == "weak") {
        sd <- ceiling((modes[2] - modes[1])/1.25)
      } else if (noise == "moderate") {
        sd <- ceiling((modes[2] - modes[1])/1)
      } else if (noise == "strong") {
        sd <- ceiling((modes[2] - modes[1])/0.75)
      }
      Theta <- t(dnorm(matrix(rep(1:D, K), nrow=K, byrow=TRUE),
                       mean = modes, sd = sd))
      Theta <- Theta/(1.25*max(Theta)) # max feature probability is 0.8
      dat   <- lapply(seeds, generate_data, N=N, p=p, Theta=Theta, K=K)
      possible.K <- mclapply(dat, run_LRT_and_AS, mc.cores=20)
      possible.K <- do.call(cbind, possible.K)
      possible.K.LRT[i, j, k, , ] <- possible.K[1:6, ]
      possible.K.AS[i, j, k, , ] <- possible.K[7:12, ]
      print(paste("Done with", i, j, k, "..."))
    }
  }
}

R.info <-sessionInfo()
save.image(file = "LRT_and_AS.RData")
```

```
###  Written Prelim, Sim Study
###  "Clustering Binary Data"
###  functions.R
###  Neal Grantham
###  06/18/14

### Seed Creation functions

get_seed <- function(i, j, k, r) {
  seed <- (i-1) + (j-1)*3 + (k-1)*9 + (r-1)*27
  return(seed)
}

invert_seed <- function(seed) {
  r <- floor(seed/27) + 1
  seed <- seed %% 27
  k <- floor(seed/9) + 1
  seed <- seed %% 9
  j <- floor(seed/3) + 1
  seed <- seed %% 3
  i <- seed + 1
  return(list(i=i, j=j, k=k, r=r))
}

### General helper functions

trim <- function(a, tol = 1e-4) {
  a[a < tol] <- tol
  a[a > 1 - tol] <- 1 - tol
  return(a)
}

lappend <- function(lst, obj) {
  lst[[length(lst) + 1]] <- obj
  return(lst)
}

###  Data Generation function

generate_data <- function(seed, N, p, Theta, K) {
  X <- NULL
  Z <- NULL
  set.seed(seed)
  for (n in 1:N) {
    Z.n <- as.vector(rmultinom(1, 1, prob=p))  # generate new allocation Z_n
    Z <- rbind(Z, Z.n)
    theta.k <- Theta %*% Z.n  # grab component parameters
    X.n <- rbinom(D, 1, prob=theta.k)
    X <- rbind(X, X.n)
  }
  return(list(X=X, Z=Z, p=p, Theta=Theta, seed=seed))
}
```

```
###  Starting values for EM and Gibbs sampler

generate_starting_values <- function(X, K) {
  N <- nrow(X)
  Nstar <- ceiling(0.33*N)
  samp <- sample(1:N, Nstar)
  Xstar <- X[samp, ]
  Zstar <- t(rmultinom(Nstar, 1, prob = rep(1, K)))
  u <- colSums(Zstar)
  p <- u/Nstar
  v <- t(Xstar) %*% Zstar
  Theta <- t(t(v)/u)
  Theta <- trim(Theta)
  return(list(p=p, Theta=Theta))
}

###  Computational functions used in EM and Gibbs

get_log_s <- function(X.n, p, Theta) {
  bern <- matrix(dbinom(X.n, 1, prob=Theta, log=TRUE), nrow=length(X.n))
  log.s <- log(p) + colSums(bern)
  return(log.s)
}

get_S <- function(X, p, Theta) {
  N <- nrow(X)
  log.s <- matrix(t(apply(X, 1, get_log_s, p=p, Theta=Theta)), nrow=N)
  means <- rowMeans(log.s)
  s <- exp(log.s - means)
  sums <- rowSums(s)
  return(s/sums)
}

evaluate_log_likelihood <- function(X, p, Theta) {
  log.s <- matrix(t(apply(X, 1, get_log_s, p=p, Theta=Theta)), nrow=nrow(X))
  llike <- sum(log(rowSums(exp(log.s))))
  return(llike)
}

permute_data <- function(dat, Theta.new) {
  require(gtools)
  K <- ncol(Theta.new)
  nu.possible <- permutations(K, K, 1:K)
  min.error <- 1e10
  for (i in 1:nrow(nu.possible)) {
    nu <- nu.possible[i, ]
    error <- sum(abs(Theta.new - dat$Theta[, nu]))
    if (error < min.error) {
      min.error <- error
      labels <- nu
    }
  }
```

```
  p.true <- dat$p[labels]
  Theta.true <- dat$Theta[, labels]
  Z.true <- dat$Z[, labels]
  return(list(p=p.true, Theta=Theta.true, Z=Z.true))
}

###  EM algorithm function

EM_algorithm <- function(dat, attempts = 100, t.max = 1000, verbose = FALSE, fix.K) {

  X <- dat$X
  Z <- dat$Z

  N <- nrow(X)
  D <- ncol(X)

  if (missing(fix.K)) {
    K <- ncol(Z)
  } else {
    K <- fix.K
  }

  llike.save <- -1e10
  for (i in 1:attempts) {
    init <- generate_starting_values(X, K)
    p.OLD <- init$p
    Theta.OLD <- init$Theta
    Z.OLD     <- matrix(diag(K)[sample(1:K, N, replace=TRUE), ], nrow=N)
    llike.OLD <- evaluate_log_likelihood(X, p.OLD, Theta.OLD)
    for (t in 1:t.max) {
      Z.NEW <- get_S(X, p=p.OLD, Theta=Theta.OLD)
      u <- colSums(Z.NEW)
      p.NEW <- u/N
      v <- t(X) %*% Z.NEW
      Theta.NEW <- trim(t(t(v)/u), tol = 1e-4)
      llike.NEW <- evaluate_log_likelihood(X, p.NEW, Theta.NEW)
      if (is.nan(llike.NEW)) {
        if (verbose) print("Divergence! Restarting with different initial values.")
        break
      }
      if (abs(llike.NEW - llike.OLD) < 1e-4) {
        if (verbose) print(paste("Convergence reached in", t, "iterations."))
        if (llike.NEW > llike.save) {
          if (verbose) print("Better local max achieved.")
          p.save <- p.NEW
          Theta.save <- Theta.NEW
          Z.save <- Z.NEW
          llike.save <- llike.NEW
        }
        break
      }
      llike.OLD <- llike.NEW
      p.OLD <- p.NEW
```

```
      Theta.OLD <- Theta.NEW
    }
  }

  maximizers <- list(p=p.save, Theta=Theta.save, Z=Z.save, llike=llike.save)
  return(maximizers)
}

###  Gibbs Sampler functions

draw_posterior_samples <- function(dat, init, t.max = 7500, burn = 2500, verbose = FALSE) {

  X <- dat$X
  Z <- dat$Z

  N <- nrow(X)
  D <- ncol(X)
  K <- ncol(Z)

  keep.p     <- vector(mode="list", length=t.max)
  keep.Theta <- vector(mode="list", length=t.max)
  keep.Z     <- vector(mode="list", length=t.max)

  keep.p[[1]]     <- p     <- init$p
  keep.Theta[[1]] <- Theta <- init$Theta
  keep.Z[[1]]     <- Z     <- diag(K)[sample(1:K, N, replace=TRUE), ]

  for (t in 2:t.max) {
    S <- get_S(X, p, Theta)
    Z <- t(apply(S, 1, rmultinom, n=1, size=1))
    u <- colSums(Z)
    p <- rdirichlet(1, 1 + u)
    v <- as.vector(t(X) %*% Z)
    first  <- 1 + v
    second <- 1 + rep(u, each=D) - v
    Theta <- matrix(rbeta(rep(1, K*D), shape1=first, shape2=second), nrow=D, ncol=K)
    ## save samples
    keep.p[[t]] <- p
    keep.Theta[[t]] <- Theta
    keep.Z[[t]] <- Z
    if (verbose & t %% 500 == 0) print(paste("Gibbs: Iteration", t, "complete..."))
  }

  if (verbose) {print("Gibbs sampling complete.")}

  ## remove values for burn-in
  keep.p     <- keep.p[-(1:burn)]
  keep.Theta <- keep.Theta[-(1:burn)]
  keep.Z     <- keep.Z[-(1:burn)]

  ## aggregate posterior samples
  post <- mapply(list, keep.p, keep.Theta, SIMPLIFY=FALSE)
  post <- mapply(lappend, post, keep.Z, SIMPLIFY=FALSE)
```

```
  return(post)
}

gibbs_sampler <- function(dat, t.max=7500, burn=5000, verbose=FALSE, ncore=2) {

  require(gtools)  # rdirichlet, permutations
  require(parallel) # mclapply, mcmapply

  X <- dat$X
  Z <- dat$Z

  N <- nrow(X)
  D <- ncol(X)
  K <- ncol(Z)

  init.list <- list(generate_starting_values(X, K), generate_starting_values(X, K),
                    generate_starting_values(X, K), generate_starting_values(X, K))

  post.list <- lapply(init.list, draw_posterior_samples, dat=dat, t.max=t.max,
                      burn=burn, verbose=verbose)#, mc.cores=ncore)
  post <- do.call(c, post.list)

  ## resolve the label-switching problem
  num.post.samples <- length(post)
  nu.best.list <- rep(list(1:K), num.post.samples)  # start with identity permutation
  nu.previous.matrix <- do.call(rbind, nu.best.list)
  nu.possible <- permutations(K, K, 1:K)  # K! possible labelings
  updated <- rep(TRUE, num.post.samples)

  for (i in 1:20) {
    if (verbose) print(paste("Relabeling: Iteration", i, "..."))
    Qhat <- get_Qhat(post=post, nu.best=nu.best.list, dat=dat, ncore=ncore)
    nu.updated <- find_best_nu(post=post[updated], Qhat=Qhat,
                               nu.possible=nu.possible, dat=dat$X, ncore=ncore)
    nu.updated.matrix <- do.call(rbind, nu.updated)
    nu.best.matrix <- do.call(rbind, nu.best.list)
    nu.best.matrix[updated, ] <- nu.updated.matrix
    updated <- !apply(nu.best.matrix == nu.previous.matrix, 1, all)
    nu.previous.matrix <- nu.best.matrix
    nu.best.list <- split(t(nu.best.matrix), gl(num.post.samples, K))
    if (!any(updated)) {
      if (verbose) print("Relabeling successful.")
      break
    }
  }

  post.reordered <- mapply(correct_labels, post, nu.best.list, SIMPLIFY=FALSE)
  post.p <- lapply(post.reordered, function(lst) lst$p)
  post.Theta <- lapply(post.reordered, function(lst) lst$Theta)
  post.Z <- lapply(post.reordered, function(lst) lst$Z)

  ## posterior means
```

```
  mean.p     <- Reduce("+", post.p)/length(post.p)
  mean.Theta <-  Reduce("+", post.Theta)/length(post.Theta)
  mean.Z     <- Reduce("+", post.Z)/length(post.Z)

  return(list(p=mean.p, Theta=mean.Theta, Z=mean.Z))
}

### Functions to deal with the label-switching problem

get_classification_probability_matrix <- function(iter, nu, X) {
  p <- iter[[1]]
  Theta <- trim(iter[[2]])
  S <- get_S(X, p[nu], Theta[, nu])
  return(S)
}

get_Qhat_helper <- function(post.and.nu.best, dat) {
  post <- list()
  nu.best <- list()
  for (i in 1:length(post.and.nu.best)) {
    post[[i]] <- post.and.nu.best[[i]][[1]]
    nu.best[[i]] <- unlist(post.and.nu.best[[i]][[2]])
  }
  lst <- mapply(get_classification_probability_matrix, post, nu.best,
                MoreArgs=list(X=dat$X), SIMPLIFY=FALSE)
  return(lst)
}

get_Qhat <- function(post, nu.best, dat, ncore) {
  m <- length(post)
  post.and.nu.best <- list()
  for (i in 1:m) {
    post.and.nu.best[[i]] <- list(post[[i]], list(nu.best[[i]]))
  }
  groups <- gl(ncore, ceiling(m/ncore), length=m)

  lst <- mclapply(post.and.nu.best.split, get_Qhat_helper, dat=dat, mc.cores=ncore)
  lst <- do.call(c, lst)
  Qhat <- Reduce("+", lst)/length(lst)
  return(Qhat)
}

find_best_nu <- function(post, Qhat, nu.possible, dat, ncore) {
  m <- length(post)
  groups <- gl(ncore, ceiling(m/ncore), length=m)
  post.split <- split(post, groups)

  lst <- mclapply(post.split, find_best_nu_helper, Qhat=Qhat,
                  nu.possible=nu.possible, dat=dat, mc.cores=ncore)
  nu.best <- do.call(c, lst)
  return(nu.best)
}
```

```
find_best_nu_helper <- function(post, Qhat, nu.possible, dat) {
  nu.best <- lapply(post, go_through_possible_nu, Qhat=Qhat,
                    nu.possible=nu.possible, dat=dat)
  return(nu.best)
}

go_through_possible_nu <- function(post, Qhat, nu.possible, dat) {
  KL <- apply(nu.possible, 1, compute_kullback_leibler_divergence, post=post,
              Qhat=Qhat, dat=dat)
  nu.best <- nu.possible[which.min(KL), ]
  return(nu.best)
}

compute_kullback_leibler_divergence <- function(nu, post, Qhat, dat) {
  S <- get_classification_probability_matrix(post, nu, dat)
  KL <- sum(S * log(S/Qhat))
  return(KL)
}

correct_labels <- function(lst, nu) {
  lst[[1]] <- lst[[1]][nu]
  lst[[2]] <- lst[[2]][, nu]
  lst[[3]] <- lst[[3]][, nu]
  names(lst) <- c("p", "Theta", "Z")
  return(lst)
}

###  Primary function to conduct simulations in known_K.R

run_EM_and_Gibbs <- function(dat) {
  set.seed(dat$seed)

  ### EM
  maximizers <- EM_algorithm(dat)
  true <- permute_data(dat, maximizers$Theta)
  Z.max.cluster <- apply(maximizers$Z, 1, which.max)
  Z.actual.cluster <- apply(true$Z, 1, which.max)
  pct.correctly.clustered.EM <- mean(Z.max.cluster == Z.actual.cluster)

  ### Gibbs
  posterior.means <- gibbs_sampler(dat, t.max=10000, burn=5000, ncore=4)
  true <- permute_data(dat, posterior.means$Theta)
  Z.post.cluster <- apply(posterior.means$Z, 1, which.max)
  Z.actual.cluster <- apply(true$Z, 1, which.max)
  pct.correctly.clustered.Gibbs <- mean(Z.post.cluster == Z.actual.cluster)

  pct.correctly.clustered <- c(pct.correctly.clustered.EM, pct.correctly.clustered.Gibbs)
  return(pct.correctly.clustered)
}

###  LRT + Bootstrap functions

do_bootstrap <- function(t, attempts, N, p, Theta, K) {
```

```
  return(sapply(t, do_bootstrap_helper, attempts=attempts, N=N, p=p,
                Theta=Theta, K=K))
}

do_bootstrap_helper <- function(t, attempts, N, p, Theta, K) {
  dat.bootstrap <- generate_data(t*K*(K+1), N, p, Theta, K)
  max.H0.bootstrap <- EM_algorithm(dat.bootstrap, attempts, fix.K=K)
  max.H1.bootstrap <- EM_algorithm(dat.bootstrap, attempts, fix.K=K+1)
  return(2*(max.H1.bootstrap$llike - max.H0.bootstrap$llike))
}

bootstrap_LRT <- function(dat, attempts=25, t.max=25, verbose=FALSE, ncore=1) {
  X <- dat$X
  Z <- dat$Z

  N <- nrow(X)
  D <- ncol(X)

  approx.pval <- rep(0, 6)
  for (k in 1:6) {
    max.H0 <- EM_algorithm(dat, verbose=verbose, fix.K=k)
    max.H1 <- EM_algorithm(dat, verbose=verbose, fix.K=k+1)
    neg2loglambda <- 2*(max.H1$llike - max.H0$llike)
    t.chunks <- split(1:t.max, gl(ncore, ceiling(t.max/ncore), length=t.max))
    neg2loglambda.bootstrap <- unlist(mclapply(t.chunks, do_bootstrap, attempts=attempts,
                                               N=N, p=max.H0$p, Theta=max.H0$Theta,
                                               K=k, mc.cores=ncore))
    approx.pval[k] <- mean(neg2loglambda < neg2loglambda.bootstrap)
  }
  return(approx.pval)
}

###  Allocation Sampler functions

X_conditional_on_K_and_Z <- function(X, Z, N) {
  X <- matrix(X, nrow=N)
  Z <- matrix(Z, nrow=N)
  lfx <- sum(X_conditional_on_K_and_Z_helper(X, Z))
  return(lfx)
}

X_conditional_on_K_and_Z_helper <- function(X, Z) {
  u <- colSums(Z)
  v <- t(X) %*% Z
  return(rowSums(t(lgamma(t(u - t(v) + 1)) + lgamma(v + 1)) - lgamma(u + 2)))
}

Z_conditional_on_K <- function(Z, N) {
  Z <- matrix(Z, nrow=N)
  K <- ncol(Z)
  u <- colSums(Z)
  lfz <- lgamma(K) - lgamma(K + N) + sum(lgamma(1 + u))
  return(lfz)
```

```
}

evaluate_log_joint_density <- function(X, Z, N) {
  lfx <- X_conditional_on_K_and_Z(X, Z, N)
  lfz <- Z_conditional_on_K(Z, N)
  return(lfx + lfz)
}

find_best_a <- function(u.k1, b0) {
  a <- seq(0.001, 5, by=0.001)
  equation <- lgamma(2*a) - lgamma(a) + lgamma(a + u.k1) - lgamma(2*a + u.k1)
  best.a <- a[which.min(abs(exp(equation) - b0/2))]
  return(best.a)
}

get_b.k1 <- function(X.n.to.realloc, X.reallocated, Z.reallocated) {
  if (is.null(X.reallocated)) {
    b.k1 <- 0.5
  } else {
    u.tilde <- colSums(Z.reallocated)
    log.with.X.n.k1 <- X_conditional_on_K_and_Z_helper(rbind(X.n.to.realloc, X.reallocated),
                                                       rbind(c(1,0), Z.reallocated))[1]
    log.with.X.n.k2 <-  X_conditional_on_K_and_Z_helper(rbind(X.n.to.realloc, X.reallocated),
                                                        rbind(c(0,1), Z.reallocated))[2]
    log.without.X.n <- X_conditional_on_K_and_Z_helper(X.reallocated, Z.reallocated)
    lnumer <- log(1 + u.tilde[1]) + log.with.X.n.k1 + log.without.X.n[2]
    ldenom <- log(1 + u.tilde[2]) + log.with.X.n.k2 + log.without.X.n[1]
    frac <- exp(lnumer - ldenom)
    b.k1 <- frac/(1 + frac)
  }
  return(b.k1)
}

allocation_sampler <- function(dat, t.max=20000, burn=10000, verbose=FALSE) {
  X <- dat$X
  Z <- dat$Z

  N <- nrow(X)
  D <- ncol(D)

  K.max <- 10
  K.init <- 1
  Z.init <- diag(K.init)[sample(1:K.init, N, replace=TRUE), , drop=FALSE]
  keep.K <- rep(0, t.max)
  keep.Z <- vector(mode="list", length=t.max)

  keep.K[1] <- K <- K.init
  keep.Z[[1]] <- Z <- Z.init

  for (t in 2:t.max) {
    move.chosen <- sample(c("AE", "GS", "M1", "M2", "M3"), 1)
    if (K == 1) {
      move.chosen <- "AE"
```

```
}

if (move.chosen == "AE") {
  prob.of.ejection <- 0.5
  if (K == 1) {
    prob.of.ejection <- 1
  } else if (K == K.max) {
    prob.of.ejection <- 0
  }
  if (runif(1) < prob.of.ejection) {  # attempt ejection
    k1 <- sample(1:K, 1)
    k2 <- K + 1
    u.k1 <- colSums(Z)[k1]
    a <- find_best_a(u.k1, b0=0.2)
    b.k2 <- rbeta(1, shape1 = a, shape2 = a)
    alloc.to.k2 <- runif(u.k1) < b.k2
    new.Z <- cbind(Z, 0)
    new.Z[new.Z[, k1] == 1, k2] <- as.numeric(alloc.to.k2)
    new.Z[new.Z[, k1] == 1, k1] <- 1 - as.numeric(alloc.to.k2)
    ## Do we accept this new allocation?
    u.tilde <- colSums(new.Z)
    u.tilde.k1 <- u.tilde[k1]
    u.tilde.k2 <- u.tilde[k2]
    log.proposal.ratio <- 2*lgamma(a) - lgamma(2*a) + lgamma(2*a + u.k1) -
      lgamma(a + u.tilde.k1) - lgamma(a + u.tilde.k2) + log(10)
    log.density.ratio <- evaluate_log_joint_density(X, new.Z, N)  -
      evaluate_log_joint_density(X, Z, N)
    R <- exp(log.density.ratio + log.proposal.ratio)
    if(runif(1) < min(c(1, R))) {
      new.k2 <- sample(1:k2, 1) # label switch for reversibility in symmetric case
      new.Z[, c(k2, new.k2)] <- new.Z[, c(new.k2, k2)]
      Z <- new.Z
      K <- K + 1
    }
  } else {  # attempt absorption
    k1.and.k2 <- sample(1:K, 2)
    k1 <- k1.and.k2[1]  # absorbing component
    k2 <- k1.and.k2[2]  # absorbed component
    u.tilde <- colSums(Z)
    u.tilde.k1 <- u.tilde[k1]
    u.tilde.k2 <- u.tilde[k2]
    new.Z <- Z
    new.Z[new.Z[, k2] == 1, k1] <- 1
    new.Z <- new.Z[, -k2]
    new.Z <- matrix(new.Z, nrow=N)
    u.k1 <- u.tilde.k1 + u.tilde.k2
    a <- find_best_a(u.k1, b0=0.2)
    log.proposal.ratio <- 2*lgamma(a) - lgamma(2*a) + lgamma(2*a + u.k1) -
      lgamma(a + u.tilde.k1) - lgamma(a + u.tilde.k2)
    log.density.ratio <- evaluate_log_joint_density(X, new.Z, N)  -
      evaluate_log_joint_density(X, Z, N)
    R.inverse <- exp(log.density.ratio - log.proposal.ratio)
    if(runif(1) < min(c(1, R.inverse))) {
```

```
      Z <- new.Z
      K <- K - 1
    }
  }
}

if (move.chosen == "GS") {
  for (n in 1:N) {
    new.Z <- Z
    ljoint <- rep(1, K)
    for (k in 1:K) {
      new.Z[n, ] <- ifelse(k == 1:K, 1, 0)
      fx <- X_conditional_on_K_and_Z(X, new.Z, N)
      fz <- Z_conditional_on_K(new.Z, N)
      ljoint[k] <- fx + fz
    }
    joint <- exp(ljoint - min(ljoint))
    Z[n, ] <- rmultinom(1, 1, prob=joint)
  }
}
if (move.chosen == "M1") {
  k1.and.k2 <- sample(1:K, 2)
  in.k1.or.k2 <- apply(Z, 1, which.max) %in% k1.and.k2
  b.k1 <- runif(1)
  alloc.to.k1 <- runif(sum(in.k1.or.k2)) < b.k1
  new.Z <- Z
  new.Z[in.k1.or.k2, k1.and.k2[1]] <- as.numeric(alloc.to.k1)
  new.Z[in.k1.or.k2, k1.and.k2[2]] <- 1 - as.numeric(alloc.to.k1)
  R <- exp(X_conditional_on_K_and_Z(X, new.Z, N) -
             X_conditional_on_K_and_Z(X, Z, N))
  if (runif(1) < min(c(1, R))) {  # accept new allocation?
    Z <- new.Z
  }
}
if (move.chosen == "M2") {
  k1.and.k2 <- sample(1:K, 2)
  k1 <- k1.and.k2[1]
  k2 <- k1.and.k2[2]
  in.k1 <- Z[, k1] == 1
  u <- colSums(Z)
  u.k1 <- u[k1]
  u.k2 <- u[k2]
  if (u.k1 > 0) {
    m <- sample(1:u.k1, 1)
    alloc.to.k2 <- sample(which(in.k1), m)
    new.Z <- Z
    new.Z[alloc.to.k2, ] <- ifelse(k1.and.k2[2] == 1:K, 1, 0)
    ## Do we accept this new allocation?
    log.proposal.ratio <- log(u.k1) - log(u.k2 + m) + lfactorial(u.k1) +
      lfactorial(u.k2) - lfactorial(u.k1 - m) - lfactorial(u.k2 + m)
    log.density.ratio <- evaluate_log_joint_density(X, new.Z, N) -
      evaluate_log_joint_density(X, Z, N)
    R <- exp(log.density.ratio + log.proposal.ratio)
```

```
    if(runif(1) < min(c(1, R))) {
      Z <- new.Z
    }
  }
}

if (move.chosen == "M3") {
  k1.and.k2 <- sample(1:K, 2)
  in.k1.or.k2 <- apply(Z, 1, which.max) %in% k1.and.k2
  number.to.realloc <- sum(in.k1.or.k2)
  if (number.to.realloc > 1) {
    b.k1 <- rep(0, number.to.realloc)
    shuffle <- sample(1:number.to.realloc, number.to.realloc)
    X.to.realloc <- X[in.k1.or.k2, , drop=FALSE][shuffle, , drop=FALSE]
    X.reallocated <- NULL
    Z.reallocated <- NULL
    for (each.n in 1:number.to.realloc) {
      X.n.to.realloc <- X.to.realloc[each.n, , drop=FALSE]
      b.k1[each.n] <- get_b.k1(X.n.to.realloc, X.reallocated, Z.reallocated)
      alloc.to.k1 <- runif(1) < b.k1[each.n]
      if (alloc.to.k1) {
        Z.n.reallocated <- c(1, 0)
      } else {
        Z.n.reallocated <- c(0, 1)
      }
      Z.reallocated <- rbind(Z.reallocated, Z.n.reallocated)
      X.reallocated <- rbind(X.reallocated, X.n.to.realloc)
    }
    ## un-shuffle the values
    Z.reallocated <- Z.reallocated[order(shuffle), ]
    b.k1 <- b.k1[order(shuffle)]
    new.Z <- Z
    new.Z[in.k1.or.k2, k1.and.k2] <- Z.reallocated
    ## Do we accept this new allocation?
    b.for.Z     <- trim(ifelse(apply(Z[in.k1.or.k2, , drop=FALSE], 1,
                                     which.max) == k1.and.k2[1],
                               b.k1, 1 - b.k1), 1e-8)
    b.for.new.Z <- trim(ifelse(apply(new.Z[in.k1.or.k2, , drop=FALSE], 1,
                                     which.max) == k1.and.k2[1],
                               b.k1, 1 - b.k1), 1e-8)
    log.proposal.ratio <- sum(log(b.for.Z) - log(b.for.new.Z))
    log.density.ratio <- evaluate_log_joint_density(X, new.Z, N) -
      evaluate_log_joint_density(X, Z, N)
    R <- exp(log.density.ratio + log.proposal.ratio)
    if(runif(1) < min(c(1, R))) {
      Z <- new.Z
    }
  }
}

keep.K[t] <- K
Z <- matrix(Z, nrow=N)
keep.Z[[t]] <- Z
```

```
    if(verbose & t %% 500 == 0) print(paste(K, "Allocation Sampler: Iteration",
                                            t, "complete..."))
  }

  return(keep.K[-(1:burn)])
}

parallel_allocation_sampler <- function(dat, t.max=20000, burn=10000,
                                        verbose=FALSE, ncore=4) {
  dat4 <- list(dat, dat, dat, dat)
  post.K <- mclapply(dat4, allocation_sampler, t.max=t.max, burn=burn,
                     verbose=verbose, mc.cores=ncore)
  post.K <- table(factor(unlist(post.K), levels=1:6))
  post.K <- post.K/sum(post.K)
  return(unlist(post.K))
}

###  Primary function to conduct simulations in unknown_K.R

run_LRT_and_AS <- function(dat) {
  set.seed(dat$seed)

  ### LRT
  approx.pval <- bootstrap_LRT(dat, ncore=4)

  ### Allocation Sampler
  posterior.K <- parallel_allocation_sampler(dat, ncore=4)

  possible.K <- matrix(c(approx.pval, posterior.K), ncol=1)
  return(possible.K)
}
```