Efficient Learning of Sparse Gaussian Mixture Models of Protein Conformational Substates

Ji Oh Yoo

July 2015

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Christopher James Langmead, Chair
Wei Wu, Faculty

Submitted in partial fulfillment of the requirements for the degree of Master of Science.

Copyright © 2015 Ji Oh Yoo
Keywords: conformational substates, Molecular Dynamics simulation, Gaussian Mixture Model, Nonparanormal Mixture Model, coreset approximation
Abstract

Molecular Dynamics (MD) simulations are an important technique for studying the conformational dynamics of proteins in Computational Structural Biology. Traditional methods for the analysis of MD simulation assumes a single conformational state underlying the data. With recent developments in MD simulation technologies, MD simulation now can produce massive and long time-scale trajectories across multiple conformational substates, and new efficient methods to analyze these trajectories and to learn structural dynamics of the substates are needed.

In this thesis, we develop new methods to learn parametric and semi-parametric, sparse generative models from the positional fluctuations of amino acid residues in the simulation. Specifically, our methods learn a mixture of sparse Gaussian or nonparanormal distributions. Each mixing component encodes the statistics of a different substate. L1 regularization is used to produce sparse graphical models that are easier to interpret than a simple covariance analysis, because the topology of the graphical model reveals the coupling structure between different parts of the molecule. Our method also employs coreset sampling to enhance scalability.

We demonstrate that our methods produce models that have a number of advantages over traditional Gaussian Mixture Models (GMM). Experiments on synthetic data show substantial improvements over GMMs on the recovery of the true network structure, while remaining competitive in terms of test likelihood and imputation error. Experiments on a large real MD data set are consistent with the results on synthetic data. We also demonstrate that benefits of using semi-parametric models in terms of likelihood and imputation metrics.
## Contents

1 Introduction .............................................................................. 1

2 Background ............................................................................. 3
   2.1 Structural Dynamics of Proteins ........................................ 3
   2.2 Molecular Dynamics Simulations ....................................... 3
   2.3 Analysis of Molecular Dynamics Simulation Data ............... 4
   2.4 Markov Random Field ..................................................... 6
   2.5 Gaussian Graphical Model ................................................. 7
   2.6 Multivariate Gaussian Representations of Protein Dynamics ... 7

3 Methods ................................................................................. 9
   3.1 Gaussian Mixture Model ................................................... 9
   3.2 Sparse Gaussian Mixture Model Using BIC ....................... 10
   3.3 Sparse Gaussian Mixture Model using Cluster Approximation with Coreset Sam-
   pling ...................................................................................... 12
      3.3.1 Coreset Sampling for Gaussian Mixture Model .......... 12
      3.3.2 Learning Sparse Gaussian Mixture Model for Clusters Approximated by
          Coreset Sampling ......................................................... 14
   3.4 Sparse Nonparanormal Mixture Model using Cluster Approximation ......................... 16
   3.5 Summary ........................................................................... 17

4 Experiments and Results ......................................................... 19
   4.1 Performance Metrics ......................................................... 19
      4.1.1 Performance Metrics on Test Dataset ......................... 19
      4.1.2 Performance Metrics against the True Model .............. 20
   4.2 Synthetic Experiments ...................................................... 20
      4.2.1 Generation of Synthetic Data ..................................... 20
      4.2.2 Synthetic Experiments on Sparse Gaussian Mixture Models Using Core-
          set Approximation ......................................................... 21
      4.2.3 Synthetic Experiments on Sparse Nonparanormal Mixture Models Using
          Coreset Approximation ................................................. 37
   4.3 Learning Sparse Mixture Models for Conformational Substates of Engrailed Home-
       odomain ........................................................................... 40
List of Figures

2.1 Three Dimensional Structure of Myoglobin(1MBO) ........................................ 4
2.2 Free Energy Landscape of a Protein According to its Virtual Conformational Coordinate. .................................................. 5
2.3 Approximation of a protein by Elastic Network Model .............................. 8

3.1 Diagram of learning process of a sparse Gaussian Mixture Model using cluster approximation with coreset sampling ....................................... 13

4.1 Negative Log-Likelihood of learned models using \( M_{gmm}, S_{bic}, S_{core}, S_{core2} \) methods. ................................................................. 21
4.2 Average imputation errors of learned models using \( M_{gmm}, S_{bic}, S_{core}, S_{core2} \) methods ................................................................. 22
4.3 Edge recovery accuracy of learned models using \( M_{gmm}, S_{bic}, S_{core}, S_{core2} \) methods. ................................................................. 23
4.4 Edge recovery F1 score of learned models using \( M_{gmm}, S_{bic}, S_{core}, S_{core2} \) methods. ................................................................. 23
4.5 Recovered edges in precision matrices of learned models using \( M_{gmm}, S_{bic}, S_{core}, S_{core2} \) methods ................................. 24
4.6 Recovered edges in precision matrices of learned models using \( M_{gmm}, S_{bic}, S_{core}, S_{core2} \) methods ................................. 25
4.7 Edge recovery accuracy and F1 score of thresholded precision matrices learned using \( M_{gmm} \) method ................................................................. 25
4.8 Negative log-likelihood, imputation error, and edge recovery rate of learned models using \( M_{gmm}, S_{core}, S_{core2} \) methods ....................... 27
4.9 Average imputation errors of learned models using \( M_{gmm}, S_{core}, S_{core2} \) methods ................................................................. 27
4.10 Edge recovery accuracy in precision matrices of learned models using \( M_{gmm}, S_{core}, S_{core2} \) methods ................................................................. 28
4.11 Edge recovery F1 score in precision matrices of learned models using \( M_{gmm}, S_{core}, S_{core2} \) methods ................................................................. 28
4.12 Negative log-likelihood, imputation error, and edge recovery rate of learned models using \( M_{gmm}, S_{core}, S_{core2} \) methods ................................................................. 29
4.13 Average imputation errors of learned models using \( M_{gmm}, S_{core}, S_{core2} \) methods ................................................................. 30
4.14 Edge recovery accuracy in precision matrices of learned models using \( M_{gmm}, S_{core}, S_{core2} \) methods ................................................................. 30
4.15 Edge recovery F1 score in precision matrices of learned models using \( M_{gmm}, S_{core}, S_{core2} \) methods ................................................................. 31
4.16 Negative log-likelihood, imputation error, and edge recovery rate of learned models using $M_{gmm}$, $S_{core}$, $S_{core2}$ methods. .......................................................... 32  
4.17 Average imputation errors of learned models using $M_{gmm}$, $S_{core}$, $S_{core2}$ methods. .......................................................... 32  
4.18 Edge recovery accuracy in precision matrices of learned models using $M_{gmm}$, $S_{core}$, $S_{core2}$ methods. .......................................................... 33  
4.19 Edge recovery F1 score in precision matrices of learned models using $M_{gmm}$, $S_{core}$, $S_{core2}$ methods. .......................................................... 33  
4.20 Negative log-likelihood of learned models using $S_{core}$, $S_{core2}$ and $S_{unif}$ methods. .......................................................... 34  
4.21 Average imputation errors of learned models using $S_{core}$, $S_{core2}$ and $S_{unif}$ methods. .......................................................... 35  
4.22 Edge recovery accuracy in precision matrices of learned models using $S_{core}$, $S_{core2}$ and $S_{unif}$ methods. .......................................................... 35  
4.23 Edge recovery rate in precision matrices of learned models using $S_{core}$, $S_{core2}$ and $S_{unif}$ methods. .......................................................... 36  
4.24 Runtime cost for parameter learning using $S_{core}$, $S_{core2}$ and $S_{unif}$ methods. .......................................................... 36  
4.25 Negative Log-Likelihood of learned models using $M_{gmm}$, $S_{core}$, $S_{npn}$ methods. .......................................................... 38  
4.26 Average imputation errors of learned models using $M_{gmm}$, $S_{core}$, $S_{npn}$ methods. .......................................................... 38  
4.27 Edge recovery accuracy of learned models using $M_{gmm}$, $S_{core}$, $S_{npn}$ methods. .......................................................... 39  
4.28 Edge recovery F1 score of learned models using $M_{gmm}$, $S_{core}$, $S_{npn}$ methods. .......................................................... 39  
4.29 Engrailed homeodomains of *Drosophila melanogaster* (1DU0) bound to DNA double helix. .......................................................... 41  
4.30 Negative log-likelihood of learned models using $M_{gmm}$, $S_{core}$, $S_{unif}$ methods on engrailed homeodomain. .......................................................... 41  
4.31 Average imputation error of learned models using $M_{gmm}$, $S_{core}$, $S_{unif}$ methods on engrailed homeodomain. .......................................................... 42  
4.32 BIC score of learned models using $M_{gmm}$, $S_{core}$, $S_{unif}$ methods on engrailed homeodomain. .......................................................... 42  
4.33 Edges recovered using $M_{gmm}$, $S_{core}$, $S_{unif}$ methods on engrailed homeodomain. .......................................................... 43  
4.34 Average runtime cost of learned models using $M_{gmm}$, $S_{core}$, $S_{unif}$ methods on engrailed homeodomain. .......................................................... 44  
4.35 Average imputation error of models learned using $S_{npn}$ method on engrailed homeodomain with different Winsorization parameters. .......................................................... 45  
4.36 Negative log-likelihood of models learned using $M_{gmm}$, $S_{core}$, $S_{npn}$ method on engrailed homeodomain. .......................................................... 45  
4.37 Average imputation error of models learned using $M_{gmm}$, $S_{core}$, $S_{npn}$ method on engrailed homeodomain. .......................................................... 46  
4.38 Edges recovered using $M_{gmm}$, $S_{core}$, $S_{npn}$ methods on engrailed homeodomain. .......................................................... 46
List of Tables

4.1 Average runtime cost in learning models using $M_{gmm}$, $S_{bic}$, $S_{core}$, $S_{core2}$ methods 26
Chapter 1

Introduction

In statistics and machine learning, extracting the underlying patterns in data and predicting the future behaviors of the target object are often the primary research goals. With the improvements in computational power and increase in storage, developing efficient algorithms for the analysis of large-scale data has become an equally important goal. These goals are especially important in the natural sciences where modern experimental platforms produce massive amounts of data. In this thesis, we focus on the analysis of large-scale data from the simulation of molecular structures, including proteins.

Proteins are among the most important molecules in Biology because they participate in virtually every biological process and structure. It is well established that the three-dimensional structure and dynamics of proteins ultimately govern their behavior. Therefore, it is useful to understand and to model the structure and dynamics of proteins. Traditional methods for analyzing the structure of proteins involve crystallizing the protein samples and conducting various types of spectroscopy. In Computational Structural Biology, the motions of proteins are often studied using a technique known as Molecular Dynamics (MD) simulations [1]. MD simulation can provide 3-dimensional coordinates of atoms in user-specified environments including different settings of temperatures and solvents. Informally, starting with an initial set of atomic coordinates, MD simulations involve iteratively adjusting the positions of the atoms of the system using Newton’s laws of motion. The output of a MD simulation is a time-series of frames, known as a trajectory, reporting the positions of the atoms at different points in time. The goal of this thesis is to introduce new algorithms for analyzing the MD simulation trajectories. Specifically, we employ techniques from Machine Learning to produce mixture models of protein structures. These models will reveal and characterize the conformational substates of the proteins (i.e., the highly probable structures) [2, 3]. These substates, in turn, can be used by Biologists to better understand how each protein performs its tasks.

The primary motivation for this research are advances in the technologies for performing MD simulations. Traditional methods for the analysis of MD simulation implicitly assume that there is a single conformational substate in the data. This assumption was once valid, when MD simulation technologies were less powerful and could only produce simulations form a single substate. Today, however, that assumption is no longer valid, in general. The technological advances in MD now make it possible to produce massive and long time-scale trajectories that often cover multiple substates of a protein [4, 5, 6, 7, 8]. Therefore, methods capable of modeling
multiple substates simultaneously are needed. Our methods use probabilistic graphical models to identify the conformational substates of a protein and to learn the underlying distributions of the substates. The output models of our methods result in generative models that can be further applied for prediction studies of the substates, given small perturbations of partial structure of the protein. Specifically, our approach is based on Gaussian Mixture Models with sparsity constraints for better interpretability. To address the issue of scalability, we use coreset sampling methods to first approximate the clusters (i.e., substates) in the data. Finally, we extend the approach with a semi-parametric method to cover the richer family of distributions than simple multivariate Gaussian distributions.

This thesis is structured as follows. In Chapter 2, we give a brief overview of dynamics of proteins, Molecular Dynamics simulations, and the traditional methods for the analysis of the simulated data. We also include the brief introduction of probabilistic graphical model that we apply in our proposed methods. In Chapter 3, we present our methods for efficient learning structures and parameters of sparse Gaussian Mixture Models for identification of the substates and the semi-parametric extension of our methods. In Chapter 4, we show the evaluations of our methods in various metrics, including likelihood and imputation errors of the learned models that are widely used in machine learning along with the runtime costs of the methods.
Chapter 2

Background

In this chapter, we give a brief overview of the structural dynamics of proteins and the identification of conformational substates of proteins from Molecular Dynamics simulated data. Then, we review the existing methods and the background of our methods using Gaussian Graphical Model to overcome the shortness of existing methods.

2.1 Structural Dynamics of Proteins

A protein is a polypeptide, a linear chain of amino acids. The structure of a protein has been the main subject of studies in biochemistry and molecular biology as the structural properties are closely related to a wide range of functions in biochemical reactions in living organisms. The overall shape of a protein is a three dimensional structure as in Figure 2.1 rather than a linear chain, constructed by local interactions including hydrogen bonds among local amino acids and non-local interactions among the partial structures of the protein. Also, the structure of a working protein is neither rigid nor static but fluctuating and dynamic as each molecule in a protein behaves according the laws of thermodynamics. The motions of a protein and its molecules visit an ensemble of states that can be partitioned into conformational substates, which share the similar overall structure and the biochemical functions but have different local structures or the rate of the functions [2, 3]. The study of these conformational substates and their relationships to the various functions is one of the active area of research, and the multiple techniques including nuclear magnetic resonance (NMR), neutron spectroscopy, and X-ray crystallography have been introduced [9, 10, 11]. However, these experimental techniques, at present, cannot capture all of the details of dynamics in each conformational substate or among the substates.

2.2 Molecular Dynamics Simulations

Molecular Dynamics (MD) simulation is a computer simulation of the movements of the atoms in a given molecular object. Given the initial three dimensional coordinates and the initial velocities of the atoms in the system, MD repeats updating the coordinates of the atoms after a short time step and storing the conformation of the atoms. For each time step, MD numerically solves the Newton’s equations of motions using the interatomic potential energy, and the length of the time
Figure 2.1: Three dimensional structure of myoglobin(1MBO) with bounded oxygen molecule. Myoglobin is a polypeptide with 153 amino acids and has a three-dimensional structure rather than linear or planar structure.

step is usually set to 1-2 femtoseconds ($10^{-15}$ sec). For reasonable size of the data for later analysis, every 1000th to 10000th simulation (1-100 picoseconds, $10^{-12}$ sec) is stored for further analysis as a time series of frames, the conformations of the system.

MD is one of the popular computational methods for studies of conformational substates of proteins as it can capture the details in biomolecular functions of proteins, and user can control the degree of the accuracy and the approximation of the simulation [1]. Recently, by the development of algorithms with higher simulation throughput and the advance in hardware, MD now can generate trajectory of the system for longer time-scale, from microseconds to milliseconds ($10^{-6}$ to $10^{-3}$ seconds) [4, 5, 6, 7, 8]. As the transitions among the conformational substates occur in microseconds to milliseconds time-scale (Figure 2.2), such long time-scale simulations can contain multiple conformational substates. Instead of manually identify substates among the frames, we need efficient algorithms to identify the substates and learn the structural properties of each state from a large-scale data.

2.3 Analysis of Molecular Dynamics Simulation Data

Our goal is to model an underlying distribution of the given Molecular Dynamics (MD) simulation data. We assume that the user provides a trajectory containing the atoms of interest (e.g., $\alpha$-carbons of the amino acid residues of a protein). Given the trajectories from MD simulation, we can model the distribution over several kinds of variables. In this thesis, we model the joint distribution over atomic fluctuations, the distance of each atom from its average position, by learning a generative model from the data.
Figure 2.2: Free Energy Landscape of a Protein According to its Virtual Conformational Coordinate. The populations of different protein conformation follow Boltzmann distribution, and the ratio of populations varies exponentially to the free energy difference of the conformations. The transitions across the smaller energy barriers take picoseconds to nanoseconds, and the transitions among the larger energy barriers take nanoseconds to microseconds. The Molecular Dynamics simulation data of longer than microseconds time-scale contains multiple conformational substates and needs an algorithm to identify the substates.
The methods for the study of the conformational substates have been developed mainly using two strategies. The first is using clustering algorithms on the physical properties proteins (coordinates of atoms or dihedral angles of bonds) to identify the substates [12, 13, 14]. However, the pair-wise comparison in the clustering algorithms make it hard to scalable for the massive long time-scale trajectories we want to analyze, and the methods give non-generative models, which have limits on the study of the behaviors of the system given local perturbations. The second strategy is using Principle Component Analysis (PCA) to reduce the dimensionality of the problem [13, 15, 16]. The PCA-related methods give a generative model, but the change of basis in PCA gives the output that is hard to interpret in terms of the original molecular system and utilize in further studies. Recently, a method for learning probabilistic graphical models with generative property for fluctuations of proteins was proposed to overcome the issues [17]. In this thesis, we extend this method to scale up to very large trajectories for multiple substates and to semi-parametric models. Our method uses the approximation step in the clustering of the frames for identification of the substates for the better scalability, does not change the basis for the better interpretability of the results, and gives a generative model based on graphical models for the applicability in further prediction studies of the target system.

2.4 Markov Random Field

The methods we introduce in this thesis consist of a mixture of Gaussian distributions where each component is encoded as a sparse Markov Random Field. Markov Random Field (MRF) is a graphical model \( M = (G, \Psi) \) based on the undirected graph \( G = (V, E) \) with a set of potential functions \( \Psi \). Each node in \( V = \{v_1, v_2, ..., v_n\} \) represents the corresponding random variable in \( X = \{X_1, X_2, ..., X_n\} \), and each potential function \( \psi_c \in \Psi \) is defined on the nodes in clique \( c \) of \( G \). The joint distribution of \( X \) is defined to be the product of all potential functions after normalization:

\[
P(X_1, X_2, ..., X_n) = \frac{1}{Z} \prod_c \psi(X_c)
\]

where \( X_c \subseteq X \) is the random variables encoded in a clique \( c \), and \( Z \) is a normalization factor called partition function.

The graphical representation of MRF with the definition of joint distribution makes it easier to interpret the conditional independencies between the variables without evaluating the marginal or conditional distributions. For two nodes \( v_i \) and \( v_j \) with the corresponding random variables \( X_i \) and \( X_j \), if \( v_i \) and \( v_j \) are connected by an edge in the graph \( G \), then the two random variables \( X_i \) and \( X_j \) are dependent. Conversely, if \( v_i \) and \( v_j \) are not connected by an edge, then \( X_i \) and \( X_j \) are conditionally independent, given their respective Markov Blankets. This property gives us an informal but useful interpretation of the pair-wise relationships between the variables; if the nodes \( v_i \) and \( v_j \) are connected by an edge, then the correlation between \( X_i \) and \( X_j \) is due to a direct relationship in the system, and if \( v_i \) and \( v_j \) are not connected by an edge but are connected by a path in \( G \), then the correlation between \( X_i \) and \( X_j \) is due to an indirect relationship. There are existing algorithms for inference and learning of the structure and parameters of MRFs, but
these methods, in general, give approximate solutions because exact inference and learning is
NP-hard on MRFs with loops \[18\].

2.5 Gaussian Graphical Model

Gaussian Graphical Model (GGM) \( M = (\mu, \Sigma) \) is a Markov Random Field with continuous random variables distributed in a single multivariate Gaussian distribution with the mean vector \( \mu \) and the covariance matrix \( \Sigma \). The pair-wise conditional independency between \( X_i \) and \( X_j \) is captured by the zero \((i, j)\)th entry in the precision matrix \( \Theta = \Sigma^{-1} \). The potential functions of GGM are encoded in \( \mu \) and \( \Sigma \), and the joint distribution is the multivariate Gaussian distribution:

\[
P(X_1, X_2, ..., X_n) = P(X) = \frac{1}{Z} \exp \left( -\frac{1}{2} (X - \mu)^T \Sigma^{-1} (X - \mu) \right)
\]

where \( Z = \sqrt{(2\pi)^n |\Sigma|} \) is a partition function.

One of the key benefits to using a GGM is that, unlike most other MRFs, learning and inference is analytically tractable. The marginal distribution \( P(X_a) \) of a subset \( X_a \subseteq X \) of variables is simply a multivariate Gaussian distribution with the mean vectors and the covariance matrix with corresponding rows and columns of \( X_a \). The conditional distribution \( P(X_a | X_b = Z_b) \) of \( X_a \) given values of \( X_b = Z_b \) is also a Gaussian distribution with parameters:

\[
\mu_{a|b} = \mu_a + \Sigma_{ab} \Sigma_{bb}^{-1} (Z_b - \mu_b) \\
\Sigma_{a|b} = \Sigma_{aa} - \Sigma_{ab} \Sigma_{bb}^{-1} \Sigma_{ba}
\]

where \( \mu_a \) and \( \mu_b \) are sub-vectors with corresponding rows in the original mean vector \( \mu \), and \( \Sigma_{aa}, \Sigma_{ab}, \Sigma_{ba}, \Sigma_{bb} \) are sub-matrices with corresponding rows and columns in the original covariance matrix \( \Sigma \).

2.6 Multivariate Gaussian Representations of Protein Dynamics

Most methods for analysis of Molecular Dynamics simulation data assume some variation of Elastic Network Model (ENM) for simplified representation of interactions between the atoms. In ENM, a subset of atoms in the system are represented as masses, and the direct relationship between the masses are represented as springs. When a folded protein is approximated with an elastic network of masses (\( \alpha \)-carbons of various residues) and springs (pair-wise direct relationship among residues), as in Figure 2.3, the positional fluctuations of the \( \alpha \)-carbons follow a multivariate Gaussian distribution \[19, 20\]. ENM can be analyzed via Normal Mode Analysis, which uses Principle Component Analysis and thus models the system with a single multivariate Gaussian distribution \[21\]. In contrast, our method identifies the multiple conformational sub-states and models the data using a mixture of sparse GGMs. By learning a mixture of GGMs, our method produces a model with the power to represent more complicated distributions than Normal Mode analysis. The use of sparse models improves the interpretability of the output models and reduces the likelihood of overfitting the data.
Figure 2.3: Approximation of a protein by Elastic Network Model. If the $\alpha$-carbons of amino acid residues are represented as masses and pair-wise direct relationships between residues are presented as springs, the fluctuation of the masses from the equilibrium follows a multivariate Gaussian distribution.
Chapter 3

Methods

For large-scale Molecular Dynamics simulations data, a single multivariate Gaussian distribution is not suitable to represent multiple conformational substates of proteins, and an algorithm that can encode more than one Gaussian distribution is needed. In this chapter, we first present traditional Gaussian Mixture Model and its parameter learning, and introduce methods to learn sparse Gaussian Mixture Model for better interpretability. We then discuss methods for scaling to large datasets via coreset sampling. Finally, we discuss methods for increasing the representative power of the method by learning mixtures of semi-parametric models.

3.1 Gaussian Mixture Model

A Gaussian Mixture Model (GMM) $M = \{ (\pi_k, \mu_k, \Sigma_k) \}_{k=1}^K$ is an extension of the Gaussian Graphical Model to represent a mixture of $K$ multivariate Gaussian distributions. Each Gaussian Graphical Model $(\mu_k, \Sigma_k)$ is associated with a weight probability $\pi_k$ that represents the probability of the $k$'th component is chosen, and the sum of the probability satisfies the unitarity condition $\sum_k \pi_k = 1$. For a mixture with $K$ components, the joint probability can be represented as:

$$p(X|M) = \sum_{k=1}^K \pi_k p(X|M_k) = \sum_{k=1}^K \pi_k N(X|\mu_k, \Sigma_k)$$

and the log-likelihood of this model given the data $D = \{ X^{(1)}, \ldots, X^{(N)} \}$ is:

$$L(M|D) = \sum_{i=1}^N \log \left( p(X^{(i)}|M) \right) = \sum_{i=1}^N \log \left( \sum_{k=1}^K \pi_k N(X^{(i)}|\mu_k, \Sigma_k) \right)$$

Unlike the GGM with single Gaussian distribution, GMM does not have a closed-form expression for maximum likelihood estimators. The popular way for parameter learning of GMM is using an Expectation-Maximization (EM) algorithm [22]. The algorithm repeats updating 1) the 'responsibility' of each datapoint to each component and 2) the parameters $M$ in turn. The $t$'th iteration can be expressed as:

$$\gamma_{i,k}^{(t)} = p(k|X^{(i)}, M^{(t)})$$
\[ \pi_k^{(t+1)} = \frac{1}{N} \sum_{i=1}^{N} \gamma_{i,k}^{(t)} \]

\[ \mu_k^{(t+1)} = \frac{\sum_{i=1}^{N} \gamma_{i,k}^{(t)} X^{(i)}}{\sum_{i=1}^{N} \gamma_{i,k}^{(t)}} \]

\[ \Sigma_k^{(t+1)} = \frac{\sum_{i=1}^{N} \gamma_{i,k}^{(t)} (X^{(i)} - \mu_k) (X^{(i)} - \mu_k)^\top}{\sum_{i=1}^{N} \gamma_{i,k}^{(t)}} \]

This algorithm stops when the change of log-likelihood for every step is less than a pre-defined tolerance. As the likelihood function for a mixture of Gaussian distribution is not convex and it might converge to a local optimum, this EM algorithm is usually run many times with random initialization of parameters in \( M \), and the estimator giving the maximum likelihood is chosen. Each iteration of the algorithm takes \( O(NK) \) time, but the number of iteration is highly dependent on the initial parameters of \( M \) and the stopping criterion.

### 3.2 Sparse Gaussian Mixture Model Using BIC

The EM algorithm for Gaussian Mixture Model usually results in a very dense model, where the entries in the precision matrix of each component \( \Theta_k = \Sigma_k^{-1} \) are mostly non-zero. Denser models usually suffer from overfitting problem, and is hard to interpret the relationships and the degree of interactions in the original problem. To reduce the density of GMM, we introduce L1 regularization term to the likelihood of GMM:

\[ \mathcal{L}_p(\Theta | D) = \sum_{i=1}^{N} \log \left( \sum_{k=1}^{c} \pi_k N(X^{(i)}|\mu_k, \Sigma_k) \right) - \lambda \sum_{k=1}^{c} \|\Sigma_k^{-1}\|_1 \quad (3.1) \]

Ruan and coworkers [23] proposed an EM-style algorithm from Eq. 3.1:

\[ \gamma_{i,k}^{(t)} = p(k|X^{(i)}, \Theta^{(t)}) \]

\[ \pi_k^{(t+1)} = \frac{1}{N} \sum_{i=1}^{N} \gamma_{i,k}^{(t)} \]

\[ \mu_k^{(t+1)} = \frac{\sum_{i=1}^{N} \gamma_{i,k}^{(t)} X^{(i)}}{\sum_{i=1}^{N} \gamma_{i,k}^{(t)}} \]

\[ \Sigma_k^{(t+1)} = \arg\max_{\Sigma_k > 0} \left\{ -\log |\Sigma_k| + \text{Tr}(\Sigma_k^{-1} A_k^{(t+1)}) - \lambda \|\Sigma_k^{-1}\|_1 \right\} \quad (3.2) \]

where

\[ A_k^{(t+1)} = \frac{\sum_{i=1}^{N} \gamma_{i,k}^{(t)} (X^{(i)} - \mu_k) (X^{(i)} - \mu_k)^\top}{\sum_{i=1}^{N} \gamma_{i,k}^{(t)}} \]
Algorithm 1: Learning Sparse GMM with L1 Regularization using BIC

**Data:** input data $D$, number of components $K$, reference model $m$

**Result:** $\Sigma_k, \mu_k, \pi_k$ for each component

split data $D$ into training set $D_{\text{train}}$ and cross-validation set $D_{\text{cv}}$.

**for** Every combination of regularization parameters $\lambda_1, ..., \lambda_c$ **do**

randomly initialize $K$ component centers;

**while** not converged within criteria **do**

evaluate each $\gamma^{(t)}_{i,k}$ using $D_{\text{train}}$;

update $\pi^{(t+1)}_k, \mu^{(t+1)}_k, \Sigma^{(t+1)}_k$ with $\lambda_1, ..., \lambda_c$ using $D_{\text{train}}$;

align components with reference model $m$ and relabel components;

end

calculate BIC using $D_{\text{cv}}$;

store the model with best BIC score so far;

end

Eq. 3.2 assumes a single regularization parameter for all components. As we expect significantly different modes of interactions among parts of a protein for each conformational substate, it is more appropriate to introduce different regularization parameter $\lambda_k$ for each component:

$$
\Sigma^{(t+1)}_k = \arg\max_{\Sigma_k > 0} \left\{ -\log |\Sigma_k| + \text{Tr}(\Sigma_k^{-1} A^{(t+1)}_k) - \lambda_k \|\Sigma_k^{-1}\|_1 \right\} 
$$

(3.3)

Solving Eq. 3.3 to get the covariance matrix for each iteration is exactly the same as getting sparse covariance estimation of a multivariate Gaussian, and we can use efficient algorithm including graphical lasso [24]. The regularization parameters $\lambda_k$’s can be selected by cross-validation using the Bayesian Information Criterion (BIC), which penalizes the likelihood with a function that is linear in the number of parameters. Lower BIC values are better than higher ones. The BIC of our model is defined as follows:

$$
\text{BIC}(\Theta) = \mathcal{L}(D|\Theta) - \log(N) \text{df}(\Theta) 
$$

(3.4)

where df is the degree of freedom in the model:

$$
df(\Theta) = \sum_{k=1}^{K} \left[ p + \sum_{i \leq j} I((\Sigma_k^{-1})_{ij} \neq 0) \right] 
$$

Algorithm 1 shows the pseudo-code for learning Sparse GMM with L1 Regularization with BIC using the update equations for each parameters. One subtlety in this algorithm is aligning and relabeling step with respect to the given reference model. Each iteration in this algorithm is an EM algorithm with different regularization parameter $\lambda_k$ for each component $k$, so it needs to preserve the ordering of the learned components for every iteration. As the EM algorithm starts from the randomly initialized parameters in $M$ and the trajectories of those centers until convergence are not predictable, it is necessary to relabel the component indices in each EM step. For relabeling, measuring the differences in weights, means, or covariance matrices alone
is not appropriate as there can be more than one clusters having the similar weights or means. We use Kullback-Leibler divergence (KL-divergence) as a measure of distance among clusters to find the closest reference cluster for each learned cluster. The KL-divergence of two multivariate Gaussian distribution can be evaluated analytically as in Eq. 3.5.

$$D_{KL}(\mathcal{N}_0, \mathcal{N}_1) = \frac{1}{2} \left\{ \text{Tr}(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu_0)^T \Sigma_1^{-1}(\mu_1 - \mu_0) - \text{dim} + \ln \left| \frac{\Sigma_1}{\Sigma_0} \right| \right\}$$ (3.5)

where \text{dim} is the number of variables in the component. For reference model, we can use ground-truth model for synthetic experiments and a model learned from traditional GMM for real datasets.

### 3.3 Sparse Gaussian Mixture Model using Cluster Approximation with Coreset Sampling

As the methods presented in the previous section use all of the given data to refine the cluster centers and the cluster assignments of the datapoints, it is not scalable for large-scale MD simulations. Also, as Algorithm 1 for sparse GMM using BIC tries to solve graphical lasso optimization problem for every EM iteration, it costs significant amount of time for regularization parameter search. Instead of refining cluster for every iteration using EM style algorithm, we can simplify the problem by first approximating the \( K \) cluster centers and then learning a sparse GGM for each cluster, as represented in Figure 3.1. In this section, we introduce an approximation method for identifying the clusters of the data using coreset sampling method [25]. Then, we propose a learning of a mixture of sparse GGMs based on the approximated clusters.

#### 3.3.1 Coreset Sampling for Gaussian Mixture Model

A coreset \( C = \{(\gamma^{(1)}, X'^{(1)}), \ldots, (\gamma^{(M)}, X'^{(M)})\} \) of the given data is a subset of the data with a set of weights representing the degree of contribution to the original data. The negative log-likelihood of \( C \) can be extended by weighting each term in the summation by the corresponding weight \( \gamma^{(j)} \):

$$\mathcal{L}(C|\Theta) = - \sum_{j=1}^{M} \log \sum_{k=1}^{c} \frac{\pi_k \gamma^{(j)}}{\sqrt{(2\pi)^d |\Sigma_k^{-1}|}} \exp \left( -\frac{1}{2} (X'^{(j)} - \mu_k)^T \Sigma_k^{-1} (X'^{(j)} - \mu_k) \right)$$

and the accuracy of the coreset \( C \) can be measured by the difference between the approximated \( \mathcal{L}(C) \) and the original \( \mathcal{L}(D) \).

Instead of the whole term \( \mathcal{L}(C) \), Feldman and coworkers [25] focus on the most important term in the negative log-likelihood. The negative log-likelihood of the data \( D \) can be decomposed to:
Figure 3.1: Diagram of learning process of a sparse Gaussian Mixture Model using cluster approximation with coreset sampling. We use coreset sampling method to approximate the clusters, and use either the original data or the coreset samples to learn a sparse GGM for each cluster.

$$\mathcal{L}(D|\Theta) = - \sum_{i=1}^{N} \log \sum_{k=1}^{c} \frac{\pi_k}{\sqrt{(2\pi)^d|\Sigma_k|}} \exp \left( -\frac{1}{2}(X^{(i)} - \mu_k)^{T}\Sigma_k^{-1}(X^{(i)} - \mu_k) \right)$$

$$= -N \log Z(\Theta) + \sum_{i=1}^{N} \log \sum_{k=1}^{c} \frac{\pi_k}{Z(\Theta)\sqrt{(2\pi)^d|\Sigma_k|}} \exp \left( -\frac{1}{2}(X^{(i)} - \mu_k)^{T}\Sigma_k^{-1}(X^{(i)} - \mu_k) \right)$$

$$= -N \log Z(\Theta) + \phi(D|\Theta)$$

where $Z(\Theta) = \sum_{k=1}^{c} \frac{\pi_k}{\sqrt{(2\pi)^d|\Sigma_k|}}$ is a normalizer. The first term in the equation above is independent of the given data $D$, and the second term $\phi(D|\Theta)$ contains all the dependencies on the data $D$. The coreset sampling algorithm constructs a weighted subset of the given data that can give $(1 + \epsilon)$-approximation of $\phi(D|\Theta)$.

For coreset $C$, this $\phi$ can be extended to:

$$\phi(C|\Theta) = \sum_{j=1}^{M} \log \sum_{k=1}^{c} \frac{\pi_k \gamma(j)}{Z(\Theta)\sqrt{(2\pi)^d|\Sigma_k|}} \exp \left( -\frac{1}{2}(X^{(j)} - \mu_k)^{T}\Sigma_k^{-1}(X^{(j)} - \mu_k) \right)$$

and the coreset algorithm gives $(1 + \epsilon)$-approximation on $\phi(D|\Theta)$ with probability $1 - \delta$:

$$(1 - \epsilon)\phi(D|\Theta) \leq \phi(C|\Theta) \leq (1 + \epsilon)\phi(D|\Theta)$$
Algorithm 2: Coreset Sampling for Gaussian Mixture Model

Data: input data $D$, $\epsilon$, $\delta$, number of component $c$, size of coreset $M$

Result: coreset $C$

$D' := D$, $B := \emptyset$

$\beta = 10dk \ln(1/\delta)$

while $|D'| < \beta$ do

sample set $S$ with size $\beta$ uniformly at random from $D'$;

remove $\lceil|D'|/2\rceil$ points from $|D'|$ that are closest to $S$;

$B = B \cup S$;

end

$B = B \cup D'$;

for each $b \in B$ do

assign $D_b$ that is closest point to $b$ in $D$;

end

for each $b \in B$ do

$x = D_b$;

assign $m(x) = \left[ \frac{5}{|D_b|} + \frac{d(x,B)^2}{\sum_{X' \in D} d(X',B)^2} \right]$

end

pick coreset samples of size $M$ from $D$ with probability proportional to $m(x)$;

for each coreset sample $X^{(j)}$, assign weight $\gamma^{(j)} = \frac{\sum_{X \in D} m(X)}{|C|m(X^{(j)})}$;

The original algorithm by Feldman and coworkers gives a coreset with size independent of the size of the original data and theoretical guarantees. But the size of the constructed coreset is $O(dk^3/\epsilon^2)$, and the implementation of this algorithm sometimes gives coreset with size larger than the original data. For the problem setting with $d = 30$ variables and $k = 5$ components and $\epsilon = 0.1$, the original algorithm samples more than 11 million samples. For the purpose of examining the behavior and the performance of coreset algorithm for sparse GMM models, we modify the algorithm to sample the given number of coreset samples. Algorithm 2 shows the pseudo-code for coreset algorithm we use in our analysis.

### 3.3.2 Learning Sparse Gaussian Mixture Model for Clusters Approximated by Coreset Sampling

For efficient learning of a sparse Gaussian Mixture Model, instead of repeating refining cluster assignments of datapoints and solving optimization problem for parameters, we can construct approximated clusters and learn a single sparse Gaussian distribution for each cluster. From coreset samples $C$ with size of $M$ from the given data, we use weighted version of K-means algorithm for approximating the clusters shown in Algorithm 2.

After approximating the clusters using the weighted K-means algorithm, we learn a mixture of sparse GGMs based on the approximated clusters. For each cluster, we select the datapoints that belong to the cluster, and learn a sparse GGM by using the graphical lasso algorithm. For
Algorithm 3: Weighted K-means Algorithm using Coreset Samples

**Data:** Coreset samples $C = \{(X^{(1)}, \gamma^{(1)}), ..., (X^{(M)}, \gamma^{(M)})\}$, number of clusters $K$, cluster centers $p_1, ..., p_K$

**Result:** Cluster assignments $f : \{X^{(1)}, ..., X^{(M)}\} \rightarrow \{1, ..., K\}$ of coreset samples

Randomly initialize $K$ cluster centers picked from coreset samples $C$;

while not converged within criteria do

   Update mapping $f^{(t)}$ for each $X^{(i)}$ that gives the label of the closest cluster center;
   
   for each cluster $k$ do
   
       Update $p_k^{(t+1)} = \sum_{X^{(i)}: f^{(t)}(X^{(i)})=k} \gamma^{(i)}X^{(i)} / \sum_{X^{(i)}: f^{(t)}(X^{(i)})=k} \gamma^{(i)}$;
   
   end

end

the datapoints to be used in the learning process, we have two options. The first option is that we use the original datapoints that belong to the cluster, and the second is we use the coreset samples of the cluster. The first option gives us more datapoints for learning, and the second option has advantage in runtime cost as it is using a smaller set of samples. Both options are explained in this section, and their behaviors are analyzed in Chapter 4.

**Learning Sparse GMM from Original Datapoints Divided in Approximated Clusters**

We first assign the original datapoints to the approximated clusters, and then learn a sparse GMM for each cluster. As we reduce the original problem of learning Gaussian mixture to separate problem of learning a single Gaussian distribution, we can use graphical lasso algorithm for each cluster. The pseudo-code for this algorithm is in Algorithm 4.

For model selection, we can use exhaustive search on candidates of the regularization parameter $\lambda_k$’s, but we use simple heuristics to reduce the runtime cost for model selection. The optimal regularization $\lambda_k$ we want is small enough to capture the relationship in precision matrix $\Sigma_k^{-1}$ but large enough to give sparse $\Sigma_k^{-1}$. For datapoints matrix $D_k$ in cluster $k$, if we randomly shuffle the columns of $D_k$, then all the pairwise correlations between the rows and the columns are diminished. Thus, to find the optimal $\lambda_k$, we randomly shuffle the columns of $D_k$ and apply graphical lasso with parameter candidates of $\lambda_k$ from smallest to largest until the learned precision matrix is diagonal. For more accurate choice of $\lambda_k$, we can repeat this selection process and take the average value for our final optimal $\lambda_k$. Also, this algorithm can be easily parallelized for each cluster $k$ and for each candidate for $\lambda_k$.

**Learning Sparse GMM from Coreset samples Divided in Approximated Clusters**

To further exploit the coreset sampling approximation, we can learn sparse Gaussian for each clusters using coreset samples instead of using original datapoints. By reducing the number of samples for each clusters, we can reduce the time cost for learning a model for each cluster while it might sacrifice some degree of accuracy of the model. For this method, we modify Algorithm 4 to use the coreset samples $C$ instead of original input data $D$, and the rest of the algorithm is
Algorithm 4: Learning Sparse GMM from Datapoints Divided to Approximated Clusters

Data: original input data $D$, cluster centers $p_1, \ldots, p_K$
Result: sparse Gaussian Mixture Model with $K$ clusters

for $X^{(i)} \in D$ do
    Assign the cluster assignment $c(X^{(i)}) = \text{label of the closest cluster to } X^{(i)}$
end

for each cluster $k$ do
    $D_k = \{ X^{(i)} : c(X^{(i)}) = k \}$
    $D'_k = \text{randomly shuffled columns of } D_k$
    for each candidate for regularization parameter $\lambda_k$ do
        Apply graphical lasso with $\lambda_k$ for $D'_k$
        if learned $\Sigma_k^{-1}$ is diagonal matrix then
            break
        end
    end
    Apply graphical lasso with the last $\lambda_k$ for $D_k$
end

The method we discussed in this section mainly consists of two parts. The first part is to approximate the cluster centers and the assignments of the datapoints to the clusters where we use coreset sampling to accelerate the process. The second part is to learn a sparse GGM for each cluster, and we use either the original datapoints for larger size of samples or the coreset samples for further reduce the runtime cost.

3.4 Sparse Nonparanormal Mixture Model using Cluster Approximation

The models introduced in the previous sections assume that fluctuations of proteins in each conformational substate follow Gaussian distribution. This assumption make the problem of parameter learning and inference easier, but it is limiting the representational power of the model. Liu and coworkers [26] proposed a semi-parametric method to extend standard Gaussian distribution to cover more families of distributions with richer representational power. A random vector $X = (X_1, X_2, \ldots, X_P)$ follows nonparanormal distribution if there exists a set of functions $\{f\}_1^P$ such that $f(X) = (f_1(X_1), f_2(X_2), \ldots, f_P(X_P))$ follows a Gaussian distribution $N(\mu, \Sigma)$, and we write $X$ follows a nonparanormal distribution $NPM(\mu, \Sigma, \{f\}_1^P)$. By the flexibility of the choices of $\{f\}_1^P$, the family of nonparanormal distributions has richer representational power than Gaussian distributions.

Assuming that the given dataset $D = \{X_1, X_2, \ldots, X_N\}$ follows a nonparanormal distribution, the estimator of $\{f\}_1^P$ is given by:
\[ \hat{f}_p(X_p) = \mu_p + \sigma_p \tilde{h}_p(X_p) \]

where \( \hat{\mu}_p \) and \( \hat{\sigma}_p \) are empirical mean and covariance for normalization:

\[
\hat{\mu}_p = \frac{1}{N} \sum_{i=1}^{N} X_p^{(i)} \\
\hat{\sigma}_p = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (X_p^{(i)} - \hat{\mu}_p)^2}
\]

and \( \tilde{h}_p \) is inverse standard Gaussian distribution function (quantile function) applied to the estimated empirical distribution of \( X_p \):

\[
\tilde{h}_p(X_p) = \Phi^{-1}(\hat{F}_p(X_p))
\]

For high-dimensional problem settings, the proposed estimator for \( F_p \) is a \textit{Winsorized} estimator to truncate the outliers for the extreme values \( X_p \):

\[
\hat{F}_p(X_p) = \begin{cases} 
\delta_N & \text{if } \hat{F}_p(X_p) < \delta_N \\
\hat{F}_p(X_p) & \text{if } \delta_N \leq \hat{F}_p(X_p) \leq 1 - \delta_N \\
1 - \delta_N & \text{if } \hat{F}_p(X_p) > 1 - \delta_N 
\end{cases}
\]  

(3.6)

where \( \hat{F}_p(X_p) \) is a marginal empirical distribution \( \hat{F}_p(x) = \frac{1}{N} \sum_{i=1}^{N} I(X_p^{(i)} \leq x) \). The choice of \( \delta_N \) gives the bias-variance trade-off of the Winsorized estimator, and we follow the suggested \( \delta_N \) in the paper \( \delta_N = \frac{1}{(4N^{1/4}\sqrt{\pi \log N})} \).

Using this proposed semi-parametric method, we can modify the Algorithm 4 to learn non-paranormal distributions after dividing datapoints to the approximated clusters. The modified algorithm is in Algorithm 5.

Another advantage of the Nonparanormal Mixture Model is that it can be used in the further prediction studies. The transformation functions \( \{f_p\}_p^P \) do not involve projecting the data to another space with smaller dimensions, and they are invertible. For predicting the missing values given other coordinates, we can do inference on the missing values from the learned Gaussian Mixture Model given the evidence, and apply the inverse transformation functions to get the values in the original problem space.

### 3.5 Summary

In this chapter, we introduced methods to learn a sparse Gaussian Mixture Model from the data along with the traditional method. The method for a sparse GMM with L1 regularization with BIC score is an extension of the traditional EM-style learning method, and it involves solving a graphical lasso optimization for every step in the EM-style algorithm. For efficiency in the learning process, we divide the problem into identifying the clusters and learning the sparse model for each cluster separately. For the identification of clusters, we use an approximation
Algorithm 5: Learning Sparse Nonparanormal Mixture Model from Datapoints Divided to Approximated Clusters

<table>
<thead>
<tr>
<th>Data: original input data $D$, cluster centers $p_1, \ldots, p_K$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result: sparse Nonparanormal Mixture Model with $c$ clusters</td>
</tr>
<tr>
<td>for $x_i \in D$ do</td>
</tr>
<tr>
<td>Assign the cluster assignment $c(x_i) = \text{label of the closest cluster}$</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>for each cluster $k$ do</td>
</tr>
<tr>
<td>$D_k = {x_i: c(x_i) = k}$</td>
</tr>
<tr>
<td>Estimate transformation function ${f_p}_{p=1}^{P}$ using Winsorized estimator</td>
</tr>
<tr>
<td>Transform the data: $T_k = f(D_k)$</td>
</tr>
<tr>
<td>$T'_k = \text{randomly shuffled columns of } T_k$</td>
</tr>
<tr>
<td>for each candidate for regularization parameter $\lambda_k$ do</td>
</tr>
<tr>
<td>Apply graphical lasso with $\lambda_k$ for $T'_k$</td>
</tr>
<tr>
<td>if learned $\Sigma^{-1}_k$ is diagonal matrix then</td>
</tr>
<tr>
<td>break</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>Apply graphical lasso with the last $\lambda_k$ for $T_k$</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

method using coreset sampling and weighted K-mean algorithm. For learning a sparse GGM, we use either all of the original samples or the coreset samples as a training set with heuristic method to find the optimal regularization parameter for graphical lasso algorithm. Finally, we extend our method to a semi-parametric method using nonparanormal distributions while the transformation functions involved in this method requires extra space complexity.
Chapter 4

Experiments and Results

In this chapter, we present the results of our experiments using methods in the previous chapter on various performance metrics compared to the baseline methods. Our experiments are on both synthetic data and engrailed homeodomain protein data generated by Molecular Dynamics simulations. We use traditional Gaussian Mixture Model (without sparsity constraints) for our baseline method and analyze the performances varying the problem settings including number of mixtures, dimensions for each mixture, and number of sample datapoints.

4.1 Performance Metrics

In this section, we list the performance metrics that are used to evaluate our method, along with the baseline approach. The performance metrics include both quantitative and qualitative measures to compare the characteristics of the learned models. Some of the metrics are against the test dataset, and the others are against the true model in the case we know the true model.

4.1.1 Performance Metrics on Test Dataset

After learning the model with training set, we measure performances on a separate test dataset using the following metrics:

- Negative log-likelihood of test dataset \( D_{test} \).

\[
\text{LL} = - \sum_{X \in D_{test}} \log p(X|M)
\]

- Average imputation error of test dataset \( D_{test} \).

Imputation is a process of predicting missing values, when the model is conditioned on partial observations. The average accuracy of imputed values is an indication of the quality of the model. We compute the average imputation error by iteratively conditioning on \( n - 1 \) variables and imputing the value of the remaining variables:

\[
\text{Error}_{\text{imputation}} = \frac{1}{P|D_{test}|} \sum_{X \in D_{test}} \sum_{j=p}^{P} |X_j - V_j|
\]
where $V_k$ is the predicted value on $j$’th coordinate from the learned model given the other coordinates of each datapoint $X$ as evidence. The predicted value $V_j$ is sampled from the learned model and this inference problem is analytically tractable with our choice of Gaussian distribution for each component. The inference is done in the following process:

1. Evaluate the probability of each component (responsibility) $p_1, p_2, ..., p_K$ given the evidence using the each component’s marginal distribution.
2. Multiply the probability of each component’s with the component’s weight.
3. Sample the component index $k$ with the probability according the the evaluated probabilities for components $p_1, p_2, ..., p_K$.
4. Sample the predicted value for missing coordinate $V_j$ from the conditional distribution of $k$’th component given the evidence.

### 4.1.2 Performance Metrics against the True Model

In synthetic experiments, we know the true model of the generated data, so we can measure the performance by comparing the learned model to the true model. For the comparison, we use the following metrics against the true model:

- **Edge recovery accuracy.**
  One of the ultimate goals of our methods is to identify the pair-wise direct interaction between the variables (amino acid residues in proteins). For each component, we measure the accuracy in recovering edges.
  \[
  \text{Accuracy} = \frac{\text{true positive} + \text{true negative}}{\text{total possible number of edges}}
  \]

- **Edge recovery F1 score.**
  When the true model has sparse precision matrices, accuracy might not be the best metric for edge recovery as methods producing any fairly sparse models could achieve high accuracy. Thus, we measure the F1 score, which is a harmonic mean of precision and recall.
  \[
  \text{F1 score} = \frac{2 \times \text{true positive}}{2 \times \text{true positive} + \text{false positive} + \text{false negative}}
  \]

### 4.2 Synthetic Experiments

In this section, we present the results of our methods on synthetic experiments to analyze the performances of three sparse Gaussian Mixture Models and the sparse Nonparanormal Mixture Model introduced in the previous chapter. Specifically, the models are denoted as following:

- **$S_{BIC}$**: sparse GMM learned with regularization parameter search with BIC
- **$S_{core}$**: sparse GMM learned from all datapoints using coreset-approximated clusters
- **$S_{core2}$**: sparse GMM learned from coreset subsamples using coreset-approximated clusters
- **$S_{npn}$**: sparse Nonparanormal Mixture Model learned from all datapoints using coreset-approximated clusters
4.2.1 Generation of Synthetic Data

For generating a mixture of Gaussian distribution, we generate mean vectors and precision matrices for each component, and the weights for clusters. Mean vectors are generated by sampling from the interval $[0, 5)$ uniformly at random for each coordinate. For precision matrices, we generate real positive definite matrix with specified edge-density. Weights for components are chosen from 2% to 40%. The training dataset and test dataset are generated from the true model by choosing a component following the weights and sampling from the selected multivariate Gaussian distribution for each datapoint.

4.2.2 Synthetic Experiments on Sparse Gaussian Mixture Models Using Coreset Approximation

In this section, we compare $S_{BIC}$, $S_{core}$, and $S_{core2}$ with the baseline model, which is a traditional GMM, denoted as $M_{gmm}$.

Experiments on Synthetic Data with Varying Number of Variables

We conduct experiments on synthetic data varying the number of variables for the mixture of Gaussian distributions, 10, 30, and 50 variables. The ground-truth models have 5 components in the mixture and 10,000 samples are used for learning. The edge-density in each component is 30% for all models. To measure the performance, 100 samples are used as a test dataset and the learning is repeated 10 times to calculate the average and the 95% confidence interval. For coreset approximation of clusters, we use $\delta = 0.1$ and $\epsilon = 0.1$ with coreset size $M = 1000$ (10% of samples).

![Figure 4.1: Negative Log-Likelihood of learned models using $M_{gmm}$, $S_{bic}$, $S_{core}$, $S_{core2}$ methods.](image)
Results using log-likelihood metrics  Figure 4.1 shows the negative log-likelihood using $M_{\text{gmm}}$, $S_{\text{BIC}}$, $S_{\text{core}}$, $S_{\text{core2}}$ methods on the three different datasets. Overall, the negative log-likelihood of all models increases as the number of variables increases, and this is because the number of parameters to learn increases proportionally to the square of the number of variables for covariance matrix of each component. Models learned by $M_{\text{gmm}}$ show the best likelihood of the test data. However, the differences between $S_{\text{bic}}$ and $S_{\text{core}}$ and the baseline method are small. The negative log-likelihood of $S_{\text{bic}}$ is 0.48% to 4.8% worse than $M_{\text{gmm}}$, and the likelihood of $S_{\text{core}}$ is 0.48% to 5.3% worse than $M_{\text{gmm}}$. We note that the confidence intervals of the likelihoods of $S_{\text{bic}}$ and $S_{\text{core}}$ overlap those of the baseline methods for datasets with 10 and 30 variables. $S_{\text{core2}}$ method shows the worst performance with likelihood 3.2% to 26% worse than the baseline method. We conclude that $S_{\text{bic}}$ and $S_{\text{core}}$ are nearly equivalent to the baseline model, in terms of likelihood.

Results using imputation error metrics  Figure 4.2 shows the average imputation error of test set using $M_{\text{gmm}}$, $S_{\text{bic}}$, $S_{\text{core}}$, $S_{\text{core2}}$ methods on the three different datasets. For all experiments, the imputation errors of $M_{\text{gmm}}$ is the smallest. The difference between the baseline model and $S_{\text{bic}}$ and $S_{\text{core}}$ are small, and sometimes insignificant. The imputation error of $S_{\text{bic}}$ is 2.2% to 6.6% higher than $M_{\text{gmm}}$, and that of $S_{\text{core}}$ is 2.2% to 7.9% higher than $M_{\text{gmm}}$. Like the likelihood, The $S_{\text{bic}}$ and $S_{\text{core}}$ imputations are within the confidence intervals of the baseline model for the 10 and 30 variable datasets. $S_{\text{core2}}$ shows the highest imputation error with 8.6% to 22% larger than $M_{\text{gmm}}$ method. Unlike the log-likelihood, there was no significant trend in the imputation error with the increased number of variables. We conclude that $S_{\text{bic}}$ and $S_{\text{core}}$ are nearly equivalent to the baseline model, in terms of imputation error.

Results using edge recovery-related metrics  Figure 4.3 and 4.4 show the accuracy and F1 score on edge recovery in each learned clusters using the different methods. As expected, the sparse models show significant improvements over the baseline model, in terms of F1 score.
Figure 4.3: Edge recovery accuracy of learned models using $M_{gmm}$, $S_{bic}$, $S_{core}$, $S_{core2}$ methods.

Figure 4.4: Edge recovery F1 score of learned models using $M_{gmm}$, $S_{bic}$, $S_{core}$, $S_{core2}$ methods.
$S_{\text{core}}$ shows the best accuracy (85% to 90%) and F1 score (0.72 to 0.86). $S_{\text{bic}}$ shows accuracy from 79% to 87% and F1 score from 0.67 to 0.74. While the accuracy of $S_{\text{core}2}$ method shows accuracy higher than 70% for all experiments, the F1 score for 30 and 50 variable are less than 0.50. We conclude that $S_{\text{bic}}$ and $S_{\text{core}}$ are superior to $S_{\text{core}2}$ and the baseline model in terms of edge recovery.

![Recovered Edges in Precision Matrices](image)

Figure 4.5: Recovered edges in precision matrices of learned models using $M_{\text{gmm}}$, $S_{\text{bic}}$, $S_{\text{core}}$, $S_{\text{core}2}$ methods.

**Recovered edges from the methods** Figure 4.5 and 4.6 show the edges recovered in the models on 10 variable datasets and 30 variable datasets using $M_{\text{gmm}}$, $S_{\text{bic}}$, $S_{\text{core}}$, and $S_{\text{core}2}$ methods. $M_{\text{gmm}}$ model learns a fully connected for all components, and this explains the worst accuracy and F1 score in edge recovery. $S_{\text{core}}$ and $S_{\text{core}2}$ methods recover many edges correctly but the learned models have sparser precision matrices than the true model.

**Thresholding-based method for sparse models** As the low accuracy and F1 score in edge recovery of $M_{\text{gmm}}$ model is due to the full precision matrices, one possible method to increase the recovery performance is to threshold the absolute values of the entries in the precision matrices learned from $M_{\text{gmm}}$ model, which is treating the small entries as a noise in the parameter learning process. Figure 4.7 shows the accuracy and F1 score in edge recovery under various threshold values. As the smallest threshold (0.0001) gives the full non-zero precision matrices and the largest threshold (1.0) gives the diagonal precision matrices, the accuracy changes from 0.3 to 0.7. The accuracy under all threshold values are lower than 85% of $S_{\text{core}}$ method. The best F1 score is achieved at threshold value 0.01, but this is still less than F1 score 0.72 by $S_{\text{core}}$ method. Thus, we conclude that the sparse methods are superior to any simple thresholding-based method to deriving sparse models.
Figure 4.6: Recovered edges in precision matrices of learned models using $M_{gmm}$, $S_{bic}$, $S_{core}$, $S_{core2}$ methods.

Figure 4.7: Edge recovery accuracy and F1 score of thresholded precision matrices learned using $M_{gmm}$ method.
Results using runtime metrics

Table 4.1 shows the runtime costs in learning models using the different methods on datasets. The $S_{bic}$ algorithm takes significantly larger amount of time for learning mainly for two reasons. First, the Algorithm 1 searches for the optimal regularization parameters by exhaustively attempting all possible combinations of parameters. So, $S_{bic}$ method takes $O(l^K)$ number of iterations where $l$ is the number of candidates for regularization parameters and $K$ is the number of components. Secondly, each iteration in the algorithm involves solving the optimization problem in estimating $\Sigma_k^{(t+1)}$ using graphical lasso algorithm. While graphical lasso is an efficient algorithm for single optimization problem, but nesting it in EM-style algorithm takes significant number of iterations for single iteration for searching regularization parameters. $S_{core}$ method shows about 5 to 16 times of runtime cost compared to $M_{gmm}$, and this is due to the coreset algorithm for approximating clusters and the search for the optimal regularization parameters. As in Algorithm 4, we use heuristic method for the search which involves searching smallest regularization parameter that gives diagonal precision matrices in the column-shuffled data. As the method $S_{bic}$ takes hours to several days for learning, we exclude $S_{bic}$ for further experiments and analyses. We conclude that $S_{core}$ is the best method among those we considered. It has similar likelihood and imputation errors as the baseline method, but performs much better in edge recovery. This improvement comes at a modest increase in runtime, which could potentially be mitigated by a parallel implementation of the algorithm.

Experiments on Synthetic Data with Varying Number of Components

We conduct experiments on synthetic data varying the number of components in the Gaussian mixtures, 3, 5, 10 components. The ground-truth models have 30 variables in the mixture and 10,000 samples are used for learning. The edge density in each component is 30% for all models. For all learning methods, 100 samples are used as a test dataset and the learning is repeated 10 times to calculate average and 95% confidence interval. For coreset approximation of clusters, we use $\delta = 0.1$ and $\epsilon = 0.1$ with coreset size $M = 1000$ (10% of samples).

Results using log-likelihood and imputation error metrics

Figure 4.8 and 4.9 show the negative log-likelihood and average imputation error of learned models using the methods on three different test set. Generally speaking, performance degrades as the number of components increases. As in the previous experiment, the log-likelihood and imputation error of $S_{core}$ is larger than $M_{gmm}$ for all experiments, and $S_{core2}$ shows the worst performance. The log-likelihood of $S_{core}$ is 3.2% to 13% worse than the baseline method, $M_{gmm}$, and $S_{core2}$ is 20% to 23% worse than $M_{gmm}$. Imputation error of $S_{core}$ is 1% to 6.3% worse than $M_{gmm}$, and $S_{core2}$ is 20% to 23% worse than $M_{gmm}$.
Figure 4.8: Negative log-likelihood, imputation error, and edge recovery rate of learned models using $M_{gmm}, S_{core}, S_{core2}$ methods.

Figure 4.9: Average imputation errors of learned models using $M_{gmm}, S_{core}, S_{core2}$ methods.
Figure 4.10: Edge recovery accuracy in precision matrices of learned models using $M_{gmm}$, $S_{core}$, $S_{core2}$ methods.

Figure 4.11: Edge recovery F1 score in precision matrices of learned models using $M_{gmm}$, $S_{core}$, $S_{core2}$ methods.
worse than $M_{gmm}$.

**Results using edge recovery-related metrics**  Figure 4.10 and 4.11 show the edge recovery accuracy and F1 score of the models, and $S_{core}$ shows significant improvements over the $M_{gmm}$ model in both accuracy and F1 score. $S_{core}$ performs best on datasets with 3 components as number of samples for each component increases as the number of components decreases under fixed size of total samples. The baseline method $M_{gmm}$ learns fully connected or densely connected model (edge density > 97% in precision matrices) and this explains the low accuracy and F1 score. $S_{core2}$ shows significant improvements over $M_{gmm}$, but the F1 score in 5 and 10 component datasets do not show any improvement or even less than $M_{gmm}$. These results further support the claim that $S_{core}$ is the best alternative to the baseline method, although it is perhaps best suited to distributions with about 5 or fewer mixing components.

**Experiments on Synthetic Data with Varying Number of Training Samples**

We conduct experiments on synthetic data varying the size of the training set, from 5,000 to 100,000. The ground-truth models have 5 components, 30 variables in each mixture and the edge density in each component is 30%. For all learning methods, 100 samples are used as a test dataset and the learning is repeated 10 times to calculate average and 95% confidence interval. For coreset approximation of clusters, we use $\delta = 0.1$ and $\epsilon = 0.1$ with coreset size $M = 10\%$ of the size of the samples.

![Figure 4.12: Negative log-likelihood, imputation error, and edge recovery rate of learned models using $M_{gmm}$, $S_{core}$, $S_{core2}$ methods.](image)

**Results using log-likelihood and imputation error metrics**  Figure 4.12 and 4.13 show the negative log-likelihood and average imputation error of the models learned by $M_{gmm}$, $S_{core}$,
Figure 4.13: Average imputation errors of learned models using $M_{gmm}$, $S_{core}$, $S_{core2}$ methods. $S_{core2}$ methods. Overall, as the size of training samples and coreset size increases, the log-likelihood of all models improves. The log-likelihood of $S_{core}$ improves from 7.1% worse to 1.1% worse than $M_{gmm}$ method, and $S_{core2}$ improves from 26% worse to 9.2% worse than $M_{gmm}$. The imputation errors of $M_{gmm}$ are all within the confidence interval and do not show any improvements, but imputation errors of $S_{core}$ and $S_{core2}$ show improvements with increase number of samples. $S_{core}$ method improves from 9.8% worse to 2.6% worse than $M_{gmm}$ method, and $S_{core2}$ improves from 25% worse to 11% worse than $M_{gmm}$.

Figure 4.14: Edge recovery accuracy in precision matrices of learned models using $M_{gmm}$, $S_{core}$, $S_{core2}$ methods.
Figure 4.15: Edge recovery F1 score in precision matrices of learned models using $M_{gmm}$, $S_{core}$, $S_{core2}$ methods.

Results using edge recovery-related metrics Figure 4.14 and 4.15 show the edge recovery accuracy and F1 score of $M_{gmm}$, $S_{core}$, and $S_{core2}$ methods. As in the previous experiments, the edge recovery accuracy and F1 score of $S_{core}$ method shows significant improvements over $M_{gmm}$ method, and both accuracy and F1 score of $S_{core}$ is higher than $S_{core2}$ in all experiments. As the size of the training samples increases, $S_{core}$ shows improvements in both accuracy (from 83% to 89%) and F1 score (from 0.67 to 0.83), but $M_{gmm}$ method do not show any difference among the experiments. $S_{core2}$ shows improvements in accuracy for all sample sizes, but F1 score shows improvements only in sample size greater than 100,000. These results are consistent with the previous results. Namely, that $S_{core}$ is the best alternative to $M_{gmm}$.

Experiments on Synthetic Data with Varying Edge Density in Precision Matrices

We conduct experiments on synthetic data varying the edge density in precision matrices from 10% to 50%. The ground-truth models have 30 variables in 5 components, and 10,000 samples are used for learning. For all learning methods, 100 samples are used as a test dataset and the learning is repeated 10 times to calculate average and 95% confidence interval. For coreset approximation of clusters, we use $\delta = 0.1$ and $\epsilon = 0.1$ with coreset size $M = 1000$ (10% of samples).

Results using log-likelihood and imputation error metrics Figure 4.16 and 4.17 show the negative log-likelihood and imputation error of the models learned by $M_{gmm}$, $S_{core}$, and $S_{core2}$ methods. As previous experiments, the log-likelihood and imputation error of $S_{core}$ and $S_{core2}$ are worse than $M_{gmm}$. As the edge density of the true model increases, the relative difference in both metric gets bigger for both $S_{core}$ and $S_{core2}$ models.
Figure 4.16: Negative log-likelihood, imputation error, and edge recovery rate of learned models using $M_{gmm}$, $S_{core}$, $S_{core2}$ methods.

Figure 4.17: Average imputation errors of learned models using $M_{gmm}$, $S_{core}$, $S_{core2}$ methods.
Figure 4.18: Edge recovery accuracy in precision matrices of learned models using $M_{gmm}$, $S_{core}$, $S_{core2}$ methods.

Figure 4.19: Edge recovery F1 score in precision matrices of learned models using $M_{gmm}$, $S_{core}$, $S_{core2}$ methods.
Results using edge recovery-related metrics  Figure 4.18 and 4.19 show the edge recovery accuracy and F1 score of $M_{gmm}$, $S_{core}$, $S_{core}^2$ methods. As the edge density of the true model increases, accuracy of $S_{core}$ method drops from 96% to 72%, and the F1 score of $S_{core}$ drops from 0.84 to 0.64. $S_{core}$ method shows better performance when the underlying Gaussian distribution for each component has sparser distribution and this explains the best performance in log-likelihood and imputation error of 10% edge density model. The $S_{core}^2$ method shows about the same or worse F1 score as $M_{gmm}$ method in datasets with 30% and 50% edge densities. For $M_{gmm}$ method, both edge recovery accuracy and F1 score show improvements with higher density datasets, but this is because $M_{gmm}$ learns almost fully connected models and the accuracy increases along with the density of the true models. As expected, these results demonstrate that the sparse methods are best used for distributions that are truly sparse.

Experiments on Synthetic Data with Varying Size for Coreset

We conduct experiments on synthetic data varying the size of the coreset in cluster approximation in coreset algorithm, from 100 to 700 coreset samples. The ground-truth models have 10 components in the mixture and 10,000 samples are used for learning. The edge-density in each component is 30%. For all learning methods, 100 samples are used as a test dataset and the learning is repeated 10 times to calculate average and 95% confidence interval. For coreset approximation of clusters, we use $\delta = 0.1$ and $\epsilon = 0.1$. We use the same number of sub-sample size for $S_{unif}$ baseline method, where it samples the same size with $S_{core}$ and $S_{core}^2$ for cluster approximation but in uniformly at random among the original data and uses all samples for learning each Gaussian distribution for each cluster.

Figure 4.20: Negative log-likelihood of learned models using $S_{core}$, $S_{core}^2$ and $S_{unif}$ methods.

Figure 4.20 to 4.23 show the performances of $S_{core}$, $S_{core}^2$, and $S_{unif}$ methods. For all experiments, negative log-likelihoods of $S_{core}^2$ are worse than $S_{core}$ and the baseline method $S_{unif}$. The log-likelihood and imputation error improve as the size of coreset samples increases, but did
Figure 4.21: Imputation errors of learned models using $S_{core}$, $S_{core2}$ and $S_{unif}$ methods.

Figure 4.22: Edge recovery rate in precision matrices of learned models using $S_{core}$, $S_{core2}$ and $S_{unif}$ methods.
Figure 4.23: Edge recovery F1 score in precision matrices of learned models using $S_{\text{core}}$, $S_{\text{core}2}$ and $S_{\text{unif}}$ methods.

Figure 4.24: Runtime cost for parameter learning using $S_{\text{core}}$, $S_{\text{core}2}$ and $S_{\text{unif}}$ methods.
not perform better when the most samples are used for the learning. The baseline method \( S_{unif} \) shows similar average likelihood to \( S_{core} \) method, but the variance is larger than \( S_{core} \) and the variance gets smaller as the size of the coreset increases. This is due to the uniformly random sampling of original data for cluster approximation of \( S_{unif} \) where the subsamples can misrepresent the heavily or lightly weighted clusters of the true distribution. For both accuracy and F1 score, \( S_{core2} \) method performs worse than both \( S_{core} \) and \( S_{unif} \) methods while the F1 score improves as the size of the coreset increases. The edge recovery accuracy and F1 score of \( S_{core} \) with less than or equal to 500 coreset subsamples (5% of the original data) shows high variance, but both metrics are stabilized with more than 1,000 subsamples (10% or the original data). The accuracy and F1 score of \( S_{unif} \) method shows similar performance with \( S_{core} \), but it still shows high variance with the most subsamples used (50% of the original data). The runtime cost of the three methods is shown in 4.24. As \( S_{core} \) and \( S_{unif} \) uses the whole dataset for parameter learning, the runtime cost of the two methods takes 3 to 6 times more than \( S_{core2} \). As \( S_{core} \) uses additional coreset algorithm for coreset sampling, it takes 12% to 15% longer than \( S_{unif} \) method. These results suggest that \( S_{core} \) is superior to \( S_{unif} \) because it tends to have lower variance in the metrics we considered. However, this comes at the cost of a modest increase in runtime.

### 4.2.3 Synthetic Experiments on Sparse Nonparanormal Mixture Models Using Coreset Approximation

Our final experiments on synthetic data examine the impact of the use of sparse Nonparanormal Mixture Model, \( S_{npn} \). We compare the performance of \( S_{npn} \) with the traditional Gaussian Mixture Model \( M_{gmm} \) and the previously proposed \( S_{core} \) as baseline methods.

#### Experiments on Synthetic Data with Varying Number of Variables

We conduct experiments on synthetic data varying the number of variables for the mixture of Gaussian distributions, 10, 30, and 50 variables. The ground-truth models have 5 components in the mixture and 10,000 samples are used for learning. The edge-density in each component is 30% for all models. To measure the performance, 100 samples are used as a test dataset and the learning is repeated 10 times to calculate the average and the 95% confidence interval. For coreset approximation of clusters, we use \( \delta = 0.1 \) and \( \epsilon = 0.1 \) with coreset size \( M = 1000 \) (10% of samples). For Winsorization parameter, we use \( \delta_n = \frac{1}{4n^{1/4}\sqrt{\pi}\log n} \) with \( n = 10000 \) as suggested [26].

#### Results using log-likelihood and imputation error metrics

Figure 4.25 and 4.26 show the negative log-likelihood and the average imputation error of learned models using \( M_{gmm} \), \( S_{core} \), \( S_{npn} \) methods on the three different datasets. For log-likelihood, \( S_{npn} \) shows better performance than \( S_{core} \) method in all experiments, and shows better performance than \( M_{gmm} \) in 10 variable dataset. For imputation errors, \( S_{npn} \) method shows smaller errors than \( S_{core} \) and larger than \( M_{gmm} \) for all experiments. \( S_{core} \) method shows 2.2% to 7.9% larger imputation error than \( M_{gmm} \), and \( S_{npn} \) shows 1.5% to 7.2% larger than \( M_{gmm} \). Thus, \( S_{npn} \) shows a slight improvement over \( S_{core} \).
Figure 4.25: Negative Log-Likelihood of learned models using $M_{gmm}$, $S_{core}$, $S_{nqn}$ methods.

Figure 4.26: Average imputation errors of learned models using $M_{gmm}$, $S_{core}$, $S_{nqn}$ methods.
Figure 4.27: Edge recovery accuracy of learned models using $M_{gmm}$, $S_{core}$, $S_{npu}$ methods.

Figure 4.28: Edge recovery F1 score of learned models using $M_{gmm}$, $S_{core}$, $S_{npu}$ methods.
Results using edge recovery-related metrics  Figure 4.27 and 4.28 show the accuracy and F1 score on edge recovery in each learned clusters using the different methods. For all datasets, $S_{npn}$ shows significant improvements over $M_{gmm}$ in both accuracy and F1 score, but its performance is lower than or even to $S_{core}$ method. While the true model is still a mixture of Gaussian distribution, the increased representational power of $S_{npn}$ achieves the better performance in log-likelihood and imputation error, but as $M_{gmm}$ still has a maximum degree of freedom in parameters, $S_{npn}$ shows lower performance than $M_{gmm}$.

4.3 Learning Sparse Mixture Models for Conformational Substates of Engrailed Homeodomain

We apply our methods of learning sparse Gaussian Mixture Model and sparse Nonparanormal Mixture Model on the simulation data of engrailed homeodomain. Homeodomain, a protein with 60 amino acids shown in 4.29 is a DNA-binding sub-structure of a larger protein that regulates specific genes as a transcription factors of the genes, and is shared across all the analyzed species under kingdom animalia [27]. Engrailed homeodomain is a protein of interest for Molecular Dynamics simulation and its application in research due to its fastest folding and unfolding rate, which is predicted and corroborated in experiments [28, 29]. The engrailed homeodomain is expected to have significant fluctuations at equilibrium and to visit many conformational substates.

The data The Molecular Dynamics simulation is performed on ANTON, a machine specifically designed for long time-scale simulation, [8]. The simulated homeodomain has 46 amino acids and the simulation is under 350 degrees Kelvin. The total time-scale of the simulation is for 50 microseconds with more than 500,000 frames, each of which contains 3-dimensional coordinates of $\alpha$-carbons of the protein. To eliminate the conformational changes due to translations and rotations, all of the frames are aligned to the average positions of $\alpha$-carbons, and the displacement of each $\alpha$-carbon of each frame from the averaged position is evaluated for our studies.

4.3.1 Learning Sparse Gaussian Mixture Model for Conformational Substates of Engrailed Homeodomain

We use a sparse Gaussian Mixture Model with coreset approximation ($S_{core}$) for learning substates of engrailed homeodomain. For baseline methods, we use the traditional GMM ($M_{gmm}$) and a sparse GMM using cluster approximation using uniformly sampled subsamples ($S_{uni}$). 500,000 frames are used training dataset, and 1,000 frames are used as a test dataset. For coreset approximation of clusters, we use $\delta = 0.1, \epsilon = 0.1$, and coreset size $M = 50000$, 10% of the total number of samples.

Selection of the number of components using BIC score  Figure 4.30 and 4.31 show the negative log-likelihood and the average imputation error of $M_{gmm}$, $S_{core}$, and $S_{uni}$ with the number of components from 3 to 10. Overall, the models with more components show better likelihood and imputation error, and this is mainly due to the increased number of parameters when more
Figure 4.29: Engrailed homeodomains of *Drosophila melanogaster* (1DU0) bound to DNA double helix.

Figure 4.30: Negative log-likelihood of learned models using $M_{gmm}$, $S_{core}$, $S_{unif}$ methods on engrailed homeodomain.
Figure 4.31: Average imputation error of learned models using $M_{gmm}$, $S_{core}$, $S_{unif}$ methods on engrailed homeodomain.

Figure 4.32: BIC score error of learned models using $M_{gmm}$, $S_{core}$, $S_{unif}$ methods on engrailed homeodomain.
clusters are used. As in the synthetic experiments, both of the likelihood and the imputation error of $M_{gmm}$ are smaller than $S_{core}$ and $S_{unif}$ as it learns fully connected model with the highest degree of freedom. The performance of $S_{core}$ and $S_{unif}$ are all within the confidence interval for all number of components. To select the appropriate number of components to avoid overfitting problems, we evaluated BIC scores for each number for components in [4.32]. As $M_{gmm}$ learns fully connected models, the BIC score of $M_{gmm}$ decreases as the number of components increases. For both $S_{core}$ and $S_{unif}$ methods, the highest BIC score is achieved at 5 components.

Figure 4.33: Edges recovered using $M_{gmm}$, $S_{core}$, $S_{unif}$ methods on engrailed homeodomain.

**Recovered edges from the methods** Figure 4.33 shows edges recovered in precision matrices in models learned using $M_{gmm}$, $S_{core}$, and $S_{unif}$ methods. $M_{gmm}$ learns almost fully connected model for all components, and $S_{core}$ and $S_{unif}$ methods learn sparse models with 49.7% and 49.5% edge density, respectively.

**Results using runtime metrics** Figure 4.34 shows the average runtime cost in parameter learning of $M_{gmm}$, $S_{core}$, $S_{unif}$ methods. For models with number of components less than 5, the runtime cost for learning $M_{gmm}$ is less than or similar to $S_{core}$ and $S_{unif}$ methods, but for 5 or more components, the runtime cost of learning $M_{gmm}$ is greater than both of the other models. $S_{core}$ method takes longer time for models with more number of components. $S_{unif}$ method is fastest among the all methods for 4 or more components, and the learning time does not vary much with the number of components.

**Summary** These results suggest that, on real data, $S_{core}$ and $S_{unif}$ are essentially equivalent. Thus, there may be no benefit to employing coreset sampling vs uniform sampling. The results on synthetic data, however, suggest that the $S_{unif}$ edge recovery metrics have higher variance than those of $S_{core}$. Since the true set of edges for real data are unknown, we cannot confirm this
directly on the Engrailed data. Therefore, the choice between $S_{\text{unif}}$ and $S_{\text{core}}$ may, in practice, be determined by the user’s tolerance for the higher runtime costs of $S_{\text{core}}$.

### 4.3.2 Learning Sparse Nonparanormal Mixture Model for Conformational Substates of Engrailed Homeodomain

Finally, we use a sparse Nonparanormal Mixture Model with coreset approximation ($S_{\text{npn}}$) for learning substates of engrailed homeodomain. For baseline methods, we use the traditional GMM ($M_{\text{gmm}}$) and a sparse GMM using cluster approximation ($S_{\text{core}}$). 500,000 frames are used training dataset, and 1,000 frames are used as a test dataset. For coreset approximation of clusters, we use $\delta = 0.1$, $\epsilon = 0.1$, and coreset size $M = 50000$, 10% of the total number of samples.

#### Selection of Winsorization parameter using imputation error metrics

To find the best Winsorization parameter $\delta_n$, we evaluate the imputation error using parameter candidates, and the result is shown in Figure 4.35. The smallest imputation error is achieved at 0.0976, and this is larger than the supposed Winsorization parameter (0.00146) [26]. This can be explained by that while conformational fluctuations of a protein from equilibrium can be assumed to be Gaussian distribution, the substates are not entirely separated as in GMM; the distributions for substates can be superposed, so the conformations in the transition between substates can be not well approximated by a mixture of Nonparanormal distribution.

#### Results using log-likelihood and imputation error metrics

Figure 4.36 and 4.37 are the negative log-likelihood and the average imputation error of test set using $M_{\text{gmm}}$, $S_{\text{core}}$, and $S_{\text{npn}}$ methods using 5 components and the selected Winsorization parameter. For log-likelihood, while
Figure 4.35: Average imputation error of models learned using $S_{npn}$ method on engrailed homeodomain with different Winsorization parameters.

Figure 4.36: Negative log-likelihood of models learned using $M_{gmm}$, $S_{core}$, $S_{npn}$ method on engrailed homeodomain.
Figure 4.37: Average imputation error of models learned using $M_{gmm}$, $S_{core}$, $S_{nqn}$ method on engrailed homeodomain.

Figure 4.38: Edges recovered using $M_{gmm}$, $S_{core}$, $S_{nqn}$ methods on engrailed homeodomain.
$S_{core}$ method shows 9.1% worse performance than $M_{gmm}$, $S_{nnp}$ method shows 3.0% improvement from $M_{gmm}$ and 11% improvement from $S_{core}$. For imputation error, $S_{core}$ shows 20% larger error than $M_{gmm}$, $S_{nnp}$ shows 17% larger error than $M_{gmm}$ and 2.9% improvement from $M_{gmm}$. The better performance of $S_{nnp}$ is due to the increased representational power of $S_{nnp}$ from the semi-parametric model than the original $S_{core}$ method. The edges recovered in precision matrices of models using the methods are shown in 4.38, and density of models learned by $S_{nnp}$ method is 48.7%.

**Summary** We find that the $S_{nnp}$ model is slightly better than $S_{core}$ on the real data. This is not surprising, since the real data are more likely to violate the assumptions of Gaussian distributions than the synthetic data.
Chapter 5

Conclusion & Future Work

Molecular Dynamics Simulations are an important technique for studying the conformational dynamics of proteins. Recent developments in MD simulation technologies enable long timescale simulations (up to milliseconds) which, in turn, produce massive and complex data sets. Thus, new computational methods are needed to analyze these trajectories. In this thesis we have developed, implemented, and tested several methods capable of producing parametric and semi-parametric, sparse generative models from MD trajectories. Specifically, our methods learn mixture models of L1 regularized GGMs or sparse nonparanormal models. Coreset sampling and k-means clustering is used to accelerate the process of learning the models.

We have demonstrated that a mixture model consisting of sparse GGMs is competitive with a traditional GMM, in terms of test likelihood and imputation errors for distributions of roughly 5 mixing components, or fewer. The real advantage of these models, however, lies in their ability to identify the most relevant couplings among the variables. This produces a model that is far easier to interpret than a traditional GMM, which produces very dense models. The runtime cost of learning a mixture model consisting of sparse GGM shows modest increase from the traditional GMM in synthetic experiments, and modest improvements on the real MD data of Engrailed Homeodomain. We have further demonstrated that on real MD data, the use of a mixture of sparse nonparanormal distributions has benefits in terms of likelihoods and imputation errors. This demonstrates the benefits of the increased representational power of the nonparanormal. The primary disadvantage of the nonparanormal is the extra space complexity needed to encode the transformation from the original problem space.

Our methods are based on the assumption that the structural fluctuations of a protein from its equilibrium follow a multivariate Gaussian distribution. While we propose a semi-parametric method using Nonparanormal distribution as real data might violate the assumption, fully non-parametric graphical models can be developed \[30\]. This would likely increase the representational power beyond the nonparanormal distributions. Additionally, one might address the space complexity issues of the nonparanormal distribution by subsampling the data, perhaps using the coreset samples.

The cluster approximation based on the coreset sampling method assumes that the equilibrium for substates are well-separated in the problem space, and the magnitudes of the eigenvalues of covariance matrices are in a reasonable range. However, in a problem where either assumption fails to hold, the approximation of cluster centers and assignments of datapoints will not perform
very well.

In addition, our method assumes that the given data are independent and identically distributed samples, but data generated by MD simulation are trajectories, which is a time series of conformations. In a long enough time-scale simulation, this assumption might hold if the simulation covers all of the possible substates and the weight for each cluster is well reflected in the dataset, but the simulation cannot guarantee any of these conditions. One might address model this time-series using delay embedding (i.e., constructs a higher dimensional data matrix with overlapping samples) [31].

Finally, the parallelized implementation of the coreset sampling algorithm and the parameter learning of sparse GGMs would reduce the runtime especially for massive and long time-scale trajectories. Analyzing the behaviors of our methods with such large sized trajectories with more components than tested in this thesis is also needed to better understand the representative power and the limitations of our methods.
Bibliography


[11] James S Fraser, Michael W Clarkson, Sheena C Degnan, Renske Erion, Dorothee Kern,


