A hybrid approach to clustering optimization

by

Dustin Hayes

B.A., Kansas State University, 2021

B.A., Kansas State University, 2019

A REPORT

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Statistics
College of Arts and Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2023

Approved by:

Major Professor
Dr. Gyuhyeong Goh

Copyright

Abstract

A known limitation of many clustering algorithms is their inability to guarantee convergence to global optimality. The K-means clustering algorithm, as a representative example, given some data $\boldsymbol{x}$ and some objective function L which we aim to optimize, only ensures that clustering assignments are found such that L is optimized locally. Consequently, one can not generally know after one application of the K-means clustering algorithm if the globally optimal or an inferior locally optimal solution was found. This limitation is not exclusive to the K-means algorithm; other algorithms, such as the EM algorithm for Gaussian mixture models, are also liable to converge to a sub-optimal solution.

In practice, one can somewhat avoid this limitation by running their clustering algorithm of choice a number of times and selecting the best solution found. However, this approach is uncertain by nature. At no point would the practitioner know whether or not the optimal solution had been found or if more attempts were required. A single algorithm which offers a higher degree of assurance that the global optimum was reached would be preferable.

In this paper we explore existing clustering methods, discuss their limitations, and present a novel clustering algorithm, with implementation in R, that achieves a higher assurance of global optimization than traditional clustering methods. By strategically passing the optimization problem between an EM algorithm and a Gibbs sampler, our algorithm takes advantage of both local optimization and global search methods to explore the complex loss-landscape of the objective function, ensuring more reliable convergence to optimal solutions.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to acknowledge Dr. Gyuhyeong Goh for his guidance. His continued support and expertise have been invaluable. I would also like to thank Dr. Haiyan Wang and Dr. Trevor Hefley for agreeing to serve as members of my committee and for all they have taught me. I will always look back on my time at the Kansas State University statistics department with deep appreciation and fondness.

Dedication

To my lovely partner, Maria Jose Vera Carvajal: Thank you for your patience and support, and for letting me practice my presentation with you even though you aren't a huge fan of statistics. I promise our evenings will be back to normal the moment all of this is over.

Chapter 1

Introduction

Clustering algorithms have become essential tools for modern data analysis, with applications in fields as diverse as bio informatics, energy studies, machine learning, networking and pattern recognition[1]. Unlike supervised algorithms, which use labeled data to learn patterns then used to apply new labels to unseen data, clustering is an un-supervised approach. The purpose is not to uncover the characteristics of those points which belong to a particular class but to instead generate meaningful clusters of similar points using their inherent properties[2]. What precisely defines the notions of a "cluster" and "similarity" is not ubiquitous among clustering algorithms. Since the mid 20th century various algorithmic strategies for clustering have emerged including K-means clustering, soft clustering algorithms, hierarchical clustering, density based methods and grid-based methods. Each of these algorithms frames the clustering problem in a different light, and each comes with its own set of advantages and limitations.

The purpose of our work is to address a significant limitation shared by a number of clustering algorithms, including the highly popular K-means and soft clustering algorithms: that convergence is only guaranteed to a locally optimal solution, not the globally optimal solution. These algorithms behave such that each iteration results in a monotonically improving value of the objective function. Although this behavior might appear to be wholly advantageous, it can and frequently does result in a situation where the algorithm gets "stuck" in at a sub-optimal solution which may not be very representative of the true underlying nature of the data. As the dimensionality and complexity of the data increases and the loss-landscape becomes increasingly multi-modal, the danger of landing in a superficial local

optimum increases as well.

We propose a novel clustering algorithm which seeks to address this limitation by combining the local search capabilities of the EM algorithm with global exploration of the loss-landscape afforded by the Gibbs sampler algorithm. We begin by discussing in general the algorithms which underpin our proposed method: K-means clustering, soft clustering, the EM algorithm, and the Gibbs sampler. We will then elucidate the implementation and purpose of each algorithmic component in the context of our hybrid algorithm. Finally, we will compare the behavior of our algorithm to existing methods with a real data analysis study and offer some closing thoughts. The implementation of our algorithm in R will also be included in appendix A.

Chapter 2

K-means Clustering

K-means clustering is among the best-known clustering practices in use today. J. MacQueen is frequently credited with introducing the technique in 1967[3]. Despite its limitations, K-means clustering continues to find use in wide variety of domains and applications.

The description of and notation for the K-means clustering algorithm presented in this section are heavily influenced by "Introduction to Statistical Learning" by James, Witten, Hastie and Tibshirani[4]. Since our proposed method shares many conceptual elements with the K-means clustering algorithm, we thought it prudent to utilize a clear, well-known text to clearly establish a baseline understanding of the concepts we will later expand on.

We begin with some data $\boldsymbol{x}$ which we seek to perform K-means clustering analysis on. We must first choose the number of clusters, K, that K-means clustering will endeavor to find. The selection of K is highly consequential. Since the K-means algorithm will always find exactly K many clusters, if K is too small, the algorithm will fail to discover meaningful relationships. If K is too large the algorithm will report spurious relationships: it will report the existence of clusters which may not be representative of the true nature of the data. A number of methods exist for choosing K. Determining which of these methods is superior, and under what circumstances, is an ongoing area of research which is somewhat beyond the scope of this work. We chose to utilize Bayesian Information Criterion (BIC), which has the advantages of being statistically grounded and computationally simple, for selecting K.

Now, we define the notation with which we will explore the K-means clustering algorithm.

- K : The number of clusters our algorithm will seek.
- N : The number of observations in our data.

- d : The number of features.
- $\boldsymbol{x}$: The data matrix, where each row corresponds to an individual observation and each column represents a variable. Therefore, $\boldsymbol{x}$ is of dimension $N \times d$.
- x_i : The i^{th} observation in the dataset.
- x_{ij} : The j^{th} feature of the i^{th} observation in the dataset.
- $C_1, \ldots, C_K$: Sets indicating cluster assignment, where C_k contains the indices of the points in the k^{th} cluster.
- $\bar{x}_k$: The mean vector of all points assigned to C_k
- $\bar{x}_{jk}$: The j^{th} feature of $\bar{x}_k$

Our purpose is to find sets $C_1, \ldots, C_K$ such that within cluster variation is minimized. Allow $W(C_k)$ to represent the variation within cluster k. We endeavor to find $C_1, \ldots, C_K$ such that $\sum_{k=1}^{K} W(C_k)$ is minimized.

There are multiple ways by which one might formalize the notion of within cluster variation W. The most common means of quantifying within cluster variation, and the one which we will employ, is *squared Euclidean distance*

$$W(C_k) = \sum_{i \in C_k} \sum_{j}^{d} (x_{ij} - \bar{x}_{jk})^2.$$

Where $\bar{x}_k$ is the mean vector of all points assigned to C_k, and $\bar{x}_{jk}$ is the j^{th} component of that vector. $\bar{x}_k$ is frequently referred to as the "centroid" of the k^{th} cluster.

2.1 The Algorithm

The K-means algorithm is a heuristic method which has been shown to be effective in many situations, despite that solving the problem directly is NP-hard[5]. The algorithm begins with the selection of K according to the method of the practitioner's choosing. Sets $C_1, \ldots, C_K$

are then initialized. The latter step is often done by assigning each data point to one of the K clusters randomly according to a discrete uniform distribution. One may alternatively randomly select K points in the dataset to serve as the initial centroids, then assign each point to the nearest centroid.

The algorithm is quite simple; each iteration consists of two steps which repeat until convergence. First, the centroids or mean vectors of each cluster $\bar{x}_k$ are calculated. Next, each point is assigned to the cluster with the closest centroid according to a chosen distance metric. The algorithm consists of repeating these two steps until convergence: we then calculate the centroids of each cluster again, and the algorithm continues as such. The algorithm terminates once clusters stabilize i.e., $C_1, \ldots, C_K$ remain unchanged after one iteration.

Algorithm 1 K-means clustering

while $C_1^{new}, \ldots, C_K^{new} \neq C_1^{old}, \ldots, C_K^{old}$ **do**
  for all clusters k **do**
    Compute centroid $\bar{x}_k = \frac{1}{|C_k|} \sum_{i \in C_k} x_i$
  end for
  for all x_i in dataset **do**
    Assign x_i to the cluster C_k^{new} with the closest centroid $\bar{x}_k$
  end for
  Update $C_1^{old}, \ldots, C_K^{old}$ to $C_1^{new}, \ldots, C_K^{new}$
end while
return $C_1, \ldots, C_K$

Each iteration of the K-means clustering algorithm will reduce $\sum_{k=1}^{K} W(C_k)$ until convergence, and convergence to some local minimum is guaranteed. Significant work has been done to optimize the K-means algorithm for use at scale[6]. These characteristics, along with the algorithm's simplicity and ease of use, have made the K-means algorithm quite popular.

Despite its popularity, the K-means algorithm has drawbacks and limitations. It is not resistant to outliers, which may heavily distort the shape of resultant clusters[4]. The assumption of euclidean space makes the use of K-means clustering inappropriate for data with categorical variables. The K-means clustering algorithm is also not well suited to non-spherical clusters. As we have mentioned previously, K-means clustering is only guaranteed

to find a locally optimal solution. Furthermore, one must always face the difficult problem of choosing the optimal K for their use case. We will discuss the latter two limitations in greater detail in this work.

The requirement that K-means clustering assigns every point to one and only one cluster might also be thought of as a limitation. Although this behavior is perfectly suitable for some applications, one can imagine a situation where greater flexibility would be desired. We may prefer to instead report the degree to which a particular point belongs to each cluster. For example, if we were to imagine a point which exists exactly midway between two clusters, we might wish to describe the point as belonging to both clusters equally, instead of forcing the point to belong to one cluster or the other.

Chapter 3

Soft Clustering

Soft clustering algorithms are in principle very similar to the traditional K-means algorithm. The primary conceptual difference is that cluster memberships are "soft"; instead of assigning each point to one and only one cluster as in K-means, we allow each point to belong to all clusters but to varying degrees. This approach is less ridged than forcing "hard" clustering assignments and permits us to report cluster assignments in a more nuanced manner.

For each data point i, allow ω_{ik} to be the i^{th} point's degree of membership to the k^{th} cluster, and allow ω_i to be a K-dimensional vector which contains all membership degrees for the i^{th} point. Each ω_{ik} takes a value between 0 and 1, where 1 represents full membership and 0 represents no membership. We require that the sum of membership degrees for each point equals 1. This is a sensible constraint, as it allows us to interpret $\{\omega_{i1}, \ldots, \omega_{iK}\}$ for each point i as a discrete probability distribution. It will become clear shortly why we desire for ω_i to have this interpretation.

$$\sum_{k=1}^{K} \omega_{ik} = 1, \quad \text{for all} \quad i.$$

Finally, we define an $N \times k$ matrix $\mathbf{\Omega}$, which contains all membership degrees.

$$\mathbf{\Omega} = \begin{bmatrix} \omega_{11} & \omega_{12} & \ldots & \omega_{1K} \\ \omega_{21} & \ldots & \ldots & \omega_{2K} \\ \vdots & \vdots & \vdots & \vdots \\ \omega_{N1} & \ldots & \ldots & \omega_{NK} \end{bmatrix}$$

$\mathbf{\Omega}$ contains all information regarding our clustering assignments for our N data points

and is analogous to sets $C_1, \ldots, C_K$ in the previous chapter.

Previously, we employed *squared Euclidean distance* to define the notion of within-cluster variation, which we set out to minimize. In this chapter and thereafter our model for clustering will permit us to adopt the objective of maximizing likelihood instead. In doing so, we benefit from a more statistically justifiable formulation which elucidates cluster membership and provides additional information regarding the underlying structure of our data.

3.1 The EM algorithm

The Expectation-Maximization (EM) algorithm is a statistical algorithm that is commonly used to model data that includes missing or latent variables. It was first introduced by A. P. Dempster, N. M. Laird and D. B. Rubin in 1977, although the authors themselves note that special cases of the EM algorithm had been developed prior to their contribution[7]. We will introduce the algorithm generally here, and will later apply it specifically to cluster analysis.

Consider a set of data $\boldsymbol{x}$, which was generated by some process which we parameterize with θ, with some log likelihood $\mathcal{L}(\theta \mid \boldsymbol{x})$. Also assume that there exists some latent or missing data Z. Our purpose is to maximize $\mathcal{L}$. Assume that directly maximizing $\mathcal{L}(\theta \mid \boldsymbol{x})$ is difficult or impossible due to our lack of knowledge of Z. The EM algorithm is capable of finding θ such that $\mathcal{L}(\theta \mid \boldsymbol{x})$ is maximized in many such situations.

The EM algorithm utilizes a function called $Q(\theta \mid \theta^{(t)})$, where $\theta^{(t)}$ represents our estimation of θ at iteration t

$$Q(\theta \mid \theta^{(t)}) = \mathbb{E}_{Z \sim p(z \mid \theta^{(t)}, \boldsymbol{x})} \left[\ln p(\boldsymbol{x}, Z \mid \theta) \right].$$

Q is the expected value of conditional distribution of the full data $\ln(p(\boldsymbol{X}, Z \mid \theta))$ with respect to the posterior distribution of Z. The EM algorithm consists of two steps which repeat until convergence is reached:

- **E-step**: Compute $Q(\theta \mid \theta^{(t)})$ using the current estimate of $\theta^{(t)}$.

- **M-step**: Find $\theta^{(t+1)} = \text{argmax}_\theta \, Q(\theta \mid \theta^{(t)})$ to update θ.

It can be shown that updating θ in this fashion always results in non-decreasing changes to the log-likelihood $\mathcal{L}(\theta \mid \boldsymbol{x})$[7]. Convergence criteria may vary. Generally, one will run the EM algorithm until θ and/or $\mathcal{L}$ change less than some small, predefined threshold after one iteration.

3.2 EM algorithm for Gaussian mixture models

In this section we will present the formulation of the Gaussian mixture model, detail how the EM algorithm can applied to it, and show that the EM algorithm when used in this fashion achieves soft clustering through iterative estimation of latent variables Z.

We first define a Gaussian mixture model, or GMM. Modeling our data $\boldsymbol{x}$ with a GMM implies that we assume our data to have been generated by K-many normal distributions with mixture weights π_k

$$x_i \sim \sum_{k=1}^{K} \pi_k \mathcal{N}(x_i \mid \mu_k, \Sigma_k).$$

where the mixture weights satisfy $\sum_{k=1}^{K} \pi_k = 1$, each μ_k is the mean vector of the k^{th} component, and Σ_k is the diagonal covariance matrix of the k^{th} component. Let $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ signify matrices containing the mean vectors and diagonal terms of the covariance matrices for each component, respectively. The mixture weights are denoted by the vector π. The log-likelihood of the entire data set $\boldsymbol{x}$ modeled with a GMM is then:

$$\ln p(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}, \pi) = \sum_{i=1}^{N} \ln \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_i \mid \mu_k, \Sigma_k) \right).$$

There are significant problems with solving for the maximum likelihood estimators directly[8]. We can, however, use the EM algorithm to arrive at good estimations of $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$, and π, the collection of which we will subsequently refer to as $\boldsymbol{\theta}$.

Let us define an N dimensional vector of latent variables called Z that indicates which

components of our mixture model generated each of our data points. Precisely, Z is composed of elements z_i which take a value in $\{1, 2, \ldots, K\}$, where the value of each element z_i indicates the mixture component which generated the i^{th} data point. Z acts as the "missing data" which motivates the use of the EM algorithm, the idea being that, if we did know Z, solving for maximum likelihood estimators would be quite simple.

Applying the EM algorithm in this circumstance will result in an estimation of parameters $\boldsymbol{\theta}$, as well an estimation of latent variables Z. We will define the probability of the i^{th} data point having been generated by the k^{th} component as follows:

$$P(z_i = k \mid x_i, \boldsymbol{\theta}) = \omega_{ik}. \tag{3.1}$$

Note that this probability can be calculated readily by applying the Bayes rule:

$$\omega_{ik} = \frac{\pi_k \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_{m=1}^{K} \pi_m \mathcal{N}(x_i \mid \mu_m, \Sigma_m)}.$$

ω denotes "membership degrees" as they were referred during our discussion of soft clustering in general. These membership weights reflect the probability that a given point was generated by a particular component of the mixture model, as represented in equation (3.1). Note that the notions of a "cluster" and of a "component" of our mixture model are, in this context, equivalent.

Just as before, we define $\boldsymbol{\Omega}$ to be the collection of these membership degrees: an $N \times K$ matrix where each entry ω_{ik} indicates the degree of membership of the i^{th} data point with respect to the k^{th} mixture component. Estimating $\boldsymbol{\Omega}$, as we will see, is essential to the E-step of the algorithm. In fact, when formulated for a GMM, Q explicitly contains these membership degrees:

$$
\begin{aligned}
Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(t)}) &= \mathrm{E}_{Z|\boldsymbol{x},\boldsymbol{\theta}^{(t)}} \ln L(\boldsymbol{\theta} \mid \boldsymbol{x}, Z) \\
&= \sum_{i=1}^{N} \sum_{k=1}^{K} P(z_i = k \mid x_i, \boldsymbol{\theta}^{(t)}) \ln(\pi_k \mathcal{N}(x_i \mid \mu_k, \Sigma_k)) \\
&= \sum_{i=1}^{N} \sum_{k=1}^{K} \omega_{ik} \ln(\pi_k \mathcal{N}(x_i \mid \mu_k, \Sigma_k))
\end{aligned}
$$

In essence, the process of estimating $\boldsymbol{\Omega}$ in the E-step also provides the information needed to determine the value of Q. Thus, the execution of the E-step is, in effect, tantamount to determining $\boldsymbol{\Omega}$, our membership degrees.

Carrying out the M-step involves taking the partial derivatives of Q with respect to each of the parameters in $\boldsymbol{\theta}$, setting the resulting equations to zero, and solving the system for $\boldsymbol{\theta}^{(t+1)}$. We will not present the derivation here; this process ultimately amounts to updating our parameters according to the formulas (3.2), (3.3), and (3.4). Note that, for simplicity, we have elected to set all off-diagonal terms of each Σ_k to zero. We will enumerate the elements of each Σ_k with a single index j, which takes a value in $\{1, \ldots, d\}$, that indicates both the row and column position of the element in the diagonal matrix. For clarity:

$$
\Sigma_k = Diag(\sigma_{1k}, \ldots, \sigma_{dk}) = \begin{bmatrix} \sigma_{1k} & 0 & \cdots & 0 \\ 0 & \sigma_{2k} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_{dk} \end{bmatrix}.
$$

This simplifying assumption does not perturb the fundamental nature of this process. Taking $N_k = \sum_{i=1}^{N} \omega_{ik} \ldots$

$$
\pi_k^{(t+1)} = N_k / N \tag{3.2}
$$

$$\mu_k^{(t+1)} = (1/N_k) \sum_{i=1}^{N} \omega_{ik} x_i \tag{3.3}$$

$$\Sigma_k^{(t+1)} = \frac{1}{N_k} \sum_{i=1}^{N} \omega_{ik} (x_i - \mu_k^{(t+1)})(x_i - \mu_k^{(t+1)})^T. \tag{3.4}$$

The EM-algorithm can now be written in a manner specific to GMMs:

- **E-step**: Compute ω_{ik} for each pairing of data point and mixture component using (3.2).
- **M-step**: Recalculate our estimates for each parameter in $\boldsymbol{\theta}$ using equations (3.2), (3.3), and (3.4).

The algorithm continues until the convergence criterion is met. The final estimate of $\boldsymbol{\theta}$ constitutes maximum likelihood estimators for our model. The final estimation of $\boldsymbol{\Omega}$ yields our soft clustering assignments.

The EM algorithm used in this fashion provides a robust method for achieving soft clustering. However, it is not without its own set of limitations, many of which are shared with other clustering algorithms. Notably, as is the case with K-means clustering, the EM algorithm only guarantees convergence to a locally optimal solution[9]. Likewise, the requirement that K well chosen persists. The choice of K precedes the implementation of the algorithm, is frequently not a simple choice, and significantly impacts the usefulness the results.

A number of methods exist for choosing K which extend to both hard and soft clustering. In the following chapter we will briefly describe our chosen method: minimization of the Bayesian information criterion (BIC). Our discussion of the local minimum problem, which is the crux of our study, will be further examined in chapter 5.

Chapter 4

Selection of K

As is the case with the K-means clustering algorithm and the EM algorithm, we are required to choose an appropriate value of K before our proposed algorithm can begin. Various methods exist for selecting an appropriate value for K. In applied contexts, one might first make an informed guess based on some domain knowledge. One might also have an idea as to how many clusters to expect based on previous analysis. Here, however, we wish to employ a systematic method that is not dependent on domain knowledge or previous experience. Instead, we wish to make our selection based on some readily calculable criterion.

We have chosen to make use of the Bayesian Information Criterion to inform our selection of K. The Bayesian Information Criterion, which we will subsequently refer to as BIC, is calculated as follows:

$$BIC = Cln(N) - 2ln(\hat{L}) \tag{4.1}$$

Where C is equal to the number of free parameters in our model, N is equal to the number of data points, and $\hat{L}$ is the maximized likelihood function. Since an increase in K will always result in an increase in the maximized likelihood function, regardless of how informative additional clusters might be, we require some means to penalize increased complexity of the model. The BIC weighs the complexity of the model, communicated by the first term $Cln(N)$, against the natural logarithm of the maximized likelihood function. We wish, therefore, to find the selection of K which results in the smallest BIC possible.

We will utilize the EM algorithm in our search of the value of K which succeeds in minimizing BIC. We may write (4.1) more precisely,

$$\text{BIC} = C\ln(N) - 2\ln\left(p(\boldsymbol{x} \mid \boldsymbol{\mu}_{\text{EM}}, \boldsymbol{\Sigma}_{\text{EM}}, \pi_{\text{EM}})\right), \tag{4.2}$$

where the subscript "EM" is used to denote the maximum likelihood estimate of a given parameter obtained by running the EM algorithm until convergence. Our method for determining K consists of running the EM algorithm and recording the BIC according to (4.2) for each value of K we suspect might be appropriate. We then choose the K which results in the lowest BIC as the most appropriate for our data, and consequentially utilize this value for the remainder of our algorithm. Note that we anticipate the relationship between K and BIC to be uni-modal i.e, we expect to see only one minimum at the appropriate value of K.

Chapter 5

Proposed Algorithm

We propose an algorithm which seeks to address a fundamental limitation of both the EM algorithm and the K-means algorithm: that convergence is only guaranteed to a locally optimal solution. Our algorithm utilizes the strategic interplay of the EM and Gibbs algorithm in an attempt to capitalize on the advantages of each. We use the EM algorithm for efficient local optimization and the Gibbs sampler to explore the loss landscape globally. We begin this section with a brief discussion of the Gibbs algorithm in general. We will then present and discuss our hybrid method.

A number of studies have attempted to improve existing clustering algorithms such that they are more likely to achieve a globally optimal solution[10], [11]. The conditions under which clustering algorithms achieve globally optimal solutions have also been studied[12]. To the best of our knowledge, this is the first work to introduce a hybrid methodology that combines the strengths of the EM and Gibbs algorithms to pursue global optimization in clustering problems.

5.1 Gibbs sampler

The Gibbs sampler is a Markov Chain Monte Carlo (MCMC) algorithm used for generating samples from multivariate distributions. It is useful when directly sampling from their joint distribution is challenging or impossible, but sampling from the full conditional distribution of each random variable is not. The Gibbs sampler is frequently implemented in a Bayesian context in order to sample from the full posterior distribution. We will utilize the Gibbs

sampler in this context as well. In the interest of providing a clear baseline for understanding, the following introductory material draws on the discussion on Gibbs sampling presented in "Bayesian Data Analysis" by Gelman, Carlan, Stern, Dunson, Vehtari and Rubin[13].

Assume that we seek to generate samples from a vector of parameters θ which have generated data $\boldsymbol{x}$

$$\theta = (\theta_1, \theta_2, \ldots, \theta_n).$$

Using a Gibbs sampler to sample from θ entails iteratively sampling from each component of θ conditioned on all other elements in θ and the data $\boldsymbol{x}$. Allow $\theta_{-k}^{(t-1)}$ to represent all components of θ except for the k^{th} component at the $(t-1)^{th}$ iteration. At the t^{th} iteration, Gibbs sampling requires sampling from

$$p(\theta_k^{(t)} \mid \theta_{-k}^{(t-1)}, \boldsymbol{x}) \quad \text{for all} \quad k.$$

The components of $\theta_{-k}^{(t-1)}$ constitute the most recent samples at iteration t except for the k^{th} component. This is to say that, if we are currently sampling from the third component of θ at the t^{th} iteration, $\theta_3^{(t)}$ will be conditioned on $\theta_{-3}^{(t-1)} = (\theta_1^{(t)}, \theta_2^{(t)}, \theta_4^{(t-1)}, \ldots, \theta_K^{(t-1)})$.

Clearly, initial values for θ must be chosen before this process can begin. These values may be chosen randomly, strategically, or by more sophisticated methods. At the beginning of the process, the samples obtained through Gibbs sampling are sensitive to our initial selection. After a sufficient number of iterations, however, our sampler will have reached what is called the "stationary" distribution, at which point we will be sampling from the true posterior distributions that we are interested in. How many samples are required to reach the stationary distribution is a function of your choice of initial samples; choices which are not very representative of the true underlying distributions can increase the time required to converge to the stationary distribution. It is not straightforward to determine how many samples will be required until convergence, or how a researcher might know whether or not the stationary distribution has been reached. As we will find, these concerns are not critical to our particular application of the Gibbs sampler.

We will take advantage of the Gibbs sampler to generate samples from the full conditionals of the parameters of our GMM. In order to do so, we must select prior distributions and derive the full conditionals of each of the parameters of interest: $\boldsymbol{\mu}, \boldsymbol{\Sigma}, \pi$ as well as Z.

As was the case when we examined the EM algorithm, we will assume that the all covariance matrices are diagonal. Before reporting the priors we used and their derived posteriors, we should introduce additional notation.

- a_0 : A hyper-parameter used to parameterize the prior of π.
- μ_0 : A hyper-parameter used to represent the mean of the prior distributions of $\boldsymbol{\mu}$.
- α_0 : A hyper-parameter used to parameterize the priors of $\boldsymbol{\Sigma}$.
- β_0 : A hyper-parameter used to parameterize the priors of $\boldsymbol{\Sigma}$.

Allow us to formulate our model once more. Consider a Gaussian mixture model consisting of K components such that data generated by any particular component is distributed as:

$$x_i \mid z_i = k \sim \mathcal{N}(\mu_k, \Sigma_k).$$

We define mixture weights, or the probability of a particular point belonging to a particular component as:

$$P(z_i = k) = \pi_k \quad \text{for} \quad k = 1, \ldots, K.$$

The likelihood of x_i is then:

$$p(x_i \mid \boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x_i \mid \mu_k, \Sigma_k).$$

We define the following prior distributions:

$$(\pi_1, ..., \pi_K) \sim Dirichlet(a_0, ..., a_0)$$

$$(\mu_{jk}, \sigma^2_{jk}) \sim \mathcal{N}(\mu_{jk} \mid \mu_0, \kappa_0\sigma^2_{jk}) Inv - Gamma(\sigma^2_{jk} \mid \alpha_0, \beta_0).$$

The priors for π, μ_{jk}, and σ_{jk} are mathematically convenient as they are conjugate priors. We will find that the posterior of each parameter of interest has the same form as the corresponding prior. Note that we are treating each dimensional component of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ individually. Hyper-parameters will be chosen such that our priors are uninformative.

Although not always the case, it is possible to precisely derive the full posterior distribution of each parameter:

$$P(z_i = k \mid others) = \frac{\pi_k \mathcal{N}(x_i \mid \mu_k, \sigma^2_k)}{\sum_{h=1}^{K} \pi_m \mathcal{N}(x_i \mid \mu_m, \sigma^2_m)} \tag{5.1}$$

$$\pi \mid others \sim Dirichlet(a_0 + N_1, ..., a_0 + N_K) \tag{5.2}$$

$$\mu_{jk} \mid others \sim \mathcal{N}(\mu_{jk} \mid \hat{\mu}_{jk}, \hat{\kappa}_k\sigma^2_{jk}) \tag{5.3}$$

$$\sigma^2_{jk} \mid others \sim Inv - Gamma(\sigma^2_{jk} \mid \hat{\alpha}_k, \hat{\beta}_{jk}) \tag{5.4}$$

Where,

$$\begin{aligned}
\hat{\kappa}_k &= (\kappa_0^{-1} + N_k)^{-1} \\
\hat{\mu}_{jk} &= \hat{\kappa}_k(\kappa_0^{-1}\mu_0 + N_k\bar{x}_{jk}) \\
\bar{x}_k &= N_k^{-1} \sum_{i:z_i=k} x_i \\
\hat{\alpha_k} &= \alpha_0 + N_k \\
\hat{\beta}_{jk} &= \beta_0 + \frac{1}{2}\left[\sum_{i:z_i=k} (x_{ij} - \bar{x}_{jk})^2 + \frac{N_k}{1 + \kappa_0 N_k}(\bar{x}_{jk} - \mu_0)^2\right].
\end{aligned}$$

The implementation of the Gibbs sampler is, in principle, quite simple. After initializing values for $\boldsymbol{\mu}, \boldsymbol{\Sigma}, \pi$, and Z, one must sample each parameter using equations (5.1) through (5.4) until the stationary distribution has been reached and an adequate number of samples have been gathered.

5.2 Hybrid Algorithm

The statistical model which underpins our algorithm is the Gaussian mixture model, which we will treat in a Bayesian setting. We will employ much of the same notation used in our discussion of the Gibbs sampler.

Our first step is to determine an appropriate value for K. We do so by running the EM algorithm on our data for increasing values of K until BIC is minimized as described in Section 4. We must also pick initial values for $\boldsymbol{\theta}$ and Z. Each clustering assignment in Z is initialized by randomly drawing an integer from the set $\{1, 2, \ldots, K\}$, where each integer is selected with uniform probability. $\boldsymbol{\mu}$ is initialized by calculating the mean of the points assigned to each mixture component k in the previous step. Each dimension of every mixture component in $\boldsymbol{\Sigma}$ is initialized with a variance of 1 and covariance 0. π is likewise initialized uniformly; each mixture weight is initially set to $\frac{1}{K}$, which ensures that $\sum_{k=1}^{K} \pi_k = 1$. Other initialization schemes would work just as well; this is merely our chosen implementation.

EM Phase

We begin by running the EM algorithm on our GMM until convergence, exactly as described in Chapter 3. Doing so will result an estimation of $\boldsymbol{\theta}_{\text{EM}}$ that maximizes log-likelihood $\mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{x})$, as well as soft clustering assignments $\boldsymbol{\Omega}_{\text{EM}}$. However, the question remains as to whether $\boldsymbol{\theta}_{\text{EM}}$ maximizes $\mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{x})$ globally or merely locally. Define $\mathcal{L}_{\text{EM}}$ as the value of the log-likelihood function evaluated at the termination of the EM algorithm

$$\mathcal{L}_{\text{EM}} = \mathcal{L}(\boldsymbol{\theta}_{\text{EM}} \mid \boldsymbol{x}).$$

Gibbs phase

We subsequently utilize a Gibbs sampler to explore the loss-landscape for superior solutions. Our reasoning is that if the Gibbs sampler can identify a solution with a higher likelihood than $\mathcal{L}_{\text{EM}}$ within a predetermined number of iterations R, we can infer that the EM algorithm did not converge to a global optimum. Conversely, if no better solution is found after R iterations, we accept $\boldsymbol{\theta}_{\text{EM}}$ as optimal and the algorithm terminates, our confidence in our final solution being related to the magnitude of our chosen R.

Let $\boldsymbol{\theta}_{\text{Gibbs}}$ denote the parameter set discovered by the Gibbs sampler at any iteration. Furthermore, allow $\mathcal{L}_{\text{Gibbs}}$ to be the log-likelihood of our data evaluated at $\boldsymbol{\theta}_{\text{Gibbs}}$. Stated mathematically, our Gibbs sampler will search for $\boldsymbol{\theta}_{\text{Gibbs}}$ such that

$$\mathcal{L}_{\text{Gibbs}} = \mathcal{L}(\boldsymbol{\theta}_{\text{Gibbs}} \mid \boldsymbol{x}) > \mathcal{L}_{\text{EM}}. \tag{5.5}$$

Once a solution which satisfies (5.5) has been found, we employ the EM algorithm again to to conduct a local search in the vicinity of $\boldsymbol{\theta}_{\text{Gibbs}}$. We initialize the EM algorithm with $\boldsymbol{\theta}_{\text{Gibbs}}$ and permit it to converge once more.

Conclusion of the Algorithm

The algorithm continues switching between the EM algorithm and the Gibbs sampler until the Gibbs sampler fails to find a better solution after R iterations, at which point the algorithm terminates and $\boldsymbol{\theta}_{EM}, \boldsymbol{\Omega}_{EM}$ are taken as our final parameters and clustering assignments, respectively. If hard clustering assignments are desired, one can simply take the column index of the maximum ω_{ik} in each row of $\boldsymbol{\Omega}_{\text{EM}}$, which correspond to the cluster which the i^{th} point is most likely to belong to.

We adopt the following notation to write the algorithm formally:

- $\boldsymbol{\theta}_{init}$: Parameters $\boldsymbol{\theta}$ after random initialization.
- Z_{init} : Clustering assignments Z after random initialization.

- EM($\boldsymbol{\theta}, Z$) : The EM algorithm represented as a function, which takes the current $\boldsymbol{\theta}, Z$ and outputs updates $\boldsymbol{\theta}_{EM}$ and $\boldsymbol{\Omega}_{\text{EM}}$.

- $\boldsymbol{\Theta}$: A variable which contains the current best estimate of $\boldsymbol{\theta}$.

Our proposed hybrid algorithm may then be written as follows:

Algorithm 2 Hybrid Algorithm

$k \leftarrow 1$
while K yielding minimum BIC is not found **do**
  Perform EM algorithm with K = k
  Calculate BIC_k
  if BIC_k is minimum and $k > 2$ **then**
    Break
  else
    $k \leftarrow k + 1$
  end if
end while
Initialize $\boldsymbol{\theta}$ and Z
$\boldsymbol{\Theta}, \boldsymbol{\Omega}_{EM} \leftarrow EM(\boldsymbol{\theta}_{init}, Z_{init})$
Calculate $\mathcal{L}_{EM} \leftarrow \mathcal{L}(\boldsymbol{\theta}_{EM}, \boldsymbol{x})$
while $r < R$ **do**
  Sample $\boldsymbol{\theta}_{\text{Gibbs}}$ and Z_{Gibbs}
  Calculate $\mathcal{L}_{Gibbs} \leftarrow \mathcal{L}(\boldsymbol{\theta}_{Gibbs}, \boldsymbol{x})$
  if $\mathcal{L}_{Gibbs} > \mathcal{L}_{EM}$ **then**
    $r \leftarrow 0$
    $\boldsymbol{\Theta}, Z \leftarrow EM(\boldsymbol{\theta}_{Gibbs}, Z_{Gibbs})$
  else
    $r \leftarrow r + 1$
  end if
end while
return $\boldsymbol{\Theta}, \boldsymbol{\Omega}_{\text{EM}}$

Chapter 6

Real Data Analysis

In the interest of demonstrating the practical applicability of our approach, we compare the performance of our proposed algorithm to the performance of the traditional EM algorithm on the "wine" data set[14], which may be found on the UCI machine learning repository. The wine data set was chosen due to its popularity among researchers, multi-modal nature, and relative brevity. We apply both the EM algorithm and our hybrid algorithm to our data thirty times, recording the maximum likelihood obtained by each algorithm. Our findings indicate that our hybrid algorithm effectively escapes the local minimum and ultimately yields a higher likelihood solution when the EM algorithm alone yields a sub-optimal solution.

6.1 The Data

The wine data set, as described in the UCI machine learning repository, is a record of the characteristics of a sample of Italian wines, with three different varieties of wines represented. Each of the 173 rows consists of thirteen features: Alcohol, Malic acid, Ash, Alcalinity of ash , Magnesium, Total phenols, Flavanoids, Nonflavanoid phenols, Proanthocyanins, Color intensity, Hue, OD280/OD315 of diluted wines, and Proline. The data also contains class labels indicating wine type. Although clustering is an unsupervised task and we will not be using these labels directly, we will set K to be equal to 3 for the purposes of our study. The classes represented in the wine data set are mostly well separated, and previous work where clustering is performed on this data affirms our choice of K[15].

6.2 Methodology

In order to perform each repetition of our experiment, we initialize values for $\boldsymbol{\theta}$ and Z, as described in chapter 5. Each point is randomly assigned an initial cluster thus initializing Z, mean vectors are calculated for each initial cluster to serve as our initial centroids $\boldsymbol{\mu}$, all covariance matrices $\boldsymbol{\Sigma}$ are initialized with diagonal terms equaling 1 and off diagonal terms equalizing zero, and mixture weights π are initialized such that each of the K mixture weights are equal and the sum of these weights is one.

Setting K to be equal to three, the EM algorithm is then applied to the wine data $\boldsymbol{x}$ and allowed to run until convergence. We consider convergence to have occurred when each component of $\boldsymbol{\theta}$ changes less than .00001 after a single iteration, at which point $\mathcal{L}_{\text{EM}}$ is calculated and recorded.

We subsequently implement our hybrid algorithm on wine data $\boldsymbol{x}$. Given that implementing the EM algorithm is the first phase of the hybrid algorithm, we use the previously generated $\boldsymbol{\theta}_{\text{EM}}$ to begin the Gibbs phase directly, thus combining the first phase of our novel algorithm with the baseline application of the EM algorithm. Doing so offers, apart from convenience, some protection against the random effect of initialization; we can be certain that any improvement that our hybrid approach offer is genuine, not a matter of fortunate random initialization. We allow the Gibbs phase 3000 attempts to find a higher likelihood solution that the most recent $\mathcal{L}_{\text{EM}}$.

The aforementioned process is repeated thirty times, each trial resulting in a final log-likelihoods corresponding to the EM algorithm run in isolation and our hybrid approach, respectively.

6.3 Results

We found that our hybrid algorithm offered an improvement over the traditional EM algorithm in 21 out of 30 replications. The EM algorithm converged to the highest likelihood solution observed over all runs in the remaining 9 replications; consequently, the hybrid al-

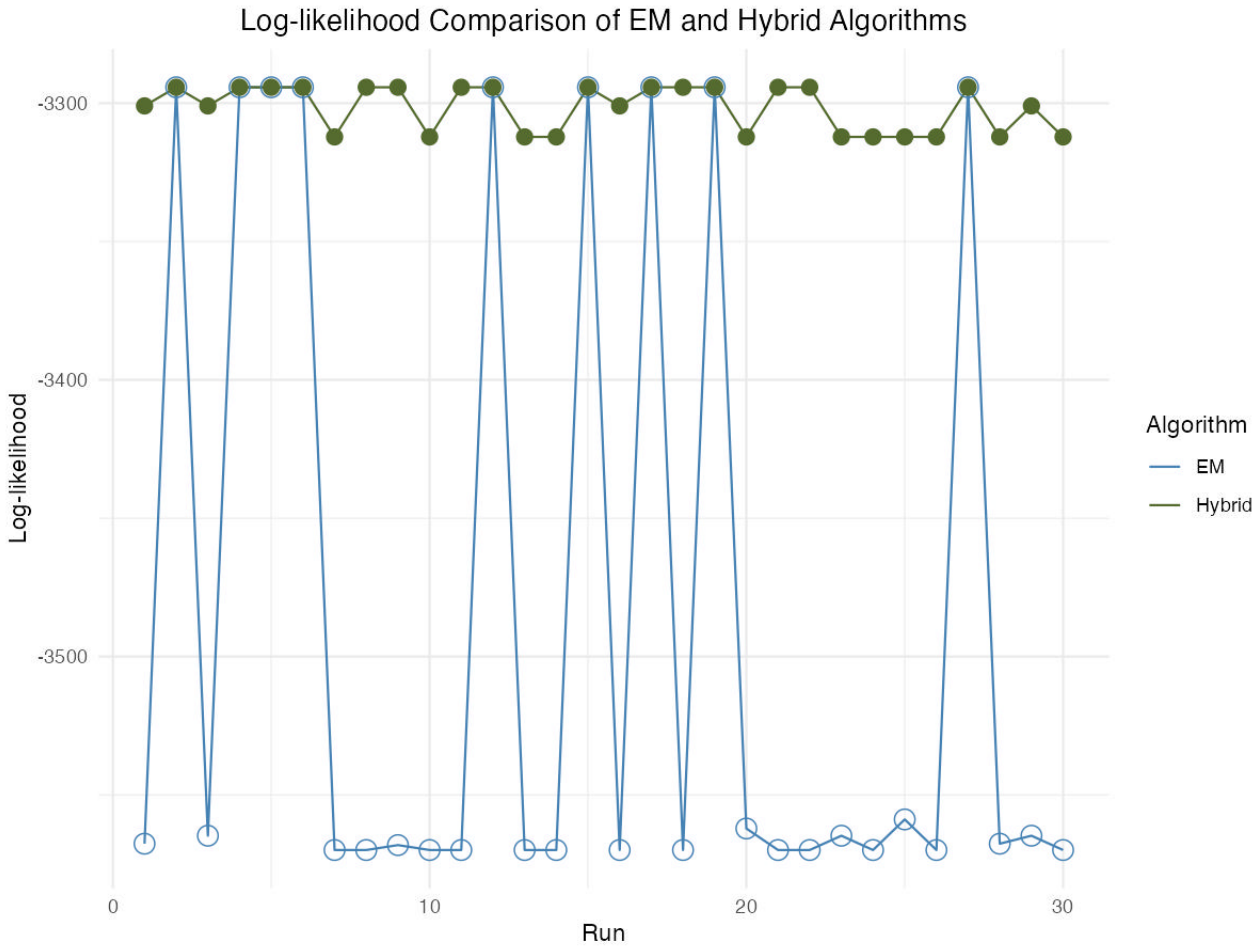

Figure 6.1: *The Hybrid algorithm (solid green) consistently improved upon the solution offered by the traditional EM algorithm (hollow blue) in terms of log-likelihood. Two distinct modes are clearly depicted, indicating that the wine data set contains at least one sub-optimal minimum to which the EM algorithm may converge.*

gorithm was unable to offer an improvement. Our results are displayed graphically in 6.1. Under the assumption that the highest likelihood solutions observed are indeed the global maximum, it is expected that the Hybrid approach would fail to perform better.

Excluding the instances where the EM algorithm achieved the apparent global optimum, there is a relatively large gap between the average performance of the Hybrid algorithm and the second most optimal solution generated by the EM algorithm alone, forming two distinct levels of optimally dominated by the Hybrid algorithm and EM algorithm respectively. Among those two levels much smaller variations are observed.

The Hybrid approach successfully enhanced our final solution on each occasion that the

	Hybrid_algo_obj_results	EM_obj_results	Difference
1	-3300.99	-3567.62	266.63
2	-3294.26	-3294.26	0.00
3	-3300.99	-3564.73	263.74
4	-3294.26	-3294.26	0.00
5	-3294.26	-3294.26	0.00
6	-3294.26	-3294.26	0.00
7	-3312.20	-3569.90	257.70
8	-3294.26	-3569.90	275.64
9	-3294.26	-3568.12	273.86
10	-3312.20	-3569.90	257.70
11	-3294.26	-3569.90	275.64
12	-3294.26	-3294.26	0.00
13	-3312.20	-3569.90	257.70
14	-3312.20	-3569.90	257.70
15	-3294.26	-3294.26	0.00
16	-3300.99	-3569.90	268.91
17	-3294.26	-3294.26	0.00
18	-3294.26	-3569.90	275.64
19	-3294.26	-3294.26	0.00
20	-3312.20	-3562.12	249.92
21	-3294.26	-3569.90	275.64
22	-3294.26	-3569.90	275.64
23	-3312.20	-3564.73	252.53
24	-3312.20	-3569.90	257.70
25	-3312.20	-3558.84	246.64
26	-3312.20	-3569.90	257.70
27	-3294.26	-3294.26	0.00
28	-3312.20	-3567.62	255.42
29	-3300.99	-3564.73	263.74
30	-3312.20	-3569.90	257.70

Table 6.1: $\mathcal{L}_{EM}, \mathcal{L}_{Hybrid}$ *and* $\mathcal{L}_{Hybrid} - \mathcal{L}_{EM}$ *for each replication of our study.*

EM algorithm converged to the lower of the two modes. Although our Hybrid approach did not achieve the maximum likelihood solution observed at each replication, it was notably more likely to do so than the EM algorithm alone. Increasing the number of samples afforded in the Gibbs phase should theoretically strengthen our assurance of global optimization at the cost of computational time.

Chapter 7

Conclusion

By combining the efficient local optimization capabilities of the EM algorithm with the global stochastic search afforded by the Gibbs sampling algorithm, we were able to construct a novel clustering algorithm which is capable of offering more robust assurance of convergence to global optimality. This assurance was empirically demonstrated with a real data study by which the EM algorithm and our proposed algorithm were directly compared. In addition, we offer the implementation of our algorithm in R, which may be found in appendix, and a discussion of those clustering methods which underpin our own algorithm.

Our real data analysis, although sufficient for the purposes of demonstrating the fundamental advantages of our algorithm, is somewhat limited in scope. We would benefit from additional empirical analysis, executed with more repetitions on a greater variety of data, so as to clarify the strengths and limitations of our method.

Additional effort may be well spent comparing our algorithm to other clustering approaches. Particularly worthwhile might be a comparison between our hybrid approach and others which aim to assuage the local minimum problem in clustering, such as the K-means++ algorithm[16].

In summary, our approach offers a promising method by which global optimallity may be assured in clustering problems. Further study is required to fully clarify the behavior, limitations, and applications of our algorithm.

Bibliography

[1] A. S. Shirkhorshidi, S. Aghabozorgi, T. Y. Wah, and T. Herawan. Big data clustering: A review. In B. Murgante et al., editors, *Computational Science and Its Applications – ICCSA 2014*, volume 8583 of *Lecture Notes in Computer Science*, Cham, 2014. Springer. doi: 10.1007/978-3-319-09156-3_49.

[2] A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: A review. *AMC Computing Surveys*, 1999.

[3] J. MacQueen. Some methods for classification and analysis of multivariate observations. *Berkeley Symp. on Math. Statist. and Prob.*, 1967.

[4] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning with Applications in R.* Springer, 2023.

[5] M. Mahajan, P. Nimbhorkar, and K. Varadarajan. The planar k-means problem is np-hard. *Theoretical Computer Science*, 2012.

[6] T. Kanungo, D.M. Mount, N.S. Netanyahu, R. Silverman C.D. Piatko, and A.Y. Wu. An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 1977.

[8] C.M Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[9] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer, 2008.

[10] S. Odashima, M. Ueki, and N. Sawasaki. A split-merge dp-means algorithm to avoid local minima. In *Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2016. Lecture Notes in Computer Science*, volume 9852, Cham, 2016. Springer.

[11] Ting Su and J. Dy. A deterministic method for initializing k-means clustering. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 784–786, 2004. doi: 10.1109/ICTAI.2004.7.

[12] Shokri Z. Selim and M. A. Ismail. K-means-type algorithms: A generalized convergence theorem and characterization of local optimality. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(1):81–87, 1984. doi: 10.1109/TPAMI.1984.4767478.

[13] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian Data Analysis*. Chapman Hall/CRC, 2013.

[14] Stefan Aeberhard and M. Forina. Wine. UCI Machine Learning Repository, 1991. DOI: https://doi.org/10.24432/C5PC7J.

[15] Mark Junjie Li, Michael K. Ng, Yiu-ming Cheung, and Joshua Zhexue Huang. Agglomerative fuzzy k-means clustering algorithm with selection of number of clusters. *IEEE Transactions on Knowledge and Data Engineering*, 20(11):1519–1534, 2008. doi: 10.1109/TKDE.2008.88.

[16] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 1027–1035. Society for Industrial and Applied Mathematics, 2007. ISBN 9780898716245.

Appendix A

Implementation of Hybrid algorithm

```
df <- read.table(file='<path_to_data>', sep=",", header=FALSE)

X <- as.matrix(df) #convert data into numeric argument

# Set K, set d <- # of dimensions and N <- # of points
K <- 3
d <- dim(X)[2]
N <- dim(X)[1]

#####################################

# Set Hyperparameters
kappa_0 <- 1000
alpha_0 <- 1
Beta_0 <- 1
mu_0 <- 0

# log-likelihood of MVN
log_dMVN <- function(point, mean, variance){
  d <- length(mean)
  density <- 0
```

```
  for (i in 1:d){
    density <- density + dnorm(point[i], mean[i], sqrt(variance[i]), log<-TRUE)
  }
  density
}

# Function for log-sum-exp trick to avoid underflow
# when normalizing probabilities.
log_sum_exp <- function(x){
  c <- max(x)
  c + log(sum(exp(x - c)))
}

########Objective Function#######
Obj <- function(data, mu, Sigma, pi.mix){
  sum(apply(data, 1, point_likelihood, mu<-mu, Sigma<-Sigma, pi.mix<-pi.mix))
}

# Likelihood of a single point
point_likelihood <- function(point, mu, Sigma, pi.mix){
  likelihood <- 0
  for (k in 1:K){
    likelihood <- likelihood + pi.mix[k]*dMVN(point, mu[k,], Sigma[k,])
  }
  log(likelihood)
}

# P(z_i = k | x_i)
log_z_cond <- function(k, point, mu, Sigma, pi.mix){
```

```
  log_numerator <- log(pi.mix[k]) + log_dMVN(point, mu[k,], Sigma[k,])
  log_denom_terms <- numeric(K)
  for (i in 1:K){
    log_denom_terms[i] <- log(pi.mix[i]) + log_dMVN(point, mu[i,], Sigma[i,])
  }
  log_denom <- log_sum_exp(log_denom_terms)
  log_out <- log_numerator - log_denom
  out <- exp(log_out)
}

# For sampling pi in Gibbs sampler
sample_pi <- function(z){
  alpha_k.vec <- numeric(K)
  for (i in 1:K){
    alpha_k.vec[i] <- alpha_0 + sum(z <- i)
  }
  pi_samples <- rdirichlet(1, alpha_k.vec)
  pi_samples
}

# For sampling Z in Gibbs sampler
sample_z <- function(data, mu, Sigma, pi.mix){
  samples <- numeric(length(data[,1]))
  mu <- mu[order(mu[,1], decreasing=F),]
  for (i in 1:length(data[,1])){
    probs <- numeric(K)
    for (k in 1:K){
      probs[k] <- log_z_cond(k, data[i,], mu, Sigma, pi.mix)
    }
```

```
    samples[i] <- sample(1:K, 1, replace=FALSE, probs)
  }
  samples
}

# For sampling mu in Gibbs sampler
sample_means <- function(X, Sigma, z){
  samples <- matrix(0, K, d)
  for (k in 1:K){
    indexes <- which(z == k, arr.ind=TRUE)
    n_k <- sum(z == k)
    if (n_k>1){
      data_k <- X[indexes,]
      sample_mean <- colSums(data_k)/(dim(data_k)[1])
      kappa.hat <- 1/(kappa_0^(-1) + n_k)
      mu.hat <- kappa.hat*((kappa_0^(-1))*mu_0 + n_k*sample_mean)
      Sigma <- diag(kappa.hat*Sigma[k,], d, d)
      samples[k,] <- mvrnorm(n=1, mu=mu.hat, Sigma)
    }
    if (n_k==1){
      data_k <- X[indexes,]
      sample_mean <- data_k
      kappa.hat <- 1/(kappa_0^(-1) + n_k)
      mu.hat <- kappa.hat*((kappa_0^(-1))*mu_0 + n_k*sample_mean)
      Sigma <- diag(kappa.hat*Sigma[k,], d, d)
      samples[k,] <- mvrnorm(n=1, mu=mu.hat, Sigma)
    }
    if (n_k==0){
      sample_mean <- rep(mu_0, d)
```

```
      kappa.hat <- 1/(kappa_0^(-1) + n_k)
      mu.hat <- kappa.hat*((kappa_0^(-1))*mu_0 + n_k*sample_mean)
      Sigma <- diag(kappa.hat*Sigma[k,], d, d)
      samples[k,] <- mvrnorm(n=1, mu=mu.hat, Sigma)
    }
  }
  samples
}

# For sampling Sigma in Gibbs sampler
sample_variance <- function(X, z){
  sample_var <- matrix(0, k, d)
  for (k in 1:K){
    indexes <- which(z == k, arr.ind<-TRUE)
    n_k <- sum(z == k)
    if (n_k>1){
      data_k <- X[indexes,]
      sample_mean <- colSums(data_k)/n_k
      alpha.hat <- rep(alpha_0 + n_k/2, d)
      Beta.term1 <- colSums((sweep(data_k,2,sample_mean))^2)
      Beta.term2 <- (n_k/(1 + kappa_0*n_k))*((sample_mean - mu_0)^2)
      Beta.hat <- Beta_0 + .5*(Beta.term1 + Beta.term2)
      for (dim in 1:d){
        sample_var[k,dim] <- 1/rgamma(1,shape=alpha.hat[dim], rate=Beta.hat[dim])
      }
    }else{sample_var[k,] <- 1/rgamma(d,shape=alpha_0, rate=Beta_0)}
  }
  sample_var
}
```

```
##############################
############EM################
##############################


EM <- function(mu, Sigma, alpha, z){#Randomly generate means by choosing random K groups
  Obj.before <- Obj(data=X, mu, Sigma, alpha)
  mu.old <- means
  Sigma.old <- Sigma
  alpha.old <- alpha
  z.old <- z
  stop.all <- FALSE
  #equal weights at the beginning
  ##############################
  mu0 <- mu #for convergence check
  alpha0 <- alpha #for convergence check
  Sigma0 <- Sigma #for convergence check
  R<-10000 #maximum EM iteration number
  for(r in 1:R){
    ############################
    #E-Step ###################
    ############################
    num.w <- matrix(0,N,K)
```

```
log_num.w <- matrix(0,N,K)
for(i in 1:N){
  for(k in 1:K){
    log_num.w[i,k] <- sum(dnorm(X[i,], mu[k,], sqrt(Sigma[k,]), log=TRUE)) + log(alp
    # num.w[i,k]<-exp(sum(dnorm(X[i,],mu[k,],sqrt(Sigma[k,]),log<-TRUE))+log(alpha[k
  }}
log_denom <- apply(log_num.w, 1, log_sum_exp)
w <- exp(log_num.w - matrix(rep(log_denom, each=K), N, K, byrow=TRUE))

############################
#M-Step ###################
############################

#1. alpha update
N.k <- colSums(w, na.rm=TRUE)
alpha <- N.k/N

#2. mu & sigma update
for(k in 1:K){
  w.ik <- matrix(w[,k],N,d)
  mu[k,] <- colSums(w.ik*X)/N.k[k]
  mu.k <- matrix(mu[k,], N, d, byrow=TRUE)
  Sigma[k,] <- colSums(w.ik*(X-mu.k)^2)/N.k[k]
  Sigma[k,which(Sigma[k,]==0)] <- 0.1^5
}
if (max(abs(mu0-mu))<0.1^5&max(abs(alpha0-alpha))<0.1^5&max(abs(Sigma0-Sigma))<0.1^5
  break
}
mu0 <- mu
```

```
    Sigma0 <- Sigma
    alpha0 <- alpha
    if(r==R){print("Warning: Algorithm does not converge")}
  }
  Obj.after <- Obj(data=X, mu, Sigma, alpha)
  results <- list("means"=mu, "variances"=Sigma,
                  "pi.mix"=alpha, "z"=apply(w,1,which.max),
                  "mu.better"=mu.old, "Sigma.better"=Sigma.old,
                  "pi.better"=alpha.old, "stop"=stop.all,
                  "Obj"=Obj.after)
}


########################################
###############Gibbs####################
########################################
library(MCMCprecision)
library(MASS)


Gibbs <- function(G.iter, start_obj, mu.init, sig.init, pi.init, z.init, Gibbs_improveme
  stop.all <- TRUE
  z_samples <- matrix(0, N, G.iter)
  pi_samples <- matrix(0, K, G.iter)
  mean_samples <- array(rep(0,K*d*G.iter), c(K,d,G.iter))
  variance_samples <- array(rep(0,K*d*G.iter), c(K,d,G.iter))
  mean_samples[,,1] <- mu.init
  variance_samples[,,1] <- sig.init
  z_samples[,1] <- z.init
```

```
pi_samples[,1] <- pi.init
fin.iter <- 1
for (g in 2:G.iter){
  if (Gibbs_improvement==TRUE){print("Gibbs has improved our solution")}
  print(g)
  z_samples[,g] <- sample_z(X, mean_samples[,,g-1],
                            variance_samples[,,g-1], pi_samples[,g-1])
  pi_samples[,g] <- sample_pi(z_samples[,g])
  mean_samples[,,g] <- sample_means(X, variance_samples[,,g-1],
                                    z_samples[,g])
  variance_samples[,,g] <- sample_variance(X, z_samples[,g])
  Obj.gibbs <- Obj(data=X, mu=mean_samples[,,g],
                   Sigma=variance_samples[,,g],
                   pi.mix=pi_samples[,g])
  last_index <- g
  print('########################')
  if (Obj.gibbs > start_obj){
    Gibbs_improvement <- TRUE
    stop.all <- FALSE
    break
  }
}
results <- list("z_samples"=z_samples[,last_index],
                "pi_samples"=pi_samples[,last_index],
                "mean_samples"=mean_samples[,,last_index],
                "variance_samples"=variance_samples[,,last_index],
                "iter.num"=fin.iter - 1,
                "Obj"=Obj.gibbs,
                "stop"=stop.all,
```

```
                "Gibbs_improvement"=Gibbs_improvement)
}

######Full implementation#######



##############################
#Initial value specification
##############################

z <- sample(1:K,N,replace<-TRUE)
mu <- matrix(0,K,d)
Sigma <- matrix(1,K,d) #set initial values of all sigma to be one
for(k in 1:K){
  mu[k,] <- apply(X[z==k,],2,mean)}
pi.mix <- rep(1/K,K) #equal weights at the beginning

EM_results <- EM(mu, Sigma, pi.mix, z)
EM_obj_results[i] <- EM_results$Obj
Gibbs_improvement <- FALSE

while(TRUE){
  EM_means <- EM_results$means
  EM_variances <- EM_results$variances
  EM_pi.mix <- EM_results$pi.mix
  EM_z <- EM_results$z
  EM_obj <- Obj(X, mu=EM_means, Sigma=EM_variances,
                pi.mix=EM_pi.mix)
  print("EM obj is")
```

```
  print(EM_obj)
  new.attempt <- Gibbs(G.iter<-1000, start_obj<-EM_obj, mu.init<-EM_means,
                       sig.init<-EM_variances, pi.init<-EM_pi.mix,
                       z.init <- EM_z, Gibbs_improvement<-Gibbs_improvement)
  Gibbs_improvement <- new.attempt$Gibbs_improvement

  if (new.attempt$stop==TRUE){
    # This sets final parameters when we get through
    # Gibbs samples without finding a better
    # optimum, signaling the end of the algorithm.
    final.means <- EM_means
    final.variances <- EM_variances
    final.pi.mix <- EM_pi.mix
    final.z <- EM_z
    final.obj <- EM_obj
    Hybrid_algo_obj_results[i] <- final.obj
    print(final.means)
    print(final.variances)
    print(final.pi.mix)
    print(final.z)
    break
  }
  else{EM_results <- EM(mu=new.attempt$mean_samples,
                        Sigma=new.attempt$variance_samples,
                        alpha=new.attempt$pi_samples,
                        z=new.attempt$z_samples)
      }
}
```