Efficient Learning of Sparse Gaussian Mixture Models of Protein Conformational Substates

Ji Oh Yoo

CMU-CS-15-124
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.

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 the benefits of using semi-parametric models in terms of likelihood and imputation metrics.

Acknowledgments

First of all, I would like to express my sincere gratitude to my advisor, professor Christopher Langmead, for his continuous support for my study in the Fifth Year Masters Program. Without his guidance and advice, it would not have been possible to finish this research. What I learned from him during this research will be one of the most valuable assets in my future life and career. I would also like to thank professor Wei Wu for her service as a thesis committee member and her valuable feedbacks and suggestions for my research.

When there was a problem with my research, it was sometimes difficult to move forward. I would like to thank all of my wonderful friends who have helped me overcome the obstacles during these difficult times. Every conversation, coffee, and game we had together helped me regain my energy so I could concentrate and finish my study.

Finally, I would like to thank my parents and my sisters for their support. Looking back, it was not an easy decision to leave my home country and to study abroad for many years. My family supported my decision and encouraged me over the distance between us, and I am deeply grateful for their unconditional love.

Contents

List of Figures

List of Tables

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 amount 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 governs 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 probables 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 from 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 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 a brief introduction of probabilistic graphical model that we apply in our proposed methods. In Chapter 3, we present our methods for efficient learning of 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 simulation 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 across 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

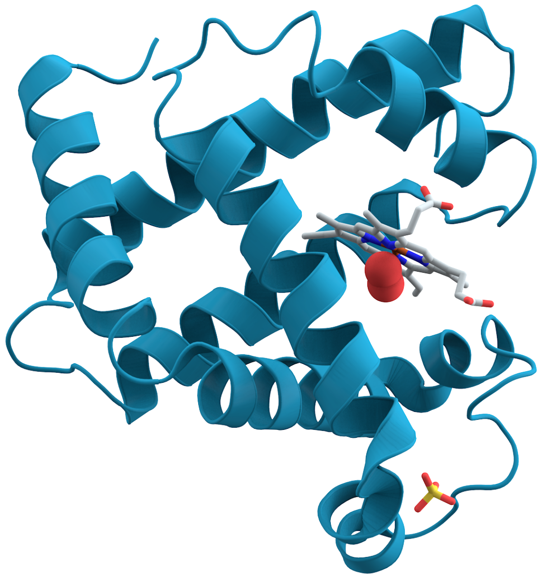


Figure 2.1: Three dimensional structure of myoglobin (1MBO) with bounded oxygen molecule. Myoglobin is a polypeptide with 153 amino acids and has a three-dimensional structure rather than a linear or planar structure.

time step is usually set to 1-2 femtoseconds (10^{-15} sec). For reasonable size of the data for later analysis, every 1,000th to 10,000th 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 identifying 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., α-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.

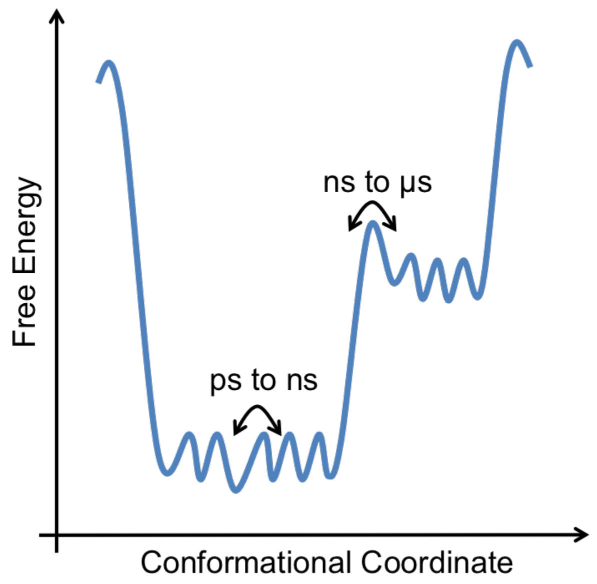


Figure 2.2: Free energy landscape of a protein according to its virtual conformational coordinate. The populations of different protein conformation follow Boltzmann distribution, and the ratio of populations varies exponentially to the free energy difference of the conformations. The transitions across 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 can contain multiple conformational substates and requires 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 of 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 makes it hard to be 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 an 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 these 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 Ψ. 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 ψ_c in Ψ 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 are 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 μ and the covariance matrix Σ. 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 μ and Σ, and the joint distribution is the multivariate Gaussian distribution:

$$P(X_1, X_2, ..., X_n) = P(X) = \frac{1}{Z} \exp\Big(-\frac{1}{2}(X-\mu)^{\intercal}\Sigma^{-1}(X-\mu)\Big)$$

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 are 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 μ_a and μ_b are sub-vectors with corresponding rows in the original mean vector μ, and $\Sigma_{aa}, \Sigma_{ab}, \Sigma_{ba}, \Sigma_{bb}$ are sub-matrices with corresponding rows and columns in the original covariance matrix Σ.

2.6 Multivariate Gaussian Representations of Protein Dynamics

Most methods for the 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 relationships between the masses are represented as springs. When a folded protein is approximated with an elastic network of masses (α-carbons of various residues) and springs (pair-wise direct relationship among residues), as in Figure 2.3, the positional fluctuations of the α-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 substates 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.

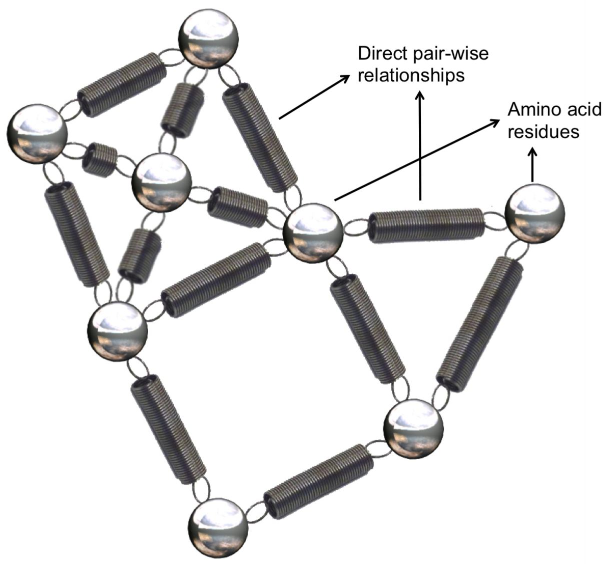


Figure 2.3: Approximation of a protein by Elastic Network Model. If the α-carbons of amino acid residues are represented as masses and pair-wise direct relationships between residues are presented as springs, the fluctuation of the masses from the equilibrium follows a multivariate Gaussian distribution.

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 representational power of our models 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 (μ_k, Σ_k) is associated with a weight probability π_k that represents the probability of the k'th component is chosen, and the sum of the probabilities 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 $D = \{X^{(1)}, ..., X^{(N)}\}$ is:

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

Unlike the GGM with a 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]. This 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:

$$\gamma_{i,k}^{(t)} = p(k|X^{(i)}, M^{(t)})$$

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

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

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

This algorithm stops when the change of log-likelihood for every step is less than a predefined 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 are 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:

$$\mathcal{L}_p(\mathbf{\Theta}|D) = \sum_{i=1}^{N}\log\Big(\sum_{k=1}^{c}\pi_k N(X^{(i)}|\mu_k,\Sigma_k)\Big) - \lambda\sum_{k=1}^{c}\|\Sigma_k^{-1}\|_1 \tag{3.1}$$

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

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

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

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

$$\Sigma_k^{(t+1)} = \operatorname*{argmax}_{\Sigma_k\succ 0}\Big\{-\log|\Sigma_k| + \mathrm{Tr}(\Sigma_k^{-1}A_k^{(t+1)}) - \lambda\|\Sigma_k^{-1}\|_1\Big\} \tag{3.2}$$

where

$$A_k^{(t+1)} = \frac{\sum_{i=1}^{N}\gamma_{i,k}^{(t)}(X^{(i)}-\mu_k)(X^{(i)}-\mu_k)^\intercal}{\sum_{i=1}^{N}\gamma_{i,k}^{(t)}}$$

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

$$\Sigma_k^{(t+1)} = \underset{\Sigma_k \succ 0}{\text{argmax}} \Big\{ -\log|\Sigma_k| + \text{Tr}(\Sigma_k^{-1} A_k^{(t+1)}) - \lambda_k \|\Sigma_k^{-1}\|_1 \Big\} \tag{3.3}$$

Solving Eq. 3.3 to get the covariance matrix for each iteration is exactly the same as the sparse inverse covariance estimation of a multivariate Gaussian, and we can use efficient algorithms including *graphical lasso* [24]. The regularization parameters λ_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. The BIC of our model is defined as follows:

$$\text{BIC}(\mathbf{\Theta}) = \mathcal{L}(D|\mathbf{\Theta}) - \log(N)\,\text{df}(\mathbf{\Theta}) \tag{3.4}$$

where df is the degree of freedom in the model:

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

and higher BIC values are better than lower ones.

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 $D_{train}$ and cross-validation set $D_{cv}$. ;
for Every combination of regularization parameters $\lambda_1, ...\lambda_c$ do
    randomly initialize $K$ component centers ;
    while not converged within criteria do
        evaluate each $\gamma_{i,k}^{(t)}$ using $D_{train}$;
        update $\pi_k^{(t+1)}$, $\mu_k^{(t+1)}$, $\Sigma_k^{(t+1)}$ with $\lambda_1, ..., \lambda_c$ using $D_{train}$;
        align components with reference model $m$ and relabel components ;
    end
    calculate BIC using $D_{cv}$ ;
    store the model with best BIC score so far ;
end
```

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 the aligning and relabeling step with respect to the given reference model m. Each iteration in this algorithm is an EM algorithm with different regularization parameter λ_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 between clusters to find the closest reference cluster for each learned cluster. The KL-divergence of two multivariate Gaussian distributions can be evaluated analytically as in Eq. 3.5.

$$D_{KL}(\mathcal{N}_0, \mathcal{N}_1) = \frac{1}{2}\Big\{ \operatorname{Tr}(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu_0)^\intercal \Sigma_1^{-1}(\mu_1 - \mu_0) - dim + \ln \frac{|\Sigma_1|}{|\Sigma_0|}\Big\} \tag{3.5}$$

where dim is the number of variables in the component. For reference model, we can use ground-truth model for synthetic experiments or 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 the regularization parameter search. Instead of refining clusters 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 method of a mixture of sparse GGMs based on the approximated clusters.

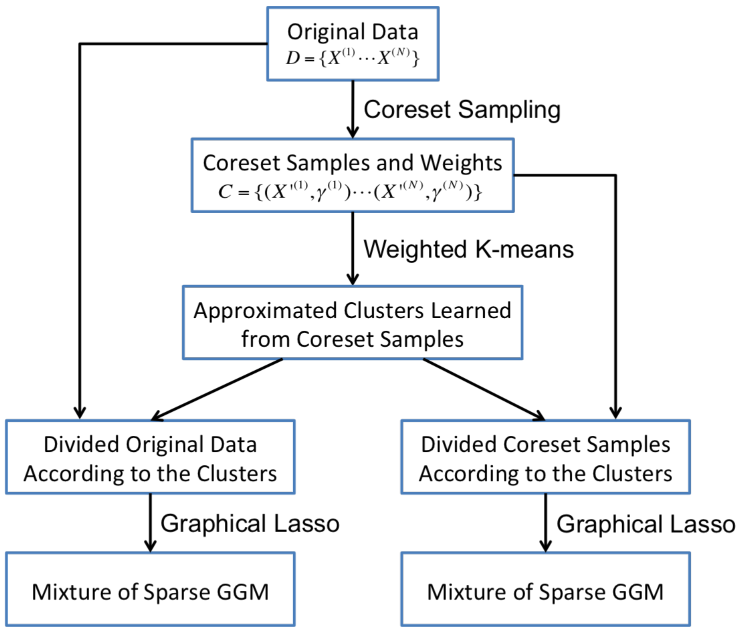
3.3.1 Coreset Sampling for Gaussian Mixture Model

A coreset $C = \{(\gamma^{(1)}, X'^{(1)}), ..., (\gamma^{(M)}, X'^{(M)})\}$ 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 $\gamma^{(j)}$:

$$\mathcal{L}(C|\mathbf{\Theta}) = -\sum_{j=1}^{M} \log \sum_{k=1}^{c} \frac{\pi_k \gamma^{(j)}}{\sqrt{(2\pi)^d |\Sigma_k^{-1}|}} \exp\Big(-\frac{1}{2}(X'^{(j)} - \mu_k)^\intercal \Sigma_k^{-1}(X'^{(j)} - \mu_k)\Big)$$

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:



Original Data
$D=\{X^{(1)}\cdots X^{(N)}\}$
Coreset Sampling
Coreset Samples and Weights
$C=\{(X'^{(1)},\gamma^{(1)})\cdots(X'^{(N)},\gamma^{(N)})\}$
Weighted K-means
Approximated Clusters Learned from Coreset Samples
Divided Original Data According to the Clusters
Divided Coreset Samples According to the Clusters
Graphical Lasso
Graphical Lasso
Mixture of Sparse GGM
Mixture of Sparse GGM

Figure 3.1: Diagram of the 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.

$$\begin{aligned}
\mathcal{L}(D|\mathbf{\Theta}) &= -\sum_{i=1}^{N}\log\sum_{k=1}^{c}\frac{\pi_k}{\sqrt{(2\pi)^d|\Sigma_k^{-1}|}}\exp\Big(-\frac{1}{2}(X^{(i)}-\mu_k)^\intercal\Sigma_k^{-1}(X^{(i)}-\mu_k)\Big)\\
&= -N\log Z(\mathbf{\Theta})+\sum_{i=1}^{N}\log\sum_{k=1}^{c}\frac{\pi_k}{Z(\Theta)\sqrt{(2\pi)^d|\Sigma_k^{-1}|}}\exp\Big(-\frac{1}{2}(X^{(i)}-\mu_k)^\intercal\Sigma_k^{-1}(X^{(i)}-\mu_k)\Big)\\
&= -N\log Z(\mathbf{\Theta})+\phi(D|\mathbf{\Theta})
\end{aligned}$$

where $Z(\mathbf{\Theta})=\sum_k\frac{\pi_k}{\sqrt{(2\pi)^d|\Sigma_k^{-1}|}}$ is a normalizer. The first term in the equation above is independent of the given data D and the second term $\phi(D|\mathbf{\Theta})$ 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 $\phi(D|\mathbf{\Theta})$.

For coreset C, this ϕ can be extended to:

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

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

$$(1-\epsilon)\phi(D|\mathbf{\Theta})\leq\phi(C|\mathbf{\Theta})\leq(1+\epsilon)\phi(D|\mathbf{\Theta})$$

Algorithm 2: Coreset Sampling for Gaussian Mixture Model

```
Data: input data $D$, $\delta$, 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 $m(x) = \left\lceil \frac{5}{|D_b|} + \frac{d(x,B)^2}{\sum_{X' \in D} d(X',B)^2} \right\rceil$ ;
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 3.

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 $f : \{X'^{(1)}, ..., X'^{(M)}\} \to \{1, ..., K\}$ 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 $p_k^{(t+1)} = \sum_{X'^{(i)}:f^{(t)}(X'^{(i)})=k} \gamma^{(i)} X'^{(i)} \Big/ \sum_{X'^{(i)}:f^{(t)}(X'^{(i)})=k} \gamma^{(i)}$ ;
      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 λ_k's, but we use a simple heuristic method to reduce the runtime cost for model selection. The optimal regularization λ_k we want is small enough to capture the relationship in precision matrix Σ_k^{-1} but large enough to give sparse Σ_k^{-1}. For datapoints matrix D_k in the 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 λ_k, we randomly shuffle the columns of D_k and apply graphical lasso with parameter candidates of λ_k from smallest to largest until the learned precision matrix is diagonal. For more accurate choice of λ_k, we can repeat this selection process and take the average value for our final optimal λ_k. Also, this algorithm can be easily parallelized for each cluster k and for each candidate for λ_k.

Learning Sparse GMM from Coreset samples Divided in Approximated Clusters

To further exploit the coreset sampling approximation, we can learn a sparse Gaussian for each cluster using the coreset samples instead of using the original datapoints. By reducing the number of samples for each clusters, we can reduce the runtime 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 the original input data D, and the rest of the

Algorithm 4: Learning Sparse GMM from Datapoints Divided to Approximated Clusters

```
Data: original input data $D$, cluster centers $p_1, ..., p_K$
Result: sparse Gaussian Mixture Model with $K$ clusters
for $X^{(i)} \in D$ do
    Assign the cluster assignment $c(X^{(i)})$ = label of the closest cluster to $X^{(i)}$;
end
for each cluster $k$ do
    $D_k = \{X^{(i)} : c(X^{(i)}) = k\}$ ;
    $D'_k$ = 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
```

algorithm is the same.

The methods we discussed in this section mainly consist 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 to 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 a Gaussian distribution. This assumption makes 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 distributions to cover more families of distributions with richer representational power. A random vector $X = (X_1, X_2, ..., X_P)$ follows a *nonparanormal* distribution if there exists a set of functions $\{f\}_1^P$ such that $f(X) = (f_1(X_1), f_2(X_2), ..., f_P(X_P))$ 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 $D = \{X_1, X_2, ..., X_N\}$ follows a nonparanormal distribution, the estimator of $\{f\}_1^P$ is given by:

$$\widehat{f_p}(X_p) = \widehat{\mu_p} + \widehat{\sigma_p}\widetilde{h_p}(X_p)$$

where $\widehat{\mu_p}$ and $\widehat{\sigma_p}$ are empirical mean and covariance for normalization:

$$\widehat{\mu_p} = \frac{1}{N}\sum_{i=1}^{N} X_p^{(i)}$$

$$\widehat{\sigma_p} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X_p^{(i)} - \widehat{\mu_p})^2}$$

and $\widetilde{h_p}$ is an inverse standard Gaussian distribution function (quantile function) applied to the estimated empirical distribution of X_p:

$$\widetilde{h_p}(X_p) = \Phi^{-1}(\widetilde{F_p}(X_p))$$

For high-dimensional problem settings, the proposed estimator for F_p is a *Winsorized* estimator to truncate the outliers for the extreme values X_p:

$$\widetilde{F_p}(X_p) = \begin{cases} \delta_N & \text{if } \widehat{F_p}(X_p) < \delta_N \\ \widehat{F_p}(X_p) & \text{if } \delta_N \leq \widehat{F_p}(X_p) \leq 1 - \delta_N \\ 1 - \delta_N & \text{if } \widehat{F_p}(X_p) > 1 - \delta_N \end{cases} \tag{3.6}$$

where $\widehat{F_p}(X_p)$ is an empirical cumulative distribution function $\widehat{F_p}(x) = \frac{1}{N}\sum_{i=1}^{N} I(X_p^{(i)} \leq x)$. The choice of δ_N gives the bias-variance trade-off of the Winsorized estimator.

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 sparse Nonparanormal Mixture Model is that it can be used in the further prediction studies. The transformation functions $\{f_p\}_1^P$ 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 a sparse model for each cluster separately. For the identification of clusters, we use an approximation method using coreset sampling and weighted K-mean algorithm. For learning a sparse GGM, we use

Algorithm 5: Learning Sparse Nonparanormal Mixture Model from Datapoints Divided to Approximated Clusters

```
Data: original input data $D$, cluster centers $p_1, ..., p_K$
Result: sparse Nonparanormal Mixture Model with $c$ clusters
for $x_i \in D$ do
    Assign the cluster assignment $c(x_i) =$ label of the closest cluster ;
end
for each cluster $k$ do
    $D_k = \{x_i : c(x_i) = k\}$ ;
    Estimate transformation function $\{f_p\}_{p=1}^{P}$ using Winsorized estimator ;
    Transform the data: $T_k = f(D_k)$ ;
    $T_k' =$ randomly shuffled columns of $T_k$ ;
    for each candidate for regularization parameter $\lambda_k$ do
        Apply graphical lasso with $\lambda_k$ for $T_k'$ ;
        if learned $\Sigma_k^{-1}$ is diagonal matrix then
            break ;
        end
    end
    Apply graphical lasso with the last $\lambda_k$ for $T_k$ ;
end
```

either all of the original samples or the coreset samples as a training set with a 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 learning 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:

$$\text{Error}_{imputation} = \frac{1}{P|D_{test}|} \sum_{X \in D_{test}} \sum_{j=p}^{P} |X_j - V_j|$$

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.
  $$\text{Accuracy} = \frac{\text{true positive + true negative}}{\text{total possible number of edges}}$$
- 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.
  $$\text{F1 score} = \frac{2 \times \text{true positive}}{2 \times \text{true positive} + \text{false positive + false negative}}$$

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 $\delta = 0.1$ with coreset size $M = 1000$ (10% of samples).

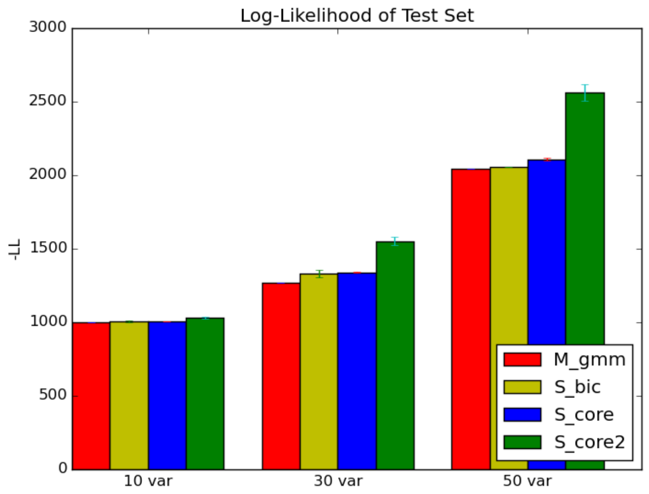


Figure 4.1: Negative Log-Likelihood of learned models using M_{gmm}, S_{bic}, S_{core}, S_{core2} methods.

Results using log-likelihood metrics Figure 4.1 shows the negative log-likelihood using M_{gmm}, S_{BIC}, S_{core}, S_{core2} methods on 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_{gmm} show the best likelihood of the test data. However, the differences between S_{bic} and S_{core} and the baseline method are small. The negative log-likelihood of S_{bic} is 0.48% to 4.8% worse than M_{gmm}, and the likelihood of S_{core} is 0.48% to 5.3% worse than M_{gmm}. We note that the confidence intervals of the likelihoods of S_{bic} and S_{core} overlap those of the baseline methods for datasets with 10 and 30 variables. S_{core2} method shows the worst performance with likelihood 3.2% to 26% worse than the baseline method. We conclude that S_{bic} and S_{core} are nearly equivalent to the baseline model, in terms of likelihood.

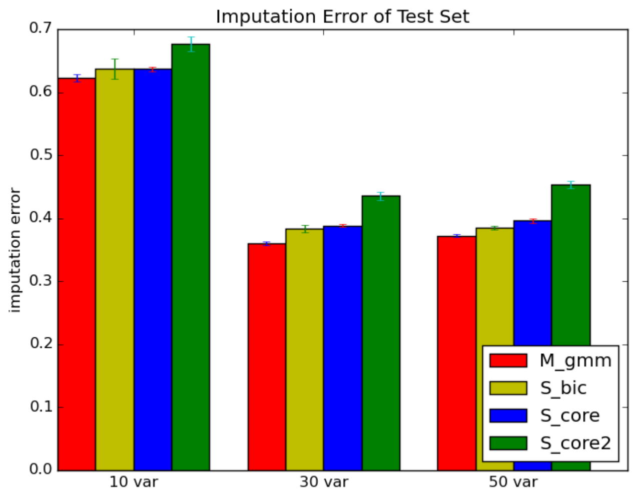


Figure 4.2: Average imputation errors of learned models using M_{gmm}, S_{bic}, S_{core}, S_{core2} methods.

Results using imputation error metrics Figure 4.2 shows the average imputation error of test set using M_{gmm}, S_{bic}, S_{core}, S_{core2} methods on the three different datasets. For all experiments, the imputation errors of M_{gmm} is the smallest. The difference between the baseline model and S_{bic} and S_{core} are small, and sometimes insignificant. The imputation error of S_{bic} is 2.2% to 6.6% higher than M_{gmm}, and that of S_{core} is 2.2% to 7.9% higher than M_{gmm}. Like the likelihood, The S_{bic} and S_{core} imputations are within the confidence intervals of the baseline model for the 10 and 30 variable datasets. S_{core2} shows the highest imputation error with 8.6% to 22% larger than M_{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_{bic} and S_{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.

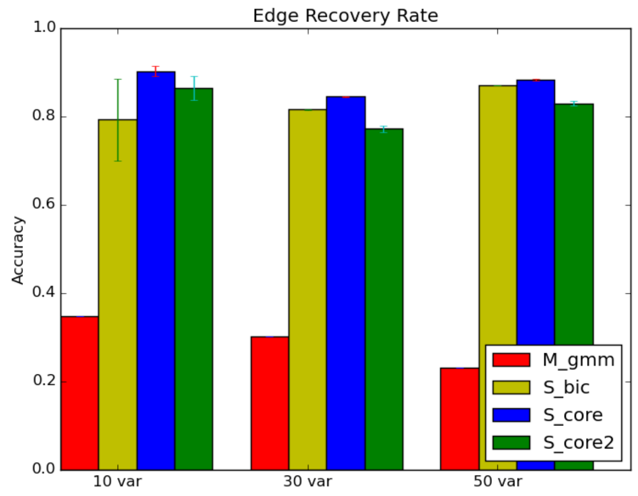


Figure 4.3: Edge recovery accuracy of learned models using M_{gmm}, S_{bic}, S_{core}, S_{core2} methods.

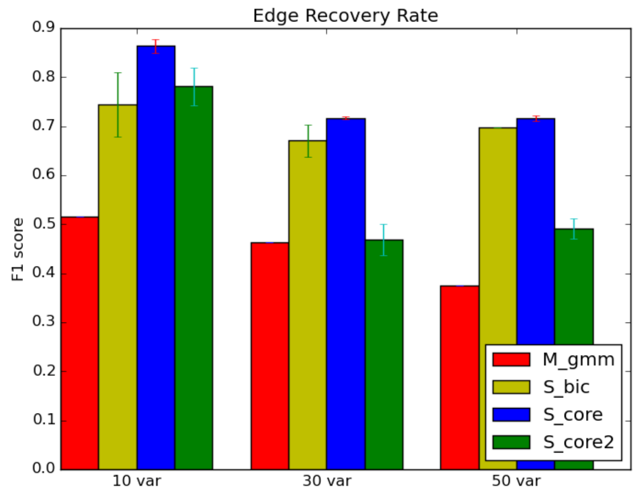


Figure 4.4: Edge recovery F1 score of learned models using M_{gmm}, S_{bic}, S_{core}, S_{core2} methods.

S_{core} shows the best accuracy (85% to 90%) and F1 score (0.72 to 0.86). S_{bic} shows accuracy from 79% to 87% and F1 score from 0.67 to 0.74. While the accuracy of S_{core2} 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_{bic} and S_{core} are superior to S_{core2} and the baseline model in terms of edge recovery.

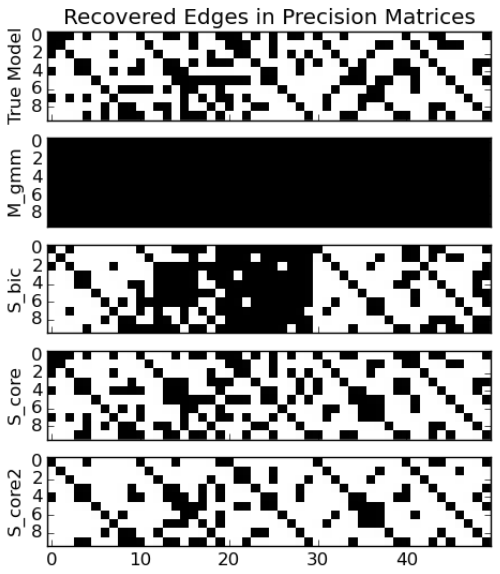


Figure 4.5: Recovered edges in precision matrices of learned models using M_{gmm}, S_{bic}, S_{core}, S_{core2} methods.

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_{gmm}, S_{bic}, S_{core}, and S_{core2} methods. M_{gmm} model learns a fully connected for all components, and this explains the worst accuracy and F1 score in edge recovery. S_{core} and S_{core2} 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_{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_{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_{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_{core} method. Thus, we conclude that the sparse methods are superior to any simple thresholding-based method to deriving sparse models.

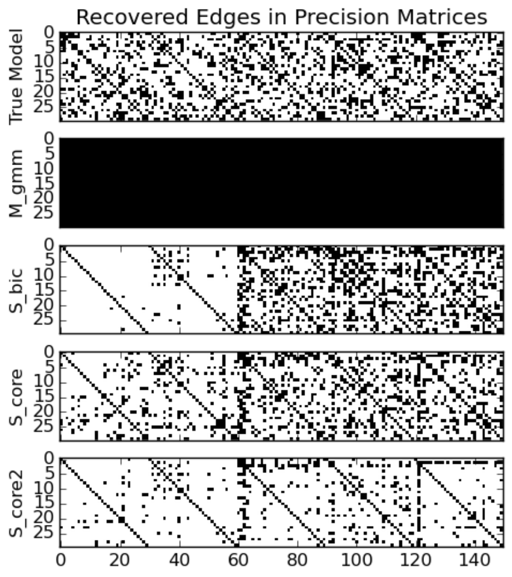


Figure 4.6: Recovered edges in precision matrices of learned models using M_{gmm}, S_{bic}, S_{core}, S_{core2} methods.

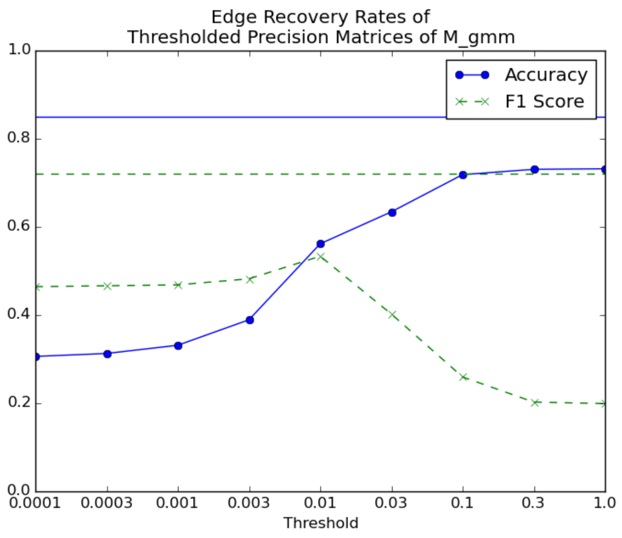


Figure 4.7: Edge recovery accuracy and F1 score of thresholded precision matrices learned using M_{gmm} method.

Runtime (sec)	10 vars	30 vars	50 vars
M_{gmm}	1.02	1.34	2.03
S_{bic}	2052	5012	18000
S_{core}	4.65	13.40	33.38
S_{core2}	1.51	2.98	13.37

Table 4.1: Average runtime cost in learning models using M_{gmm}, S_{bic}, S_{core}, S_{core2} methods.

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 $\delta = 0.1$ 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%

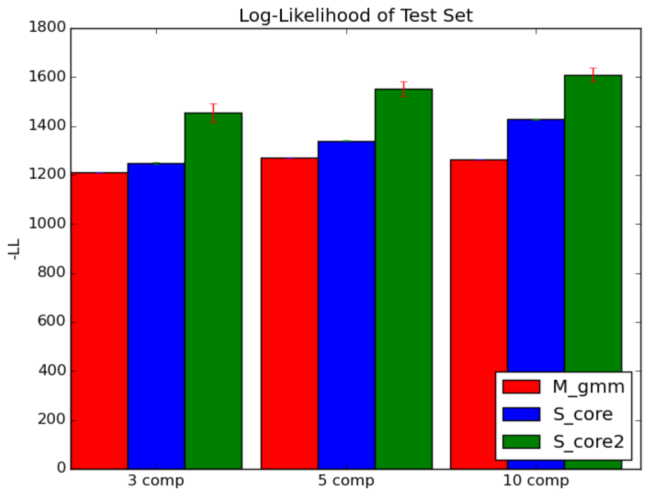


Figure 4.8: Negative log-likelihood, imputation error, and edge recovery rate of learned models using M_{gmm}, S_{core}, S_{core2} methods.

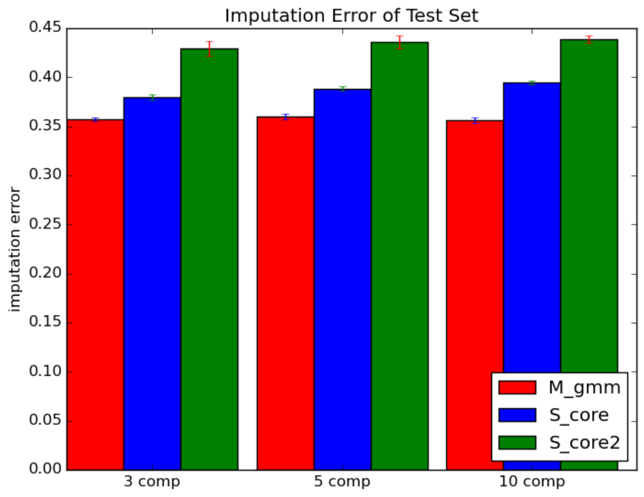


Figure 4.9: Average imputation errors of learned models using M_{gmm}, S_{core}, S_{core2} methods.

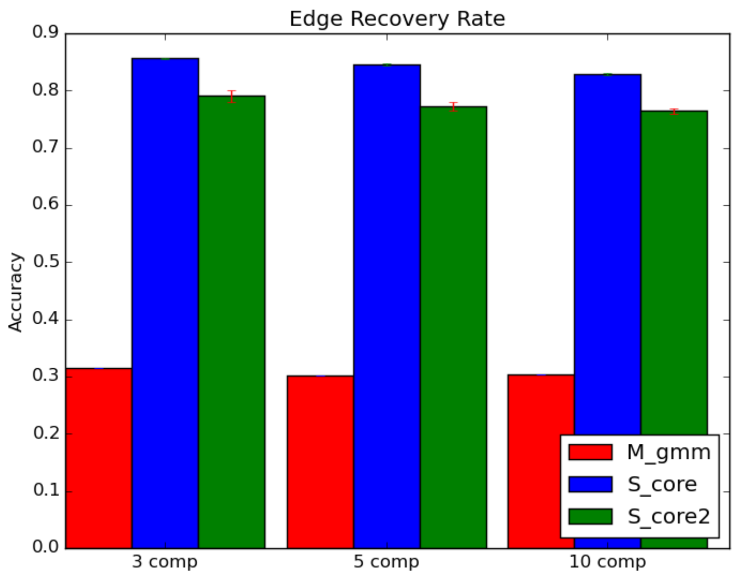


Figure 4.10: Edge recovery accuracy in precision matrices of learned models using M_{gmm}, S_{core}, S_{core2} methods.

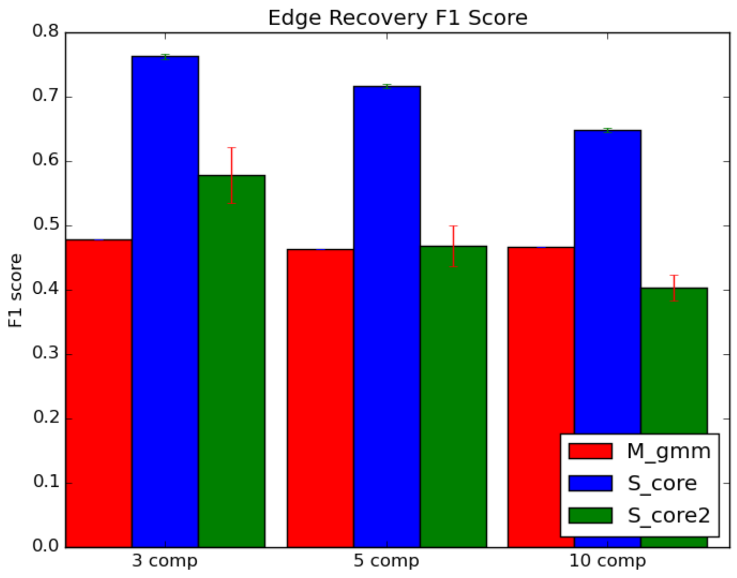


Figure 4.11: Edge recovery F1 score in precision matrices of learned models using M_{gmm}, S_{core}, S_{core2} methods.

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 $\delta = 0.1$ with coreset size $M = 10\%$ of the size of the samples.

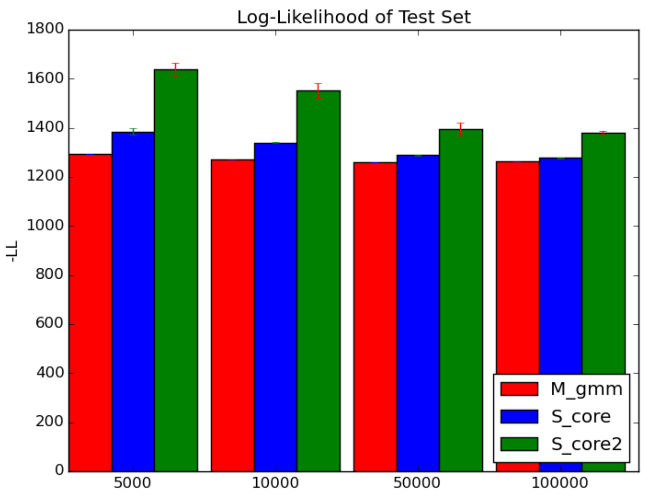


Figure 4.12: Negative log-likelihood, imputation error, and edge recovery rate of learned models using M_{gmm}, S_{core}, S_{core2} methods.

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},

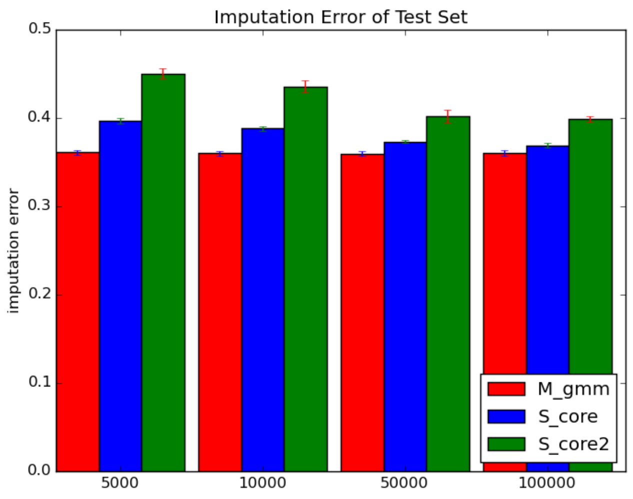


Figure 4.13: Average imputation errors of learned models using M_{gmm}, S_{core}, S_{core2} methods.

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}.

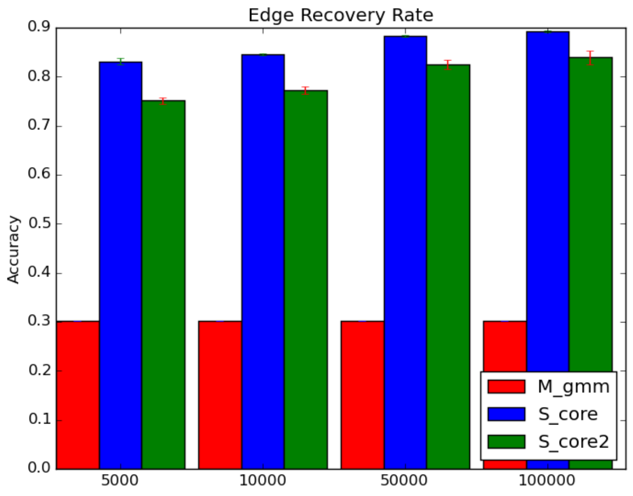


Figure 4.14: Edge recovery accuracy in precision matrices of learned models using M_{gmm}, S_{core}, S_{core2} methods.

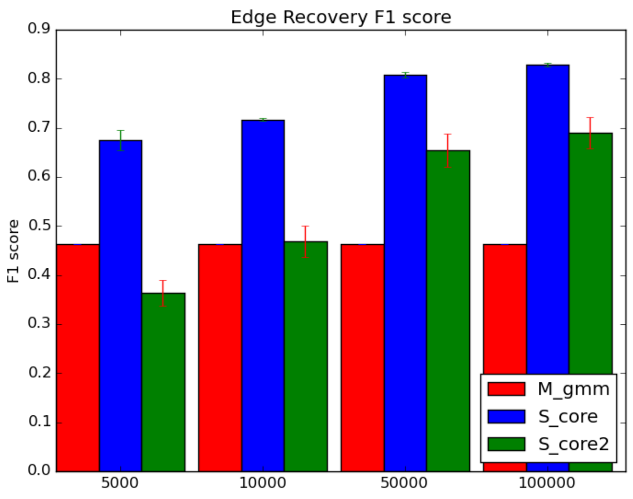


Figure 4.15: Edge recovery F1 score in precision matrices of learned models using M_{gmm}, S_{core}, S_{core2} methods.

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 $\delta = 0.1$ 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.

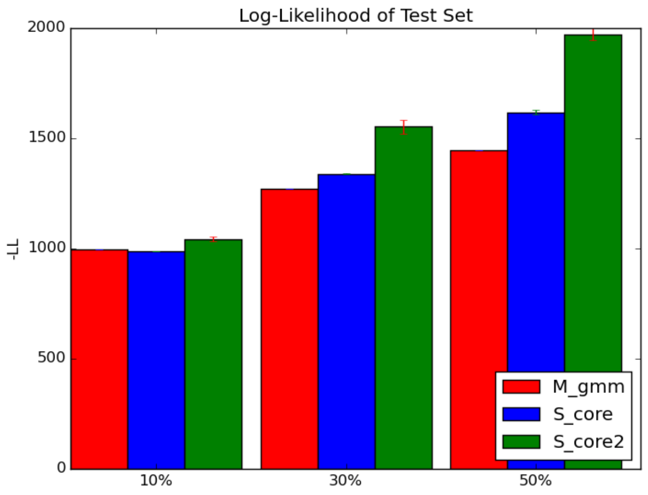


Figure 4.16: Negative log-likelihood, imputation error, and edge recovery rate of learned models using M_{gmm}, S_{core}, S_{core2} methods.

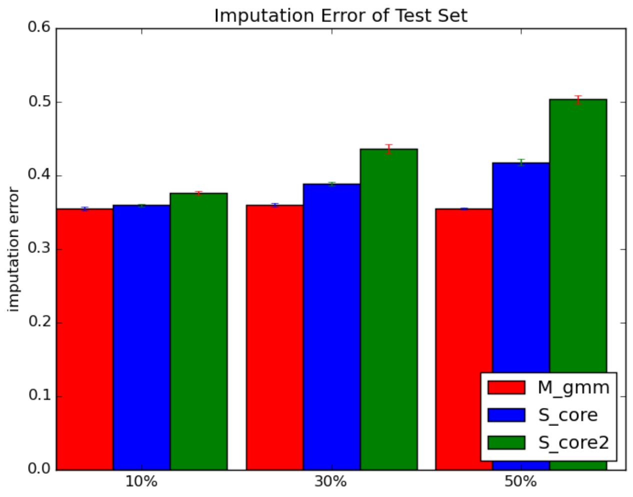


Figure 4.17: Average imputation errors of learned models using M_{gmm}, S_{core}, S_{core2} methods.

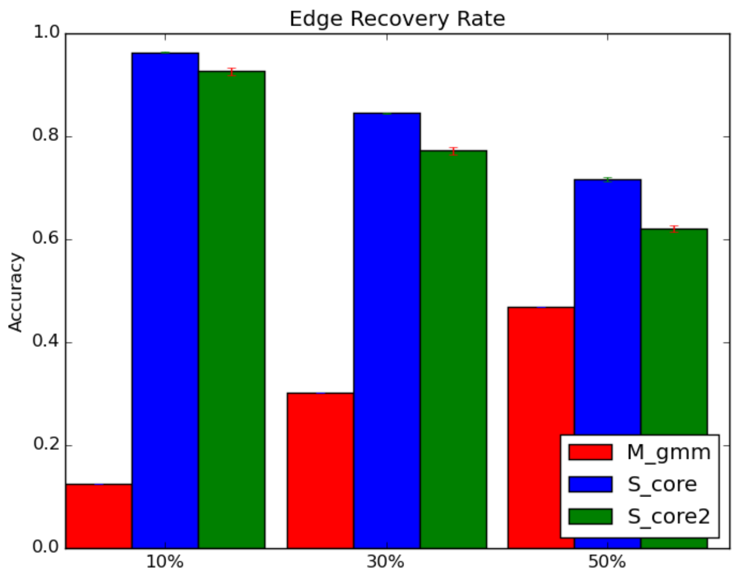


Figure 4.18: Edge recovery accuracy in precision matrices of learned models using M_{gmm}, S_{core}, S_{core2} methods.

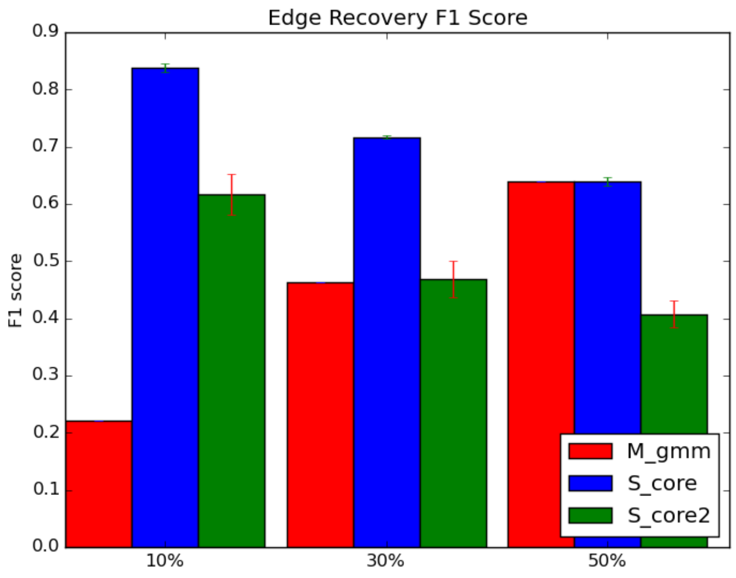


Figure 4.19: Edge recovery F1 score in precision matrices of learned models using M_{gmm}, S_{core}, S_{core2} methods.

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_{core2} 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_{core2} 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 $\delta = 0.1$. To further analyze the benefits of coreset sampling in our methods, we use two more baseline methods, S_{unif} and S_{unif2}, where they sample the sub-samples uniformly at random instead of coreset sampling algorithm. From those random sub-samples, S_{unif} and S_{unif2} both identify the approximated clusters, then S_{unif} uses all original samples to learn a Gaussian distribution for each cluster, but S_{unif2} uses only those sub-samples to learn the Gaussian distributions.

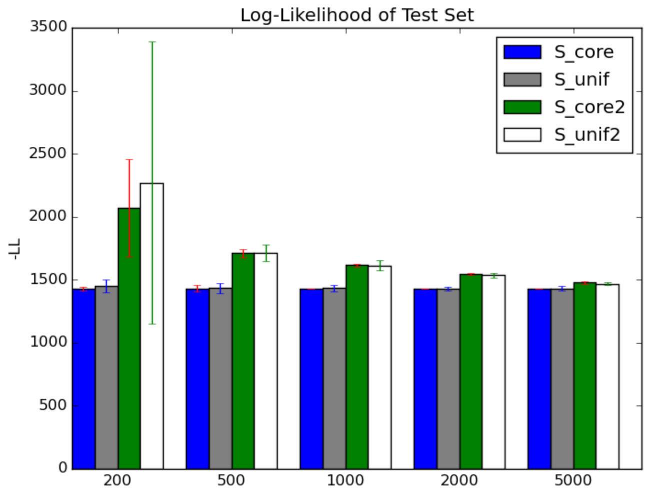


Figure 4.20: Negative log-likelihood of learned models using S_{core}, S_{unif}, S_{core2}, and S_{unif2} methods.

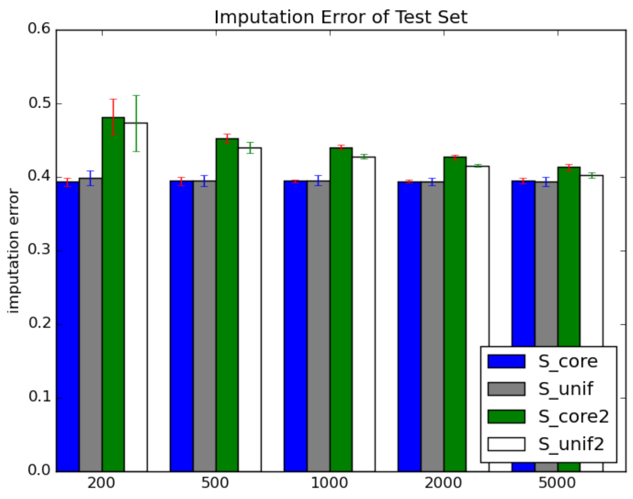


Figure 4.21: Imputation errors of learned models using S_{core}, S_{unif}, S_{core2}, and S_{unif2} methods.

Results using log-likelihood and imputation error metrics Figure 4.20 and 4.21 show the log-likelihoods and the average imputation errors of S_{core}, S_{unif}, S_{core2}, and S_{unif2} methods. For log-likelihoods and imputation errors, S_{core2} and S_{unif2} are worse than S_{core} and S_{unif} methods, and this is due to the less size of the training data set in the learning of Gaussian distributions. The performances of S_{core2} and S_{unif2} increase as the size of the sub-samples increases, did not perform better than their counterparts, S_{core} and S_{unif}, when the most sub-samples are used. The baseline method S_{unif} shows similar average log-likelihoods and imputation errors to S_{core} method, but the variance is larger than S_{core} and the variance gets smaller as the size of the sub-samples increases. This is due to the uniformly random sampling of original data for cluster approximation of S_{unif} where the randomly chosen sub-samples can misrepresent the heavily or lightly weighted clusters of the true distribution. For S_{unif2} method, its log-likelihoods and imputation errors are competitive or sometime better than S_{core2} method. While the coreset sampling algorithm gives a better representation of the distribution of the original dataset, the graphical lasso in S_{core2} does not use the weights of the coreset samples. This can explain the competitive performance of S_{unif2} method compared to S_{core2} method.

Results using edge recovery-related metrics Figure 4.22 and 4.23 show the edge recovery accuracy and F1 score of S_{core}, S_{unif}, S_{core2}, and S_{unif2} methods. The edge recovery accuracy and F1 score of S_{core} with less than or equal to 500 coreset sub-samples (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). Both accuracy and F1 score of S_{core2} and S_{unif2} are worse than S_{core} and S_{unif} in all experiments as in the log-likelihoods and the imputation errors. The accuracy of S_{core2} and S_{unif2} show the similar average and variance, but S_{core2} shows the better F1 score than S_{unif2}.

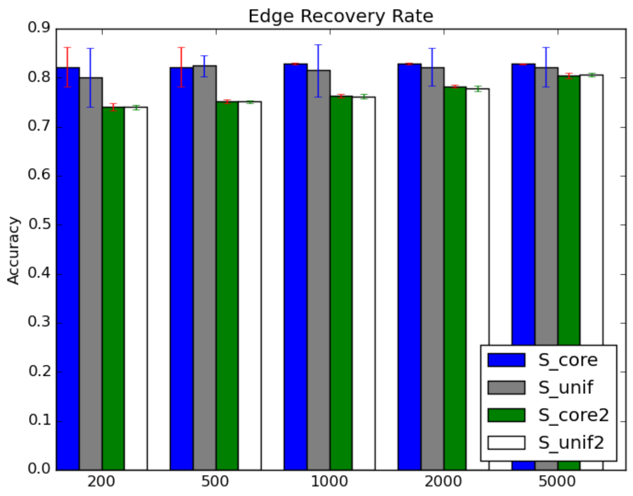


Figure 4.22: Edge recovery rate in precision matrices of learned models using S_{core}, S_{unif}, S_{core2}, and S_{unif2} methods.

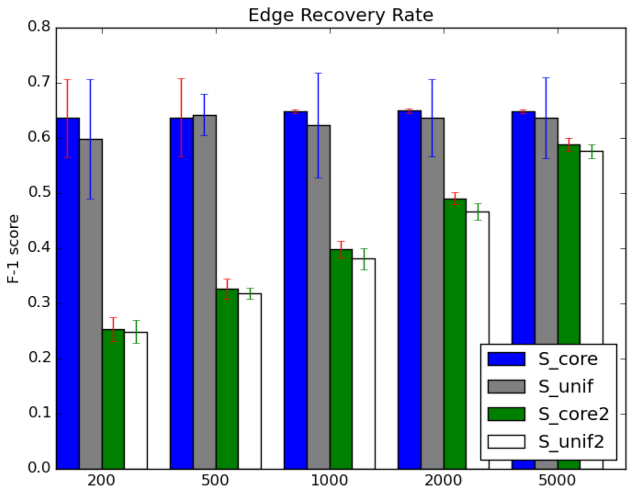


Figure 4.23: Edge recovery F1 score in precision matrices of learned models using S_{core}, S_{unif}, S_{core2}, and S_{unif2} methods.

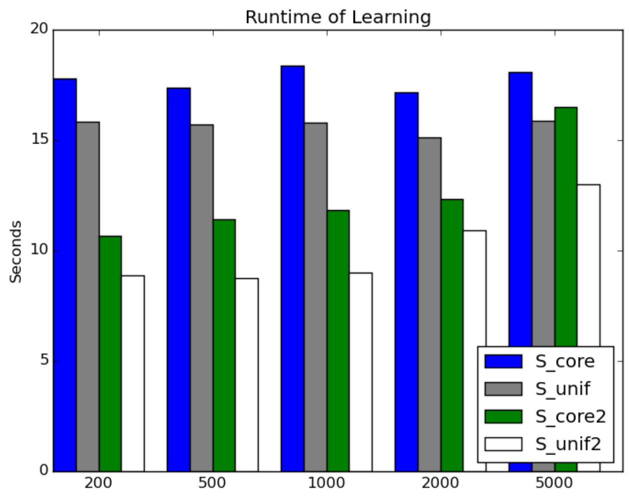


Figure 4.24: Runtime cost for parameter learning using S_{core}, S_{unif}, S_{core2}, and S_{unif2} methods.

Results using runtime metrics The runtime cost of the four 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} and S_{unif2}. As S_{core} and S_{core2} uses additional coreset algorithm for coreset sampling, it takes more runtime cost than S_{unif} and S_{unif2}. S_{core} takes 12% to 15% longer time than S_{unif}, and S_{core2} takes 13% to 20% longer time than S_{unif2} method.

Summary These results suggest that the coreset samples have benefits that it reduces the variance of performance metrics and produces more stable results when used in the learning process, compared to the random sub-samples. S_{core} tends to have lower variance than S_{unif} in the metrics we considered. S_{core2} and S_{unif2} show worse performance than S_{core} and S_{unif} as S_{core2} and S_{unif2} use only the sub-samples for learning each Gaussian distribution, but when small portion of the sub-samples is used, S_{core2} shows lower variance than S_{unif2}. However, These benefits of the coreset sampling algorithm come at the cost of a modest increase in runtime. Also, S_{core2} has the advantage of reducing the runtime costs in the learning process compared to S_{core}, but as the graphical lasso does not utilize the weights of the coreset sub-samples, the performance of S_{core2} does not show noticeable improvements over its counterpart, S_{unif2}.

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 $\delta = 0.1$ 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].

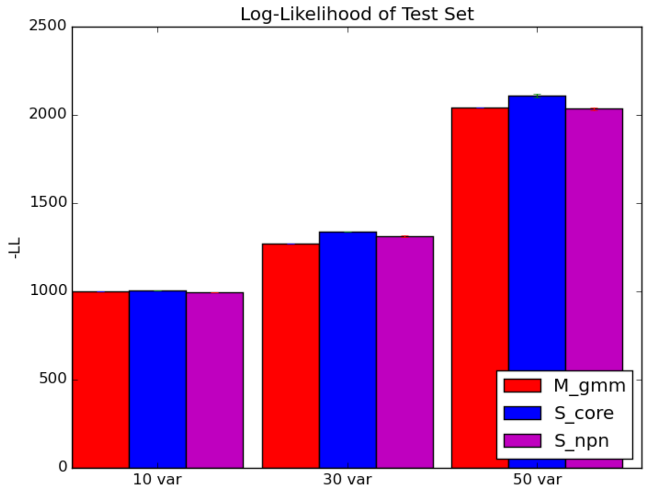


Figure 4.25: Negative Log-Likelihood of learned models using M_{gmm}, S_{core}, S_{npn} methods.

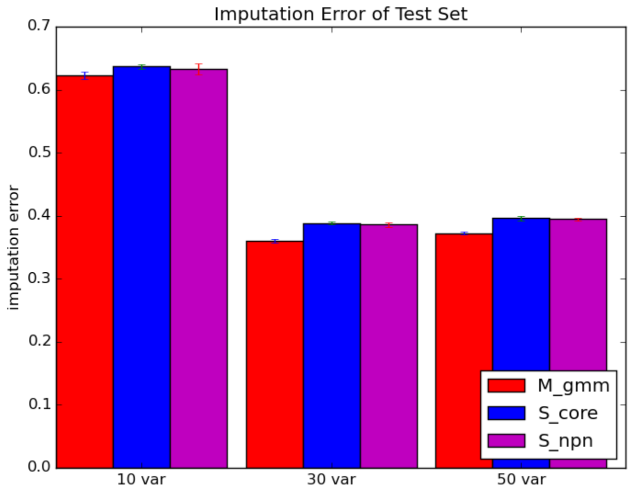


Figure 4.26: Average imputation errors of learned models using M_{gmm}, S_{core}, S_{npn} methods.

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}.

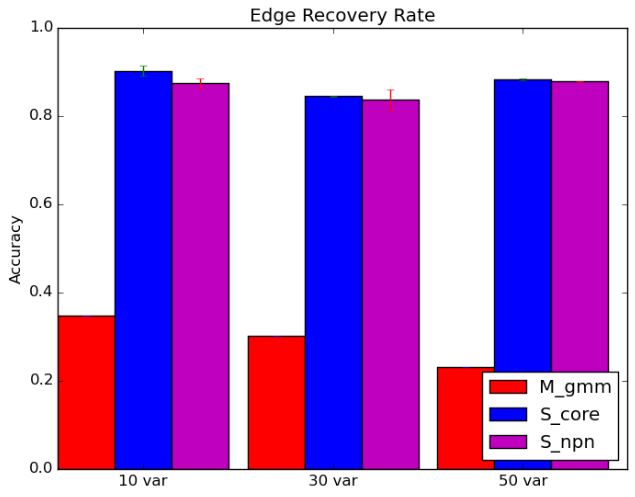


Figure 4.27: Edge recovery accuracy of learned models using M_{gmm}, S_{core}, S_{npn} methods.

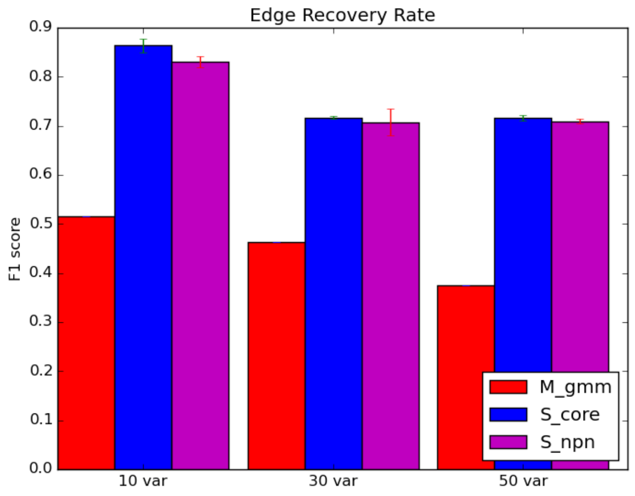


Figure 4.28: Edge recovery F1 score of learned models using M_{gmm}, S_{core}, S_{npn} methods.

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 53 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 of 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 (from residue 8 to 53) 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 α-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 α-carbons, and the displacement of each α-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_{unif}). 500,000 frames are used training dataset, and 1,000 frames are used as a test dataset. For coreset approximation of clusters, we use $\delta = 0.1$, 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_{unif} 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

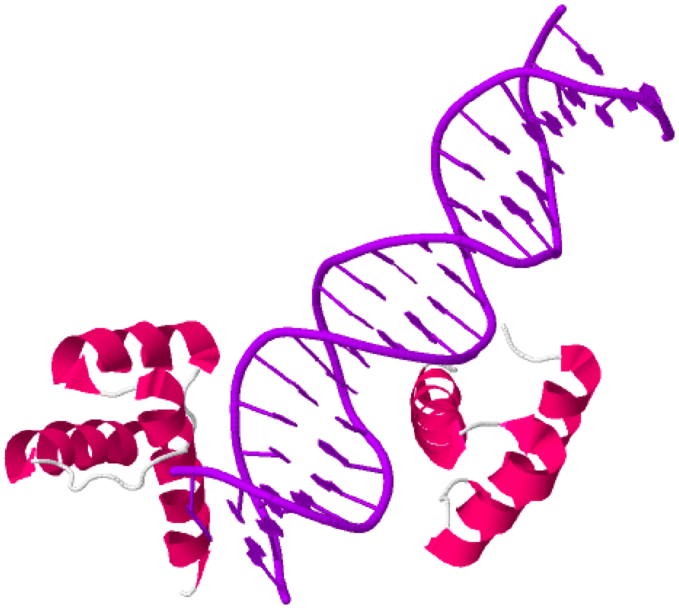


Figure 4.29: Engrailed homeodomains of *Drosophila melanogaster* (1DU0) bound to DNA double helix.

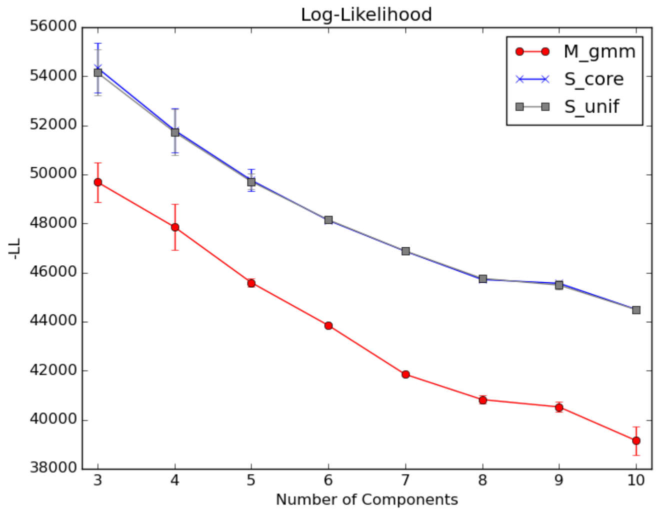


Figure 4.30: Negative log-likelihood of learned models using M_{gmm}, S_{core}, S_{unif} methods on engrailed homeodomain.

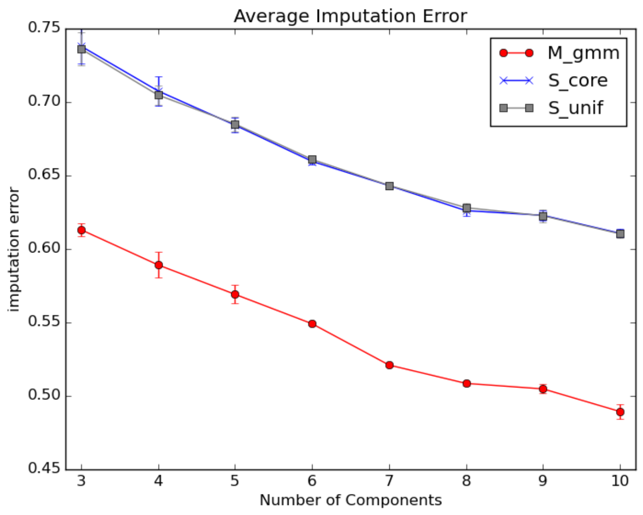


Figure 4.31: Average imputation error of learned models using M_{gmm}, S_{core}, S_{unif} methods on engrailed homeodomain.

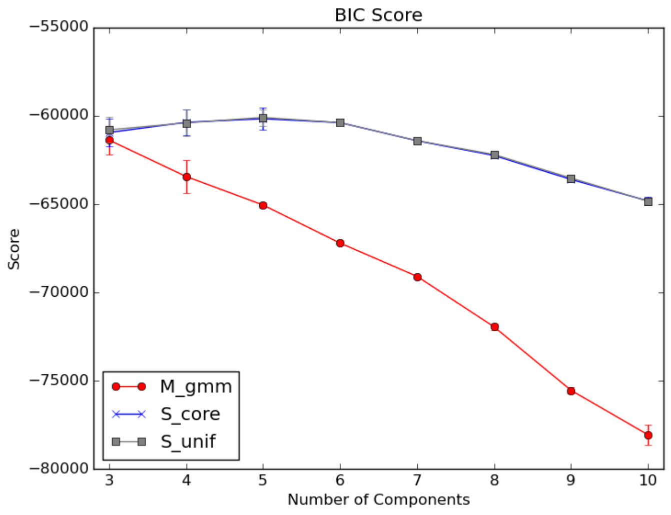


Figure 4.32: BIC score error of learned models using M_{gmm}, S_{core}, S_{unif} methods on engrailed homeodomain.

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.

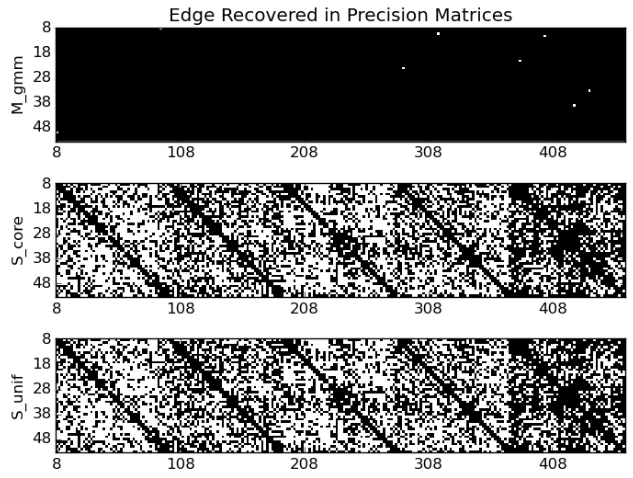


Figure 4.33: Edges recovered using M_{gmm}, S_{core}, S_{unif} methods on engrailed homeodomain.

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 51.3% and 50.5% 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 of employing coreset sampling compared to random 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

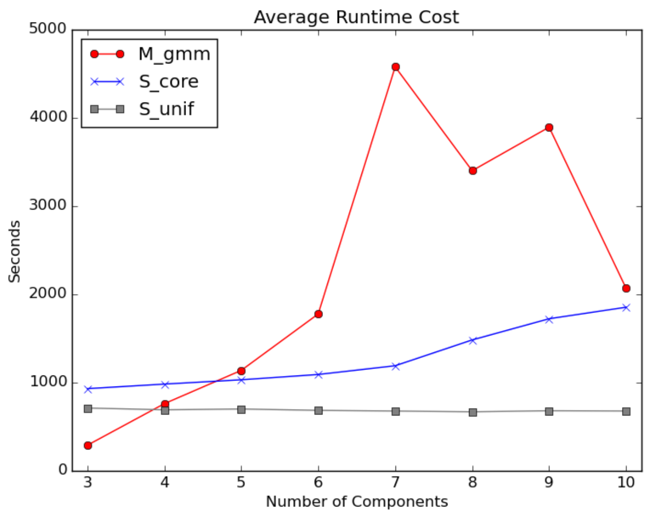


Figure 4.34: Average runtime cost of learned models using M_{gmm}, S_{core}, S_{unif} methods on engrailed homeodomain.

confirm this directly on the Engrailed Homeodomain data. Therefore, the choice between S_{unif} and S_{core} may, in practice, be determined by the user's tolerance for the higher runtime costs of S_{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_{npn}) for learning substates of engrailed homeodomain. For baseline methods, we use the traditional GMM (M_{gmm}) and a sparse GMM using cluster approximation (S_{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 $\delta = 0.1$, 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 δ_n, we evaluate the imputation error using parameter candidates, and the result is shown in 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 of S_{npn} method Figure 4.36 and 4.37 are the negative log-likelihood and the average imputation error of test set using M_{gmm}, S_{core}, and S_{npn} methods using 5 components and the selected Winsorization parameter. For log-likelihood, while S_{core} method shows 9.1% worse

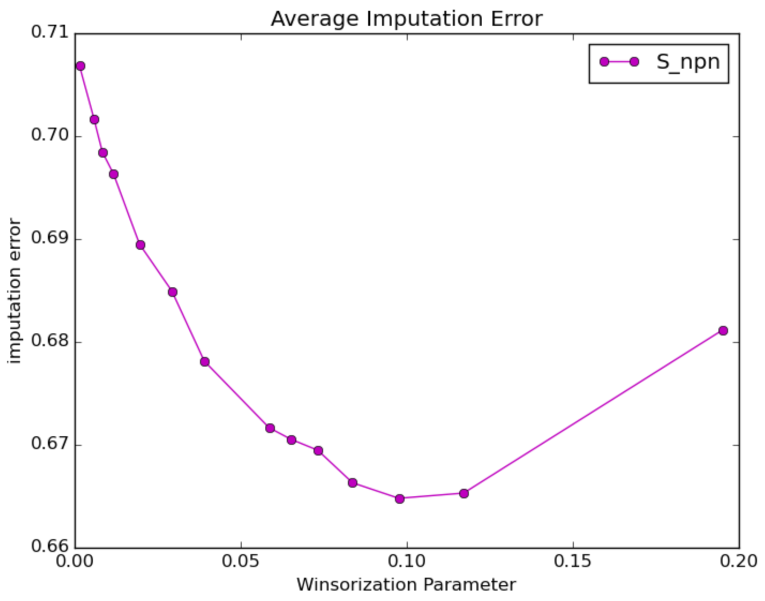


Figure 4.35: Average imputation error of models learned using S_{npn} method on engrailed homeodomain with different Winsorization parameters.

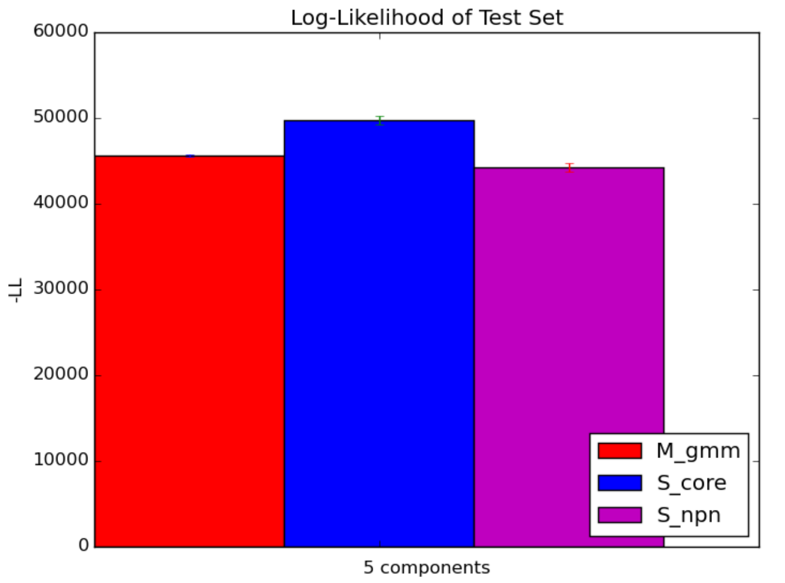


Figure 4.36: Negative log-likelihood of models learned using M_{gmm}, S_{core}, S_{npn} method on engrailed homeodomain.

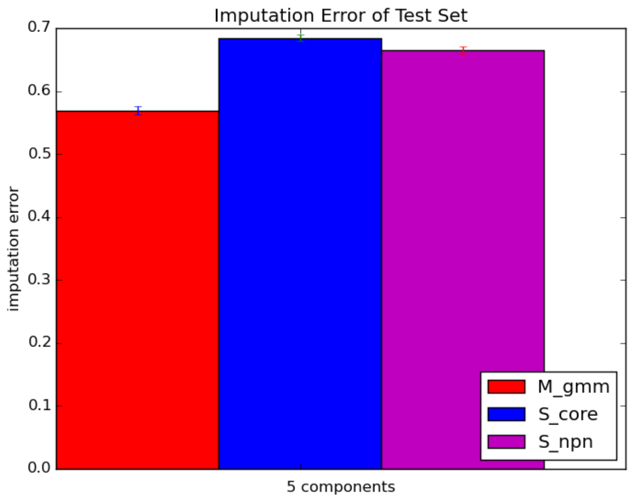


Figure 4.37: Average imputation error of models learned using M_{gmm}, S_{core}, S_{npn} method on engrailed homeodomain.

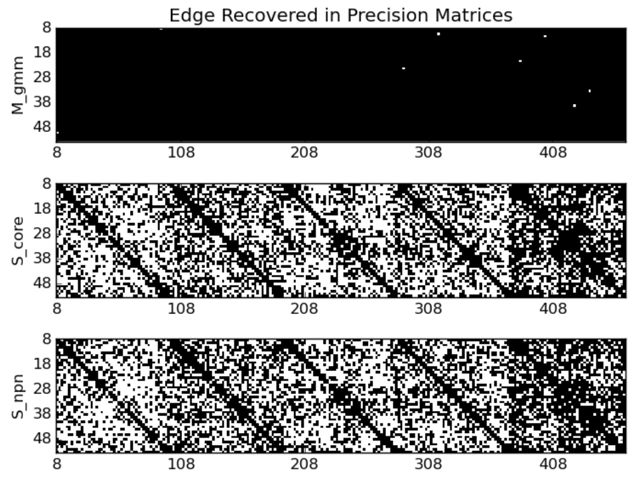


Figure 4.38: Edges recovered using M_{gmm}, S_{core}, S_{npn} methods on engrailed homeodomain.

performance than M_{gmm}, S_{npn} 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_{npn} shows 17% larger error than M_{gmm} and 2.9% improvement from M_{gmm}. The better performance of S_{npn} is due to the increased representational power of S_{npn} from the semi-parametric model than the original S_{core} method. The edges recovered in precision matrices of models using the methods are shows in 4.38, and density of models learned by S_{npn} method is 51.3%.

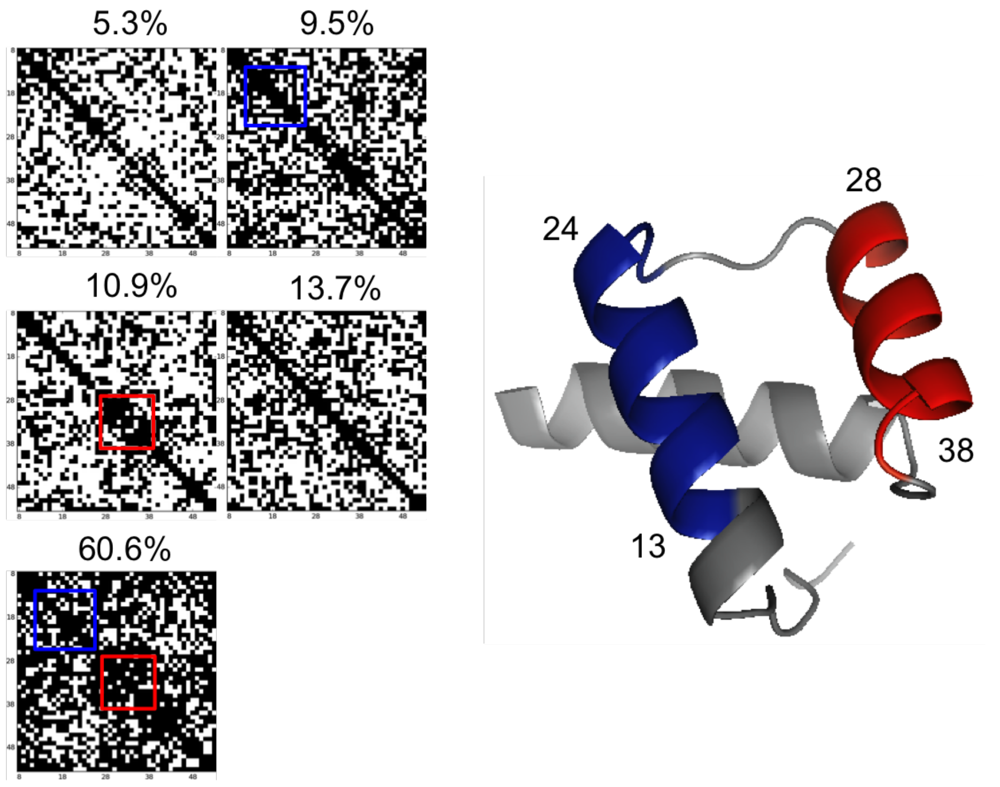


Figure 4.39: Recovered precision matrices by S_{npn} for 5 conformational substates with their weight probabilities and selected patterns matched with residues of Engrailed Homeodomain (1ENH).

Figure 4.39 shows the recovered precision matrices of conformational substates with their weight probabilities learned by S_{npn} method. The last substate has the highest weight probability (60.6%) than the other four substates, and it has the most densest precision matrix (69.5%) than that of other four states. Some of the blocks of edges in the precision matrices are marked in the Figure 4.39 with the corresponding amino acids residues of Engrailed Homeodomain (1ENH). A block of edges among the residues from 13 to 24 are recovered in the second and the fifth substates, and another block of edges from residues from 28 to 38 are recovered in the third and the fifth substates. These blocks of edges represent the direct interactions among those residues, and those residues are in the first and the second alpha helices of Engrailed Homeodomain. These results suggest that the different substates are partially differentiated by the degree of intra-helical couplings in the first and the second alpha helices.

Summary We find that the S_{npn} 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 miliseconds) 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 GGMs 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 methods assume that the given data are independent and identically distributed samples, but the 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 to 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 and variables than tested in this thesis is also needed to better understand the representational power and the limitations of our methods.

Implementation

The implementation of the methods we developed is released in PyPI, and the source code is available at `https://github.com/yoojioh/gamelanpy`.

Funding Acknowledgement

This work was supported by NSF Grant IIS-0905193 and NIH Grant US NIH P41 GM103712.

Bibliography

[1] Martin Karplus and J Andrew McCammon. Molecular dynamics simulations of biomolecules. *Nature Structural & Molecular Biology*, 9(9):646–652, 2002. 1, 2.2

[2] Hans Frauenfelder, Gregory A Petsko, and Demetrius Tsernoglou. Temperature-dependent x-ray diffraction as a probe of protein structural dynamics. 1979. 1, 2.1

[3] Hans Frauenfelder, Fritz Parak, and Robert D Young. Conformational substates in proteins. *Annual review of biophysics and biophysical chemistry*, 17(1):451–479, 1988. 1, 2.1

[4] James C Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D Skeel, Laxmikant Kale, and Klaus Schulten. Scalable molecular dynamics with namd. *Journal of computational chemistry*, 26(16):1781–1802, 2005. 1, 2.2

[5] Kevin J Bowers, Edmond Chow, Huageng Xu, Ron O Dror, Michael P Eastwood, Brent A Gregersen, John L Klepeis, Istvan Kolossvary, Mark A Moraes, Federico D Sacerdoti, et al. Scalable algorithms for molecular dynamics simulations on commodity clusters. In *SC 2006 Conference, Proceedings of the ACM/IEEE*, pages 43–43. IEEE, 2006. 1, 2.2

[6] Vijay S Pande, Ian Baker, Jarrod Chapman, Sidney P Elmer, Siraj Khaliq, Stefan M Larson, Young Min Rhee, Michael R Shirts, Christopher D Snow, Eric J Sorin, et al. Atomistic protein folding simulations on the submillisecond time scale using worldwide distributed computing. *Biopolymers*, 68(1):91–109, 2003. 1, 2.2

[7] John E Stone, James C Phillips, Peter L Freddolino, David J Hardy, Leonardo G Trabuco, and Klaus Schulten. Accelerating molecular modeling applications with graphics processors. *Journal of computational chemistry*, 28(16):2618–2640, 2007. 1, 2.2

[8] David E Shaw, Martin M Deneroff, Ron O Dror, Jeffrey S Kuskin, Richard H Larson, John K Salmon, Cliff Young, Brannon Batson, Kevin J Bowers, Jack C Chao, et al. Anton, a special-purpose machine for molecular dynamics simulation. *ACM SIGARCH Computer Architecture News*, 35(2):1–12, 2007. 1, 2.2, 4.3

[9] Katherine Henzler-Wildman and Dorothee Kern. Dynamic personalities of proteins. *Nature*, 450(7172):964–972, 2007. 2.1

[10] David D Boehr, Ruth Nussinov, and Peter E Wright. The role of dynamic conformational ensembles in biomolecular recognition. *Nature chemical biology*, 5(11):789–796, 2009. 2.1

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

and Tom Alber. Hidden alternative structures of proline isomerase essential for catalysis. *Nature*, 462(7273):669–673, 2009. 2.1

[12] Jianyin Shao, Stephen W Tanner, Nephi Thompson, and Thomas E Cheatham. Clustering molecular dynamics trajectories: 1. characterizing the performance of different clustering algorithms. *Journal of Chemical Theory and Computation*, 3(6):2312–2334, 2007. 2.3

[13] Stephan Frickenhaus, Srinivasaraghavan Kannan, and Martin Zacharias. Efficient evaluation of sampling quality of molecular dynamics simulations by clustering of dihedral torsion angles and sammon mapping. *Journal of computational chemistry*, 30(3):479–492, 2009. 2.3

[14] Xavier Daura, Wilfred F van Gunsteren, and Alan E Mark. Folding–unfolding thermodynamics of a β-heptapeptide from equilibrium simulations. *Proteins: structure, function, and bioinformatics*, 34(3):269–280, 1999. 2.3

[15] Arvind Ramanathan, Pratul K Agarwal, Maria Kurnikova, and Christopher J Langmead. An online approach for mining collective behaviors from molecular dynamics simulations. *Journal of Computational Biology*, 17(3):309–324, 2010. 2.3

[16] Arvind Ramanathan, Ji Oh Yoo, and Christopher J Langmead. On-the-fly identification of conformational substates from molecular dynamics simulations. *Journal of Chemical Theory and Computation*, 7(3):778–789, 2011. 2.3

[17] Narges S Razavian, Hetunandan Kamisetty, and Christopher J Langmead. Learning generative models of molecular dynamics. *BMC genomics*, 13(Suppl 1):S5, 2012. 2.3

[18] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009. 2.4

[19] Ivet Bahar, Ali Rana Atilgan, and Burak Erman. Direct evaluation of thermal fluctuations in proteins using a single-parameter harmonic potential. *Folding and Design*, 2(3):173–181, 1997. 2.6

[20] Turkan Haliloglu, Ivet Bahar, and Burak Erman. Gaussian dynamics of folded proteins. *Physical review letters*, 79(16):3090, 1997. 2.6

[21] RM Levy, AR Srinivasan, WK Olson, and JA McCammon. Quasi-harmonic method for studying very low frequency modes in proteins. *Biopolymers*, 23(6):1099–1112, 1984. 2.6

[22] Jeff A Bilmes et al. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *International Computer Science Institute*, 4(510):126, 1998. 3.1

[23] Lingyan Ruan, Ming Yuan, and Hui Zou. Regularized parameter estimation in high-dimensional gaussian mixture models. *Neural computation*, 23(6):1605–1622, 2011. 3.2

[24] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008. 3.2

[25] Dan Feldman, Matthew Faulkner, and Andreas Krause. Scalable training of mixture models via coresets. In *Advances in Neural Information Processing Systems*, pages 2142–2150, 2011. 3.3, 3.3.1

[26] Han Liu, John Lafferty, and Larry Wasserman. The nonparanormal: Semiparametric estimation of high dimensional undirected graphs. *The Journal of Machine Learning Research*, 10:2295–2328, 2009. 3.4, 4.2.3, 4.3.2

[27] Walter J Gehring, Markus Affolter, and Thomas Burglin. Homeodomain proteins. *Annual review of biochemistry*, 63(1):487–526, 1994. 4.3

[28] Ugo Mayor, Christopher M Johnson, Valerie Daggett, and Alan R Fersht. Protein folding and unfolding in microseconds to nanoseconds by experiment and simulation. *Proceedings of the National Academy of Sciences*, 97(25):13518–13522, 2000. 4.3

[29] Ugo Mayor, J Günter Grossmann, Nicholas W Foster, Stefan MV Freund, and Alan R Fersht. The denatured state of engrailed homeodomain under denaturing and native conditions. *Journal of molecular biology*, 333(5):977–991, 2003. 4.3

[30] Narges Sharif Razavian. Continuous graphical models for static and dynamic distributions: Application to structural biology. 2013. 5

[31] Emil Eirola and Amaury Lendasse. Gaussian mixture models for time series modelling, forecasting, and interpolation. In *Advances in Intelligent Data Analysis XII*, pages 162–173. Springer, 2013. 5