EVALUATION OF DIFFERENT CLUSTERING AND CLASSIFICATION ALGORITHMS FOR CONTIUNOUS AND DISCRETE DATA SETS

A Major Qualifying Project

Submitted to the Faculty

Of the

WORCESTER POLYTECHNICAL INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Jacob Peskoe

Abstract

The objective of this project was to investigate and compare several existing methods for clustering and classifying data as well as some of my own design. Under the employment of BAE Systems, Inc., I tested algorithms designed to identify clusters and determine parameter estimates from continuous data that were known to be in the form of Gaussian mixtures. I also looked into methods for classifying discrete data that were known to originate from multiple unique sources, in hopes of being able to categorize these data automatically based on previous knowledge. Through my findings, I was able to determine which algorithm performed the best. I also found multiple viable methods for automatically categorizing the discrete data that BAE Systems was interested in.

Table of Contents

1. Background

There has always been an interest and need to analyze data, by searching for patterns, trends, and any other information available. The advent of high-throughput computers with high-speed access to terabytes of data has enabled the mining of data. Advancements in artificial intelligence has further led to the creation of algorithms that can "learn" from one data set and apply that "knowledge" to additional data sets.

During the summer, I was employed by BAE Systems, Inc., who was interested in understanding a variety of data sets. Some sets consisted of continuous data, generated from underlying Gaussian mixture distributions. Other data derived from discrete distributions of unknown form. My employer was interested in techniques to identify the number of different clusters for the continuous data and categories for the discrete data. I was asked to investigate what techniques were available that would allow them to identify the characteristics of clusters or categories. My supervisor identified a few techniques to research for analyzing continuous data and a couple for discrete. I researched these methods and further developed some new techniques, one a hybrid of others and some original methods.

In my research, I found material on many of the existing techniques that described their derivation and implementation. However, I was unable to find comparisons of the methods that would highlight their strengths and weaknesses and thereby assist me in determining which algorithms were most appropriate for BAE's needs. This gave rise to a project that would examine and compare algorithms and ultimately guide other analysts (at BAE in particular) in selecting the most appropriate methods for their applications.

2. Problem Statement

For this project, I generated "synthetic" data sets that were designed to resemble the real data, which were proprietary. I then applied various classification algorithms to these data. I investigated the performance of each algorithm against the data sets and evaluated the strengths and weaknesses of each. Furthermore, I considered methods of combining algorithms to determine if results can be

improved. Finally, I provided some general recommendations to guide a data analyst in the overall use of these algorithms.

3. Statistical Concepts

There are a number of statistical concepts that are used throughout this project. Some reflect aspects of the data being examined and others refer to techniques for understanding the data. A brief description of these concepts is provided here.

When examining a set of data, it is not always known whether the data derive from a single distribution or multiple distributions. In the case where there is more than one underlying distribution, the data follow what is known as a mixture distribution. A model for data representing a mixture distribution is referred to as a mixture model.

In instances where there exists a set of data for which the categorization or classification is already known, that data can be used to "train" a computer algorithm. The data are referred to as "training data." Then, when a new data point is introduced, that computer algorithm can apply what it has "learned" from the training data to categorize or classify the new data point.

There are a number of approaches to inferring parameters when modeling unknown distributions. Among them are Maximum Likelihood Estimation (MLE), Expectation Maximization (EM), and Bayesian Inference.

MLE is a method of estimating model parameters from observed data. For a set of empirical data, the likelihood function is the probability that the underlying distribution, described by a set of parameters, produced that set of data. The MLE is the parameter (or set of parameters) that maximizes this likelihood function.

EM is an iterative technique for finding the MLE (Bilmes, 1998). It is typically applied to an observed population of data when there is believed to be any latent or unobserved variables. For this project, the latent variables describe from which distribution the data derive. The technique searches for the MLE, examining the log of the likelihood function (called the log-likelihood) rather than the likelihood function directly. (The log-likelihood function is easier to manage computationally and it is maximized at the

same point that the likelihood function is maximized.) It alternates between optimizing the parameters of the distributions and optimizing the assignments of the data to distributions. The iterations continue until the improvement in parameter estimates falls below a given threshold.

Bayesian Inference is an approach that is based on Bayes' Theorem (Birnbaum, 1962). It seeks to maximize the posterior probability of the model parameters rather than simply the likelihood function. The major feature of Bayesian Inference that separates it from MLE and EM is that it considers prior knowledge about the parameters. This prior knowledge is referred to as the prior distribution.

4. The Data

Due to the proprietary nature of the real data collected by BAE systems, I have generated representative data, or synthetic data, for use in this project. All analyses and results presented in this project reflect the synthetic data only. For additional information on how the synthetic data were generated, see Appendix A – Generating Synthetic Data.

The types of continuous data that BAE Systems works with are often multi-dimensional and therefore not be easily viewed in a two-dimensional space. The data may derive from different populations, the number of which is unknown. It is also unknown from which population any particular observation derives. The types of data BAE Systems frequently examine involve Gaussian noise; therefore I am specifically concerned with modeling Gaussian mixture distributions.

I received real discrete data, which had been categorized by experts. BAE Systems was interested in developing methods that could automatically categorize new data, based on the characteristics of the data already assigned to the categories. This concept of using previously categorized data to assist in categorizing new data is referred to as "supervised learning." The already categorized data is referred to as "training data."

The discrete data can be thought of as an incomplete network of nodes and connections. A combination of nodes and connections represents an observation or sub-network. Nodes are of different known types. All that is known about a sub-network is the number and types of nodes. Each sub-network must be classified into its proper category.

5. Methodology

To evaluate the performance of the algorithms addressed in this paper, I programmed them into MATLAB and tested each of them on synthetic data. The algorithms for discrete data were also tested against the real data. All of the real data that was used was collected by my project team at BAE Systems and all of the tools I used (computer and software) were provided by the company. The exact nature of the real data is confidential, as well as the data itself, so all of the data and experimental results shared in this paper are from synthetic datasets.

For the continuous data, we were not examining a particular set of real data, but wanted to test clustering algorithms for a variety of Gaussian mixtures. For this reason, I simply generated multiple datasets varying in size, number of clusters, cluster means, and cluster covariances. However, we did have a set of real discrete data. In order to generate the discrete synthetic datasets, I needed to draw random samples from generative models that fit the real data. Since part of the purpose of this project is to compare the different methods of learning these models, I had to create multiple synthetic datasets using a variety of methods in order to have a fair comparison.

For the discrete datasets, I used leave-one-out cross-validation as a technique to measure how well each algorithm performed. This technique involves selecting one data point (sub-network) to "leave out", using the remainder of the population to "train" the algorithm, and then using the left-out data point as a test subject. I was able to examine how many sub-networks each algorithm correctly classified because the underlying categories were known. This process was repeated until every sub-network was left out once. For each algorithm, I noted how many times the correct category was identified as the "most likely" as well as how many times the correct category was among the top three. I also measured speed in terms of the amount of time it took each algorithm to run.

6. Algorithms for Examining Continuous Data

For the continuous data, the objective was to identify clusters in the data and infer information about the underlying distributions that created the clusters. BAE Systems was most interested in modeling Gaussian mixtures, so I limited my investigation to these types of distributions. There were five approaches I examined, with one having two variations, resulting in a total of six different

algorithms. Descriptions of the five approaches are provided below. Detailed implementations of some of the algorithms are provided in Appendix B – Implementation of Algorithms.

1) *K*-Means Clustering

The most straight-forward approach to identifying clusters was the *k*-means clustering algorithm. The basic idea of *k*-means clustering is to group data points into *k* separate clusters (where *k* is an input parameter), such that we minimize the sum of the distances between each point and the mean of its respective cluster (Hartigan & Wong, 1979). To do this, the algorithm iterates between 2 main steps: calculate the mean for each cluster and then reassign the data points to clusters using the newly calculated means. This iteration is continued until no new assignments of data points occur.

2) EM for Gaussian Mixtures

Like the *K*-means clustering technique, EM for Gaussian Mixtures requires the number of clusters (*k*) to be specified. The implementation I used (MATLAB's built-in EM function) initializes the cluster means by selecting *k* data points at random. The covariance matrices (one for each cluster) are each initialized as the same diagonal matrix, where the j^{th} element on the diagonal is equal to the variance of the j^{th} dimension of the whole data set.

Probability distributions for cluster assignments are calculated for each data point, based on the initial parameter estimates (means and covariance matrices). Using these probability distributions, EM then finds the expected log-likelihood function and calculates new parameter estimates that maximize that log-likelihood function. With the new parameter estimates, new probability distributions for cluster assignments are calculated and therefore a new expected log-likelihood function is derived. New parameter estimates are once again obtained by maximizing the new expected log-likelihood function. This process is repeated until the improvement in the parameter estimates falls below a desired threshold.

3) Kurihara's Implementation of Variational Bayesian Inference for Gaussian Mixture Models

In my investigations of clustering algorithms, I found a collection of open-source algorithms by Kenichi Kurihara. I selected one that uses Variational Bayesian Inference for Gaussian Mixture Models and another that uses approximations to a Dirichlet Process (DP) (explained in the next subsection).

Bayesian Inference involves maximizing the posterior probability distribution, which can often be intractable and therefore difficult to calculate. Variational Bayesian Inference methods approximate the posterior probability distribution by factorizing the joint probability distribution into two independent distributions (Beal, 2003). Although it is known that the distributions are not in fact independent, this factorization is easier to handle computationally. Furthermore, the means and variances of the actual and approximated distributions should be fairly close.

Like the *K*-Means and EM approaches, this technique required knowing the number of clusters. While I have the open source code for the algorithm, I do not have proper documentation of the techniques used. I did not have the resources necessary to reverse engineer the code, so I cannot speak further to the specific approaches used.

4) Dirichlet Process Gaussian Mixture Models (DPGMM)

For the continuous data in this project, I was concerned with only Gaussian distributions. My goal is to cluster data points so that all data points generating from the same Gaussian distribution are assigned to the same cluster. A Dirichlet process (DP) will generate a series of distributions.

The "Polya Urn Scheme" can be viewed as a simplified realization of the DP that aids in understanding. The process begins with an urn containing a number of black balls. A ball is drawn from the urn at random. If a black ball is selected (which it will be for the first iteration), then a ball of a different color is chosen at random from a distribution of colored balls. Both balls are then put back into the urn. If a non-black ball is selected, then another ball of the same color is put back in the urn along with the selected ball. This process is repeated over and over again. Over time the chances of drawing a black ball become infinitely small and approach zero.

In the context of this project, each unique ball color represents a unique Gaussian distribution (and associated parameters). Therefore, over many iterations, the mixture of colors in the urn represents the mixture of the distributions. The distribution of colored balls outside of the urn represents the prior distribution (see section 2 on Bayesian Inference). The population of observations represents draws from the urn.

The DPGMM algorithms I considered were all iterative. Observations are considered one at a time and each is assigned to an existing cluster (distribution) or a new cluster. The assignment follows a distribution, which is dependent upon the parameter estimates of the existing clusters as well as the number of previously assigned observations to each cluster. If a new cluster is assigned, the parameter estimates for the new cluster are drawn from the prior distribution. Once all observations are assigned to clusters, new estimates of the cluster parameters are created by sampling values from the posterior distributions (each cluster having its own posterior distribution). If desired, the observations can be considered again and reassigned to an existing cluster or a new cluster. As stated earlier, the assignment follows a distribution that is dependent upon the (newly developed) parameter estimates for the existing clusters as well as the number of previously assigned observations to each cluster. With the new assignments, new parameter estimates are created. This process can be repeated and over time, the resulting mixture model approaches the model with the maximum posterior probability, given both the observed data and the assumed prior.

The prior and posterior distributions referred to above are not easily sampled from directly. Therefore, a number of techniques for sampling from these distributions have been developed. Many of these involve what are known as Markov Chain Monte Carlo (MCMC) methods (Neal, 2000). One such method, known as Gibbs Sampling, I examined for one-dimensional Gaussian mixtures. I found it too difficult to extend to more dimensions. Since I was interested in multi-dimensional data, I did not pursue this method any further. Another MCMC method, known as Metropolis-Hastings, I considered for two-, three- and ten-dimensional Gaussian mixtures. The third technique, from Kurihara's collection, was the variational DPGMM. Where MCMC methods sample indirectly from the troublesome joint distributions, Kurihara's method factorizes the joint distributions and then samples from them (as was described earlier in Kurihara's Variational Bayesian Inference).

The DPGMM algorithm I investigated with Metropolis-Hastings technique is adapted from Neal's Algorithm 8 (Neal, 2000). It requires what is known as a concentration parameter. This parameter influences both the number of clusters that are identified and the proportion of observations assigned to each cluster. Using a larger value tends to result in more clusters. It can be thought of as representing the number of black balls that are in the urn in the Polya Urn Scheme, although the value of the parameter is not limited to integers. In fact, a common default value is 1, which does not favor either a larger or smaller number of clusters.

Another input required by the DPGMM with Metropolis-Hastings method is the number of sets of auxiliary parameters. These parameters are needed when assigning an observation to either an existing or a new cluster. In order to calculate the probabilities of each possible assignment, the likelihood of the models for each cluster must be determined. This is difficult to calculate for a new cluster, since the cluster parameters are not defined. So sets of auxiliary parameters are drawn from the prior distribution to serve as the potentially new cluster parameters. If the observation is not assigned to that new cluster, the auxiliary parameters are discarded. The more sets of auxiliary parameters drawn, the better the chances are of creating a new cluster with appropriate parameters. However, this drawing occurs once for every observation in a single iteration of the algorithm. I found that selecting more than three sets slowed the computational time considerably. This presents a trade-off between the number of sets of auxiliary parameters and the total number of iterations for the complete algorithm.

One thing to note about the MCMC sampling methods in general (not just when applied to DPGMM algorithms) is that they require their own input parameters. The methods sample again and again, in an iterative fashion. As such, they require what is known as a "burn-in" period as well as a "lag". Each iteration is dependent upon the results of the previous iteration. The more iterations that occur, the less dependent the next iteration is on the very first value. The burn-in period represents the number of iterations that occur before the first sample is taken. This reduces the influence of the initial value on the samples taken. Similar to the burn-in period, the lag serves to reduce the dependency between two consecutive samples. It equals the number of iterations that need to occur after one sample is taken and before the next sample is drawn.

5) EM with a Dirichlet Prior

After observing performance characteristics of the previous techniques, I wanted to combine the speed and accuracy of EM with the ability to infer the number of clusters that the Dirichlet process provides. The concept is simple. Step 1) Run EM several times, varying the number of clusters each time. Each run produces parameter estimates, the likelihood of the model, and the mixture proportions of each cluster within the model. Because the tendency of the likelihood function will be to increase as the number of clusters increases, there is a need to limit the number of clusters. Otherwise, each cluster may contain only a single data point. I chose to introduce the Dirichlet prior. Step 2) Calculate the posterior probability for each model by multiplying the likelihood by the Dirichlet prior. The Dirichlet prior is a distribution of the mixture proportions of each cluster. Step 3) Select the model with the greatest posterior probability. The details of this approach and its derivation are proprietary and therefore not included in this paper.

To run this hybrid algorithm, a user does not need to know the number of clusters but must specify a range of possible values for the clusters. Furthermore, the Dirichlet prior implementation requires that the user specify the concentration parameter, as described for the Metropolis-Hastings technique.

7. Analyses and Results for Continuous Data

Table 1 below summarizes the input parameters required by the various methods examined for the continuous data. As seen in the table, the first three methods require that the number of clusters be known or closely approximated. This attribute highly limits their applicability to many scenarios. Conversely, the DPGMM methods derive the number of clusters from the data. The EM with a Dirichlet Prior, while not deriving the number of clusters, requires the user to specify a range of possibilities to consider. Also, while the first three methods may be limited in their application, they require little more input information. The Kurihara's VDPGMM requires no input parameters, whereas the other DPGMM methods require several additional parameters. Due to these similarities and differences, I analyzed and compared the first three methods against each other and the last four methods against one another.

Algorithm	Input Requirements					
	# Clusters	Stopping Mechanism	Concentration Parameter	# Auxiliary Parameters	Burn-in Period	Lag
K-Means	Y	# Replications	N	N	N	N
EM for Gaussian Mixtures	Y	# Replications; Threshold on Likelihood Improvement	N	N	N	N
Kurihara's Implementation of Variational Bayesian Inference for Gaussian Mixture Models	Y	N	N	N	N	N
Dirichlet Process Gaussian Mixture Models (DPGMM)						
Metropolis-Hastings Sampling	N	# Iterations	Y	Y	Y	Y
Kurihara's VDPGMM	N	N	N	N	N	N
EM with a Dirichlet Prior	Range	# Replications; Threshold on Likelihood Improvement	Y	N	N	N

Table 1 -- Input Parameters for Continuous Data Algorithms

The MCMC DPGMM method (Metropolis-Hastings) requires the user to also define a prior distribution for drawing the auxiliary parameters. I used the uniform distribution as well as a Gaussian distribution for these priors.

The algorithms were tested on a number of different synthetic data sets. A subset of those data sets was selected to be described in this paper. The selected data sets, hereafter referred to as scenarios, effectively reveal the strengths and weaknesses of the algorithms.

As previously mentioned, the synthetic data were all derived from Gaussian mixture models. In addition to varying the parameters of the Gaussian distributions, I varied the number of dimensions, clusters, and observations within each data set. Table 2 below shows how these additional characteristics varied for each scenario. Except for #1 and #3, each scenario also included a combination of disparate clusters and overlapping clusters. Scenarios 1 and 3 included only disparate clusters.

Scenario	Dimensions	Clusters	Observations
1	2	2	100
2a	2	2	1,000
2b	2	2	1,000
3	2	8	200
4	2	8	1,000
5	2	8	100,000
6	3	4	1,000
7	3	4	10,000
8	4	8	400
9	10	4	200
10	10	4	1,000
11	10	8	1,000

Table 2 -- Scenario Characteristics

The algorithms were applied to the observations in each scenario and the resulting models were analyzed against the underlying models that generated the observations. Each algorithm was then evaluated with respect to consistency (when the method is applied repeatedly, the results are consistent), speed (computational speed[1]), and correctness (how close the model clusters are to the known clusters). The DPGMM models were also evaluated with respect to how well they determined the correct number of clusters.

K-Means Clustering

Consistency This algorithm selects the best result over the number of replications specified by the user. Due to the randomization of initial values, two separate runs, each with one replication, may produce different or inconsistent results. Two separate runs, each with two or three replications, may still produce inconsistent results. As the number of replications increase, however, the results of the separate runs are likely to become more consistent. Various factors (e.g., dimension and # of clusters) also affect consistency. However, by performing 50 replications for each run, the algorithm was able to overcome these shortcomings and produce results that were consistent.

Speed This algorithm completed 50 replications for most scenarios in a matter of a few seconds. However, scenario 5 with 100,000 observations took 112 seconds.

[1] Computational speed depends upon the computer running the algorithm. I used an HP Compaq 8510w laptop, 2.2 GHz, with 3 GB of RAM, with Intel® Core™2 Duo CPU

Correctness When the clusters were easily distinguished visually, this algorithm performed very well. However, the closer the clusters were to one another, the more difficulty it had determining the correct clusters.

Figure 1 shows an example of two disparate clusters from scenario 1, clearly distinguished by *k*-means clustering. The ellipses represent the Gaussian distributions that generated the observations, with the center of the ellipse located at the mean and the eccentricity of the ellipse defined by the covariance matrix. The different colors show the different clusters as defined by the clustering algorithm.

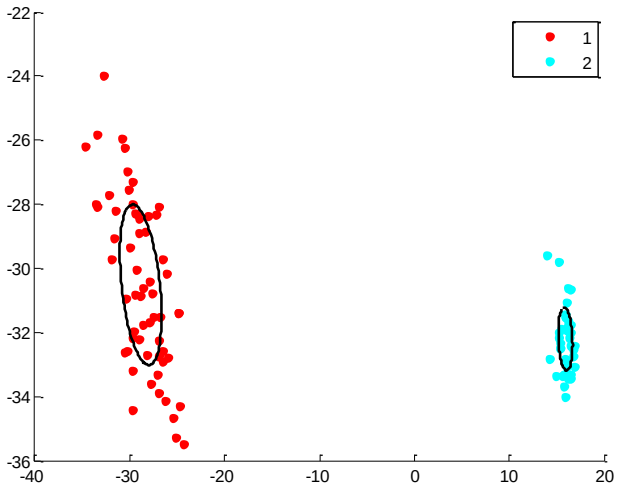

Figure 1 -- Scenario #1 with K-Means Clustering

Figure 2 shows the observations for scenario #2. Again, the shapes and positions of the ellipses reflect the parameters of the underlying Gaussian distributions. The coloring reflects the actual assignment of each observation to the correct cluster. Figure 3 shows the results of the k-means clustering algorithm when applied to the data in Figure 2. The ellipses in Figure 3 are the same ellipses from Figure 2, whereas the color coding reflects the assignments of observations to clusters as determined by the clustering algorithm. While not every data point was assigned to the correct cluster, overall the algorithm performed well.

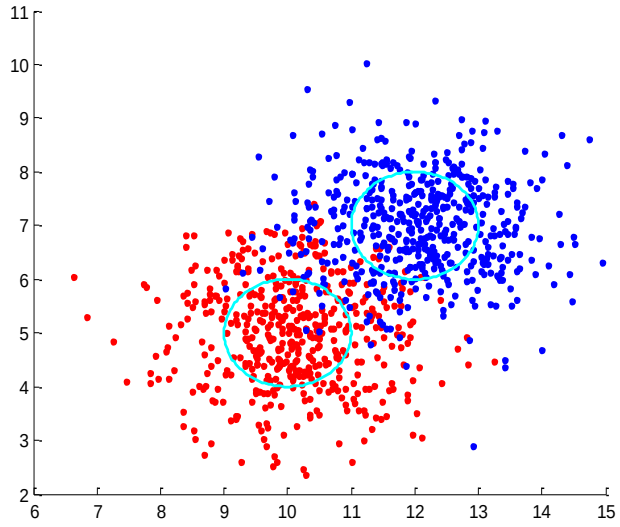

Figure 2 -- Scenario #2a Observations

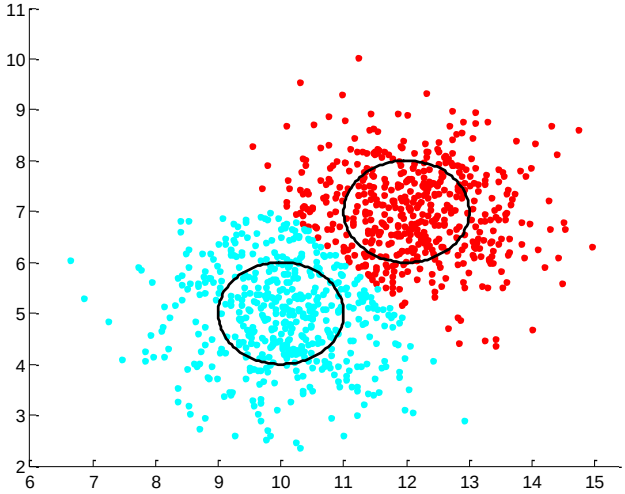

Figure 3 -- Scenario #2a with K-Means Clustering

Figure 4 shows the results of aplying the *k*-means algorithm to scenario #5. As the ellipses indicate, this scenario involved many overlapping clusters. In fact, some clusters are surrounded by other clusters. When this occurs, the *k*-means algorithm will not be able to detect that internal cluster. Since it

is designed to find the specified number of clusters, it will break apart another cluster in order to find the correct number. The broken apart cluster is seen in Figure 4 in the read and yellow data points.

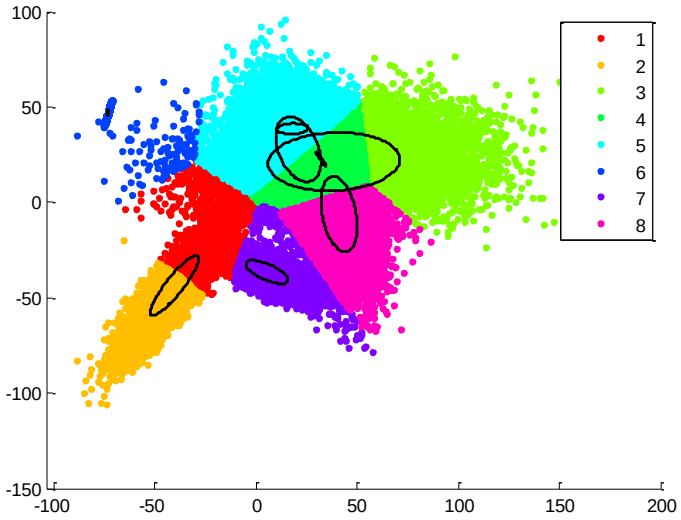

Figure 4 -- Scenario #5 with K-Means Clustering

Figure 5 shows the observations for scenario #2b, where one cluster is completed embedded within the second cluster, and the embedded cluster (data shown in red) has a smaller variance. Figure 6 shows the results of applying the *k*-means clustering to this data set. The algorithm divided the observations into the required number of clusters (two), but these are clearly not the correct clusters.

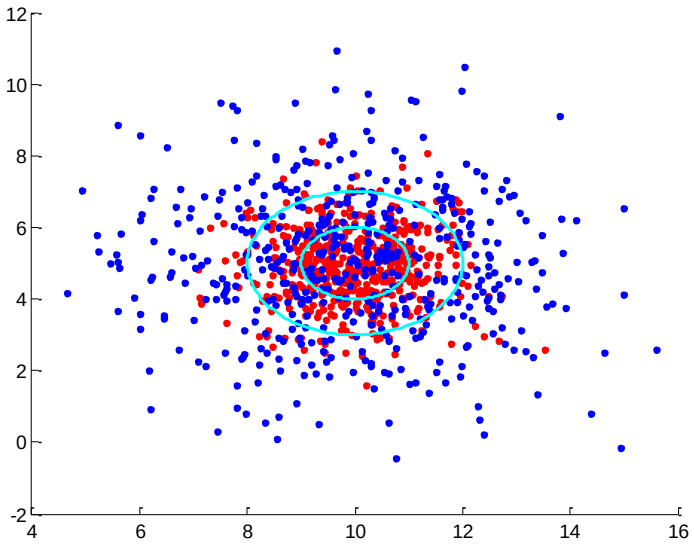

Figure 5 -- Scenario #2b Observations

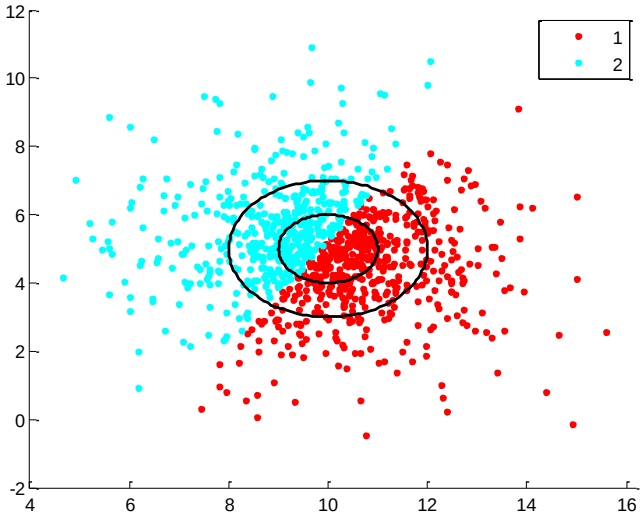

Figure 6 -- Scenario #2b with K-Means Clustering

EM for Gaussian Mixtures

Consistency This algorithm is similar to the *k*-means clustering in that initial values are randomized, so small numbers of replications can produce inconsistent results. Once again, I performed 50

replications for each run to get consistent results. Furthermore, the threshold on the likelihood improvement can also affect the consistency. One aspect of the EM approach is that the parameter estimates converge to local optima. If the threshold is set too high, the algorithm may stop before the estimates have fully converged. I found that setting the threshold to 0.01 for the log-likelihood generally ensured that the estimates converged in time.

Speed The EM for Gaussian mixtures algorithm performed more slowly than the *k*-means clustering; however, for all but scenario #5, the run times were also within a few seconds. The runtime for scenario #5 was 88 seconds.

Correctness This algorithm performed as well or better than *k*-means clustering in all 11 scenarios. It was able to distinguish highly overlapping and embedded clusters.

Figure 7 shows how well EM for Gaussian mixtures performed on scenario #2b. The red ellipses nearly coincide with the light blue ellipses in the graphic. Unlike the results for *k*-means clustering in Figure 6, these results indicate that EM for Gaussian mixtures can handle even embedded clusters.

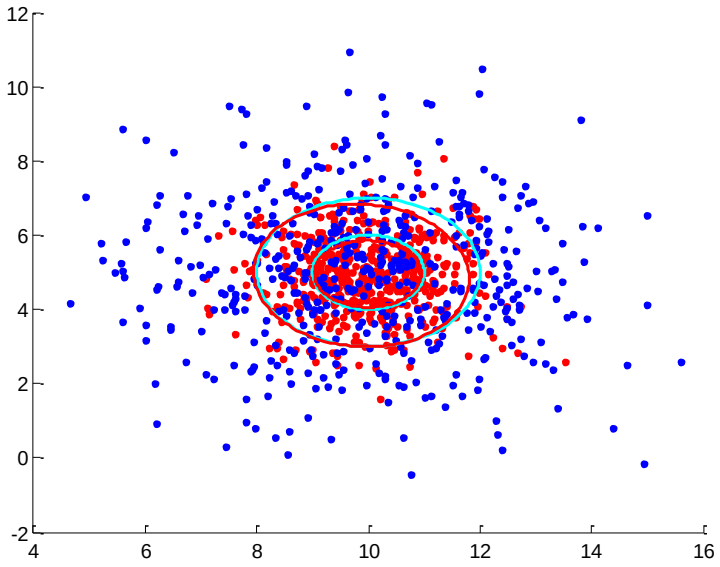

Figure 7 -- Scenario #2b with EM for Gaussian Mixtures

Kurihara's Variational Bayesian Inference

Consistency This algorithm does not rely on a user-provided stopping mechanism as did the *k*-means clustering and EM for Gaussian mixtures. It may initialize parameter values randomly, but I have not been able to determine how this is done. I ran the algorithm multiple times on the same data and different results were achieved. Some of these results were good, while others were not, certainly indicating inconsistency of results.

Speed The Kurihara's Variational Bayesian Inference method was the fastest of all algorithms examined for continuous data. The runtime for scenario #5 was 30 seconds.

Correctness This algorithm is capable of producing very good results. However, since the results can be inconsistent, it is difficult to determine whether a given result is better than another. Figure 8 and Figure 9 below show separate outcomes for the same scenario, #5. Both figures show a very good fit, but neither is definitively better than the other. Furthermore, a third run could produce greater discrepancies.

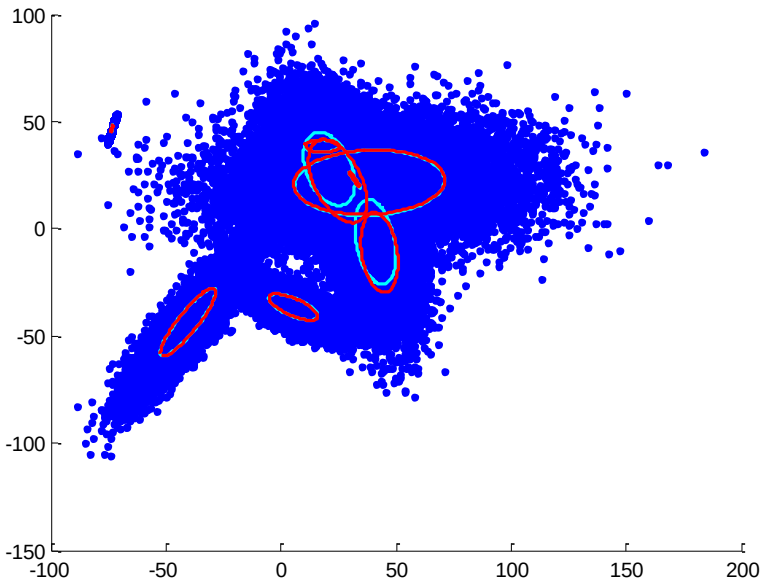

Figure 8 -- Scenario #5 with Kurihara's Variational Bayesian Inference – First Run

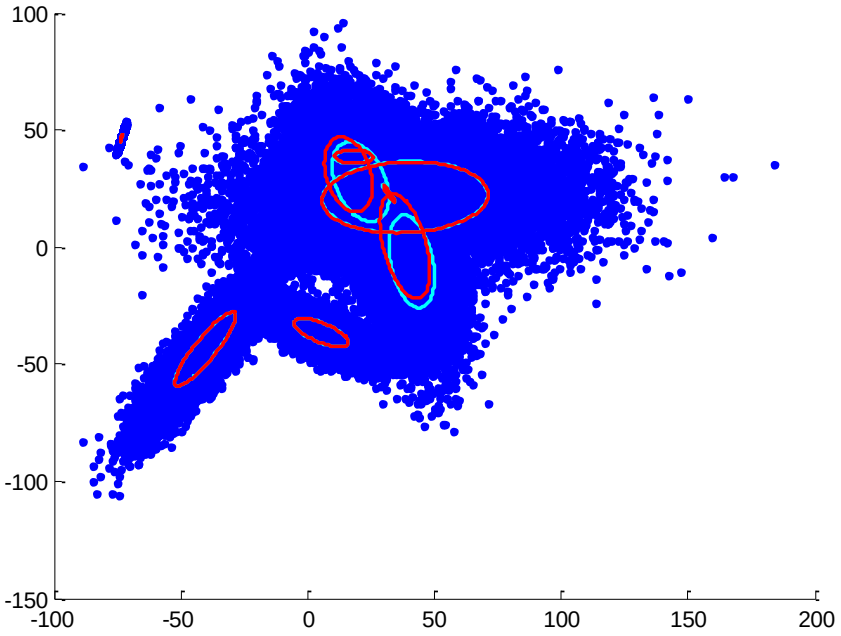

Figure 9 -- Scenario #5 with Kurihara's Variational Bayesian Inference – Second Run

I was curious as to whether this algorithm could handle a large number of clusters, dimensions, and observations. I created an additional scenario with 20 clusters, in 10 dimensions, with 10,000 data points. Figure 10 shows the results in a 2-dimensional view. While not all the clusters are perfectly matched, the algorithm performed well on a number of clusters. This was performed in 7.8 seconds.

This prompted me to examine the other two algorithms on this additional scenario. I ran the EM for Gaussian mixtures with my default parameters of 50 replications and a 0.01 threshold for the log-likelihood improvement. The results are shown in figure 11 and appeared to be a bit better than Kurihara's Variational Bayesian Inference. However, the run time was slower at 63.6 seconds. I then adjusted the input parameters in an attempt to find a better fit. I used 500 replications with a threshold of 0.00001. Figure 12 shows these results. While the results are clearly better in the second run, the run time was 1318 seconds, which is over 20 minutes. It should be noted that, since Kurihara's method has not input parameters, there was no guarantee that additional runs would improve the results.

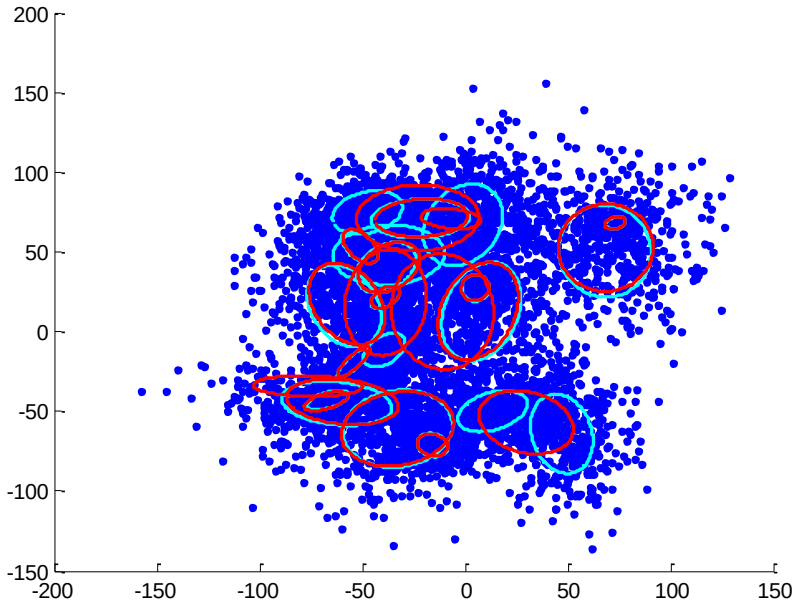

Figure 10 -- Additional Scenario with Kurihara's Variational Bayesian Inference

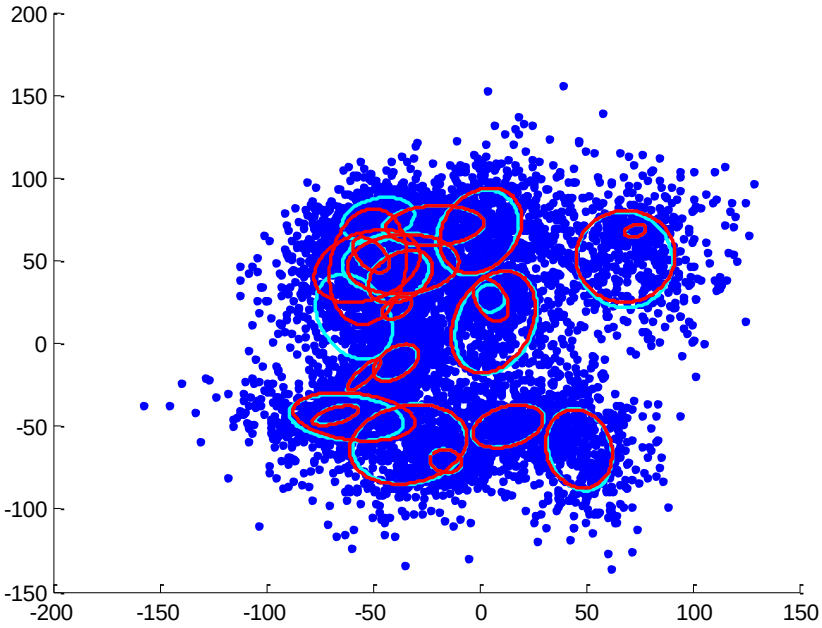

Figure 11 -- Additional Scenario with EM for Gaussian Mixtures 1st Run

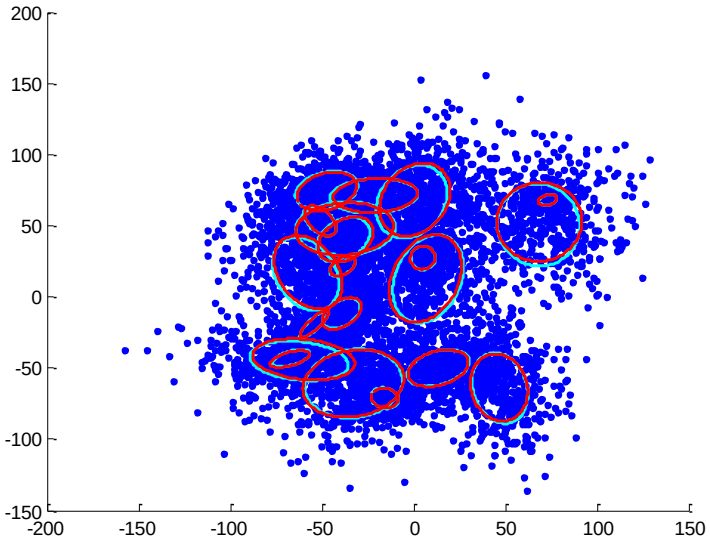

Figure 12 -- Additional Scenario with EM for Gaussian Mixtures 2nd Run

Figure 13 shows the results for the k-means, which were successful for at least some of the clusters and were produced in 16.4 seconds. It is difficult to compare the k-means clustering algorithm to the other two for this scenario, due to the differences in the visualizations of the results. For this reason, I did not attempt further runs with this method.

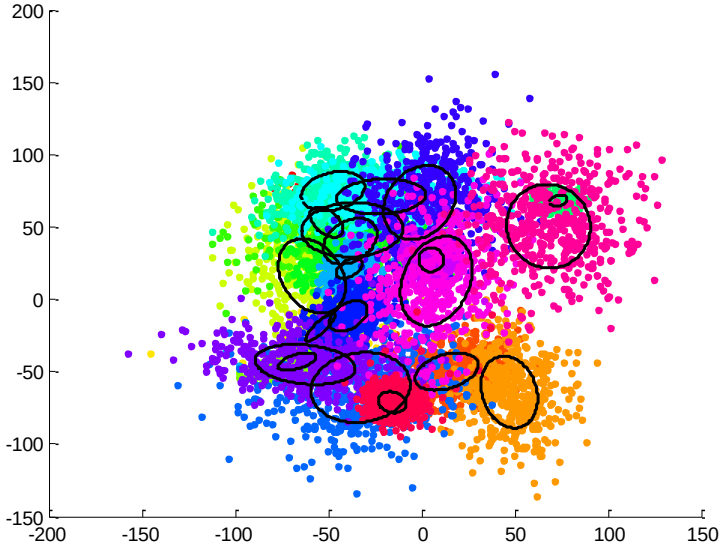

Figure 13 -- Additional Scenario with K-Means Clustering

DPGMM – Metropolis-Hastings

Consistency This method produced very inconsistent results, as it did not produce the same results for any two separate runs. Although this was expected, the results varied more than I hoped. Every time a new parameter is estimated, the value is drawn from a continuous prior or posterior distribution; thus the chance of obtaining the same results twice is zero.

Speed This algorithm is computationally intensive. The largest data set I ran it on (scenario #7) contained 10,000 observations and it took 1680 seconds (28 minutes). This was for only two iterations. The results could have improved with more iterations but that would have required significantly more time.

The concentration parameter also influences the speed. This is because (as mentioned earlier) it influences the number of clusters created. With a greater number of clusters being created, more computations are required. I kept the concentration parameter at one for most of the runs because I made no assumptions about the number of clusters. For scenario #7, I decreased the concentration parameter for a second run because the first run had resulted in twice as many clusters as there really were in the data. This reduction improved the results (number of clusters) and reduced the time.

Finally, the number of auxiliary sets impacts computational expense. I kept the number of auxiliary sets equal to two for most of the runs. Where I did increase the number to three to obtain better results, it took significantly longer.

Correctness This algorithm rarely found the correct number of clusters, often resulting in too many clusters. With a lower concentration parameter and increased iterations, the algorithm produced fewer clusters, though sometimes still too many. Those clusters that it did find, however, had fairly accurate parameter estimations.

Figure 14 shows the results for the initial run for scenario #6, with the number of iterations set to two and the number of auxiliary sets also set to two. The figure is a 2-dimensional view of 3-dimensional data, but the four clusters can be easily viewed. Scenario #6 has only four clusters but the algorithm found six.

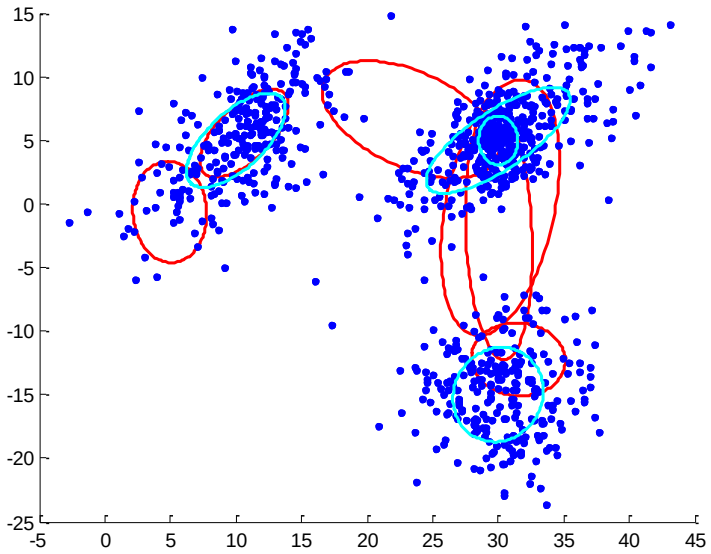

Figure 14 -- Scenario #6 First Run for DPGMM with Metropolis-Hastings

Figure 15 shows the results for a better fit of the data for scenario #6, after increasing the number of auxiliary sets by one and the number of iterations by one. The ellipses for the model clusters in the right half nicely align with the ellipses representing the true underlying distributions. The modeled ellipse for the cluster on the left, however, does not align as well. This indicates that the parameter estimates are not as close to the actual parameters.

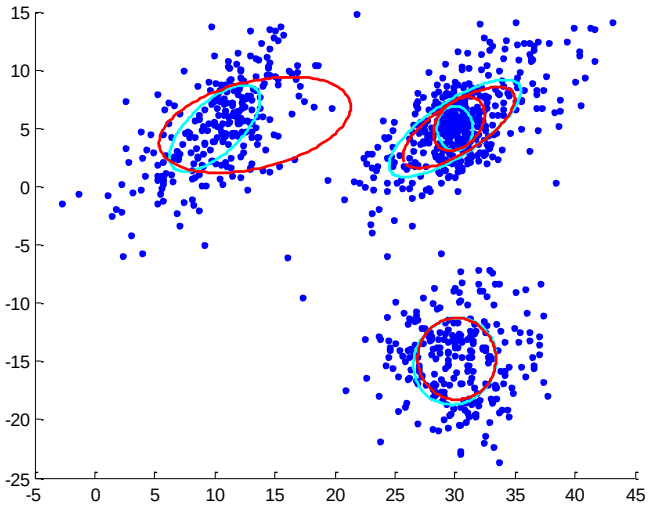

Figure 15 -- Scenario #6 Second Run for DPGMM with Metropolis-Hastings

Figure 16 shows the results from scenario #3. Nine clusters were found where there were only eight. The three concentrated, disparate clusters matched well. The cluster in the lower left quadrant was incorrectly broken into two clusters, as was the cluster directly to the right. The other three clusters were grouped into two clusters. I considered these results to be mixed.

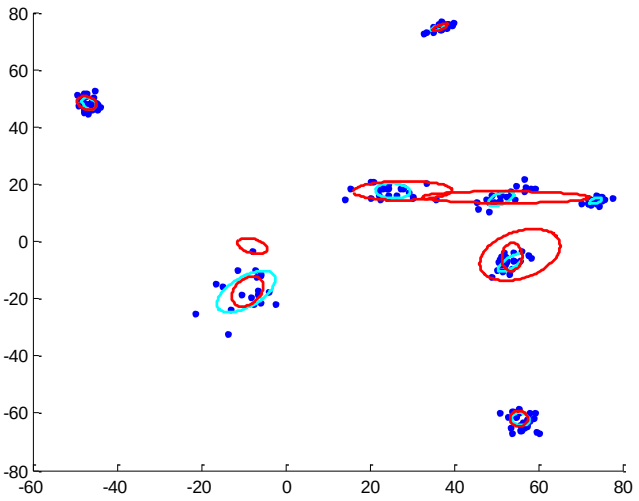

Figure 16 -- Scenario #3 for DPGMM with Metropolis-Hastings

Kurihara's VDPGMM

Consistency This algorithm was completely consistent. Different runs on the same data produced the same results.

Speed Kurihara's VDPGMM is very fast, even compared with the *k*-means clustering and EM for Gaussian mixtures. For example, running scenario #5 took only 71 seconds. However, for smaller data sets, it typically took at least one second to run, whereas *k*-means and EM for Gaussian mixtures took only fractions of a second. Kurihara's VDPGMM was not as fast as his own Variational Bayesian Inference method, however. It is clearly the fastest among the DPGMM algorithms.

Correctness This method of Kurihara's performed the best, getting the correct number of clusters in all but one run. Furthermore, it produced highly accurate parameter estimates.

Figure 17 shows the results of applying Kurihara's VDPGMM method to scenario #5. Though not perfect (one cluster was split in two), it estimates the parameters of the clusters identified by the green arrows better than Kurihara's Variational Bayesian Inference method. This scenario was not even attempted for the DPGMM Metropolis-Hastings because the run would simply take too long, especially if the input parameters needed to be adjusted.

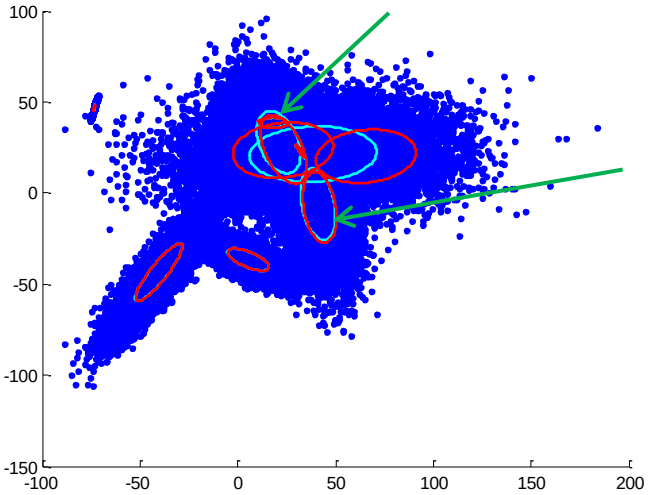

Figure 17 -- Scenario #5 for Kurihara's VDPGMM

Figure 18 below shows the same view of scenario #6 that was shown for DPGMM with Metropolis-Hastings (Figure 15 and Figure 16), but with the data fit using Kurihara's VDPGMM method. As seen in Figure 18, the fit was very good. Figure 19 and Figure 20 show the additional two 2-dimensional views of these results, illustrating that it successfully clustered the data in all dimensions.

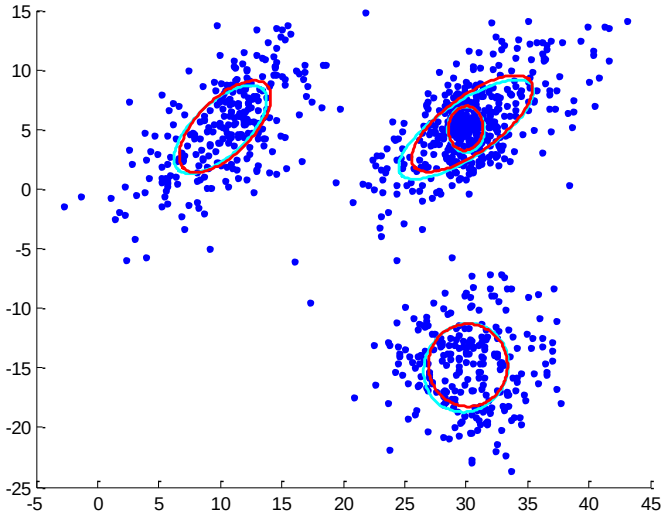

Figure 18-- Scenario #6 for Kurihara's VDPGMM (first view)

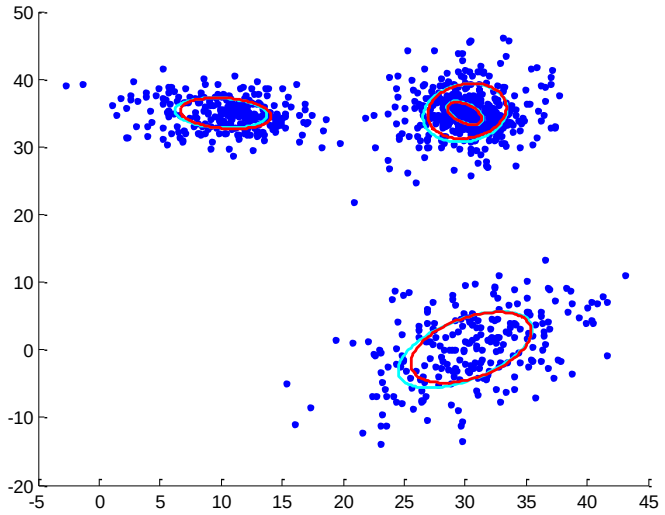

Figure 19-- Scenario #6 for Kurihara's VDPGMM (second view)

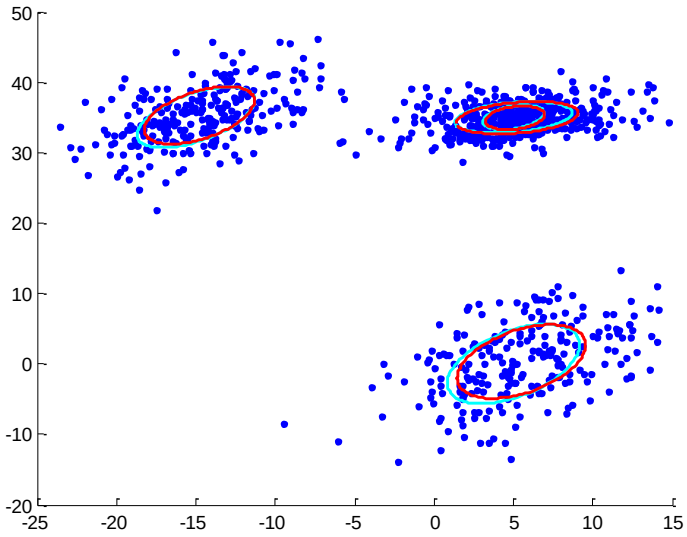

Figure 20 -- Scenario #6 for Kurihara's VDPGMM (third view)

Since Kurihara's VDPGMM performed so well, I decided to run it against the additional scenario. Figure 21 shows just how well the algorithm handled the large number of clusters and large number of dimensions. And it took only 24.7 seconds.

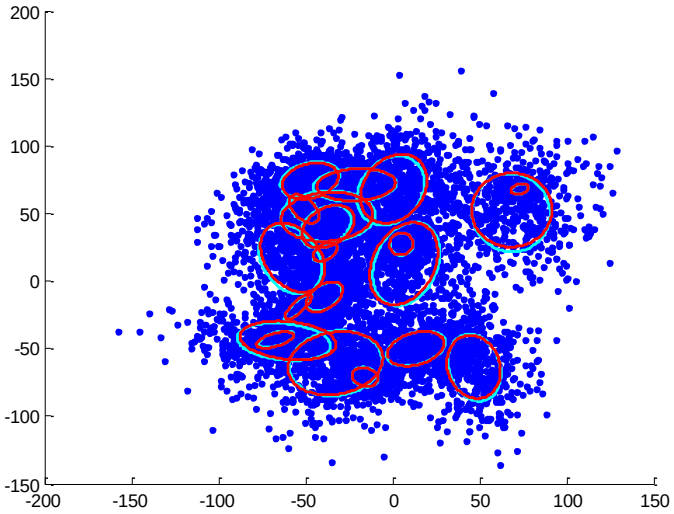

Figure 21-- Additional Scenario with Kurihara's VDPGMM

EM with Dirichlet Prior

Consistency Because this method uses EM for Gaussian mixtures, it has the same inconsistency issue with small numbers of replications and a weak (set too high) threshold. When the replications were set at 50 and the threshold set at 0.01, the results were generally consistent.

Speed While significantly faster than DPGMM Metropolis-Hastings method, this algorithm could not compete with Kurihara's VDPGMM. Because of the intensive computational needs of this method, the largest data set used was scenario #7, which took 41.3 seconds, compared to the 28 minutes that the Metropolis-Hastings method took. Kurihara's method for scenario #7 took only 1.0 seconds.

Correctness The results for this method were mixed. It generally found the correct number of clusters, though sometimes it found as many as three additional clusters. Despite the additional clusters, the parameter estimates for many of the other clusters were highly accurate.

Figure 22 shows the results from running EM with a Dirichlet prior on Scenario #3, illustrating improvement over the DPGMM Metropolis-Hastings method. Clearly, there is still room for further improvement, which can be achieved by adjusting the inputs (e.g. concentration parameter).

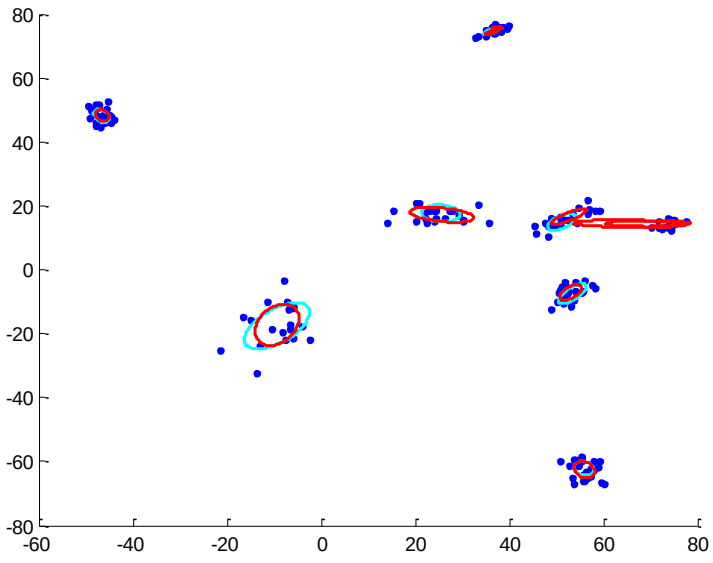

Figure 22-- Scenario #3 for EM with Dirichlet Prior

Figure 23 shows the results for EM with a Dirichlet prior when applied to Scenario #6. By comparing this with Figure 18, it is clear that the parameter estimates were nearly identical to those produced by Kurihara's VDPGMM algorithm.

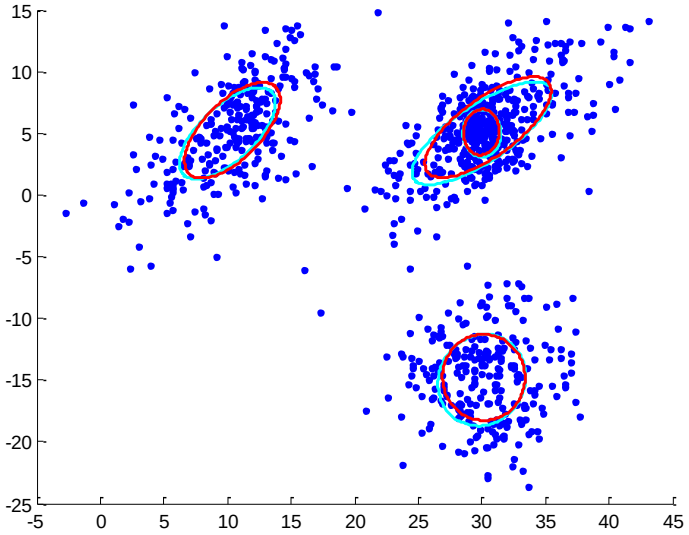

Figure 23 -- Scenario #6 for EM with Dirichlet Prior

8. Conclusions and Recommendations for Clustering Continuous Data

While my analyses did not cover all possible scenarios, I was able to identify both strengths and weaknesses in the various algorithms I investigated. In this section, I provide my recommendations for the use of these methods. I also provide possible areas for further investigation or research.

When the number of clusters is known, it is often worth running Kurihara's Variational method for Dirichlet Process Gaussian Mixture Models on the data, even though the number of clusters does not need to be specified for this method. If the method correctly determines the number of clusters, then the results should be the best. If the number of clusters is wrong, these results should be discarded and one of the methods requiring that the user specify the number of clusters should be considered. The only reason not to first try Kurihara's VDPGMM method is if speed is more important than the precision of the parameter estimates.

The three algorithms that require the number of clusters be specified by the user are *k*-means clustering, EM for Gaussian Mixture Models, and Kurihara's Variational Bayesian Inference method. Since I have found Kurihara's method to be inconsistent, I do not recommend using this algorithm. When deciding between *k*-means clustering and EM, the factors to consider include the disparity of the clusters and the speed. If the clusters are known to be disparate and speed is important, I recommend using k-means clustering. If speed is not important, regardless of the disparity of the clusters, I recommend using EM. It allows the user to make the trade-off between precision and speed, by adjusting the input parameter for the likelihood threshold.

When the number of clusters is not known, I recommend Kurihara's VDPGMM method. It has proven to be consistent, fast, and accurate in all scenarios I examined, even with large numbers of clusters, high dimensionality, and large data sets. The algorithm I created that combined EM with a Dirichlet Prior was an improvement over the Metropolis-Hastings DPGMM; however, it did not perform as well as Kurihara's VDPGMM method.

I have not recommended the use of either the Metropolis-Hastings DPGMM method or Kurihara's Variational Bayesian Inference method; however, I believe there are ways to improve upon each. I have also not recommended the use of EM with Dirichlet Prior; however, I do not have suggestions for improving on that method.

For the Metropolis-Hastings DPGMM method, it is worth investigating methods for determining the appropriate inputs: concentration parameter, prior distributions, etc. I tried a number of different values for these inputs and it was clear that some worked better than others. I may have obtained better results overall had I started with more optimal input values.

My primary objection to Kurihara's Variational Bayesian Inference method was its inconsistency. Implementing a way to perform several replications and choose among the best results, similar to what EM and *k*-means do, may overcome this inconsistency.

9. Algorithms for Examining Discrete Data

As described earlier, the discrete data represented sub-networks, consisting of nodes of different types and connections. The objective of my analysis was to assign each sub-network (observation) to its proper category. Because the information on connections was limited, I was interested in finding methods of examining the data that did not depend upon knowledge of the connections. Thus categorization of each sub-network was based solely on its number and types of nodes.

Each sub-network is associated with a vector, the length of which corresponds to the number of different node types. The value for the j^{th} element of the vector is equal to the number of nodes of type j in that sub-network. These vectors can be viewed as histograms.

All the algorithms for categorizing discrete data included in this project required training data. I did examine one technique that did not require training data, but its performance was so poor in comparison to the other algorithms on some initial data sets that I did not pursue it further.

1) Histogram-to-Histogram Matching (HTHM)

The concept behind the histogram-to-histogram matching is placing sub-networks in categories with similar sub-networks from the training data. This is done by comparing the histogram (vectors containing the number of nodes of each type) of the uncategorized sub-net to those of all the training data. “Similar” histograms have shorter “distances” between them. The uncategorized sub-network is placed in the same category as the training data point with the “closest” histogram. The “distance” between histograms can be measured in a variety of ways. I examined six different ones, the details of which are included in Appendix B – Implementation of Algorithms. I selected the distance measure that resulted in the best overall performance to use in my analysis.

2) Variational Latent Dirichlet Allocation (VLDA) with HTHM

This technique is similar to the HTHM method in that it involves comparing the histogram of the uncategorized sub-net to histograms derived from the training data. However, rather than examine every histogram in the training data, this technique derives representative (normalized) histograms from the sub-nets in each category by using a Variational Latent Dirichlet Allocation algorithm. There may be only one histogram from any given category, or there may be multiple histograms if subcategories are distinguished. In my implementation of this algorithm I limited the number of subcategories to no more

than five. For measuring the distance between two histograms, I used the same distance measure used for the HTHM method.

3) Mixture Model Fitting

I developed this technique in an effort to find a more accurate or robust method than those previously described. I wanted a technique that could allow for a mixture of underlying discrete distributions, similar to the continuous data, where there was a mixture of underlying Gaussian distributions.

The technique treats the number of nodes of a particular type in one sub-network as a random variable, independent of the number of nodes of a different type in that same sub-network. Furthermore, whereas the VLDA uses the relative frequency of node types (in the histograms), my method considers the absolute count of nodes, making this method more robust. I characterize each category using mixture models rather than histograms.

Models are fit to the training data using two different model types, Binomial and Poisson mixtures. Similar to the Gaussian mixtures, the Binomial and Poisson mixtures consist of data clusters where each cluster is defined by a single underlying distribution. Since the number of clusters in my data was not known, I chose to apply the Dirichlet process (DP) and EM with a Dirichlet prior technique that I had investigated for the continuous data. These techniques were redesigned to be applied to Binomial and Poisson mixtures rather than Gaussian mixtures. Applying them to each distribution mixture type resulted in four different algorithms. Since these algorithms are proprietary, the details of their implementations are not included in this paper.

As with the continuous data, the DP and EM with Dirichlet prior techniques for discrete data require a concentration parameter. The DP technique further requires the user to specify the number of iterations for each process. The EM with Dirichlet prior also requires the user to specify a range for the number of clusters, as well as a threshold for improvements in the likelihood (as with any EM technique).

10. Analysis and Results for Discrete Data

As discussed in Section 9, I investigated six algorithms for analyzing discrete data: 1) HTHM, 2) VLDA with HTHM, 3) Binomial Mixture Modeling with EM (BMM EM), 4) Binomial Mixture Model with DP (BMM DP), 5) Poisson Mixture Model with EM (PMM EM), and 6) Poisson Mixture Model with DP (PMM DP). Data sets were generated using six different methods. Each method was applied to generate data sets containing ten different category counts. The rationale behind the category counts is explained further in Appendix A – Generating Synthetic Data. Six methods times ten category counts resulted in 60 data sets. Each algorithm was assessed against each data set.

Each algorithm was examined in terms of how well it performed and how long it took to complete. Unlike the analysis of the continuous data, consistency in the algorithms was not an issue. The HTHM method will always produce the same result when run on the same data set. While VLDA and the DP techniques can produce different results, I found that rarely occurred in this analysis. For the EM with Dirichlet prior method, I deliberately performed 50 replications to overcome the consistency issue, as discussed in the sections for the continuous data analysis.

To evaluate how well each algorithm performed, I ran each on every data set and noted how many times the correct category was identified as the "most likely" as well as how many times the correct category was among the top three. Recall for the "leave one out" cross-validation technique, an algorithm is run against a data set over and over again, until each data point has been left out once. This produces a number of results for each data set for that algorithm. For each algorithm and data set combination, I calculated the percent of time the algorithm identified the correct category as the most likely. Since data sets were generated using six different methods for each number of categories, I could look at the average performance of a given algorithm across the six methods. Table 3 shows those averages by the algorithm as well as by the number of categories. The table is color-coded to show the best performance in green and the worst performance in red. Colors ranging from green to yellow to red represent performance decreasing from the best to the worst. I also calculated how often the correct category as among the top three identified by each algorithm. The averages from results are given in Table 4.

Average % Best Choice was Correct						
# Categories	BMM EM	BMM DP	PMM EM	PMM DP	HTHM	VLDA-HTHM
6	86%	88%	91%	89%	84%	83%
7	80%	85%	89%	86%	81%	78%
9	78%	82%	85%	83%	78%	76%
10	74%	76%	78%	77%	71%	68%
11	77%	79%	82%	77%	70%	68%
13	76%	79%	82%	77%	75%	71%
19	69%	72%	76%	70%	66%	64%
21	69%	71%	73%	71%	64%	61%
23	72%	73%	79%	74%	86%	26%
31	62%	64%	69%	62%	60%	52%

Table 3 -- Table of Averages for "Best Choice" Performance

Average % of Times Correct Category in Top 3						
# Categories	BMM EM	BMM DP	PMM EM	PMM DP	HTHM	VLDA-HTHM
6	96.8%	99.2%	100.0%	99.5%	96.0%	98.8%
7	94.8%	97.4%	98.7%	98.6%	92.8%	95.5%
9	93.0%	95.3%	96.6%	96.9%	92.6%	93.4%
10	94.2%	95.8%	97.5%	96.5%	91.2%	93.3%
11	93.5%	96.3%	98.2%	97.7%	91.2%	94.9%
13	91.1%	95.3%	96.9%	96.6%	91.7%	94.0%
19	86.5%	92.0%	95.2%	92.3%	86.9%	88.1%
21	88.2%	92.3%	94.1%	91.9%	85.3%	87.5%
23	94.8%	96.1%	97.4%	96.1%	93.3%	94.6%
31	81.5%	86.6%	90.8%	86.7%	82.8%	76.1%

Table 4 -- Table of Averages for "Top 3" Performance

It is easily seen from the tables that the performance of all algorithms dropped (almost monotonically) as the number of categories increased. This is somewhat to be expected because, with a larger number of categories to choose from, an algorithm is less likely to select the correct one, especially if the categories are less distinguishable. This is even less surprising for the data sets in my study because they were built upon one another. For example, a data set containing 13 categories has the same categories as another data set with 11, but with two new categories. (For additional information on how the data sets were generated, see Appendix A – Generating Synthetic Data.)

Overall results were rather consistent across a given number of categories. That is, the range of averages seen among the algorithms for any category size was not large. However, PMM EM consistently performed at the top for both measures of performance.

It is easily seen in Table 3 that, for data sets with 23 categories, VLDA-HTHM performed unusually poorly in identifying the correct categories. Performance this poor was seen in no other circumstances. What is interesting to note, however, is that for the same data sets the algorithm's top three choices frequently included the correct category, as seen in Table 4. I examined the data sets and could discern that the normalized histograms that VLDA produces to compare were exactly the same. All the other algorithms use absolute counts rather than relative frequencies, so they did not encounter this problem.

I also examined the algorithms in terms of speed. Table 5 gives the average run times for cross-validation of each algorithm across the six data sets for the ten different numbers of categories. HTHM was by far the fastest performing algorithm, by at least two orders of magnitude. BMM EM was by far the slowest. The remaining four are within the same order of magnitude but differences exist.

Average Cross-Validation Run Times (Seconds)						
# Categories	BMM EM	BMM DP	PMM EM	PMM DP	HTHM	VLDA-HTHM
6	1118	42	19	55	0.1	9
7	1474	55	25	73	0.2	12
9	2310	85	39	111	0.2	19
10	2483	88	41	115	0.2	23
11	2731	101	47	130	0.3	25
13	3154	112	54	148	0.3	30
19	4665	155	74	192	0.5	54
21	4765	169	82	209	0.5	63
23	6339	206	110	259	0.7	103
31	27787	791	381	1027	6.6	301

Table 5 -- Average Cross-Validation Run Times

When the run time results are combined with the performance results, it is easy to see that PMM EM excels overall. While its speed does not rival that of HTHM, it is comparable to the speed of VLDA-HTHM and better than the rest. On the other hand, VLDA-HTHM's performance was as poor as that of HTHM, but its speed was nowhere near as fast.

11. Conclusions and Recommendations for Categorizing Discrete Data

All the algorithms performed well on at least some of the data sets. They were all capable of narrowing down the number of categories. Therefore, they can all be of use if there are additional means for choosing among a small number (such as three) of categories.

Based on my analysis, the best algorithm overall was PMM EM. It was the top performer and its speed was close to the second fastest. However, the performances of the others were not far behind. HTHM also stood out because of its exceptional speed.

There does not appear to be a good reason to select VLDA-HTHM. Its performance was comparable to that of HTHM, but its speed was much worse. Furthermore, it was shown to have problems when two categories share the same normalized histogram. This is an inherent flaw in the method, not found in HTHM or the other algorithms.

While the performance of BMM EM was reasonable, it took so long to run that it presented no advantage over the other mixture modeling methods. If the computational challenges could be overcome, the method would be more viable.

The PMM EM, BMM DP, and PMM DP algorithms all performed well enough by my standards. But evaluating performance depends very much on individual needs. Further research into ways to leverage HTHM's speed and the performance of some of the mixture modeling methods might result in a hybrid algorithm that improves upon the techniques examined here.

12. Appendix A – Generating Synthetic Data

In order to generate the synthetic data, I needed to sample values from generative models that accurately represent the distributions of interest. For the continuous case, I was free to set the Gaussian mixture model parameters to whatever values I chose. For the discrete case, I needed to infer the models from the real data I was given. Once I had my models, I used MATLAB to sample pseudo-random values from these models. Further detail on how I generated the discrete data is provided below.

Since the underlying distributions for the real (discrete) data were of unknown form, I considered three different types of models that could potentially describe the real data: Binomial mixture models, Poisson mixture models, and histograms. For each of these model types, I considered two different approaches for fitting the models to the data, resulting in six different methods for generating the synthetic discrete data.

It is important to note that every synthetic data set I generated contained exactly the same categories as the corresponding real data set, with the numbers of sub-networks within each category also being the same between the real and synthetic data sets. I chose to do this so that the only difference between the real and synthetic data sets was in the observed values within the data sets.

It is also important to note that the real data actually contained additional categories, beyond the thirty-one categories included in the synthetic sets. However, many of these additional categories only contained one or two sub-networks. Any attempt at testing the algorithms with these categories would produce unreliable results, especially considering the fact that “leave-one-out” cross-validation would leave out entire categories (making correct classification impossible for those categories). For this reason, I examined only subsets of the data in which each category contained at least three sub-networks.

Furthermore, I suspected each algorithm’s performance may vary depending on the amount of available training data. I therefore decided to examine several subsets of the data, each with a different minimum number of sub-networks per category. These subsets are represented by the various synthetic data sets, with the smallest set containing six categories (minimum of twelve sub-networks per category) and the largest set containing thirty-one categories (minimum of three sub-networks per category). This is the reasoning behind the numbers of categories in each of the synthetic data sets.

However, I must point out that each subset of the real data is contained within each of the larger subsets (i.e. the set with seven categories contains all the data in the set with six, the set with nine contains all the data in the set with seven, and so on). This is not the case with the synthetic data sets, since I generated each of those sixty sets independently.

Binomial and Poisson Mixture Models

For the Binomial and Poisson mixture model types, I used the same model fitting techniques that I applied in my classification algorithms: Expectation Maximization (with a Dirichlet prior) and Dirichlet Process Mixture Modeling. In brief, I considered each class separately and fit one model for each of the node types. This produced a group of models for each class, where each model in a group only represents the distribution over node counts for a single node type and only for the class that the group belongs to. Every time I generated synthetic data for a new (hypothetical) sub-network, I drew a sample value from each model in the group individually. The only differences among the Binomial and Poisson methods were in the model parameters and how they were estimated.

Histogram/Relative Frequency Models

For the histogram type of model, I generated the data based on the relative frequencies of node types, which were obtained using two different methods. In the first method, I sampled from the empirical distribution of each sub-network in the data. In other words, for each sub-network in the real data, I normalized its histogram, and sampled values (node types) from the normalized histogram, counting the number of times each node type occurs. It is worth noting that this creates a one-to-one correspondence between the synthetic sub-networks and the real ones. In the second method, I used VLDA to infer representative models for each class, in the same way that was applied in my classification algorithm. Because both of these methods only consider relative frequencies, I needed to provide the number of times each model should be sampled from. I decided that the most appropriate number of samples for each synthetic sub-network would be the total number of nodes in the corresponding real sub-network. Since I couldn't use these exact numbers, I simply imposed a factor of randomness by adding Gaussian noise (with variance equal to 20% of the true number) and rounding off the results to the nearest integers.

13. Appendix B – Implementation of Algorithms

K-means Clustering

Let: $x = \{x_1, x_2, \ldots, x_n\}$ be the set of data, where each x_i is an m-dimensional vector containing the values for the i^{th} observation in the data

n = the number of observations in the data

k = the number of clusters

$\mu = \{ \mu_1, \mu_2, \ldots, \mu_k\}$ be the cluster means, where each μ_j is an m-dimensional vector containing the mean values for the j^{th} cluster

$z = \{z_1, z_2, \ldots, z_n\}$ be the set of cluster assignments, where each z_i is an integer between 1 and k indicating the cluster containing the i^{th} observation in the data

m = the dimension of the data

Procedure:

(1) Initialize μ by randomly choosing values within the range of the data

(2) **Repeat**

(3) **for** i = 1 **to** n

(4) **for** j = 1 **to** k

(5) $$d_{i,j} = \sqrt[m]{\sum_{h=1}^{m} \|x_{i,h} - \mu_{j,h}\|}$$

(6) $$z_i = \arg\min_j \{d_{i,j}\}$$

(7) **for** j = 1 **to** k

(8) μ_j = mean{ all x_i for which z_i = j }

(9) **Until** there are no changes in z

Expectation Maximization for Gaussian Mixtures

Although the EM algorithm I used was just the built-in function for MATLAB, I provide a general implementation of the EM algorithm for multi-dimensional Gaussian mixtures.

Let: $x = \{x_1, x_2, \ldots, x_n\}$ be the set of data, where each x_i is an m-dimensional vector containing the values for the i^{th} observation in the data

m = the dimension of the data

n = the number of observations in the data

k = the number of clusters

$\mu = \{\mu_1, \mu_2, \ldots, \mu_k\}$ be the cluster means, where each μ_j is an m-dimensional vector containing the mean values for the j^{th} cluster

$S = \{S_1, S_2, \ldots, S_k\}$ be the cluster covariance matrices, where each S_j is the m by m covariance matrix for the j^{th} cluster

$w = \{w_1, w_2, \ldots, w_k\}$ be the cluster weights, where each w_j is a proportion (between 0 and 1) equal to the proportion of the data that are associated with the j^{th} cluster. The sum of all cluster weights must equal 1

$z = \{z_{1,1}, z_{1,2}, \ldots, z_{1,n}, \ldots, z_{2,n}, \ldots, z_{k,n}\}$ be a set of indicator variables for cluster association. Each $z_{j,,i}$ is either a 1, indicating that observation x_i is associated with cluster j, or a 0 indicating they are not associated

$\theta = \{w, \mu, S\}$ be the set of model parameter estimates

$f(x_i \mid \theta_j)$ be the probability density function for multi-dimensional Gaussian distributions with parameters from θ_j, evaluated for the i^{th} observation in the data:

$$f(x_i|\theta_j) = \frac{1}{\sqrt{(2\pi)^k |S_j|}} * \exp(-\frac{1}{2}(x_i - \mu_j)^T S_j^{-1}(x_i - \mu_j)$$

$Q(\theta \mid x, \theta^{old})$ be the expectation of the log-likelihood function for the model parameters, given the data and the previous parameter estimates:

$$Q(\theta|x, \theta^{old}) = E\left[\ln \prod_{i=1}^{n} \prod_{j=1}^{k} \{w_j * f(x_i|\theta_j^{old})|z_{j,i} = 1\}\right],$$

where the expression in the brackets is only evaluated when the indicator is equal to 1, otherwise it is exempt from the calculation of the product.

(continued on next page…)

$<z_{j,i}>$ = $E[z_{j,i} | x_i, \theta^{old}]$ be the expected value of the indicator variable for the i^{th} observation and the j^{th} cluster

δ = the desired stopping threshold for improvement in the log-likelihood

Procedure:

(1) **Initialize**

(2) θ^{old} = random values drawn from the domain of possible parameter values

(3) $<z_{j,i}>$ for all i and all j:

$$\langle z_{j,i} \rangle = \frac{w_j^{old} * f\left(x_i | \theta_j^{old}\right)}{\sum_{g=1}^{k} w_g^{old} * f\left(x_i | \theta_g^{old}\right)}, \quad (Eq. 1)$$

(4) $L^{old} = Q(\theta | x, \theta^{old}, <z>)$

(5) w_j' for all j:

$$w_j' = \frac{1}{n} * \sum_{i=1}^{n} \langle z_{j,i} \rangle, \quad (Eq. 2)$$

(6) μ_j' for all j:

$$\mu_j' = \frac{\sum_{i=1}^{n} \langle z_{j,i} \rangle * x_i}{\sum_{i=1}^{n} \langle z_{j,i} \rangle}, \quad (Eq. 3)$$

(7) S_j' for all j:

$$S_j' = \frac{\sum_{i=1}^{n} \langle z_{j,i} \rangle \left(x_i - \mu_j\right)\left(x_i - \mu_j\right)^T}{\sum_{i=1}^{n} \langle z_{j,i} \rangle}, \quad (Eq. 4)$$

(8) $L = Q(\theta | x, <z>, w', \mu', S')$

(9) $d = L - L^{old}$

(10) **Repeat**

(11) $L^{old} = L$

(12) $\theta^{old} = \{ w', \mu', S' \}$

(13) **For** j = 1 to k

(14) **For** i = 1 to n

(15) Update $\langle z_{j,i} \rangle$ with (Eq.1)

(16) Update w_j' with (Eq.2)

(17) Update μ_j' with (Eq.3)

(18) Update S_j' with (Eq.4)

(19) $L = Q(\theta \mid x, \langle z \rangle, w', \mu', S')$

(20) $d = L - L^{old}$

(21) **Until:** $d < \delta$

Inference Algorithm for Dirichlet Process Mixture Models

This algorithm is equivalent to Neal's Algorithm 8 (Neal, 2000) for inferring the parameters for Dirichlet Process mixture models. It introduces auxiliary variables as a means for simplifying the task of assigning data points to clusters. It is a general algorithm that can be implemented for any type of mixture distribution, which is what I did for the Gaussian, Poisson, and Binomial mixture modeling algorithms.

Let: $x = \{x_1, x_2, \ldots, x_n\}$ be the set of data, where x_i is the i^{th} observation in the data

m = the dimension of the data

n = the number of observations in the data

k = the number of clusters in the model *Note: this number changes during the course of the algorithm

$\theta = \{ \theta_1, \theta_2, \ldots, \theta_k\}$ be the set of model parameter estimates for each cluster, where θ_j contains the parameter estimates for the j^{th} cluster

$c = \{c_1, c_2, \ldots, c_n\}$ be the set of indicator variables, where $c_i = j$ indicates that the i^{th} observation is associated with the j^{th} cluster in the model

$F(\theta_j \mid x_i)$ be the likelihood function for cluster parameters given the data, evaluated for the j^{th} cluster's parameters when the i^{th} observation is given. Note: it is equal to the probability density function for a distribution with parameters θ_j, evaluated at x_i.

G be the prior distribution for the model parameters

$\phi = \{ \phi_1, \phi_2, \ldots, \phi_H \}$ be the sets of auxiliary parameters

H = the number of sets of auxiliary parameters

α = the concentration parameter

(continued on next page…)

N = {N_1,N_2, …, N_k} be the number of observations assigned to each cluster in the model, where N_j is the number of observations assigned to the j^{th} cluster

T = the desired number of iterations

Procedure:

(1) **Repeat**

(2) **For** i = 1 to n

(3) **If** $N_{c_i} = 1$ (observation i is in a cluster with no other observations)

(4) **Then** treat that cluster's parameters as the first set of auxiliary parameters → $\phi_1 = \theta_{c_i}$, remove θ_{c_i} from θ, and draw the remaining sets from the prior distribution → $\{\phi_2, \ldots, \phi_H\} \sim G$

(5) **Otherwise** draw the sets of auxiliary parameters from the prior distribution → $\{\phi_1, \ldots, \phi_H\} \sim G$

(6) **Calculate** the probability of cluster assignment for observation i for each of the clusters in θ and for each of the clusters in ϕ:

θ → $P(c_i = j \mid x_{i,}\theta_j, N_j) = b * \frac{N_j}{n-1+\alpha} F(\theta_j|x_i)$,
ϕ → $P(c_i = k + h \mid x_{i,}\, \phi_h) = b * \frac{\alpha/H}{n-1+\alpha} F(\phi_h|x_i)$,
where *b* is the appropriate normalizing constant

(7) **Sample** a new value for c_i from the discrete distribution defined by these probabilities → $c_i \sim P(c_i)$

(8) **If** $c_i > k$ (an auxiliary cluster was selected)

(9) **Then** add the selected set of parameters to the current model → $\theta_{k+1} = \phi_{c_i - k}$,

→ k = k+1

(10) **Update** values in N as needed

(11) **For** j = 1 to k

(12) **Sample** new values for the parameter estimates in θ_j from the posterior distribution → $\theta_j \sim P(\theta_j \mid x, c, G)$

(13) **Until** the T^{th} iteration has completed

Variational Latent Dirichlet Allocation

(directly based on the algorithm described by Blei, Ng, and Jordan in their paper on Latent Dirichlet Allocation, 2003)

To help explain the LDA algorithm I put it in the context of modeling topics for words in multiple documents of a corpus.

Model Parameters:

$\mathbf{w}_d$: set containing all of the words in document d, $\mathbf{w}_d=\{w_{1,d}, w_{2,d}, \ldots\ldots, w_{N,d}\}$ (known)

N_d: total # of words in document d (known)

D: total # of documents in the corpus (known)

θ_d: discrete topic weights for document d (unknown)

$\mathbf{z}_d$: set containing the topic assignments for each word in document d, where each word $w_{n,d}$ in document d belongs to topic $z_{n,d}$ (unknown)

k: total number of topics (for our purposes, we assume k is a known constant, but that is not always the case)

α: parameter for Dirichlet distribution (given initial value, but unknown)

β: parameter for Dirichlet distribution (given initial value, but unknown)

LDA assumes this generative process for each document d in a corpus:

1. Choose $N_d \sim$ Poisson(ξ). [Note: for our purposes we ignore the randomness of N]

2. Choose $\theta_d \sim$ Dir(α).

3. For each of the N words w_n:

(a) Choose a topic $z_n \sim \text{Multinomial}(\theta_d)$.

(b) Choose a word w_n from $P(w_n \mid z_n , \beta)$, a multinomial probability conditioned on the topic z_n.

*Key idea - find $\{ \theta_d , \mathbf{z}_d \}$ for each document that maximizes the posterior probability:

$$P(\theta_d , \mathbf{z}_d \mid \mathbf{w}_d , \alpha , \beta) = \frac{P(\theta_d , \mathbf{z}_d , \mathbf{w}_d \mid \alpha , \beta)}{P(\mathbf{w}_d \mid \alpha , \beta)}$$

Unfortunately, this equation is generally intractable to compute. For this reason, we use the variational expectation maximization (EM) method for approximate inference and parameter estimation. The idea is to introduce a set of variational parameters and use Jensen's inequality to get a lower bound on the log likelihood. This lower bound can be adjusted by changing the values of the variational parameters, so the procedure is to optimize these parameters to achieve the tightest possible lower bound.

For our variational model, we introduce a Dirichlet parameter $\gamma = \{ \gamma_1 , \ldots , \gamma_k \}$ and a set of multinomial parameters $\phi = \{\phi_1 , \ldots , \phi_N\}$ as free variational parameters. This allows us to write the variational distribution as such:

$$q(\theta_d , \mathbf{z}_d \mid \gamma , \phi) = q(\theta_d \mid \gamma) * \prod_{n=1}^{N} q(z_n \mid \phi_n)$$

The optimizing values of these parameters are found by minimizing the Kullback-Liebler (KL) divergence between the variational distribution and the true posterior distribution. This minimization can be achieved by iteratively updating the variational parameters using these two equations:

$$\phi_{n,i} \propto \beta_{n,i} * \exp\{ E[\log(\theta_{d,i}) \mid \gamma] \}$$

$$\gamma_i = \alpha_i + \sum_{n=1}^{N} \varphi_{n,i}$$

The expected value inside the multinomial update can be computed as such:

$$E[\log(\theta_{d,i}) \mid \gamma] = \Psi(\gamma_i) - \Psi(\sum_{j=1}^{k} \gamma_j)$$

[where Ψ is the first derivative of the log-Gamma function]

With these equations we are able to set up a procedure for optimizing the variational parameters:

(1) initialize $\phi_{n,i}(t=0) := 1/k$ for all values of i and n

(2) initialize $\gamma_i(t=0) := \alpha_i + N/k$ for all values of i

(3) **repeat**

(4) **for** n = 1 **to** *N*

(5) **for** i = 1 **to** *k*

(6) $\phi_{n,i}(t+1) := \beta_{n,i} * \exp\{ \Psi(\gamma_i(t)) \}$

(7) normalize $\phi_n(t+1)$ to sum to 1

(8) $\gamma(t+1) := \alpha + \sum_{n=1}^{N} \varphi_n(t+1)$

(9) **until** convergence

We then take the optimized values of the variational parameters to be fixed, and we maximize the resulting lower bound on the log likelihood with respect to the model parameters α and β. After that, we take α and β to be fixed again, and we maximize the lower bound with respect to the variational parameters γ and ϕ once more. We alternate between these two steps until the lower bound converges. Alternating between optimizing the variational parameters and the model parameters forms our variational EM procedure.

For our purposes, we first run this procedure on a set of training data and we save the resulting values for α and β. We then run the procedure again except on our set of testing data, setting the initial values for α and β equal to those that resulted from the training data. The resulting values for ϕ are the probabilities of each of our data samples (words) belonging to each class (topic). Thus far, we have only used this (and other algorithms) for single "documents", but if we did, we would iterate the procedure for each document (allowing them to share values for α and β) and repeat until convergence.

Potential Distance Measures for HTHM:

Earth Mover's Distance (EMD)

(EMD can be used for calculating distance between histograms, but is far more computationally intensive than alternative metrics and has not proven to provide better results.)

To understand Earth Mover's Distance, it helps to visualize histograms of data as piles of dirt. The purpose of Earth Mover's Distance is to provide a general metric for the difference between two signatures. A signature is much like a histogram except that instead of partitioning the data into bins across the domain, they separate the data into individual feature clusters. Each feature cluster $S_j = (m_j , w_j)$ is represented by its mean or mode (m_j, location of the dirt pile) and by its weight (w_j, the percent of the total dirt that is in this particular pile). Since our data is in the form of discrete counts, we simply need to convert those counts into weights that sum to 1.

The calculation of the EMD between two signatures can be formally described with the following linear programming problem:

Let: $P = \{ (p_1, w_{p1}) , \ldots , (p_m, w_{pm}) \}$ be the first signature with m clusters, where p_i is the cluster representative and w_{pi} is the weight of the cluster;

$Q = \{ (q_1, w_{q1}) , \ldots , (q_n, w_{qn}) \}$ be the second signature with n clusters;

$D = [d_{ij}]$ be the ground distance matrix where d_{ij} is the ground distance between clusters p_i and q_j.

We want to find a flow $\mathbf{F} = [f_{ij}]$, where f_{ij} is the flow between p_i and q_j that minimizes the overall cost

$$\mathrm{WORK}(P, Q, \mathbf{F}) = \sum_{i=1}^{m} \sum_{j=1}^{n} d_{ij} f_{ij},$$

Subject to the following constraints:

$$(1) \quad f_{ij} \geq 0, \qquad 1 \leq i \leq m, \; 1 \leq j \leq n$$

$$(2) \quad \sum_{j=1}^{n} f_{ij} \leq w_{pi}, \qquad 1 \leq i \leq m$$

$$(3) \quad \sum_{i=1}^{m} f_{ij} \leq w_{pj}, \qquad 1 \leq j \leq n$$

$$(4) \quad \sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij} = \min\left(\sum_{i=1}^{m} w_{pi}, \sum_{j=1}^{n} w_{qj}\right)$$

Constraint (1) allows moving "dirt" from *P* to *Q* and not vice versa. Constraint (2) limits the amount of dirt that can be sent by the clusters in *P* so that no cluster sends more than its weight. Constraint (3) limits the dirt received by clusters in *Q* so that not cluster receives more than its weight. Finally, Constraint (4) requires that maximum amount of dirt than can be moved is moved (a.k.a. the "total flow"). Once this problem is solved, the resulting **F** is the optimal flow. The Earth Mover's Distance between *P* and *Q* is then defined as the work divided by the total flow:

$$\mathrm{EMD}(P,Q) = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} d_{ij} f_{ij}}{\sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij}}$$

Our algorithm that uses EMD is relatively simple:

(1) For each sample S_i in the set of testing data,

(2) calculate EMD(S_i,Q_j) for each sample Q_j in the set of training data,

(3) determine which sample *Q** in the training data resulted in the lowest EMD,

(4) assign sample S_i to whatever class *Q** is in.

(5) Done.

Other Distance Measures

-**KL Divergence**: measure of difference between a true distribution, *P* (test data), and an approximated distribution, *Q* (model based on training data).

$$D_{KL}(P|Q) = \sum P(x) \ln \frac{P(x)}{Q(x)}$$

-**Jeffrey Divergence**: same as KL Div. except symmetrical, where *M* = (*P*+*Q*)/2 .

$$D_J(P|Q) = \sum \left(P(x) \ln \frac{P(x)}{M(x)} + Q(x) \ln \frac{Q(x)}{M(x)} \right)$$

-**Minkowski Distance**: generalized measure of distance between two vectors, where r=1 results in the absolute difference and r=2 results in the Euclidian distance.

$$D_M(P,Q) = \left(\sum |P(x) - Q(x)|^r \right)^{1/r}$$

-**Chi-Squared Statistic**: measure of how unlikely it is that one distribution was drawn from the population represented by the other.

$$D_{\chi^2}(P,Q) = \sum \frac{(P(x) - M(x))^2}{M(x)}$$

-**Cosine Difference**: angular measure of difference between two vectors.

$$D_{cos}(P,Q) = 1 - \cos(\theta) = 1 - \frac{P \cdot Q}{\|P\| \, \|Q\|}$$

---

14. References

Beal, M. J. (2003). *Variational Algorithms for Approximate Bayesian Inference.* Thesis, University of London, The Gatsby Computational Neuroscience Unit. Retrieved from http://www.cse.buffalo.edu/faculty/mbeal/papers/beal03.pdf

Bilmes, J. A. (1998, April). A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models. Berkely, California, United States: International Computer Science Institute. Retrieved from http://lasa.epfl.ch/teaching/lectures/ML_Phd/Notes/GP-GMM.pdf

Birnbaum, A. (1962). On the Foundation of Statistical Inference. *Joural of the American Statistical Association*, 269-306. Retrieved from http://www.pp.rhul.ac.uk/~cowan/stat/pcl/Birnbaum1962.pdf

Blei, D. M., & Jordan, M. I. (2006). Variational Inference for Dirichlet Process. *Bayesian Analysis, 1*(1), 121-144. Retrieved from http://www.cs.berkeley.edu/~jordan/papers/blei-jordan-ba.pdf

Blei, D. M., Jordan, M. I., & Ng, A. Y. (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 993-1022. Retrieved from http://www.seas.harvard.edu/courses/cs281/papers/blei-ng-jordan-2003.pdf

Hartigan, J. A., & Wong, M. A. (1979). A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics), 28*(1), 100-108.

Neal, R. M. (2000). Markov Chain Sampling Methods for Dirichlet Process Mixture Models. *Journal of Computational and Graphical Statistics, 9*(2), 249-265. Retrieved from http://www.stat.purdue.edu/~rdutta/24.PDF

Kurihara, K., Welling, M., & Vlassis, N. (2007). Accelerated variational Dirichlet process mixtures. *Advances in Neural Information Processing Systems*, 19, 761. Retrieved from http://sato-www.cs.titech.ac.jp/reference/Kurihara07nips.pdf

Lee, G., & Scott, C. (2012). EM algorithms for multivariate Gaussian mixture models with truncated and censored data. *Computational Statistics & Data Analysis*. Retrieved from http://www-personal.umich.edu/~gyemin/pubs/tcem_tr.pdf

Website for Kenichi Kurihara's software: https://sites.google.com/site/kenichikurihara/academic-software/variational-dirichlet-process-gaussian-mixture-model